Luiz Fernando Gomes Soares
Simone Diniz Junqueira Barbosa

PUC RIO

Programando em NCL 3.0

DESENVOLVIMENTO DE APLICAÇÕES PARA O MIDDLEWARE GINGA

TeleMídia

TV DIGITAL E WEB

2ª. Edição

NCL (Nested Context Language) é a linguagem declarativa do middleware Ginga, recomendação ITU-T para serviços IPTV e padrão ISDB-T_B do Sistema Nipo-Brasileiro de TV Digital Terrestre.

Programando em NCL é o *livro oficial* da linguagem NCL, oferecendo uma visão passo a passo da linguagem, desde a concepção de aplicações bem simples, até as mais complexas, envolvendo adaptação de conteúdo, o uso de múltiplos dispositivos de exibição em redes residenciais, e geração de aplicações ao vivo, viabilizando as chamadas redes sociais.

Por ser uma linguagem declarativa de alto nível de abstração voltada a aplicações para TV Digital e para a Web, NCL pode ser facilmente apreendida, mesmo por quem não tem conhecimento de programação. Este livro tem como foco programadores-visuais, desenvolvedores de animações, Web designers, e, certamente, programadores de software. No entanto, nenhum pré-requisito de programação é exigido para o desenvolvimento de grande parte das aplicações.

Programando em NCL é dividido em três partes: uma parte introdutória, onde o leitor é apresentado informalmente, através de exemplos, à linguagem. Uma segunda parte, onde cada conceito da linguagem é explorado em toda sua potencialidade, e uma terceira parte onde conceitos mais avançados de NCL são discutidos, incluindo sua linguagem de script Lua, uma das mais utilizadas no mundo na área de jogos e entretenimento.

Luiz Fernando Gomes Soares e Simone Diniz Junqueira barbosa são professores do Departamento de Informática da PUC-RIO. Autores de vários livros, artigos em revistas e congressos nacionais e internacionais, têm atuado nas áreas de sistemas multimídia, interface humano computador e TV digital, áreas nas quais têm ministrado cursos em todo o país e no exterior.

NOTA: Muito zelo e técnica foram empegados na edição desta obra. No entanto, podem ocorrer erros de digitação, impressão ou dúvida conceitual. Em qualquer das hipóteses, solicitamos a comunicação ao nosso Serviço de Atendimento, enviando mensagem para ncl@telemidia.puc-rio.br, para que possamos esclarecer ou encaminhar a questão.

Nem a Pontifícia Universidade Católica do Rio de Janeiro nem os autores assumem qualquer responsabilidade por eventuais danos ou perdas a pessoas ou bens, originados do uso desta publicação.

Agradecimento

A todos aqueles que acreditaram no sonho e ajudaram na construção e consolidação da linguagem NCL e do middleware Ginga.

Luiz Fernando Gomes Soares

Simone Diniz Junqueira Barbosa

Apresentação

O impacto da TV digital é muito mais significativo do que a simples troca de um sistema de transmissão analógico para digital, proporcionando uma melhora da qualidade de imagem e de som. Mais do que isso, um sistema de TV digital permite um nível de flexibilidade inatingível com a difusão analógica. Um componente importante dessa flexibilidade é a possibilidade de expandir as funções do sistema por *aplicações* construídas sobre a base de um sistema padrão de referência.

Tais aplicações são programas computacionais residentes em dispositivos receptores ou provenientes de dados enviados conjuntamente com o áudio principal e o vídeo principal de um programa televisivo. A integração de uma capacidade computacional ao dispositivo receptor permite o surgimento de uma vasta gama de novos serviços e de programas de TV compostos não apenas pelo áudio principal e vídeo principal, mas também por outros áudios e vídeos, imagens, textos etc., sincronizados em uma aplicação muitas vezes guiada por interações do usuário telespectador, ao qual poderá ser delegado o controle do fluxo de um programa.

A construção de tais aplicações baseadas na linguagem NCL (Nested Context Language), linguagem declarativa padrão do Sistema Nipo-Brasileiro de TV Digital Terrestre e Recomendação H.761 da União Internacional de Telecomunicações para serviço IPTV, é o foco deste livro.

O livro tem como propósito introduzir o conceito de TV digital, analisando em profundidade não apenas o desenvolvimento de aplicações para tais sistemas, mas também o desenvolvimento de aplicações hipermídia para a Web. ***Programando em NCL*** é o livro oficial da linguagem NCL, oferecendo uma visão passo a passo da linguagem, desde a concepção de aplicações bem simples, até as mais complexas, envolvendo adaptação de conteúdo, o uso de múltiplos dispositivos de exibição em redes, e geração de aplicações ao vivo.

Sobre o Conteúdo

O livro está dividido em três partes. A *Parte I* apresenta uma introdução à TV digital, e guia o leitor, por meio de exemplos, em uma apresentação

informal da linguagem NCL. A *Parte II* explora em detalhes cada conceito da linguagem, em toda sua potencialidade, apresentando os elementos e atributos de NCL. A *Parte III* discute os conceitos mais avançados de NCL, incluindo sua linguagem de script Lua, uma das mais utilizadas no mundo na área de jogos e entretenimento.

O livro começa com uma introdução aos sistemas de TV digital e aos seus padrões de referência para codificação de áudio e de vídeo, para multiplexação e para modulação. Compondo o conjunto de padrões de referência, o *Capítulo 1* apresenta então a camada que será assunto do restante do livro: o *middleware*. São discutidos os *middlewares* de vários sistemas existentes e levantados os requisitos impostos a eles pelas aplicações. Ainda nesse capítulo, são discutidos os diversos paradigmas utilizados na especificação de aplicações e como os *middlewares* se dividem em ambientes declarativos e ambientes imperativos, conforme o suporte dado a um desses paradigmas. Por fim, uma discussão é apresentada sobre *middlewares* declarativos, sua importância e abrangência, quando então a linguagem NCL é apresentada.

O *Capítulo 2* apresenta o modelo conceitual de dados da linguagem NCL, denominado *Nested Context Model*. Nesse capítulo as diversas entidades básicas do modelo são informalmente apresentadas, e é dada uma pequena noção de como elas serão mapeadas em elementos da linguagem.

O *Capítulo3* traz uma introdução passo a passo a diversos elementos que compõem o perfil NCL 3.0 para TV digital. Nessa introdução, não existe a preocupação de definir cada elemento da linguagem e cada um de seus atributos, o que é feito na *Parte II* do livro. O intuito é já prover ao leitor a habilidade de desenvolver programas NCL simples e capacitá-lo a melhor entender e exercitar os demais conceitos apresentados na *Parte II*. A introdução é feita através de um único exemplo, construído passo a passo junto com o leitor.

O *Capítulo 4* introduz, de forma genérica, os vários perfis e módulos da linguagem NCL. O capítulo é apenas uma introdução à *Parte II* do livro dedicada à apresentação do perfil EDTV (Enhanced Digital TV), definido para sistemas de TV digital.

A *Parte I* se encerra com quatro apêndices. O *Apêndice A* discute mais detalhadamente os diversos métodos de compactação e compressão de dados, apresentando os padrões para textos, imagens, áudios e vídeos. O *Apêndice B* traz uma introdução ao padrão DSM-CC (*Digital Storage Media Command and Control*) para transporte de dados. O *Apêndice C* apresenta em detalhes o modelo NCM introduzido pelo *Capítulo 2*. Finalmente, o *Apêndice D* apresenta um documento NCL especificando uma base de conectores, conceito discutido no *Capítulo 3* e detalhado no *Capítulo 10*.

A *Parte II* do livro é dedicada à apresentação de cada elemento e cada atributo da linguagem NCL.

O *Capítulo 5* descreve a estrutura das aplicações NCL, apresentando todos os elementos e atributos do módulo *Structure* do perfil EDTV da linguagem NCL.

O *Capítulo 6* descreve os elementos e atributos para definição de regiões de apresentação em classes de dispositivos de exibição, apresentando todos os elementos e atributos do módulo *Layout* do perfil EDTV da linguagem NCL.

O *Capítulo 7* descreve os elementos e atributos responsáveis pela definição dos parâmetros de exibição de uma mídia em uma dada região. Ou seja, apresenta os elementos e atributos responsáveis pela configuração de *como* os objetos de mídia deverão ser apresentados. O capítulo também descreve como pode ser feita a navegação entre objetos de mídia por meio das teclas (setas) de navegação, e como efeitos de transição podem ser associados às apresentações dos objetos. Assim, o capítulo apresenta todos os elementos e atributos dos módulos *Descriptor*, *Timing*, *KeyNavigation*, *Transition* e *TransitionBase* do perfil EDTV da linguagem NCL.

O *Capítulo 8* apresenta os tipos de objetos de mídia permitidos em NCL, incluindo alguns tipos de objetos especiais. Nesse capítulo são também apresentados os objetos contextos, que permitem a estruturação do código para aumentar tanto a legibilidade quanto as oportunidades de reúso. No capítulo são apresentados todos os elementos e atributos dos módulos *Media* e *Context* do perfil EDTV da linguagem NCL.

O *Capítulo 9* descreve as interfaces oferecidas em NCL pelos objetos de mídia e de contexto. São definidas as âncoras de conteúdo, que permitem o uso de trechos dos objetos em relacionamentos, para um sincronismo mais fino do que quando se usa o objeto inteiro. São também definidas as propriedades, que possibilitam aos relacionamentos a manipulação de atributos dos objetos durante a execução da aplicação. O capítulo apresenta todos os elementos e atributos dos módulos *MediaContentAnchor*, *PropertyAnchor* e *CompositeNodeInterface* do perfil EDTV da linguagem NCL.

O *Capítulo 10* apresenta os elos e conectores que definem o sincronismo e, em particular, a interatividade entre os objetos de uma aplicação NCL. Conectores definem relações sem especificar seus participantes. Elos definem relacionamentos referindo-se a um conector e definindo os participantes da relação. O capítulo apresenta todos os elementos e atributos dos módulos *Link* e *Animation*, e da área funcional *Connectors* do perfil EDTV da linguagem NCL.

O *Capítulo 11* descreve como adaptar uma aplicação durante sua execução, através da avaliação de regras e a correspondente seleção de objetos ou de formas de apresentação de objetos, apresentando todos os elementos e atributos da área funcional *Presentation Control* e do módulo *SwitchInterface* do perfil EDTV da linguagem NCL.

O *Capítulo 12* descreve como metadados podem ser especificados em aplicações, apresentando todos os elementos e atributos do módulo *Metainformation* do perfil EDTV da linguagem NCL.

O *Capítulo 13* apresenta mecanismos adicionais de reúso fornecidos pela NCL: reúso de conteúdo, importação de documentos, importação de bases de um documento e reúso de objetos, intra e interdocumentos. O capítulo apresenta todos os elementos e atributos da área funcional *Reuse* do perfil EDTV da linguagem NCL.

A *Parte II* se encerra com o *Apêndice E*, que introduz o conceito de *Normal Play Time*, ou *NPT*, como definido pelo padrão MPEG-2.

A *Parte III* do livro discute os conceitos mais avançados e inovadores de NCL, e é voltada para geração de aplicações ao vivo, aplicações para múltiplos dispositivos, e extensões da linguagem para o aninhamento de objetos cujos conteúdos são códigos escritos em outras linguagens, em especial a linguagem Lua, linguagem de script de NCL.

O *Capítulo 14* discute como objetos com conteúdo hipermídia especificado por uma linguagem declarativa podem ser definidos, como eles podem se relacionar com outros objetos em um documento NCL e como exibidores (*players*) para esses objetos se comportam.

O *Capítulo 15* apresenta como dispositivos de exibição podem se registrar em classes ativas e passivas, e as várias formas de um documento NCL especificar em que dispositivos serão exibidos os seus objetos, bem como o comportamento do formatador NCL e dos vários exibidores de mídia nesses dispositivos, ilustrando os conceitos sempre com exemplos.

O *Capítulo 16* descreve os Comandos de Edição NCL. Por meio de Comandos de Edição, aplicações NCL podem ser criadas e modificadas em tempo de exibição, ou seja, podem ser editadas ao vivo.

O *Capítulo 17* discute como objetos com código imperativo podem ser definidos, como eles podem se relacionar com outros objetos em um documento NCL e como exibidores (*engines*) para esses objetos se comportam. Objetos e exibidores NCLua (objetos imperativos com código Lua), por definição parte dos perfis da linguagem NCL para TV digital, são introduzidos no capítulo.

O *Capítulo 18* se dedica aos objetos NCL com código Lua, os chamados *scripts* NCLua, apresentando uma série de exemplos de casos de

usos. Esse capítulo de co-autoria com Francisco Sant'Anna e Carlos de Salles Soares Neto foi por eles gentilmente cedidos ao livro.

A *Parte III* se encerra com três apêndices de conteúdo avançado. O *Apêndice F* discute as várias formas como os comandos de edição, introduzidos pelo *Capítulo 16*, podem ser transportados, tanto através de estruturas de dados recebidas sem solicitação, como através de estruturas de dados recebidas sob demanda. O *Apêndice G* traz uma discussão a respeito das estruturas de dados usadas em procedimentos de escalonamento, realizados tanto pelos receptores quanto pelos provedores de conteúdo das aplicações, e como elas podem ser deduzidas de uma outra estrutura de dados única chamada HTG – Hypermedia Temporal Graph. Finalmente, o *Apêndice H* apresenta a interpretação dada pelos vários exibidores de mídia, incluindo os objetos de mídia com código imperativo e declarativo, a comandos enviados pelo formatador NCL e a comandos de edição NCL. O apêndice também apresenta a interpretação dada pelo formatador quando as ações são submetidas a nós de composição NCL.

A Quem se Destina o Livro

Por ser uma linguagem declarativa de alto nível de abstração voltada a aplicações para TV Digital e para a Web, a NCL pode ser facilmente apreendida, mesmo por quem não tem conhecimento de programação. Este livro tem como foco programadores-visuais, desenvolvedores de animações, Web designers, e, certamente, programadores de software. No entanto, nenhum pré-requisito de programação é exigido para o desenvolvimento de grande parte das aplicações.

O livro é o resultado de um amplo trabalho desenvolvido no Laboratório TeleMídia do Departamento de Informática da Pontifícia Universidade Católica do Rio de Janeiro e seu conteúdo tem sido usado tanto em cursos de graduação quanto de pós-graduação em diversas áreas, como Informática, Comunicação e Artes. O livro, no entanto, não tem como alvo apenas a área acadêmica.

Como Utilizar o Livro

Existem várias formas de utilização deste livro, em cursos regulares ou como leitura auto-didática.

Aqueles interessados apenas em entender como opera um sistema de TV digital, os vários padrões envolvidos e como um leitor sem conhecimentos prévios de programação poderia desenvolver aplicações para o middleware

Ginga devem se concentrar na Parte I do livro, usando a Parte II como consulta ao desenvolver suas aplicações.

O leitor interessado em sistemas hipermídia e na utilização da linguagem NCL para desenvolvimento para Web poderia pular os Capítulos 1 e 4 da Parte I e se concentrar nos detalhes oferecidos pela Parte II do livro. Em uma primeira leitura, os Capítulos 11 a 13 da Parte II poderiam ser ignorados, bem como toda a Parte III. Para o desenvolvimento de aplicações mais complexas, no entanto, são essenciais os Capítulos 11 e 13 da Parte II, e os Capítulos 17 e 18 da Parte III.

O leitor interessado na utilização da linguagem NCL para desenvolvimento para TV digital tem um perfil parecido com o caso anterior, mas, nesse caso, os Capítulos 1 e 4 da Parte I são essenciais, mesmo em uma primeira leitura.

Profissionais para o desenvolvimento de aplicações NCL para TV digital (terrestre, IPTV, P2PTV, WebTV e Internet TV), quer sejam programadores de software, programadores-visuais, desenvolvedores de animações, Web designers etc. devem ler todo o livro e a sugestão é que seja na ordem apresentada. A Parte II e III serão sempre úteis como consulta freqüente a detalhes da linguagem.

Agradecimentos

Gostaríamos neste ponto de agradecer a todos os colegas e alunos do Departamento de Informática da PUC-Rio pela contribuição dada a este trabalho, principalmente àqueles que ao longo dos anos contribuíram na especificação da linguagem e na implementação do middleware Ginga. Em especial gostaríamos de agradecer às equipes dos laboratórios TeleMídia e SERG da PUC-Rio que viabilizaram a padronização mundial da linguagem NCL e a implementação de referência do middleware Ginga. Citar nomes aqui é um risco que não queremos correr por esquecimento.

Não poderíamos deixar de agradecer mais uma vez a Francisco Sant'Anna e Carlos de Salles Soares Neto que, em co-autoria com os autores do livro, desenvolveram o último capítulo sobre Lua e abriram mão de seus direitos de propriedade intelectual. Agradecemos também a Romualdo Resende Costa pela co-autoria do Apêndice G.

Temos muito a agradecer a Marcelo Ferreira Moreno, a Márcio Ferreira Moreno e a toda equipe do laboratório TeleMídia, por eles coordenada, pelo desenvolvimento das diversas ferramentas de autoria e da implementação de referência do middleware Ginga que deram suporte ao desenvolvimento das aplicações NCL que compõem esse livro.

Gostaríamos ainda de agradecer à Editora Campus e toda sua equipe pelo empenho e dedicação na realização da primeira edição deste livro.

Foi graças ao Departamento de Informática da PUC-Rio que tivemos toda a infra-estrutura necessária para a realização desta tarefa. Sem o apoio financeiro proporcionado pelo Ministério da Ciência e Tecnologia, em especial do programa CETIC, e do Ministério das Telecomunicações, por meio do FUNTTEL, este livro não poderia sequer ter sido iniciado.

Finalmente não poderíamos deixar de agradecer aos nossos cônjuges Isa Haro Martins e Luís Fernando Diniz Junqueira Barbosa pela compreensão pelas muitas horas de convívio roubadas durante a escrita do livro.

Luiz Fernando Gomes Soares

Simone Diniz Junqueira Barbosa

Acrônimos

AAC	Advanced Audio Coding
ABNT	Associação Brasileira de Normas Técnicas
ACAP	Advanced Common Application Platform
ADTB	Advanced Digital Television Broadcasting
API	Application Programming Interface (interface de programação de aplicações)
ARIB	Association of Radio Industries and Businesses
ATSC	Advanced Television Systems Committee
AVC	Advanced Video Coding
BDTV	Basic Digital Television (perfil TVD Básico)
BER	Bit Error Ratio (taxa de erro de bits)
BML	Broadcast Markup Language
BST-OFDM	Band Segmented Transmission – Orthogonal Frequency Division Multiplexing
COFDM	Coded Orthogonal Frequency Division Multiplexing
CPU	Central Processing Unit
DD	Dolby Digital
DOM	Document Object Model
DSM-CC	Digital Storage Media – Command and Control
DVB-T	Digital Video Broadcasting – Terrestrial
EDTV	Enhanced Digital Television (perfil TVD Avançado)
FDM	Frequency Division Multiplexing (multiplexação por divisão de frequência)

HDTV	High-Definition Television (TV de alta definição)
HE-AAC	High-Efficiency Advanced Audio Coding
HTML	HyperText Markup Language
IANA	Internet Assigned Numbers Authority
IPTV	Internet Protocol Television
ISDB-T	Integrated Services Digital Broadcasting – Terrestrial
ISDB-T_B	International Standard for Digital Broadcasting -Terrestrial
ISI	Inter-Symbol Interference
ITU-R	International Telecommunication Union – The Radiocommunication Sector
ITU-T	International Telecommunication Union – Telecommunication Standardization Sector
JIT	Just in Time
JPEG	Joint Photographic Experts Group
JVM	Java Virtual Machine
MHP	Multimedia Home Platform
MIME	Multipurpose Internet Mail Extensions
MJPEG	Motion JPEG
MP2	MPEG-1 Audio Layer II
MP3	MPEG-1 Audio Layer III
MPEG-2	Motion Picture Experts Group
NCL	Nested Context Language
NCM	Nested Context Model
OFDM	Orthogonal Frequency Division Multiplexing
OQAM	Offset Quadrature Amplitude Modulation
P2PTV	Peer-to-Peer TV
PAT	Program Association Table
PDA	Personal Digital Assistant
PDF	Portable Document Format

PMT	Program Map Table
PS	Parametric Stereo
PSI	Program-Specific Information
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase-Shift Keying
RDF	Resource Description Framework
SBR	Spectral Band Replication
SBTVD-T	Sistema Brasileiro de TV Digital – Terrestre
SDTV	Standard-Definition Television
SMIL	Synchronized Multimedia Integration Language
SVG	Scalable Vector Graphics
TS	Transport Stream
TVD	TV Digital
URI	Uniform Resource Identifier
UTC	Universal Time Coordinated
VSB	Vestigial Side Band
W3C	World Wide Web Consortium
WMA	Windows Media Audio
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language

Sumário

PARTE I
INTRODUÇÃO À TV DIGITAL E À LINGUAGEM NCL

APÊNDICES

Figuras

Listagens

Tabelas

PARTE I

Introdução à TV Digital e à Linguagem NCL

Capítulo 1

TV Digital: Fundamentos e Padrões

Um sistema de TV digital (seja ele para TV aberta ou terrestre, IPTV, WebTV, P2PTV etc.) é composto por uma série de módulos funcionais. Em sistemas para TV digital aberta, cada um dos módulos funcionais segue, em geral, padrões internacionais ou nacionais de operação. Serviços IPTV tendem também a seguir Recomendações da União Internacional de Telecomunicações (ITU-T ─ International Telecommunication Union). Este capítulo apresenta os fundamentos da TV digital interativa e os padrões de referência usuais para sistemas de TV digital aberta e IPTV, com destaque para o Sistema Brasileiro de TV Digital, em particular seu middleware, denominado *Ginga*.[1]

[1] Este capítulo se baseia em Soares e Barbosa (2008). O uso do material foi gentilmente cedido pela Sociedade Brasileira de Computação.

1.1 Introdução

A mudança da TV analógica para a TV digital se faz sentir, primeiramente, pela melhor qualidade de imagem e de som.

Em um sistema de transmissão não-guiado (caso da TV digital terrestre analógica e digital), o canal de transmissão introduz diversas interferências e ruídos no sinal original, limitando a capacidade do sistema, como apresentamos a seguir.

O ruído aleatório está presente em todo o espectro de frequências e não pode ser evitado. Na transmissão analógica, ele provoca queda na qualidade do sinal recebido, causando o aparecimento de "chuviscos" na imagem. A queda da qualidade depende da relação entre as potências do sinal e do ruído (relação S/N). À medida que a relação diminui — e isso pode acontecer pela atenuação do sinal quando nos afastamos da fonte —, diminui também a qualidade do sinal recebido. Nos sistemas digitais, o ruído pode modificar um nível digital do sinal transmitido a ponto de ele passar a ser confundido com outro nível. Uma baixa relação S/N pode, assim, aumentar a probabilidade de erro de bit. Para evitar o problema, todos os padrões de transmissão dos sistemas de TV digital terrestre utilizam códigos corretores de erro. Se a taxa de erro estiver abaixo de um limiar, o código corretor é capaz de corrigir todos os erros introduzidos pelo canal e não haverá queda da qualidade da imagem. Entretanto, se a taxa de erros estiver acima do limiar, o sistema não será capaz de corrigir o código transmitido e poderá até mesmo introduzir erros, em vez de corrigi-los. Por isso, é usual dizer que, na TV digital, teremos um sinal perfeito, sem nenhum "chuvisco" ou nenhum sinal. Como a relação S/N decai com a distância do transmissor, caso o sistema não esteja bem dimensionado, pode haver problemas de cobertura, com a introdução das chamadas "áreas de sombra".

A deterioração de um sinal recebido também pode dar-se pelos múltiplos percursos seguidos desde sua origem, como habitual nos sistemas de transmissão por radiodifusão. Cada um dos percursos pode apresentar atenuação e atraso diferentes dos demais, fazendo com que o sinal recebido seja formado pela sobreposição dos vários sinais provenientes dos diferentes caminhos, como ilustra a Figura 1.1. O efeito em uma TV analógica é o aparecimento de "fantasmas". Já na TV digital, múltiplos percursos podem produzir uma interferência entre símbolos (ISI — *Inter-Symbol Interference*), isto é, a sobreposição entre os bits transmitidos, como mostra o sinal recebido na casa receptora da Figura 1.1. Se o intervalo de duração de um símbolo (um símbolo representa um padrão de bits) transmitido for muito pequeno, bem como o intervalo de guarda entre os símbolos, a ponto de a diferença de retardo entre os múltiplos percursos causar a superposição de símbolos diferentes, a taxa de erros de recepção poderá aumentar. Se nenhuma

contramedida for tomada, a ISI pode inviabilizar a recepção. Como sempre, na TV digital, é tudo ou nada.

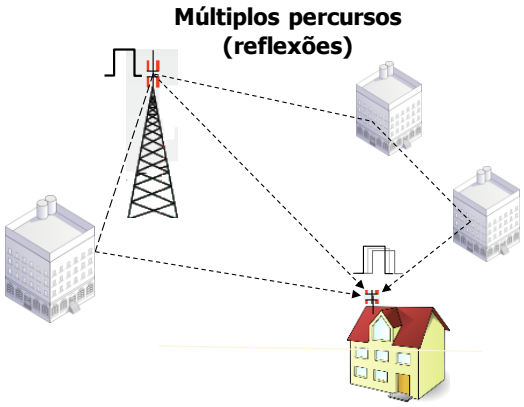


Figura 1.1 Efeito de múltiplos percursos.

O ruído impulsivo é outra fonte de degeneração de um sinal. Ruído impulsivo é aquele decorrente, por exemplo, de interferências de motores elétricos (aparelhos eletrodomésticos, motores industriais, elevadores etc.), veículos automotores, transformadores de distribuição de energia elétrica, descargas atmosféricas etc. Como o ruído aleatório, o ruído impulsivo provoca, na transmissão analógica, a queda na qualidade do sinal recebido (aparecimento de "chuviscos" na imagem). No caso dos sistemas de TV digital, se o ruído demorar tempo suficiente para causar uma rajada de erros em símbolos consecutivos do sinal, o sistema corretor de erros pode não ser capaz de corrigir o código transmitido, levando à queda da recepção. Sistemas de TV digital devem ser robustos o suficiente para evitar o problema.

Enfim, por pior que o sinal analógico chegue à televisão, o telespectador consegue recebê-lo. No caso do sinal digital, ou ele chega perfeito ou não será recebido. Por isso, é importante avaliar qual sistema de transmissão é o mais robusto, para garantir a chegada adequada do sinal digital nas residências. Note também que algumas dessas interferências, sempre presentes na TV digital terrestre, podem não ser significante em meios de transmissão guiados, como, por exemplo, nos meios utilizados na TV a cabo.

Melhor qualidade de imagem e som também se fazem sentir com a aplicação de técnicas de compressão de dados em sinais digitais, permitindo a produção de sinais de maior resolução na fonte, ou seja, sinais de melhor qualidade.

A TV analógica tem uma qualidade de imagem próxima à da TV digital padrão (SDTV), isto é, 30 quadros por segundo, com cada quadro sendo composto por 480 linhas ativas entrelaçadas, em uma TV com relação de aspecto igual a 4×3. O critério para o dimensionamento desses parâmetros de imagem sempre é atender a acuidade visual (o que o olho humano consegue discernir) dentro das limitações tecnológicas. Como mostra a Figura 1.2, a qualidade SDTV foi projetada com um tamanho de pixel tal que a acuidade visual de um minuto de grau fosse respeitada, a uma distância de sete vezes a altura da tela. Nessa distância, o ângulo de visualização horizontal é de 10 graus (suficiente para que o ser humano capture informações).

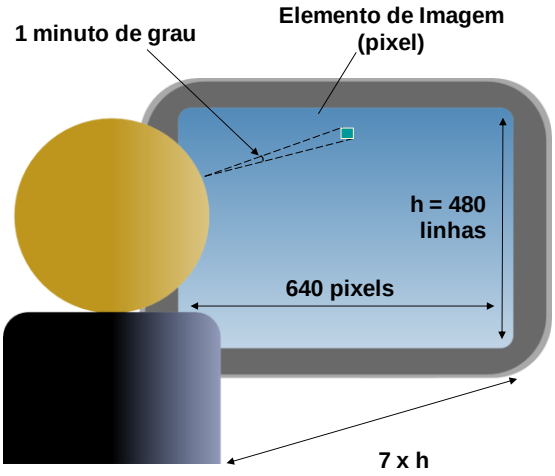


Figura 1.2 Parâmetros da SDTV.

Na qualidade SDTV, recomendação BT 601-4 [ITU-R BT.601-4, 1994] (ver Apêndice A), são gerados aproximadamente 162 Mbps (29,97 quadros/seg × 480 linhas/quadro × 704 pixels/linha × (8 bits de luminância + 8 bits de crominância)). Essa taxa de bits deve poder ser transmitida em um canal de TV de 6 MHz (no caso da TV terrestre do Brasil), que suporta uma taxa de bits da ordem de 19,3 Mbps. As técnicas atuais de compressão permitem essa taxa de compressão (162:19,3) e muito mais. Temos assim, com a tecnologia digital, a possibilidade de ter uma qualidade de imagem ainda melhor que a obtida na SDTV.

De fato, a TV digital terrestre possibilita, no mesmo canal de 6 MHz, a transmissão da qualidade chamada de alta definição (HDTV), com os seguintes parâmetros: 30 quadros por segundo, 1.920 pixels por linha, 1.080 linhas, com a acuidade visual de 1' de grau obtida a uma distância de

aproximadamente três vezes a altura da tela, que passa a ter uma relação de aspecto de 16×9. Nessa distância e com essa relação de aspecto, o ângulo de visualização horizontal é de 30 graus, possibilitando ao telespectador maior sensação de presença na cena (imersão).

Graças às técnicas de compressão de dados, a melhora em um sistema de TV digital também se faz sentir na qualidade do áudio. Dentro da mesma faixa de 6 MHz de um sistema terrestre é também possível transmitir áudio no padrão 5.1 (multicanal), dando ao telespectador, agora sob o aspecto da sensibilidade auditiva, maior sensação de imersão na cena.

As técnicas de compressão digital permitem também a alternativa de se ter vários programas de menor qualidade de definição na TV digital terrestre; por exemplo, de se ter vários programas de qualidade SDTV, em vez de se ter um único programa (áudio principal e vídeo principal) de maior qualidade (HDTV) ocupando toda a faixa de 6 MHz. Essa nova possibilidade advinda do emprego da tecnologia digital é o que chamamos de *multiprogramação*.

Todos os programas em um mesmo canal de 6 MHz podem estar relacionados a um mesmo conteúdo ou não. Um exemplo de relacionamento poderia ser a transmissão de um jogo de futebol, no qual uma câmera poderia estar posicionada atrás de um dos gols, captando também o som proveniente da torcida ali posicionada, outra câmera poderia estar posicionada no outro gol, ainda uma outra no centro do estádio etc. Ao telespectador seria dada a opção de escolher qual câmera visualizar em um determinado momento ou mesmo visualizar todas simultaneamente. Quando o conteúdo da multiprogramação está relacionado, muitas vezes o processo é chamado de *multicâmera*.

Em uma multiprogramação, entretanto, os vários programas podem ser completamente independentes, o que nos leva a revisitar o conceito de *canal*. Na TV analógica, o canal de 6 MHz (ou seja, o canal de frequência) era confundido com o canal de programação (normalmente associado a uma radiodifusora). Agora não precisa ser mais assim. Em um mesmo canal de frequência podemos ter vários programas diferentes.

O impacto da TV digital é muito mais significativo, no entanto, do que a simples troca de um sistema de transmissão analógico para digital, e muito mais do que uma melhora da qualidade de imagem e de som. Mais do que isso, um sistema de TV digital permite um nível de flexibilidade inatingível com a difusão analógica. Um componente importante dessa flexibilidade é a possibilidade de expandir as funções do sistema por *aplicações* construídas sobre a base de um sistema padrão de referência.

Tais aplicações são programas computacionais, residentes em um dispositivo receptor ou provenientes de dados enviados conjuntamente (multiplexados) com o áudio principal e o vídeo principal de um programa

televisivo. Assim, uma das características mais importantes da TV digital é a integração de uma capacidade computacional significativa no dispositivo receptor, permitindo o surgimento de uma vasta gama de novos serviços, como a oferta de guias eletrônicos de programas, o controle de acesso e a proteção de conteúdo, a distribuição de jogos eletrônicos, o acesso a serviços bancários (*T-banking*), serviços de saúde (*T-health*), serviços educacionais (*T-learning*), serviços de governo (*T-government*) e, em especial, os *programas não-lineares*.

Um programa não-linear é um programa de TV composto não apenas pelo áudio principal e vídeo principal, mas também por outros dados transmitidos em conjunto. Esses dados se constituem de outros áudios e vídeos, além do principal, imagens, textos etc., e uma aplicação relacionando temporalmente e espacialmente todos esses *objetos de mídia*, incluindo o vídeo principal e o áudio principal. Esses relacionamentos podem ser guiados por interações do usuário telespectador, ao qual poderá ser delegado o controle do fluxo de um programa televisivo, determinando se um conteúdo específico deve ser exibido ou não e, em sendo, a forma como será exibido. Como o fluxo de um programa televisivo[2] deixa de ser contínuo em sua concepção e com vários caminhos alternativos de exibição, esse programa é chamado de não-linear.

Resumindo, na TV analógica, vídeo, áudio e alguma informação de dados limitada (como o closed captioning) são transmitidos multiplexados, de tal forma que um receptor, de projeto relativamente simples, possa decodificar e reagrupar os vários elementos do sinal para produzir um programa. Como tal, um programa completo é transmitido em sua forma final. Em um sistema de TV digital, contudo, níveis adicionais de processamento são necessários. O receptor processa o fluxo digital extraindo do sinal recebido uma coleção de elementos do programa (vídeo principal, áudio principal, outros vídeos e áudios, imagens, textos etc.), que denominamos anteriormente *objetos de mídia*, que compõem o serviço selecionado pelo consumidor. Essa seleção é feita usando informações do sistema e do serviço, que também são transmitidas. Uma vez que os objetos de mídia tenham sido recebidos, eles são exibidos de acordo com um documento de especificação (parte da aplicação), também recebido junto com os dados, que ao ser processado se encarrega da exibição final.

A capacidade computacional necessária ao novo sistema pode ser integrada no próprio dispositivo exibidor: um aparelho de TV digital, um

---

[2] Como o leitor já deve ter percebido, muitos termos usados em TV digital devem ser adjetivados para sua inteira compreensão. Visando simplificar o texto, vamos adotar a seguinte convenção: chamaremos o programa televisivo simplesmente de *programa* ou *programa de TV*; um programa computacional, incluindo seus dados, será chamado de *aplicação* ou *aplicativo*. Note que, na TV digital, um programa não-linear é uma aplicação contendo o áudio e vídeo principal entre os seus objetos de mídia.

celular, um PDA etc. Como há um grande legado de TVs analógicas, uma solução simples para oferecimento desses novos serviços nesses sistemas receptores legados é utilizar, junto à TV, um "sistema de processamento" capaz de tratar corretamente o sinal recebido, decodificá-lo e exibi-lo de forma consistente: não apenas o áudio e vídeo principal, mas as aplicações e os serviços avançados. Esse sistema de processamento é chamado de *conversor digital* (ou *set-top box*).

A Figura 1.3 apresenta o diagrama em blocos de um receptor de TV digital terrestre, que pode estar embutido ou ser externo ao aparelho exibidor, como mencionamos. O sinal recebido, depois de sintonizado e demodulado (retirado do canal de frequência), é separado (demultiplexado) e entregue para os decodificadores de áudio e de vídeo, e para processamento em uma CPU. Note, na figura, que o receptor tem acesso a outra rede (rede externa na figura) através da qual pode receber ou enviar dados, conforme comandado pela aplicação recebida. O canal de acesso a essa rede é chamado de *canal de retorno* ou *canal de interatividade*, conforme veremos no decorrer do capítulo. Em um sistema de IPTV ou P2PTV, usualmente o sinal não é modulado, entrando direto em um módulo demultiplexador após ser sintonizado; mais ainda, é também usual que o canal de interatividade seja constituído pela mesma rede de transmissão por onde vêm os dados multiplexados de uma transmissora.

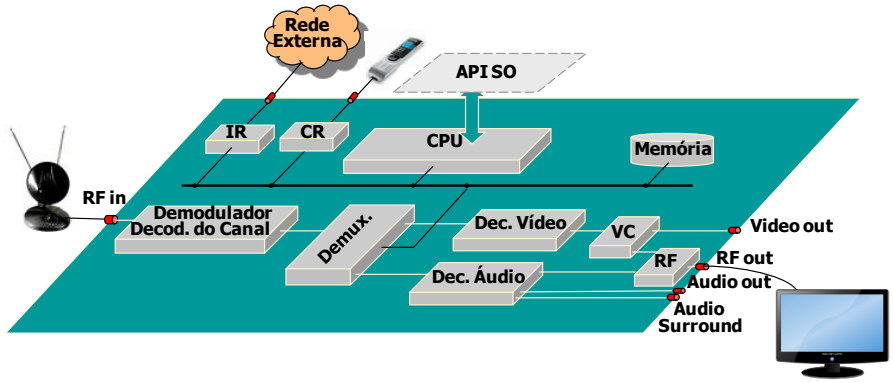


Figura 1.3 Receptor de TV digital.

1.2 Sistema de TV Digital: Visão Geral

Um sistema de TV digital é um sistema típico cliente/servidor. Em um sistema de TV digital terrestre, o servidor compõe o ambiente de uma

radiodifusora (parte esquerda da Figura 1.4) ou de um provedor de conteúdo, e o cliente, o ambiente do usuário telespectador (parte direita da Figura 1.4).

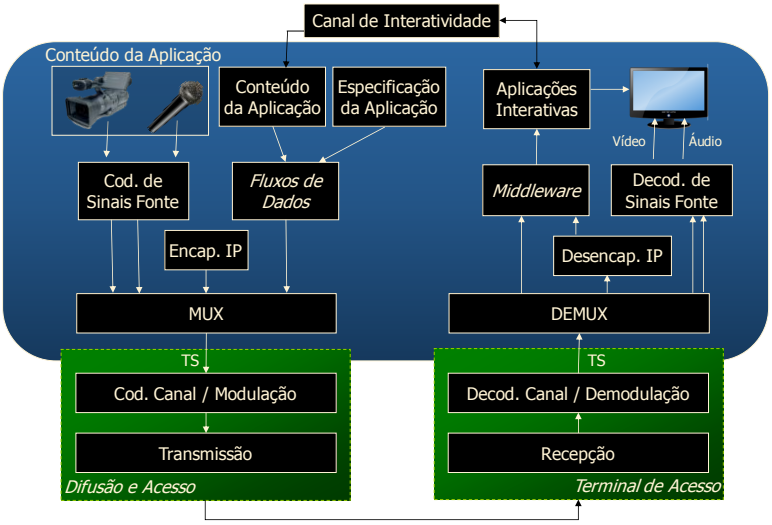


Figura 1.4 Sistema de TV digital terrestre.

Um programa é composto de um áudio principal e um vídeo principal, capturado ao vivo de uma câmera, como na Figura 1.4 (lado esquerdo) ou proveniente de um servidor de vídeo. Um programa pode também conter dados adicionais, incluindo o documento da aplicação que define o relacionamento entre os vários objetos de mídia definidos nesses dados. Os dados adicionais podem vir encapsulados no formato IP ou em outro formato, como veremos oportunamente. O vídeo e o áudio são entregues aos codificadores digitais, responsáveis pela geração dos respectivos fluxos de vídeo principal e áudio principal comprimidos. Esses fluxos, mais os fluxos dos outros dados, são então multiplexados em um único sinal, denominado fluxo de transporte (TS — *Transport Stream*). Após, no caso de um sistema terrestre, o fluxo de transporte é modulado para um canal de frequência e transmitido no ar. Já em um serviço IPTV, em geral, o fluxo de transporte é transmitido diretamente em pacotes IP sem qualquer modulação.

Do lado da recepção (lado direito da Figura 1.4), o sinal é recebido e demodulado (retirado do canal de frequência sintonizado), no caso da TV terrestre, e entregue ao demultiplexador, que separa os fluxos de áudio principal e vídeo principal (entregando-os a decodificadores apropriados) dos fluxos de dados, que são entregues para processamento.

O processamento dos dados recebidos pode demandar novos dados, obtidos pelo canal de interatividade. Dados também podem ser enviados pela aplicação à emissora ou outro destino qualquer na rede do canal de interatividade.

Um sistema de TV digital é composto por um conjunto de padrões que regulam cada um dos procedimentos descritos na Figura 1.4, chamados padrões de referência. Nas subseções seguintes discutiremos vários desses padrões, seguindo a Figura 1.5 no sentido de cima para baixo.

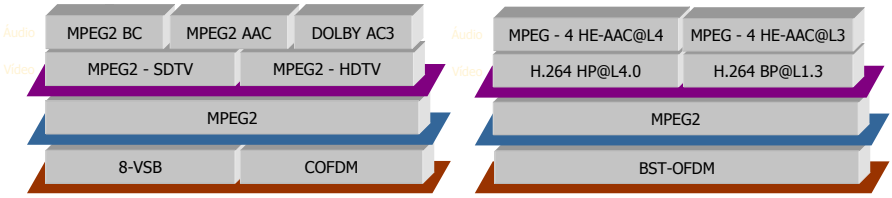


Figura 1.5 Padrões de referência para TV digital terrestre.

1.2.1 Codificação do Áudio

Um sinal digital carrega, em geral, muita informação redundante. Se eliminarmos essa redundância, conseguiremos reduzir em muito a quantidade de bits gerados.

Quando eliminamos apenas a redundância de um sinal, não há perda de informação, e dizemos que houve uma compactação ou compressão sem perdas. No entanto, podemos também diminuir a quantidade de bits com alguma perda de informação. Dependendo de quem for o usuário de uma informação, parte da informação pode ser considerada pouco útil. Raramente é necessário manter o sinal original 100% intacto, no caso das mídias de vídeo, áudio e imagens estáticas, uma vez que o usuário final perderia de qualquer forma parte da informação por limitações físicas; por exemplo, limitações do ouvido e olho humanos. A quantidade de informação que podemos perder pode ser dependente do usuário, mas também pode depender da tarefa em desenvolvimento: por exemplo, perder um pouco da nitidez de um vídeo em uma videotelefonia pode ser perfeitamente aceitável, enquanto a perda da qualidade do vídeo pode ser inadmissível em uma aplicação médica. Quando na redução dos dados gerados há perda de informação, dizemos que houve uma compressão com perdas ou simplesmente compressão. Quando essa perda não é perceptível pelo ser humano, dizemos que houve uma compressão perceptualmente sem perdas.

Em um sistema de TV digital, técnicas de compressão perceptualmente sem perdas são empregadas no áudio gerado, levando em conta o modelo psicoacústico humano. O modelo divide o domínio de frequência audível em várias bandas, e sobre elas aplica filtros, levando em conta a sensibilidade do ouvido humano, o mascaramento de frequência, o mascaramento temporal e a forma como o ser humano percebe áudio multicanal, como discutido no Apêndice A. O resultado é um áudio de alta qualidade e com baixa taxa de bits gerada.

O sistema americano de TV digital terrestre usa o padrão de codificação AC3/ATSC [ATSC AC3, 2005], antigo nome da codificação chamada hoje de Dolby Digital (DD). A codificação, proprietária da empresa Dolby, utiliza seis canais de áudio, sendo cinco com alta qualidade (20 a 20 KHz) e um apenas para as baixas frequências (20 a 120 Hz).

O sistema europeu de TV digital terrestre usa o padrão MP2 (MPEG-1 Layer 2) [ISO/IEC IS 11172-3, 1993], mas algumas implementações seguem o padrão MPEG-2 Advanced Audio Coding (AAC) [ISO/IEC 13818-7, 1997], também adotado pelo sistema terrestre japonês.

O MPEG-2 AAC, também conhecido como MPEG-2 Parte 7 ou MPEG-4 Parte 3, foi projetado como um codec (codificador/decodificador), de desempenho melhor em relação ao MP3 (MPEG-1 Layer 3) [ISO/IEC IS 11172-3 1993], para codificação de áudio em taxas de bits médias a altas. A AAC é considerada o estado da arte para áudio de alta qualidade em uma taxa de bits típica de 128 Kbps. Abaixo dessa taxa, a qualidade do áudio começa a degradar, o que pode ser compensado por técnicas de melhoramento, como SBR (*Spectral Band Replication*) e PS (*Parametric Stereo*).

A SBR é uma técnica de extensão que permite a mesma qualidade de som do AAC a aproximadamente metade da taxa de bits. A combinação de AAC e SBR é chamada de HE-AAC (High efficiency-AAC) versão 1 [ISO/IEC 14496-3, 2004], também conhecida como “aacPlus v1”.

A PS aumenta a eficiência de codificação ainda mais, através de uma representação paramétrica da imagem estéreo de um sinal de entrada. A combinação de AAC, SBR e PS é chamada de HE-AAC (High efficiency-AAC) versão 2 [ISO/IEC 14496-3, 2004].

Note que o HE-AAC, definido no padrão MPEG-4, é um superconjunto do núcleo AAC, que estende a alta qualidade de áudio AAC para taxas de bits mais baixas. Decodificadores HE-AAC v2 são capazes de decodificar fluxos HE-AAC v1 e fluxos AAC. Por sua vez, decodificadores HE-AAC v1 são também capazes de decodificar fluxos AAC.

O sistema brasileiro de TV digital terrestre adotou o padrão MPEG-4 para a codificação do áudio principal de um programa [ABNT NBR 15602-2, 2011], com as características apresentadas na Tabela 1.1.

Tabela 1.1 Codificação de áudio no sistema brasileiro de TV digital terrestre

	Receptores Fixos e Móveis	**Receptores Portáteis**
Padrão	ISO/IEC 14496-3 (MPEG-4 AAC)	ISO/IEC 14496-3 (MPEG-4 AAC)
Nível e perfil	AAC@L4 (para multicanal 5.1) HE-AAC v1@L4 (para estéreo)	HE-AAC v2@L3 (dois canais)
Taxa de amostragem	48kHz	48kHz

1.2.2 Codificação do Vídeo

Um sinal de vídeo digital carrega muita informação redundante, espacialmente (redundância *intraquadro*) e temporalmente (redundância *interquadros*).

Em um quadro de vídeo não ocorre, em geral, mudança abrupta de um pixel para um pixel consecutivo, exceto nos contornos dos objetos. Em particular, podemos empregar a codificação JPEG [ISO/IEC 10918-1, 1994] em cada quadro do vídeo para tirarmos proveito dessa redundância espacial. Essa técnica, aplicada quadro a quadro, constitui a base da codificação chamada MJPEG (Motion JPEG). Entretanto, ao empregarmos essa codificação, estaremos levando em conta apenas a redundância intraquadro, quando a maior redundância pode estar nas informações contidas em quadros consecutivos (redundância interquadros), o que é explorado no padrão MPEG vídeo.

No padrão MPEG-2 vídeo [ISO/IEC 13818-2, 2000], cada bloco (conjunto de 8×8 pixels) pode ser codificado usando apenas a informação intraquadro. Quadros em que todos os blocos são codificados dessa forma são denominados *quadros I*.

Um macrobloco (conjunto de 16×16 pixels) pode também ser codificado de forma preditiva. A predição MPEG pode ser feita baseada em quadros passados e em quadros futuros da sequência de um vídeo. Quando a predição é feita baseada em um quadro passado, é codificado o erro de predição (diferença do macrobloco que se quer codificar para o macrobloco de referência do quadro passado), usando os mesmos procedimentos usados para os blocos intraquadros, e o vetor de movimento (que dá a posição relativa do macrobloco que se quer codificar para o macrobloco de referência do quadro

passado). Quadros codificados usando esse tipo de predição são chamados *quadros P*. No MPEG-2 vídeo, a predição é sempre baseada no primeiro quadro do tipo I ou P anterior.

Macroblocos também podem ser codificados baseados no primeiro quadro I ou P, posterior ou anterior. Nesse caso teremos dois quadros de referência para a procura do melhor casamento. A codificação pode ser realizada baseada no quadro anterior, no quadro posterior ou, ainda, na interpolação dos quadros anterior e posterior. Quadros codificados usando esse tipo de predição são chamados *quadros B*.

O padrão MPEG-2 vídeo admite vários formatos de quadros e diferentes resoluções para as componentes de crominância (subamostragem de crominância). De fato, o MPEG-2 especifica vários conjuntos de parâmetros de restrição, que são definidos nos seus *perfis* e *níveis*. Um perfil especifica as facilidades de codificação que serão utilizadas (por exemplo, tipos de quadros, subamostragem etc). Um nível especifica a resolução dos quadros, as taxas de bits etc.

Para a TV digital, o MPEG-2 vídeo define o perfil principal (utiliza os quadros I, P e B e uma subamostragem de cor 4:2:0) e os níveis principal (720 pixels/linha × 480 linhas × 30 quadros/seg) e alto (1.920 pixels/linha × 1080 linhas × 30 quadros/seg), respectivamente para SDTV e HDTV. Os sistemas americano, europeu e japonês de TV digital terrestre adotam o MPEG-2 vídeo como seu padrão para codificação do vídeo.

O sistema brasileiro de TV digital terrestre emprega uma técnica de codificação mais recente: o H.264 [ISO/IEC 14496-10, 2005], também conhecido como MPEG-4 Parte 10 ou MPEG-4 AVC (*Advanced Video Coding*). O objetivo do projeto H.264/AVC foi criar um padrão capaz de prover um vídeo de boa qualidade a uma taxa substancialmente mais baixa do que os padrões anteriores (MPEG-2 entre eles), sem aumentar muito sua complexidade, para facilitar uma implementação barata e eficiente. Um objetivo adicional era fornecer flexibilidade suficiente para permitir sua aplicação em uma variedade de sistemas de taxas de transmissão altas e baixas, e também de resoluções de vídeo altas e baixas.

O H.264 contém várias facilidades novas que permitem uma compressão de vídeo muito mais eficiente e flexível. As técnicas empregadas fazem do H.264 um padrão significativamente melhor do que os padrões anteriores, sob uma variedade de circunstâncias e ambientes de aplicação, em particular o ambiente de TV digital, onde ele oferece um desempenho bem melhor do que o MPEG-2 vídeo. Em especial nas situações de alta resolução e alta taxa de bits, o padrão H.264, para a mesma qualidade de vídeo, gera uma taxa 50% ou ainda menor do que a taxa gerada pelo padrão MPEG-2.

Da mesma forma que o padrão MPEG-2 vídeo, o H.264 é dividido em perfis e níveis. No caso do sistema brasileiro de TV digital terrestre, são usados os perfis alto (HP) para os receptores fixos e móveis e o perfil base (BP) para receptores portáteis [ABNT NBR 15602-1, 2007], conforme indica a Tabela 1.2.

Tabela 1.2 Codificação de vídeo no sistema brasileiro de TV digital

	Receptores Fixos e Móveis	**Receptores Portáteis**
Padrão	ITU-T H.264 (MPEG-4 AVC)	ITU-T H.264 (MPEG-4 AVC)
Nível e perfil	HP@L4.0	BP@L1.3
Número de linhas do nível	480 (4:3 e 16:9), 720 (16:9), 1080 (16:9)	SQVGA (160×120 ou 160×90), QVGA (320×240 ou 320×180) e CIF (352×288); todos em 4:3 e 16:9
Taxa de quadros	30 e 60 Hz	15 e 30 Hz

1.2.3 Sistema de Transporte

Tanto os sistemas americano, europeu e japonês quanto o sistema brasileiro de TV digital terrestre padronizam a forma como informações audiovisuais, juntamente com os dados, devem ser multiplexadas em um único fluxo. Todos os sistemas mencionados adotam o mesmo padrão de multiplexação, MPEG-2 System [ISO/IEC 13818-1, 2000], com poucas variações. Muitos sistemas IPTV também adotam o MPEG-2 System.

O MPEG-2 System adiciona aos fluxos elementares de áudio principal e vídeo principal informações para suas exibições sincronizadas. A sincronização é realizada seguindo o paradigma de *eixo do tempo* (*timeline*) pela adição de *selos de tempo* (*timestamps*) a conjuntos de amostras codificadas de vídeo e áudio, baseadas em um relógio compartilhado. A Figura 1.6 ilustra o procedimento.

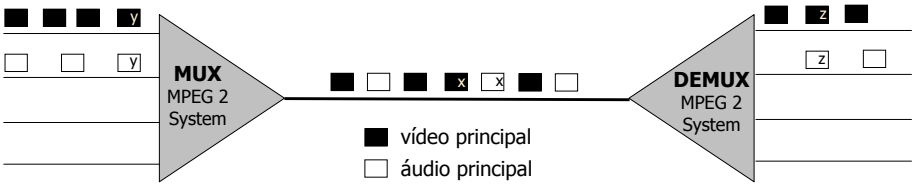


Figura 1.6 Multiplexação do áudio principal e vídeo principal.

A geração e a multiplexação dos fluxos de dados (dos outros dados além do áudio e vídeo principal) também são determinadas pelo padrão. Vejamos primeiro a multiplexação, partindo de fluxos já gerados.

1.2.3.1 Multiplexação de Dados

Os fluxos de dados podem ser transportados através de serviços síncronos (fracamente acoplados), sincronizados (fortemente acoplados) ou assíncronos (desacoplados).

O serviço de transporte *síncrono* assume que os fluxos de dados são sincronizados entre si, seguindo o paradigma de eixo do tempo (*timeline*) pela adição de selos de tempo (*timestamps*). Os fluxos de dados, no entanto, não estão relacionados à temporização dos fluxos de áudio principal e vídeo principal. A Figura 1.7 ilustra o procedimento. Note que os selos de tempo associados aos fluxos de dados são diferentes daqueles associados aos fluxos de áudio principal e vídeo principal.

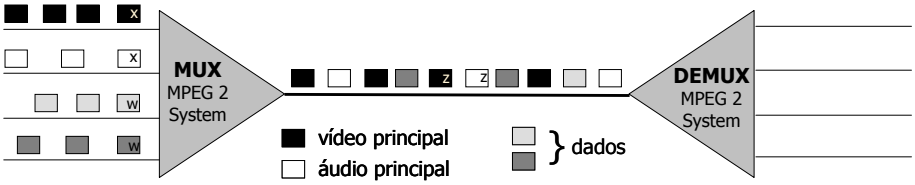


Figura 1.7 Transporte de dados síncronos.

Exibição de comerciais, estatísticas de esportes, sistemas de ajuda, entre muitos outros, são exemplos de aplicações que somente possuem sentido no contexto da exibição do vídeo principal. Portanto, com a possibilidade de envio desses dados por difusão, os padrões de TV digital devem estabelecer mecanismos que permitam a sincronização desses dados com os fluxos de áudio principal e vídeo principal.

O serviço de transporte *sincronizado* assume que os fluxos de dados são sincronizados entre si e também com os fluxos de áudio principal e vídeo principal, sempre seguindo o paradigma de eixo do tempo (*timeline*), pela adição de selos de tempo (*timestamps*). A Figura 1.8 ilustra o procedimento. Note que os selos de tempo associados aos fluxos de dados são iguais àqueles associados aos fluxos de áudio principal e vídeo principal.

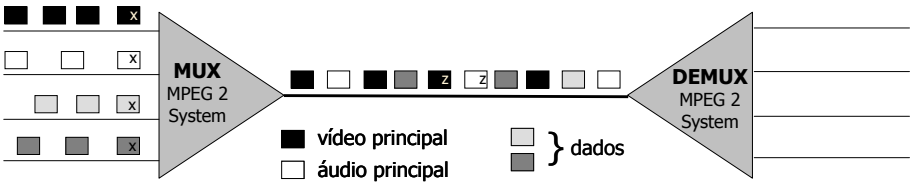


Figura 1.8 Transporte de dados sincronizados.

Fluxos de dados no transporte síncrono e sincronizado só permitem o sincronismo quando o instante de tempo de sincronização é determinístico. Aplicações interativas, nas quais a sincronização é dada em um tempo aleatório determinado pelo usuário telespectador, aplicações nas quais o conteúdo é gerado em tempo de exibição e não se pode determinar o tempo exato em que eventos ocorrerão e aplicações cujo conteúdo é determinado em tempo real (em tempo de exibição) não podem ser sincronizadas usando o serviço síncrono ou o serviço sincronizado. O suporte, nesse caso, é dado pelo serviço de transporte assíncrono.

Serviços *assíncronos* implicam que nenhuma marca de tempo (*timestamp*) é associada aos dados. No entanto, pode haver sincronismo entre os vários objetos transportados e entre esses objetos e os fluxos de áudio e/ou vídeo principal. Para tanto, o paradigma de sincronização *timeline* é abandonado, sendo substituído pelo paradigma de causalidade/restrição (também chamado de orientado a evento).

Nos serviços assíncronos, junto com os dados é mandado o documento da aplicação (hachurado na Figura 1.9), que especifica, em uma linguagem de programação específica, o comportamento relativo dos dados, do áudio principal e do vídeo principal, no tempo e no espaço.

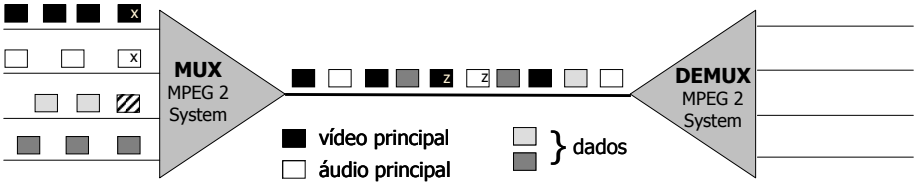


Figura 1.9 Transporte de dados assíncronos.

O serviço assíncrono é a única forma de sincronização de objetos com tempo de sincronização indeterminado. Ele, no entanto, exige uma linguagem para especificação do sincronismo e, no receptor, uma máquina capaz de interpretar e executar os comandos de sincronização especificados nessa

linguagem. O padrão MPEG-2 System especifica os vários tipos de serviço, mas não especifica a linguagem usada para sincronização no serviço assíncrono. Esse assunto será tema da Seção 1.3. Note também que é possível em uma transmissão que parte dos dados sigam um serviço e parte outros.

1.2.3.2 Fluxo Elementar de Dados

Até agora partimos do pressuposto de que os fluxos de dados já estavam gerados e empacotados. O padrão MPEG-2 especifica também várias formas pelas quais esses dados podem ser transportados e opções para o processo de geração desses fluxos, como discutido a seguir.

Fluxos de áudio e vídeo a serem transportados usando os serviços síncrono e sincronizado são gerados de forma similar à geração dos fluxos de áudio e vídeo principal.

Todo fluxo elementar deve ser quebrado em pacotes de forma a poderem ser multiplexados no fluxo de transporte, como discutimos na seção anterior. Para ajudar a solucionar esse problema, o MPEG define o conceito chamado de *seções privadas*, ou simplesmente *seções*, usadas para empacotar os dados a serem transportados no serviço assíncrono para posterior multiplexação. As seções seguem um formato padronizado [ISO/IEC 10918-1, 1994]. Cada seção pode conter até 4.096 bytes de dados e um cabeçalho que informa ao receptor quantas seções estão sendo usadas para transportar um fluxo de dados específico e como os dados devem ser remontados.

Como a sintonização de um canal específico de TV (e, assim, de um fluxo multiplexado MPEG-2 System) pode ser realizada em qualquer instante, dados sem solicitação (pushed data) que não tenham relação temporal especificada por meio de selos de tempo devem ser enviados ciclicamente. Se assim for feito, o recebimento desses dados será independente do instante de sintonização.

Para dar suporte ao envio cíclico de dados, prevalece nos sistemas de TV digital terrestre a utilização dos protocolos carrossel de dados e carrossel de objetos, especificados no padrão DSM-CC (*Digital Storage Media — Command and Control*) [ISO/IEC 13818-6, 1998]. Carrosséis de dados e de objetos são transportados em seções privadas específicas MPEG-2. Cabe observar que outros protocolos para difusão de *dados sem solicitação* (*pushed data*) são utilizados principalmente em sistemas de IPTV e P2PTV.

Um carrossel de dados DSM-CC permite o envio de dados não-estruturados. A estruturação fica por conta do sistema de TV digital que o utiliza. O sistema japonês de TV digital faz uso dessa facilidade do padrão. Um carrossel de objetos permite o envio cíclico de um sistema de arquivos. Assim, ao sintonizar um determinado canal, o receptor deve ser capaz de

decodificar os dados recebidos e colocá-los em uma área de memória para que possam ser utilizados, preservando a estrutura de arquivos e diretórios enviada. Dados enviados tanto nos sistemas americano e europeu quanto no sistema brasileiro de TV digital terrestre utilizam o carrossel de objetos.

Cabe observar que mais de um carrossel pode ser transmitido simultaneamente e que um carrossel de objetos pode fazer uso de recursos (arquivos, diretórios e outros objetos [ISO/IEC 13818-6, 1998]) que estejam sendo transferidos em outros carrosséis. Mais ainda, aplicações transferidas em arquivos de um carrossel podem fazer referência a recursos presentes no mesmo carrossel ou a recursos de outros carrosséis. Por exemplo, uma página HTML transmitida em um carrossel pode referenciar uma imagem cujo conteúdo é transmitido em um outro carrossel de objetos.

O carrossel de objetos é de fato um protocolo de transmissão cíclica de dados. O resultado é um fluxo elementar de dados que contém o sistema de arquivos transmitido de forma cíclica. Assim, se um determinado receptor não recebeu um bloco de dados em particular (devido a uma falha na transmissão ou por ter sintonizado o canal após a transmissão desse bloco), basta esperar pela sua retransmissão correta.

Como mencionado anteriormente, em um transporte de dados assíncrono, toda especificação de sincronismo entre os dados, o áudio principal, o vídeo principal e os outros dados enviados pelos serviços síncrono ou sincronizado é enviada em um documento de especificação da aplicação (bloco hachurado na Figura 1.9). Esse documento também pode ser transportado em uma seção privada MPEG (por exemplo, em um carrossel DSM-CC). Para que a aplicação entre em execução, um comando deve ser enviado ao receptor. O padrão MPEG-2 System também especifica como isso pode ser realizado através do envio de *eventos de sincronismo DSM-CC* (ou simplesmente eventos DSM-CC).

O Apêndice B traz uma discussão técnica mais detalhada sobre o padrão DSM-CC. O Apêndice F traz uma discussão detalhada das várias opções de transporte de dados e eventos de comando oferecidas pelo middleware Ginga (apresentado na Seção 1.3). Eventos de comando são discutidos e exemplificados no Capítulo 16.

1.2.3.3 Fluxo de Transporte

Cada fluxo elementar MPEG-2 System (áudio principal, vídeo principal, fluxo do carrossel etc.) tem um identificador único. As especificações MPEG-2 System definem ainda o termo *programa*, chamado de *serviço* no contexto da TV digital, como um grupo composto de um ou mais fluxos elementares com uma mesma base temporal. O fluxo de transporte multiplexado pode

conter vários serviços (programas) simultaneamente, cada qual podendo ter uma base de tempo diferente.

Simplificadamente, multiplexar serviços em um fluxo de transporte significa organizar os pacotes dos vários fluxos elementares pertencentes aos serviços contemplados em um único fluxo. Para isso, é necessário inserir no fluxo de transporte informações que permitam ao decodificador MPEG-2 identificar a qual serviço um dado fluxo elementar pertence. Essas informações são dispostas como um conjunto de tabelas de informação específica de programa (*PSI — Program Specific Information*). Uma PSI particular, denominada *PMT* (*Program Map Table*), contém a lista de identificadores dos fluxos elementares que compõem um serviço. Cada PMT encontrada representa um serviço disponível. As PMTs são localizadas através de outra PSI, denominada *PAT* (*Program Association Table*), que contém identificadores dos fluxos elementares contendo as PMTs. O fluxo elementar que possui a PAT possui identificador fixo com o valor hexadecimal 0x00. A Figura 1.10 ilustra a multiplexação MPEG-2 System, incluindo os fluxos elementares das tabelas PSI.

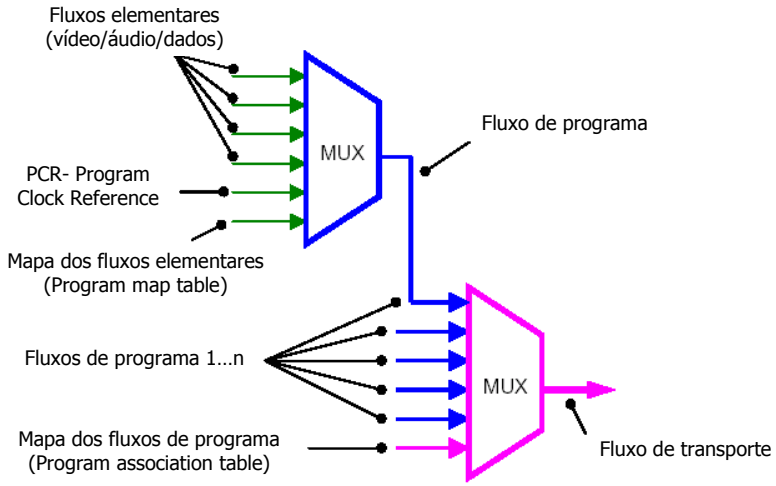


Figura 1.10 Fluxo de transporte MPEG-2 System.

1.2.4 Modulação

Um dos padrões mais importantes em um sistema de TV digital terrestre é o que define o processo de modulação, responsável por receber um fluxo de transporte e posicioná-lo em um canal de frequência.

As técnicas de modulação digital podem ser divididas em dois grupos: aquele com portadora única, como no sistema americano 8-VSB (Vestigial Side Band)/ATSC [ATSC A-53, 2007] e no sistema chinês 4-16 ou 64-OQAM (*Quadrature Amplitude Modulation*))/ADTB [ADTB, 2001], e aquele com múltiplas portadoras, como no sistema europeu COFDM/DVB-T [ETSI EN 300 744 V1.1.2, 1997] e no sistema japonês BST-OFDM/ISDB-T [ISDB-T, 1998], que podem também ser moduladas em fase (QPSK, 16QAM e 64QAM). No caso do sistema japonês, as portadoras também podem ser moduladas com DQPSK. O sistema brasileiro BST-OFDM/SBTVD-T [ABNT NBR 15601, 2007] utiliza a mesma modulação do sistema japonês.

A acomodação de um canal de HDTV numa banda de apenas 6 MHz exige não apenas técnicas eficientes de compressão, como discutido na Seção 1.2.1, mas também técnicas de modulação com múltiplos níveis. Alguns sistemas simples de modulação digital transportam apenas um bit de informação por símbolo. Em outras palavras, cada símbolo deve ser representado por um de dois possíveis estados (níveis), representando o bit 1 ou 0. Nesse caso, a taxa de bits do sistema é a mesma que a taxa de símbolos. Entretanto, outros sistemas podem ter muitos estados possíveis para cada símbolo. Geralmente, o número de estados é tal que consiste numa potência de dois e, assim, a taxa de bits do sistema é algum múltiplo inteiro da taxa de símbolos. Os sistemas de modulação digital são frequentemente indicados pelo tipo de modulação precedido por um número que representa o número de estados de cada símbolo. Por exemplo, 4QAM descreve uma modulação com quatro possíveis estados para cada símbolo. Quatro estados podem carregar dois bits de informação (00, 01, 10 e 11); assim, a taxa de bits de um sistema 4QAM é duas vezes a taxa de símbolos.

Embora seja tentador, não podemos aumentar impunemente o número de níveis, aumentando a taxa de transmissão possível, porque, ao aumentarmos o número de níveis, diminuímos a sensibilidade a ruídos, aumentando a probabilidade de interferência entre símbolos, o que reduz o desempenho do sistema quanto à taxa de erro de bits (BER).

Nas técnicas de portadora única, os símbolos digitais são transmitidos serialmente, o que significa que o intervalo de sinalização (janela de tempo) associado a cada símbolo é muito pequeno quando as taxas de transmissão são altas. As propostas atuais de HDTV, que usam taxas de aproximadamente 19,3 Mbps, trazem problemas técnicos de difícil resolução para a radiodifusão, sendo inevitáveis os fenômenos de múltiplos percursos,

já discutidos na Seção 1.1. Como apresentado naquela seção, um intervalo de sinalização pequeno aumenta a probabilidade de a interferência intersímbolos causar erros de recepção do sinal. Quando o intervalo de sinalização é muito pequeno, também aumenta a sensibilidade a ruído impulsivo, uma vez que erros podem ser causados em sequências de bits consecutivos. O combate a esses problemas exige o uso de equalizadores adaptativos muito complexos. O desempenho desses equalizadores é fundamental para que o sistema funcione de maneira adequada. Caso o número de percursos existentes seja maior do que a capacidade de atuação do equalizador, a taxa de erro se torna elevada a ponto de colocar o sistema fora de operação.

Por outro lado, as técnicas com múltiplas portadoras prometem um desempenho muito melhor frente ao ruído impulsivo e múltiplos percursos, já que cada símbolo pode ter o seu período de transmissão aumentado de tal maneira que seja muito maior que a duração dos impulsos de ruído e o intervalo de dispersão da propagação. Para tanto, um fluxo digital de alta taxa de bits é dividido em um grande número de canais com baixa taxa de bits. Esses canais são usados para modular as subportadoras individuais e são transmitidos em paralelo (isto é, simultaneamente), ao contrário da técnica de portadora única, onde os dados são enviados em série. Essa diferença fundamental permite ampliar o tempo de transmissão de cada dado para combater os efeitos da dispersão temporal e do ruído impulsivo.

A técnica utilizada na modulação das subportadoras é uma variação da multiplexação por divisão de frequência (FDM). Em um sistema FDM, as portadoras são suficientemente espaçadas de modo a poderem ser recebidas utilizando filtros convencionais. Para tornar a filtragem possível, intervalos de guarda têm de ser introduzidos entre essas portadoras, o que resulta em uma diminuição da eficiência espectral. Já na técnica utilizada na modulação das subportadoras (OFDM — *Orthogonal Frequency Division Multiplexing*), em vez de se utilizar um intervalo de guarda entre subportadoras, elas são sobrepostas, mas sem que haja interferência entre elas. Para que isso seja possível, as subportadoras devem ser matematicamente ortogonais (linearmente independentes), daí a denominação OFDM, resultando em um ganho espectral de até de 50% em relação à técnica FDM.

Para se tornar pouco sensível ao problema de multipercurso, a modulação OFDM adota um intervalo de guarda com duração superior à dispersão produzida nos múltiplos percursos. Satisfeita essa condição, não haverá interferência entre símbolos. Assim, a modulação OFDM não requer equalizadores complexos para que se tenha sucesso na recepção em canais com múltiplos percursos.

Vejamos um exemplo bem simples. Se enviarmos um milhão de símbolos por segundo usando uma modulação com portadora única, a duração de cada símbolo será de 1 microssegundo, impondo graves restrições

à interferência de múltiplos percursos. Se os mesmos símbolos forem enviados em mil subcanais, em paralelo, a duração de cada símbolo será de 1 milissegundo. Suponhamos agora que um intervalo de guarda de 1/8 de símbolo seja introduzido entre cada símbolo. Interferências entre símbolos serão evitadas se o tempo de recepção entre o primeiro e o último eco for menor que o intervalo de guarda, ou seja, 125 microssegundos, o que equivale a uma diferença de caminho de propagação de aproximadamente 37,5 quilômetros.

Como mencionamos na Seção 1.1, todos os padrões de transmissão para TV digital terrestre utilizam códigos corretores de erro. A modulação OFDM é invariavelmente usada em conjunto com a codificação do canal, daí a denominação COFDM. Tanto no sistema DVB-T quanto no ISDB-T, a codificação de canal envolve a codificação de Reed-Solomon e de Treliça.

A técnica de modulação BST-OFDM (*Band Segmented Transmission Orthogonal Frequency Division Multiplexing*) proposta no sistema japonês (ISDB-T, ISDB-TSB e ISDB-C) melhora a técnica COFDM em dois sentidos, introduzindo a *segmentação de banda* e a *intercalação no tempo* (*time interleaving*).

A intercalação no tempo espalha ao longo de um sinal transmitido símbolos consecutivos gerados no sinal original. Dessa forma, quando acontecem erros, eles não afetam símbolos consecutivos, mas são de fato espalhados com relação ao sinal original. Evitando-se a concentração de erros, evita-se que corretores de erros não sejam capazes de recuperá-los.

A segmentação de banda explora o fato de que algumas portadoras podem ser moduladas de forma diferente de outras. Assim, o canal de TV de 6 MHz pode ser segmentado e modulado com a técnica mais apropriada para cada serviço. É possível, por exemplo, enviar sinais de vídeo em um canal com modulação otimizada para recepção móvel, enviar outros sinais de vídeo em outro canal otimizado para recepção fixa etc.

1.2.5 Canal de Retorno ou Canal de Interatividade

Um sistema de TV digital terrestre pode operar sem canal de retorno (ou canal de interatividade). Nesse caso, as aplicações podem usar (ou navegar em, por semelhança com a Web) apenas os dados transmitidos por difusão. Caso as aplicações permitam a interação do usuário, o serviço oferecido é chamado de *interatividade local*.

Padrões de referência de um sistema de TV digital podem incluir, contudo, o uso de um canal de retorno.

O canal de retorno pode ser *unidirecional*, permitindo ao receptor apenas o envio de dados. Um segundo nível de interatividade é então definido, permitindo ao usuário telespectador o envio de dados, por exemplo, solicitando a compra de um determinado produto, votando em um determinado assunto etc.

O canal de retorno também pode ser *bidirecional assimétrico*, possibilitando ao receptor fazer o carregamento (download) de dados utilizados pelas aplicações. Nesse caso, uma aplicação pode receber dados por difusão ou pela rede de retorno. Um terceiro nível de interatividade é então fornecido, permitindo ao usuário telespectador o acesso a dados não provenientes por difusão, por exemplo, permitindo a navegação na Web.

Um canal de retorno bidirecional pode também permitir o envio de dados em banda larga (upload). Nesse caso, o receptor pode passar a atuar como uma pequena emissora. Esse nível de interatividade, chamada de *interatividade plena*, possibilita, entre outras coisas, o que vem sendo chamado de "TV social (*social TV*)" ou "TV em comunidade (*community TV*)", que se caracteriza por um grupo de usuários telespectadores de um mesmo programa que podem trocar dados entre si.

O sistema brasileiro de TV digital terrestre permite, em suas normas, os quatro níveis de interatividade, assim como os sistemas americano, europeu e japonês de TV digital terrestre.

1.3 Middleware

Na Seção 1.2.3 vimos como uma aplicação, em particular um programa não-linear, pode ser enviada por difusão e sem solicitação através de seções privadas MPEG-2. Essa especificação também poderia ser obtida sob demanda, como usual em serviços IPTV ou Web TV. A especificação da aplicação, entre outras funções, deve ser a responsável pela sincronização espacial e temporal dos vários objetos de mídia que compõem a aplicação.

Uma aplicação poderia ser executada diretamente sobre o sistema operacional de um receptor (ver Figura 1.3). No entanto, os sistemas operacionais de propósito geral não estão preparados para dar um bom suporte às aplicações de TV digital. Além disso, uma aplicação de TV deve ser capaz de ser executada em qualquer plataforma de hardware e sistema operacional.

Para tornar as aplicações independentes da plataforma de hardware e software de um fabricante de receptor específico e para dar melhor suporte às aplicações voltadas para a TV, uma nova camada é acrescentada nos padrões de referência de um sistema de TV digital. A essa camada denominamos

middleware. A Figura 1.11 apresenta os padrões de referência do sistema nipo-brasileiro de TV digital terrestre, incluindo seu middleware de nome *Ginga*.

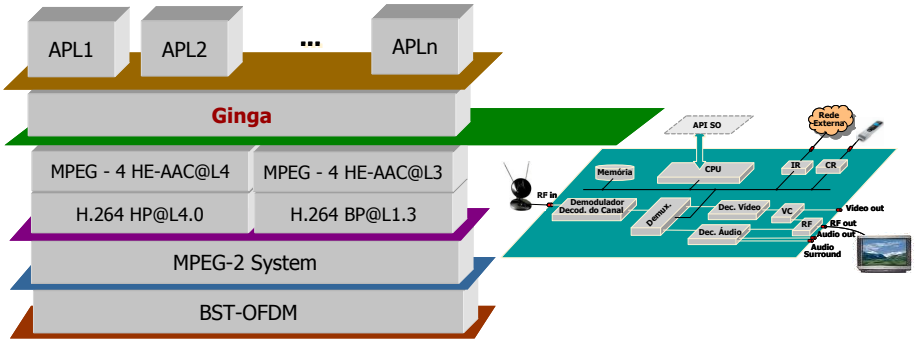


Figura 1.11 Padrões de referência do sistema brasileiro de TV digital terrestre.

O middleware é um dos componentes mais importantes de um sistema de TV digital porque, na prática, é ele que regula as relações entre duas indústrias de fundamental importância: a de produção de conteúdos e a de fabricação de aparelhos receptores. Do ponto de vista do software, podemos dizer, sem exagero, que ao definir o middleware estamos, de fato, definindo um sistema de televisão. Dominar o conhecimento dessa tecnologia é estratégico para um país, pois o não-domínio certamente acarretaria também o não-domínio de seu uso.

1.3.1 Requisitos

Uma das funções de um middleware é fornecer suporte às aplicações. O suporte é fornecido através de uma *interface de programação de aplicações* (*API — Application Programming Interface*), cuja funcionalidade oferecida deve ser regida pelas necessidades das aplicações a serem executadas no ambiente de TV digital. Dentre essas aplicações, obviamente, estão os programas não-lineares, foco principal de qualquer sistema de TV digital.

Levantar os requisitos das aplicações é, assim, levantar os requisitos de um middleware, e é o que trataremos nesta seção.

Na Seção 1.2.3 discutimos as várias formas de sincronização de dados e vimos como o suporte à sincronização é essencial quando o serviço utilizado é o serviço assíncrono, único serviço em que a sincronização com tempos não-determinísticos é possível. O suporte à sincronização, com ou sem a interação

do usuário telespectador, se faz necessário em todos os programas não-lineares e na grande maioria das outras aplicações de TV.

Como exemplo de sincronização, tomemos a exibição de um jogo de futebol. Durante a exibição do vídeo principal, um torcedor poderia entrar em foco com um copo na mão. Sincronizado com esse exato instante poderia aparecer um objeto de mídia texto com o aviso "Se beber, não dirija" ou mesmo uma imagem com a propaganda de uma marca de bebida. No decorrer do jogo poderia acontecer um lance duvidoso. Quando isso acontecesse, poderia aparecer, sincronamente, um ícone do "tira-teima" que, quando acionado (sincronismo com a interação do telespectador), redimensionaria o vídeo principal, exibiria o lance em outro objeto de mídia vídeo e mostraria a animação gráfica "tira-teima" em outra janela da tela, como ilustra a Figura 1.12. De tempos em tempos, poderia aparecer na tela um ícone que, se acionado pela interação síncrona do telespectador, exibiria os resultados dos outros jogos da rodada do campeonato. Em um jogo, é comum a cena de um jogador amarrando sua chuteira. Se isso acontecesse no nosso exemplo, sincronizado com esse exato momento, poderia aparecer um outro objeto de mídia, por exemplo um outro vídeo, fazendo a propaganda da marca da chuteira e, sincronizado com o final desse vídeo, um formulário para compra que, se preenchido, seria enviado pelo canal de retorno ao fornecedor, que se encarregaria de entregar o pedido na casa do telespectador.

Figura 1.12 Exemplo de programa não-linear.

Esse exemplo ilustra vários aspectos possíveis em um programa não-linear:

- Propagandas não-interativas (de aparecimento obrigatório): propaganda da bebida.
- Propagandas interativas (de aparecimento sob comando do usuário telespectador): propaganda da chuteira.
- Informações adicionais obrigatórias ao conteúdo do programa: o aviso "se beber, não dirija".

- Informações adicionais opcionais ao conteúdo do programa: a exibição dos resultados da rodada do campeonato e a exibição do "tira-teima".
- Uso do canal de retorno: no caso, para compra da chuteira.

Mais do que apresentar exemplos de uso, esse exemplo ilustra a importância do sincronismo de mídia, com e sem a interação do usuário. Isso nos leva ao primeiro requisito de um middleware: suporte ao sincronismo de uma forma geral e, como caso particular, à interação do usuário.

No mesmo exemplo, poder-se-ia questionar que a televisão analógica já permite algumas das facilidades, como por exemplo, o aparecimento das propagandas (bebida) sem a interação do usuário. Isso é verdade, mas para tanto, na TV analógica, a propaganda teria de ser multiplexada com o vídeo principal na origem, na emissora, gerando um novo vídeo principal. Para o receptor isso seria indiferente, uma vez que ele continua tratando um único objeto de mídia, o vídeo principal. Imaginemos, no entanto, a possibilidade de exibição dessa propaganda de forma adaptativa. Por exemplo, se uma criança estivesse assistindo ao jogo, apareceria a propaganda de um guaraná; se fosse um adulto, apareceria a propaganda de uma cerveja. Isso é possível com a TV digital, mas seria impossível na analógica. A adaptação poderia ser em função do usuário, do dispositivo que está exibindo a informação ou mesmo do local onde está o telespectador. Por exemplo, a propaganda da bebida para um usuário adulto em Ipanema poderia trazer "beba cerveja no Garota de Ipanema", já para um telespectador no centro da cidade do Rio de Janeiro, a propaganda poderia trazer "beba cerveja no Amarelinho".

O parágrafo anterior nos traz um segundo requisito importante a ser oferecido por um middleware: suporte à adaptação de conteúdo e da forma como um conteúdo é exibido.

Em uma aplicação para TV, a interatividade deve ser usada com parcimônia. Diferentemente de uma aplicação voltada para computador, uma aplicação de TV deve levar em consideração que:

- a transmissão de grande parte das informações é por difusão e não personalizada (isso é 100% verdadeiro com a interatividade apenas local);
- a TV, em geral, se assiste a uma distância razoável da tela (isso pode não ser verdade no caso de dispositivos portáteis);
- os dispositivos de interação (controle remoto) são ainda pobres em termos de expressividade e usabilidade;
- o vídeo principal é a principal fonte de sincronismo, incluindo a interação;

- assistir a um programa é muitas vezes uma atividade coletiva (novamente no caso de dispositivos portáteis, isso pode não ser verdade);
- a TV é usada para lazer e o telespectador não quer nada de muito complexo em seu uso.

Em uma assistência coletiva, além da dificuldade imposta pelo ambiente à interação, o aparecimento de informações adicionais pela demanda de um telespectador pode aborrecer seu companheiro ao lado. O uso de dispositivos de exibição pessoais (controle remoto com tela de baixa resolução, celulares, PDAs etc.) poderia amenizar o problema. Como exemplo, tome novamente a Figura 1.12. Ao pedido de um usuário para ver o tira-teima, esse poderia aparecer em seu dispositivo particular, no caso exemplificado pela Figura 1.12 em um celular, e não aparecer na tela principal da televisão, perturbando os demais telespectadores.

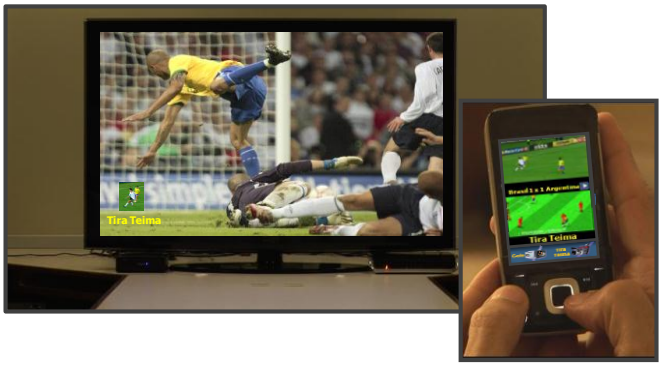


Figura 1.13 Múltiplos dispositivos de exibição.

Isso nos leva a um terceiro requisito a ser oferecido por um middleware: suporte a múltiplos dispositivos de exibição.

Dispositivos receptores pessoais podem também comunicar entre si. Poderiam, por exemplo, permitir a inclusão de comentários textuais ou outros objetos de mídia na aplicação de TV recebida, e que o novo programa assim criado fosse distribuído aos demais participantes de uma comunidade (grupo de telespectadores), via, por exemplo, o canal de retorno, constituindo o que chamamos anteriormente de “TV social” ou “TV em comunidade”.

A inserção de objetos de mídia sincronizados em um programa não-linear em tempo de exibição é importante não só por possibilitar a aplicação de TV em comunidade, descrita no parágrafo anterior, onde a edição ao vivo é realizada no cliente telespectador, mas também por possibilitar a geração de

programas não-lineares ao vivo pela emissora de TV. Em muitos programas, a decisão de quais objetos de mídia comporão o serviço pode ser decidida ao vivo. Retornando ao nosso exemplo do jogo de futebol, suponha que no momento em que um jogador marcar um gol será exibida a estatística dos gols em toda a sua carreira. Ora, a princípio não se sabe quem vai marcar o gol nem se vai haver gol. A decisão de qual objeto de mídia com a estatística será transmitido e quando só pode ser tomada após a ocorrência do gol.

Isso nos leva a um quarto requisito que deve ser oferecido por um middleware: suporte à edição ao vivo (em tempo de exibição).

Comentamos anteriormente que o vídeo principal é a fonte mais importante de sincronismo, incluindo a interação do usuário. Diferentemente das aplicações usuais de navegação na Web, que são usualmente baseadas em texto, as aplicações de TV são usualmente baseadas no conteúdo do vídeo (programa de TV) principal. Embutir informação de navegação em vídeo é problemático, quando não impossível. Assim, permitir a definição de relacionamentos entre objetos de mídia sem "embutir" tais relacionamentos nos conteúdos é um requisito importante.

Isso nos traz a um quinto requisito: suporte à definição de relacionamentos de sincronismo espacial e temporal separado da definição do conteúdo dos objetos de mídia relacionados. Na literatura isso é conhecido como definição baseada na estrutura (*structure-based*), em contraste com a definição embutida no conteúdo dos objetos de mídia, conhecida como baseada no conteúdo de mídia (*media-based*).

Em resumo, um middleware deve oferecer um bom suporte para:

- o sincronismo de uma forma geral e, como caso particular, a interação do usuário;
- a definição de relacionamentos de sincronismo espacial e temporal separado da definição do conteúdo dos objetos de mídia relacionados;
- a adaptação do conteúdo e da forma como o conteúdo é exibido;
- o uso de múltiplos dispositivos de exibição;
- a edição ao vivo (em tempo de exibição).

No caso do Brasil, a TV pode representar um grande veículo *complementar* para inclusão digital, permitindo às classes menos privilegiadas não só o acesso à informação, mas também a geração (produção) de informação (conteúdo). Assim, acresce aos requisitos de um middleware o bom suporte a aplicações ditas cidadãs, como as voltadas para as áreas de saúde, educação, cultura etc.

1.3.2 Ambientes de Programação

As aplicações de TV digital vão desde aquelas em que a aplicação transmitida não tem nenhuma relação semântica com o programa principal em exibição (como, por exemplo, a exibição de mensagens de notícias urgentes durante a exibição de um programa de TV) até os programas não-lineares (onde todos os objetos de mídia são sincronizados entre si, incluindo o áudio principal e o vídeo principal), passando pelas aplicações com vários objetos de mídia relacionados entre si e que só fazem sentido dentro do contexto do programa sendo assistido, mas não tendo nenhum relacionamento temporal com o áudio principal e o vídeo principal.

Aplicações são usualmente desenvolvidas usando dois paradigmas de programação distintos: o *declarativo* e o *não-declarativo*.

Linguagens de programação declarativas (linguagens que seguem o paradigma declarativo) são linguagens de mais alto nível de abstração, muitas vezes ligadas a um domínio ou objetivo específico. Nas linguagens declarativas, o programador fornece apenas o conjunto das tarefas a serem realizadas, não estando preocupado com os detalhes de como o executor da linguagem (interpretador, compilador ou a própria máquina real ou virtual de execução) realmente implementará essas tarefas. Linguagens declarativas resultam em uma declaração do resultado desejado, em vez da sua decomposição em uma implementação algorítmica e, portanto, normalmente não necessitam de tantas linhas de código para definir uma certa tarefa e são menos sujeitas a erros de programação. Entre as linguagens declarativas mais comuns estão a NCL (*Nested Context Language*) [ABNT NBR 15606-2, 2011] [ITU-T H.761, 2011], a SMIL [W3C REC-SMIL2-20051213, 2008], a SVG [W3C REC-SVG11-20110816, 2011] e a XHTML [W3C REC-xhtml1-20020801, 2002].

Numa programação não-declarativa, devemos informar cada passo a ser executado. Pode-se afirmar que, em uma especificação seguindo o paradigma não-declarativo, o programador possui maior poder sobre o código, sendo obrigado a estabelecer todo o fluxo de controle e execução de seu programa. Entretanto, para isso, ele deve ser qualificado e conhecer bem os recursos de implementação. Linguagens não-declarativas podem seguir diferentes modelos. Temos, assim, as linguagens baseadas em módulos, orientadas a objetos etc. Entre as linguagens não-declarativas mais comuns no domínio da TV digital estão C, Java e ECMAScript.

O universo das aplicações de um sistema de TV digital pode ser particionado em um conjunto de aplicações declarativas e um conjunto de aplicações não-declarativas. Uma aplicação declarativa é aquela em que o tipo do conteúdo de sua "entidade inicial" é declarativo. Por outro lado, uma aplicação não-declarativa é aquela cujo tipo do conteúdo de sua "entidade

inicial" é não-declarativo. Uma aplicação declarativa pura é aquela na qual o conteúdo de "todas suas entidades" é do tipo declarativo. Uma aplicação não-declarativa pura é aquela na qual o conteúdo de "todas suas entidades" é do tipo não-declarativo. Uma aplicação híbrida é aquela cujo conjunto de entidades possui tanto conteúdo do tipo declarativo quanto não-declarativo.

A especificação de uma tarefa usando uma linguagem declarativa é, a princípio, muito mais fácil que o desenvolvimento usando uma linguagem não-declarativa e também muito menos sujeita a erros de programação. Como mencionado, linguagens declarativas possuem alto nível de abstração, não exigindo grande *expertise* para o projeto de programas, ao contrário das linguagens não-declarativas que, em geral, exigem um programador especialista. Tudo isso, no entanto, é apenas um lado da verdade.

Linguagens declarativas muitas vezes são focadas em um domínio ou um objetivo específico. Quando o foco da linguagem "casa" com o foco do problema a resolver, tudo o que mencionamos no parágrafo anterior é verdade e a linguagem declarativa tem uso preferencial sobre uma linguagem não-declarativa. Quando, no entanto, o foco do problema "não casa" com o foco da linguagem declarativa, sua resolução pode ser muito difícil, se não impossível, usando a linguagem. Nesse caso, o uso de uma linguagem de propósito geral, por exemplo uma linguagem imperativa, é preferível.

Praticamente todos os middlewares para TV digital terrestre e IPTV oferecem suporte para o desenvolvimento de aplicações usando os dois paradigmas de programação. Alguns middlewares só oferecem o suporte para aplicações declarativas (puras ou híbridas). Dá-se, informalmente, o nome de *ambiente declarativo* a esse suporte. Todos os middlewares padronizados pelo ITU-T para serviços IPTV contêm apenas o ambiente declarativo. Outros middlewares, no entanto, oferecem o suporte apenas a aplicações não-declarativas. Dá-se, informalmente, o nome de *ambiente imperativo* a esse suporte. A maioria dos middlewares para TV terrestre, no entanto, é composta dos ambientes imperativos e declarativos. A Tabela 1.3 ilustra os ambientes dos middlewares dos sistemas americano, europeu, japonês e brasileiro, para receptores fixos e móveis, em seus padrões para TV digital terrestre. Note que o middleware Ginga obrigatoriamente requer o ambiente declarativo Ginga-NCL, mas admite extensões. Uma dessas extensões é o ambiente imperativo Ginga-J, adotado no Brasil como obrigatório para receptores fixos.

Tabela 1.3 Ambientes de aplicações para receptores fixos e móveis

Middleware	Sistema de TVD	Ambiente Declarativo	Ambiente Imperativo
ACAP	Americano/ATSC	ACAP-X [ATSC A-101, 2005] (linguagem declarativa = XHTML like; linguagem não-declarativa = ECMAScript)	ACAP-J [ATSC A-101, 2005] (linguagem não-declarativa = Java
MHP	Europeu/DVB-T	DVB-HTML [ETSI TS 102 812 V1.2.2, 2006] (linguagem declarativa = XHTML like; linguagem não-declarativa = ECMAScript)	MHP [ETSI TS 102 812 V1.2.2, 2006] (linguagem não-declarativa = Java)
ARIB-BML	Japonês/ISDB-T	ARIB – BML [ARIB B-24, 2004] (linguagem declarativa = BML (XHTML like); linguagem não-declarativa = ECMAScript)	Opcional (GEM [ETSI TS 102 819 V1.3.1, 2005] like); não implementado)
Ginga	Latino-Americano /ISDB-T_B	Ginga-NCL [ABNT NBR 15606-2, 2011] (linguagem declarativa = NCL; linguagem não-declarativa = Lua)	Ginga-J [ABNT NBR 15606-4, 2010] (linguagem não-declarativa = Java

A Tabela 1.4 ilustra os ambientes dos middlewares dos sistemas japonês e brasileiro para receptores terrestres portáteis.

Tabela 1.4 Ambientes de aplicações para receptores portáteis

Middleware	Sistema de TVD	Ambiente Declarativo	Ambiente Imperativo
ARIB-BML	Japonês/ISDB-T	ARIB – BML [ARIB B-24, 2004] (linguagem declarativa = BML (XHTML like; linguagem não-declarativa = ECMAScript)	X
Ginga	Latino-Americano/ ISDB-T_B	Ginga-NCL [ABNT NBR 15606-5, 2011] (linguagem declarativa = NCL; linguagem não-declarativa = Lua)	Opcional

A Tabela 1.5 ilustra os ambientes dos middlewares Recomendações ITU-T para serviços IPTV.

Tabela 1.5 Ambientes de aplicações para serviços IPTV

Middleware	Ambiente Declarativo	Ambiente Imperativo
LIME	LIME [ITU-T H.762, 2010] (linguagem declarativa = LIME (XHTML like; linguagem não-declarativa = ECMAScript)	X
Ginga	Ginga-NCL [ITU-T H.761, 2011] (linguagem declarativa = NCL; linguagem não-declarativa = Lua)	X

Terminando esta seção, cabe enfatizarmos que tanto o ambiente declarativo quanto o imperativo (também algumas vezes chamado de procedural) de um middleware devem dar suporte a todos os requisitos discutidos na Seção 1.3.1. Cabe também salientar que todos os middlewares que apresentam os dois ambientes permitem que cada um deles use as facilidades do outro através de uma ponte definida por meio de APIs bilaterais padrão.

1.3.3 Middlewares Declarativos

Como mostra a Tabela 1.3, parte dos ambientes declarativos para TV digital terrestre e IPTV é baseada em XHTML + ECMAScript. XHTML [W3C REC-xhtml1-20020801, 2002] é uma linguagem inicialmente projetada para navegação em informações textuais pela interação do usuário. XHTML privilegia a interatividade em detrimento da sincronização no seu sentido mais amplo e em detrimento da adaptabilidade de conteúdo que, por não fazerem parte do foco da linguagem, só podem ser definidas por procedimentos imperativos usando a linguagem ECMAScript [ECMA 262, 1999]. Como vimos, isso torna o desenvolvimento das aplicações mais difícil para usuários leigos e mais propensa a erros de programação.

XHTML só permite a definição de relacionamentos de interatividade envolvendo parte de um conteúdo (âncora) de um objeto de mídia, se esse objeto for textual ou se a âncora for puramente espacial. No caso de um texto, o relacionamento deve ser embutido no conteúdo, ou seja, XHTML é uma linguagem do tipo *media-based*, como definimos na Seção 1.3.1. Como não é possível embutir os relacionamentos, por exemplo, em um objeto de vídeo, a interação deve ser possibilitada pela seleção de todo o objeto, ou seja, habilitada em qualquer momento da exibição do vídeo ou, então, novamente devemos fazer uso de procedimentos não-declarativos utilizando ECMAScript.

A linguagem XHTML não oferece suporte para edição ao vivo através de comandos do provedor de conteúdos (ambiente das emissoras). No entanto, todos os middlewares apresentados baseados em XHTML oferecem esse suporte, através de extensões à linguagem e uso de eventos DSM-CC (conforme mencionamos na Seção 1.2.3 e discutimos no Apêndice B) e eventos DOM, mas novamente fazendo uso de objetos ECMAScript embutidos.

Diferentemente dos ambientes declarativos baseados em XHTML, Ginga-NCL [ABNT NBR 15606-2, 2011], o ambiente declarativo do middleware do sistema nipo-brasileiro de TV digital terrestre e padrão ITU-T para serviços IPTV, é baseado em uma linguagem declarativa, uma aplicação XML [W3C REC-xml- 20060816, 2006], que oferece um verdadeiro suporte declarativo para todos os requisitos listados na Seção 1.3.1. Diferentemente da linguagem XHTML, a linguagem NCL (*Nested Context Language*) [ABNT NBR 15606-2, 2011] [ITU-T H.761, 2011], linguagem-base do Ginga-NCL, é uma linguagem do tipo *structure-based* (ver Seção 1.3.1), que define uma separação bem demarcada entre o conteúdo e a estrutura de uma aplicação, provendo um controle não-invasivo da ligação entre o conteúdo, o leiaute e sua apresentação.

O foco da linguagem declarativa NCL é mais amplo do que o oferecido pela linguagem XHTML. Através de elementos da linguagem, suporte declarativo é oferecido a todos os requisitos anteriormente citados:

- a definição de relacionamentos de sincronismo espacial e temporal separado da definição do conteúdo dos objetos de mídia relacionados, através dos elementos <area>, <property>, <port> e <portSwitch> e dos elementos <link> e <connector>. A interação do usuário é tratada apenas como caso particular de sincronização temporal;
- a adaptação do conteúdo e da forma como o conteúdo é exibido, através dos elementos <switch> e <descriptorSwitch>;
- múltiplos dispositivos de exibição, através do elemento <regionBase>;
- a edição ao vivo (em tempo de exibição), através de nclEditingCommands associados a descritores de eventos.

Como a NCL tem uma separação mais acurada entre o conteúdo e a estrutura de uma aplicação, ela não define nenhum objeto de mídia em si. Ao contrário, ela define a cola que prende os objetos de mídia em apresentações multimídia. Uma aplicação NCL apenas define como os objetos de mídia são estruturados e relacionados, no tempo e no espaço. Como uma linguagem de cola, ela não restringe ou prescreve os tipos de conteúdo dos objetos de mídia. Nesse sentido, podemos ter objetos de imagem (GIF, JPEG etc.), de vídeo (MPEG, MOV etc.), de áudio (MP3, WMA etc.), de texto (TXT, PDF etc.),

de código declarativo (HTML, SMIL, NCL aninhados etc.), de código imperativo e funcional (Xlet, Lua etc.), entre outros, como objetos de mídia NCL. Quais objetos de mídia são suportados depende dos exibidores de mídia que estão acoplados ao *formatador NCL* (exibidor NCL).[3] Um desses exibidores é o decodificador/exibidor MPEG-4, normalmente implementado em *hardware* no receptor de televisão digital. Dessa forma, o vídeo principal e o áudio principal MPEG-4 são tratados como todos os demais objetos de mídia que podem estar relacionados utilizando NCL.

Outro objeto de mídia possível é aquele baseado em XHTML. A NCL não substitui, mas embute documentos (objetos) baseados em XHTML. Como acontece com outros objetos de mídia, qual linguagem baseada em XHTML terá suporte em um formatador NCL é uma escolha de implementação e, portanto, depende de qual navegador XHTML será incorporado no formatador NCL e atuará como exibidor dessa mídia.

A NCL permite ainda objetos de mídia escritos em outras linguagens declarativas, além da linguagem XHTML, como objetos com código SMIL, SVG etc.

A NCL possui uma linguagem de script (a linguagem Lua), com um desempenho muito superior à ECMAScript em todos os quesitos importantes para uma aplicação em TV digital, mencionados nos próximos parágrafos. Além de objetos NCLua (com código Lua), a NCL permite objetos com outros códigos imperativos como, por exemplo, objetos NCLet com código Java (Xlets), parte da ponte entre o ambiente declarativo e o ambiente imperativo do middleware Ginga.

Lua [ABNT NBR 15606-2, 2011] é uma linguagem de programação funcional e imperativa, procedural, pequena e leve, projetada para expandir aplicações em geral, para ser usada como linguagem extensível e para ser embarcada em softwares complexos.

Lua combina programação procedural com poderosas construções para descrição de dados, baseadas em tabelas associativas e semântica extensível. A linguagem é tipada dinamicamente, interpretada a partir de bytecodes, e tem gerenciamento automático de memória com coleta de lixo. Essas características fazem de Lua uma linguagem ideal para configuração, automação (*scripting*) e prototipagem rápida, característica de grande importância para as aplicações de TV, cujo desenvolvimento rápido é fundamental.

[3] Renderizador de documentos, agente do usuário e exibidor (player) são outros nomes atribuídos ao formatador de documentos. O formatador NCL é o módulo central do ambiente Ginga-NCL.

Lua é portável. Seu núcleo é muito pequeno e totalmente ANSI C. A título de comparação, enquanto máquinas JavaScript (ECMAScript) ocupam de 936 Kbytes (SpiderMonkey — *JavaScript to Mozilla*) a 1,7 Mbytes (Rhino), Lua ocupa aproximadamente 120 Kbytes (retirando o parser e as bibliotecas, Lua pode chegar a 60 Kbytes); LuaJIT (*Lua Just in Time*), uma extensão da máquina Lua com 32 KB a mais de código, ocupa um total de 150 Kbytes.

Lua é facilmente embarcável. Como mencionado, seu interpretador é uma biblioteca C, embarcável em várias linguagens, entre elas C/C++ (linguagem da implementação de referência do Ginga-NCL) e Java (linguagem do ambiente imperativo do Ginga).

Lua é uma das linguagens dinâmicas mais eficientes: eficiente para descrição de dados, eficiente em ocupação de memória de seus programas e rápida. Outras linguagens usam a frase "as fast as Lua" como aspiração. Benchmarks independentes mostram a eficiência de Lua. A Figura 1.14 apresenta uma comparação de Lua com JavaScript (ECMAScript) do SpiderMonkey, segundo o benchmark em http://shootout.alioth.debian.org/. As comparações são feitas sobre implementações de problemas propostos no site e submetidos por qualquer um. Nos gráficos, cada par de linhas representa um dos problemas propostos: linhas pretas para uso de memória; linhas brancas para tempo de CPU. As linhas vão na direção do vencedor. Quanto maior a linha, maior a diferença na comparação.

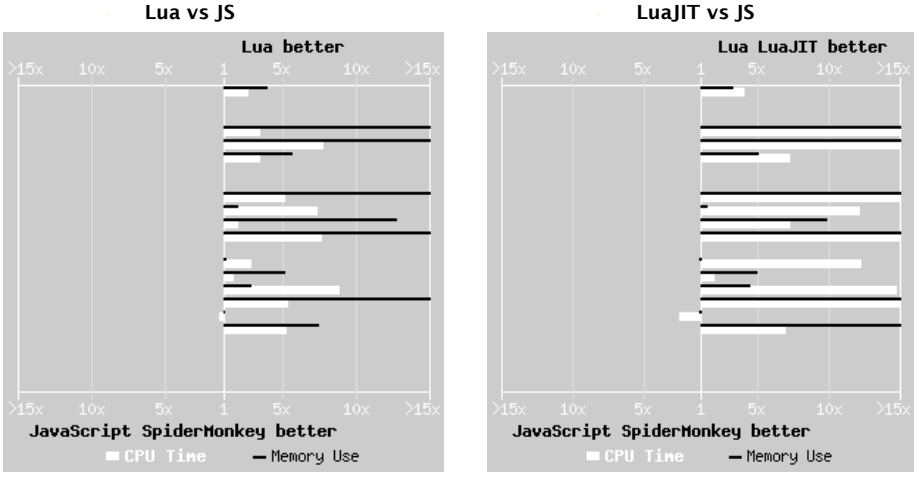


Figura 1.14 Desempenho de Lua e de LuaJIT (http://shootout.alioth.debian.org/) comparado à JavaScript.

Outras características de Lua, importantes na área de entretenimento, são: coleta de lixo incremental, evitando a longa pausa necessária em uma coleta de lixo atômica; referências fracas, permitindo melhor controle sobre as referências para objetos que podem ser coletados; e suporte a corrotinas como alternativa ao uso de multithreading para programação concorrente, que permite ao programador ter maior controle sobre a troca de contexto entre as linhas de execução concorrentes, além de não necessitar de mecanismos de sincronização, como semáforos e monitores. Tipicamente, isso permite o desenvolvimento de programas mais eficientes e com comportamento mais determinístico. Vale a pena destacar que Javascript não possui nenhum recurso para programação concorrente.

Lua é hoje a linguagem de script padrão no seguimento de entretenimento. Alguns dos jogos mais famosos da atualidade utilizam Lua.

Bibliografia

ABNT NBR 15601 (2007). Associação Brasileira de Normas Técnicas, "Televisão digital terrestre — sistema de transmissão". Sistema Brasileiro de TV Digital Terrestre, *NBR 15601*.

ABNT NBR 15602-1 (2007). Associação Brasileira de Normas Técnicas, "Televisão digital terrestre — codificação de vídeo, áudio e multiplexação, Parte 1: Codificação de vídeo". Sistema Brasileiro de TV Digital Terrestre, *NBR 15602-1*.

ABNT NBR 15602-2 (2007). Associação Brasileira de Normas Técnicas, "Televisão digital terrestre — codificação de vídeo, áudio e multiplexação, Parte 2: Codificação de áudio". Sistema Brasileiro de TV Digital Terrestre, *NBR 15602-2*.

ABNT NBR 15606-2 (2011). Associação Brasileira de Normas Técnicas, "Televisão digital terrestre — codificação de dados e especificações de transmissão para radiodifusão digital, Parte 2: Ginga-NCL para receptores fixos e móveis — Linguagem de aplicação XML para codificação de aplicações". Sistema Brasileiro de TV Digital Terrestre, *NBR 15606-2*.

ABNT NBR 15606-5 (2011). Associação Brasileira de Normas Técnicas, "Televisão digital terrestre — codificação de dados e especificações de transmissão para radiodifusão digital, Parte 5: Ginga-NCL para receptores portáteis — Linguagem de aplicação XML para codificação de aplicações". Sistema Brasileiro de TV Digital Terrestre, *NBR 15606-5*.

ABNT NBR 15606-7 (2011). Associação Brasileira de Normas Técnicas, "Televisão digital terrestre — Codificação e transmissão de dados para

radiodifusão digital, Parte 7: Ginga-NCL – Diretrizes operacionais para as Normas ABNT NBR 15606-2 e ABNT NBR 15606-5.

ADTB (2001). Zhang, Wenjum; Xia, Jingsong; Wang, Kuang e Ge, Jianhua. "Advanced Digital Television Broadcasting System."

ARIB B-24 (2004). Terrestrial Integrated Services Digital Broadcasting, "XML-based Multimedia Coding Scheme". *ARIB Standard B-24 Data Coding and Transmission Specifications for Digital Broadcasting*, version 4.0.

ATSC AC-3 (2005). Advanced Television Systems Committee, "Digital Audio Compression Standard (AC-3, E-AC-3)".,Washington, D.C., junho.

ATSC A-101 (2005). Advanced Television Systems Committee "Advanced Application Platform (ACAP)".,*ATSC Standard: Document A/101*, Washington, D.C., agosto.

ATSC A-53 (2007). Advanced Television Systems Committee "A/53: ATSC Digital Television Standard, Parts 1-6".

ECMA 262 (1999). ECMA International, "ECMAScript Language Specification".

ETSI *EN 300 744 V1.1.2* (1997). Digital Video Broadcasting, "Framing structure, channel coding and modulation for digital terrestrial television". European Telecommunications Standards Institute, *EN 300 744 V1.1.2*.

ETSI TS 102 812 V1.2.2 (2006). Digital Video Broadcasting, "Digital Video Broadcasting (DVB); Multimedia Home Platform (MHP) Specification 1.1.1". European Telecommunications Standards Institute, *TS 102 812 V1.2.2*.

ETSI TS 102 819 V1.3.1 (2005). Digital Video Broadcasting (DVB), "Globally Executable MHP (GEM) Specification 1.0.2". European Telecommunications Standards Institute, *TS 102 819 V1.3.1*.

ISDB-T (1998). Terrestrial Integrated Services Digital Broadcasting, "Specification of Channel Coding, Framing Structure and Modulation".

ISO/IEC 10918-1 (1994). International Organization for Standardization/International Eletrotecnical Committee, "Digital Compression and Coding of Continuous-Tone Still Images, Part 1: Requirements and Guidelines", *ISO/IEC IS 10918-1*.

ISO/IEC 11172-3 (1993). International Organization for Standardization/International Eletrotecnical Committee, "Information Technology — Coding of Moving Pictures and Associated Digital Storage Media at up to About 1.5 Mbits/s, Part 3: Audio", *ISO/IEC 11172-3*.

ISO/IEC 13818-1 (2000). International Organization for Standardization/International Eletrotecnical Committee, “Information Technology — Generic coding of moving pictures and associated audio information, Part 1: Systems”, *ISO/IEC 13818-1*.

ISO/IEC 13818-2 (2000). International Organization for Standardization/International Eletrotecnical Committee, “Information Technology — Generic coding of moving pictures and associated information, Part 2: Video”, *ISO/IEC 13818-2*.

ISO/IEC 13818-6 (1998). International Organization for Standardization/International Eletrotecnical Committee, “Information Technology — Generic coding of moving pictures and associated audio information, Part 6: Extensions for DSM-CC”, *ISO/IEC 13818-6*.

ISO/IEC 13818-7 (1997). International Organization for Standardization/International Eletrotecnical Committee, “Information Technology — Generic coding of moving pictures and associated audio information, Part 7: Advanced Audio Coding (AAC)”, *ISO/IEC 13818-7*.

ISO/IEC 14496-3 (2004). International Organization for Standardization/International Eletrotecnical Committee, “Information Technology — Coding of Audio-Visual Objects, Part 3: Audio”, *ISO/IEC 14496-3*.

ISO/IEC 14496-10 (2005). International Organization for Standardization/International Eletrotecnical Committee, “Information Technology — Coding of Audio-Visual Objects, Part 10: Advanced Video”, *ISO/IEC 14496-10*.

ITU-R BT.601-4 (1994). International Communication Union, “Encoding Parameters of Digital Television for Studios”, *Recommendation BT.601-4*, BT Series Volume, Geneva.

ITU-T H.761 (2011). “Nested Context Language (NCL) and Ginga-NCL for IPTV Services.” Recommendation H.761, Genebra.

ITU-T H.762 (2009). “Lightweight interactive multimedia environment”. Recommendation H.762, Genebra.

Soares, L.F.G e Barbosa, S.D.J. “TV digital interativa no Brasil se faz com Ginga: fundamentos, padrões, autoria declarativa e usabilidade”. Jornada de Atualização em Informática, 2008. Sociedade Brasileira de Computação. Julho de 2008; pp. 105-174. ISBN: 978-85-7669-188-4.

W3C REC-SMIL2-20051213 (2008). World Wide Web Consortium, "Synchronized Multimedia Integration Language — SMIL 2.1 Specification", *W3C Recommendation SMIL2-20051213*.

W3C REC-SVG11-20110816 (2011). World Wide Web Consortium, "Scalable Vector Graphics – SVG 1.1 Specification (2nd. Edition)", W3C Recommendation.

W3C REC-xhtml1-20020801 (2002). World Wide Web Consortium, "XHTML™ 1.0 The Extensible HyperText Markup Language", *W3C Recommendation xhtml1-20020801*.

W3C REC-xml- 20060816 (2006). World Wide Web Consortium, "Extensible Markup Language (XML) 1.0", W3C Recommendation *xml-20060816*.

Capítulo 2

Modelo Conceitual NCM

Toda linguagem declarativa é baseada em um modelo conceitual de dados. Um modelo conceitual deve representar os conceitos estruturais dos dados, assim como os eventos e relacionamentos entre os dados. O modelo deve também definir as regras estruturais e as operações sobre os dados para manipulação e atualização das estruturas.

Após introduzir muito resumidamente o modelo básico da linguagem XHTML, linguagem declarativa-base dos sistemas americano, europeu e japonês de TV digital terrestre, este capítulo apresenta uma visão geral do modelo NCM (*Nested Context Model*) e, ao final, relaciona as entidades do modelo NCM aos elementos da linguagem NCL correspondentes.[1]

[1] Este capítulo se baseia em Soares *et al.* (2005). O uso do material foi gentilmente cedido pelo Departamento de Informática da PUC-Rio.

2.1 Entidades Básicas XHTML

O modelo conceitual básico XHTML é muito simples e composto de apenas quatro entidades básicas, como mostra a Figura 2.1.

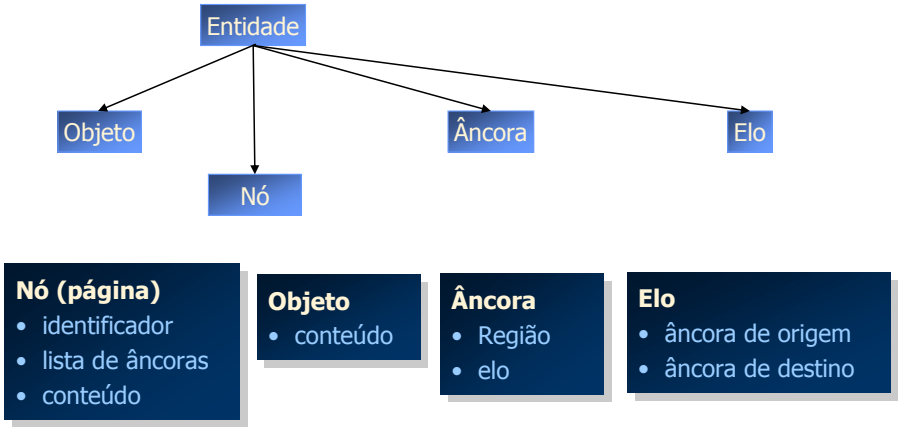


Figura 2.1 Modelo conceitual XHTML básico.

Resumidamente, um *nó*, usualmente chamado de *página*, possui um identificador único (um URI — *Uniform Resource Identifier* [IETF RFC 2396, 1998]), uma lista de âncoras e um conteúdo. O conteúdo é formado por um texto com marcação de formatação [W3C REC-xhtml1-20020801, 2002 e W3C REC-CSS2-19980512, 1998] e outros objetos embutidos.

Os *objetos* embutidos podem ser imagens, scripts (ECMAScript é o usual em ambientes de TV digital), applets etc.

Uma *âncora* é uma região (conjunto de informações) marcada de um nó, portanto definida "embutida" no nó. Âncoras podem ser definidas pela marcação de trechos do texto de um nó ou pela marcação de todo o conteúdo de um objeto. Em XHTML não é possível a definição declarativa de âncoras temporais internas a um objeto, mas apenas âncoras espaciais (especificadas por coordenadas).

Relacionamentos são definidos embutidos no conteúdo de um nó, mais precisamente como atributos de uma âncora do nó. *Elos* definem relacionamentos de interação entre uma âncora de origem e âncoras de destino. Elos são definidos embutidos em âncoras de origem.

Embora tendo por base o modelo simples apresentado, a linguagem XHTML utiliza um modelo conceitual estendido, no qual várias outras entidades são responsáveis pelo leiaute e formatação (listas, tabelas,

formulário etc.). No entanto, nenhuma forma de relacionamento de sincronismo ou de adaptação de conteúdo foi acrescida.

De fato, o modelo conceitual básico XHTML tem na simplicidade sua grande vantagem, e foi ela a responsável pela grande disseminação da linguagem. No entanto, foi concebido para navegação na Web, onde predominantemente se navega em informações textuais e com a interação do usuário. Como já mencionamos, qualquer outra forma de relacionamento tem de ser oferecida por objetos imperativos (ECMAScript, Java etc.) acionados por eventos definidos sobre elementos e atributos da linguagem [W3C REC-DOM-Level-2-Core-20001113, 2000]. Adicionalmente, não existe nenhuma entidade do modelo que dê suporte à adaptação de conteúdo.

2.2 Entidades Básicas do NCM

A linguagem declarativa NCL tem por base o modelo NCM (*Nested Context Model*) [Soares *et al.*, 2005], cuja descrição em detalhes é fornecida no Apêndice C. Nesta seção faremos apenas uma introdução aos conceitos do NCM, de um modo bastante informal. A Figura 2.2 apresenta as entidades básicas do modelo. Na figura, os termos são propositalmente deixados em inglês para um melhor casamento com os elementos da linguagem NCL.

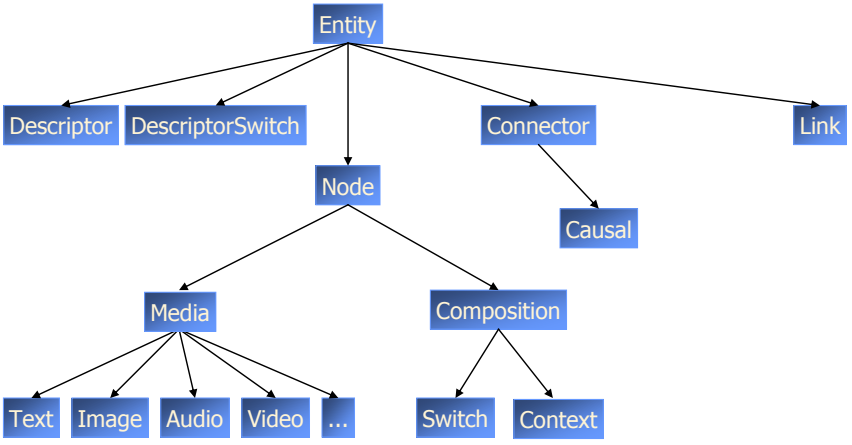


Figura 2.2 Modelo conceitual NCM básico.

Um *nó* (*node* na Figura 2.2) NCM, também chamado *objeto* NCM, possui um identificador, um conteúdo e um conjunto de âncoras. O *conteúdo* de um nó é formado por um conjunto de *unidades de informação*. A noção exata do que constitui uma unidade de informação é parte da definição do nó

e depende de sua especialização, conforme será exemplificado mais adiante. Uma *âncora* (*anchor* na Figura 2.3) é um subconjunto das unidades de informação de um nó.

Um *nó de mídia*, ou nó de conteúdo (*media* na Figura 2.2), representa o que chamamos no Capítulo 1 de *objeto de mídia*. Nós de mídia devem ser especializados em subclasses para melhor definir a interpretação do conteúdo (texto, imagem, áudio, vídeo, objetos com código Lua, objetos com código imperativo Java, objetos com código declarativo HTML etc.). Por exemplo, em um nó de vídeo, o conjunto de unidades de informação pode ser formado pelos quadros que, em sequência, compõem seu conteúdo. Uma âncora, nesse caso, poderia definir um trecho do vídeo. Uma âncora especial é aquela que delimita todo o conteúdo.

Âncoras em NCM são definidas em separado do conteúdo de um nó. Como já mencionamos no Capítulo 1, o NCM é um modelo baseado na estrutura (*structure-based*) e não no conteúdo da mídia (*media-based*), como XHTML.

Usando como metáfora o conjunto de animais racionais, podemos fazer a seguinte analogia. Podemos dizer que o conjunto de animais racionais é composto por seres humanos (objetos de mídia). Todo ser humano tem um identificador, por exemplo, seu CPF. Partes do ser humano são suas âncoras, coração, cérebro etc. O corpo, como um todo, é a âncora que delimitaria todo o conteúdo. Da mesma forma que nós de mídia têm tipos (áudio, vídeo etc.), os seres humanos também podem ser classificados quanto ao sexo, raça etc.

Os seres humanos possuem propriedades, como, por exemplo, o dinheiro que carregam, seu cargo no trabalho etc. Da mesma forma, nós NCM possuem *propriedades* (*property* na Figura 2.3). Por exemplo, para nós de mídia, sua cor de fundo, sua posição na tela etc. são suas propriedades. Os seres humanos se relacionam através de suas propriedades e âncoras. De forma análoga, os objetos de mídia relacionam entre si. As propriedades e âncoras de um objeto de mídia constituem suas *interfaces*.

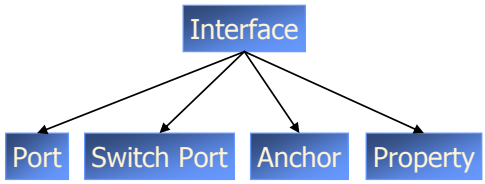


Figura 2.3 Interfaces de um nó NCM.

Os seres humanos se vestem de acordo com a situação. Por exemplo, se vai a uma festa de gala, um homem pode usar terno e gravata; porém, se vai à praia, usa um calção. Da mesma forma, objetos (nós) de mídia podem ser exibidos de forma diferente, conforme a situação. A forma como um objeto de mídia deve ser exibido, onde e por quem é especificada na entidade *descritor* (*descriptor* na Figura 2.2). É nessa entidade que são dadas as características iniciais da apresentação. Iniciais porque elas podem mudar com o tempo. Por exemplo, o posicionamento de um objeto pode mudar, e, de forma análoga, um ser humano pode mudar sua vestimenta com o tempo (no decorrer da animação da festa de gala, o homem pode, por exemplo, tirar a gravata). Devemos notar que o descritor também define onde o objeto deve ser apresentado, incluindo aí o dispositivo de exibição. Assim, essa entidade é a base para o suporte a múltiplos dispositivos de exibição.

Note que a forma como um ser humano se veste pode ser adaptada à situação. Do mesmo modo, a forma como um nó é exibido depende da situação. Assim, podemos associar vários descritores a um mesmo nó de mídia, que serão escolhidos de acordo com a situação, de acordo com uma *regra* (*rule*). O conjunto de descritores alternativos deve ser definido dentro da entidade *switch de descritores* (*descriptorSwitch* na Figura 2.2). Essa entidade é que vai permitir à NCL a definição declarativa da forma como um conteúdo deve ser apresentado, *adaptada* a um certo contexto de exibição.

Objetos de mídia podem se relacionar, assim como os seres humanos. Exemplos de relacionamentos foram definidos no Capítulo 1; por exemplo, os relacionamentos temporais e espaciais entre objetos de mídia, tão mencionados naquele capítulo.

Um relacionamento é definido por uma relação e pelos participantes da relação. Voltando à nossa metáfora, uma relação entre seres humanos poderia ser definida como "homem é casado com mulher". Em um relacionamento, um ser humano do tipo homem vai exercer o papel homem na relação e outro do tipo mulher vai exercer o papel correspondente à mulher. Assim, temos o relacionamento "João é casado com Maria". Devemos notar que, em uma sociedade que não admite a poligamia, apenas um ator pode exercer cada papel nessa relação. Tomemos agora mais três relações entre seres humanos: "ser humano é amigo de ser humano", "ser humano tipo cardiologista cuida do enfarte de ser humano", "ser humano paga dinheiro a ser humano". Na primeira dessas relações, vários seres humanos podem exercer o mesmo papel, por exemplo: Tadeu é amigo de José e de Antônio. Conclusão: relacionamentos podem ser do tipo n:m. Na segunda relação descrita, o relacionamento se dá entre partes do ser humano, por exemplo: "Marcos (um cardiologista) cuida do enfarte do *coração* de Severino". Conclusão: relacionamentos relacionam âncoras; como caso particular, a âncora que representa todo o conteúdo. No terceiro caso de relação descrito, o

relacionamento se dá entre ser humano e uma propriedade do ser humano, por exemplo: Gustavo paga dinheiro a Marcelo (aumentando o valor de propriedade "quantidade de dinheiro" do Marcelo). Conclusão: os papéis de uma relação podem ser exercidos por quaisquer das interfaces (âncoras ou propriedades). Como conclusão geral: *um relacionamento é n:m e relaciona interfaces*.

Façamos agora um paralelo da nossa metáfora com os objetos de mídia NCM. Em NCM, uma relação é definida pela entidade *conector* (*connector* na Figura 2.2). Um conector define uma relação através de seus papéis (*roles*) e da "cola" (*glue*) entre os papéis. Por exemplo: um trecho de nó de mídia, *ao ter sua apresentação iniciada, deve iniciar a apresentação* de um trecho de nó de mídia (a cola foi destacada em itálico). Os papéis "um trecho do nó de mídia" podem ser exercidos por qualquer âncora, como no exemplo de relacionamento: no exato momento de uma novela (trecho de um nó de vídeo) em que uma atriz dá uma mordida em um pedaço de pizza, deve ser iniciada a propaganda da pizzaria (um nó de imagem por inteiro).

No NCM, os conectores podem ser causais e de restrição, mas no caso da NCL 3.0, em seu perfil para TV digital, os conectores são sempre causais (a "cola" é sempre de causalidade), ou seja, quando as condições em um conjunto de papéis forem satisfeitas, um conjunto de ações deve ser aplicado em um conjunto de papéis. Conectores definem relações n:m entre interfaces de nós.

No NCM, a ligação entre o papel de um conector e uma interface de um nó se dá através da entidade *elo* (*link* na Figura 2.2). Assim, um elo é composto por um conector e por ligações (*binds*) entre os papéis dos conectores e interfaces de nós que exercem o papel. Através de elos, relacionamentos de sincronismo temporal e espacial podem ser definidos, em particular relacionamentos com interação do usuário.

Como dissemos, um descritor, ou um switch de descritores, pode ser associado a um nó definindo como, onde e por quem será exibido. Essas entidades podem ser associadas ao nó através de um atributo do nó, ou através de um elo, mais precisamente através de uma ligação (*bind*) de um elo. No caso de dupla associação, as definições contidas nas associações realizadas por elos se sobrepõem àquelas realizadas por atributos de nós.

Voltando à nossa metáfora, os seres humanos em geral pertencem a um conjunto de organizações ou estruturas. Por exemplo, João mora em uma casa, pertence a um clube, toma regularmente um certo ônibus etc. Vamos chamar essas estruturas que agrupam os seres humanos de contêineres. Note que um contêiner pode conter outros contêineres e mais seres humanos. Por exemplo, uma rua pode conter várias casas e seres humanos transeuntes; em uma dessas casas pode morar João e Maria. Uma rua está dentro de um

bairro, que está dentro de uma cidade, que pertence a um estado etc. Devemos notar que uma organização não é necessariamente hierárquica e que um objeto pode pertencer a mais de uma estrutura (contêiner). Mais ainda, devemos notar que, dentro de uma organização, os seres humanos se relacionam. Por exemplo, no trabalho (contêiner empresa), Joaquim é patrão de Manoel.

De forma análoga, é bastante interessante poder organizar objetos de mídia em conjuntos. Tomemos um exemplo típico de uma aplicação de TV. Uma novela é composta de capítulos, que por sua vez são compostos de cenas, que são compostos por tomadas, compostas de trechos de vídeo, trechos de áudio, textos de legenda, informações adicionais sobre um ator, propagandas enxertadas (que podem ser contêineres incluindo outros objetos de mídia) e outros objetos de mídia. Note que objetos de mídia podem se relacionar em cada um desses níveis organizacionais. Organizar os objetos independentes da forma temporal e espacial como serão apresentados é muito importante. Os contêineres NCM são chamados de *composição* (*composition* na Figura 2.2) e são a base da organização lógica de um documento NCM.

Um *nó de contexto* (*context* na Figura 2.2), ou *objeto de contexto*, é um tipo de nó de composição cujo conteúdo inclui um conjunto de nós de mídia e um conjunto de nós de composição, recursivamente. Daí o nome de modelo de *contextos aninhados*. Um nó de contexto possui como atributos adicionais um conjunto de elos relacionando seus objetos (nós).

Voltando à nossa metáfora, um ser humano dentro de um contêiner pode querer se relacionar com outro ser humano em outro contêiner. Por exemplo, João pode estar em um quarto de uma casa e querer conversar com Maria, que está em outro quarto. Para tanto, João deve abrir portas de quartos até chegar à Maria. Portanto, contêineres devem ter portas para permitir relacionamentos com o exterior.

De forma análoga, um nó de composição, em particular um nó de contexto, possui nas suas interfaces, além de âncoras e propriedades, *portas* (ver Figura2.3). Essas portas são mapeadas em outras interfaces interiores à composição e são elas que permitem à composição expor, controladamente, as interfaces de seus nós internos. Assim, cada *elo* contido no conjunto de elos de um nó de contexto *C* pode definir um relacionamento entre interfaces de nós *recursivamente contidos* em *C*.

Vamos ver através de um exemplo como isso é útil. Tomemos novamente o exemplo do jogo de futebol da Seção 1.3.1 do Capítulo 1. Relembrando, sincronizado com o exato momento em que um jogador amarra sua chuteira, aparece um outro objeto de mídia, por exemplo um outro vídeo, fazendo a propaganda da marca da chuteira e, sincronizado com o final desse vídeo, um formulário para compra. Ora, como essa propaganda pode

aparecer em vários momentos do jogo, mais ainda, em vários jogos diferentes, é conveniente defini-la em um contexto (contexto da propaganda na Figura 2.4) que poderá ser reusado em outro documento (reúso é abordado nos Capítulos 3 e 13). Vamos assumir que a propaganda ocorreu em dois momentos, definidos pelos trechos de vídeo (âncoras A e B na Figura 2.4) do jogo. Nesses momentos, o vídeo dentro do contexto da propaganda deve ser iniciado, o que pode ser feito pela porta P do contexto, como mostra a Figura 2.4.

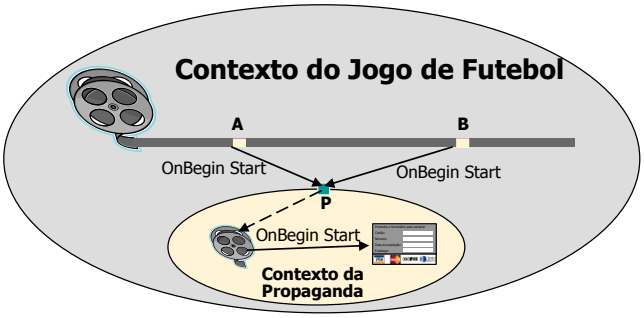


Figura 2.4 Interfaces de um nó NCM.

Finalmente, ainda dentro de nossa metáfora, os seres humanos podem estar presentes em um contêiner e ser selecionados para um relacionamento, dependendo de uma regra a ser avaliada em um momento preciso. Tomemos como exemplo um consultório dentário, onde a sala de espera pode conter vários seres humanos. Um dentista vai se relacionar (tratar) em um dado instante de tempo com apenas um ser humano, escolhido, por exemplo, pela ordem de chegada.

De forma análoga, em uma aplicação pode ser necessária a escolha entre nós (objetos). Por exemplo, imagine no caso da Figura 2.4 que, dependendo da língua do usuário telespectador, fosse escolhido um formulário para compra da chuteira em inglês ou em português. Com o objetivo de permitir a especificação de alternativas de nós a serem exibidos, o NCM define uma entidade chamada nó switch. O *nó switch* (*switch* na Figura 2.2), ou *objeto switch*, é uma outra especialização de nós de composição. O conteúdo de um nó switch é um conjunto que pode incluir nós de contexto e nós de mídia. O nó switch tem um atributo adicional que define, para cada nó contido no seu conjunto de nós, uma regra associada. As regras são definidas em uma lista ordenada. Em um dado instante de exibição, o primeiro nó que tenha a sua regra avaliada como verdadeira deve ser eleito a *alternativa selecionada*.

Além da já mencionada lista ordenada de portas, comum a todo nó de composição, nós switch possuem uma lista ordenada de *portas-switch* (*switch port* na Figura 2.3). Portas-switch contituem mais um ponto de interface. Elas definem um conjunto de mapeamentos para interfaces de nós contidos no nó switch. A definição de portas-switch permite que elos sejam definidos ancorando em nós switch, independentemente do nó filho que será selecionado. As portas tradicionais do switch permitem que elos sejam associados a alternativas específicas. Se essa alternativa não for selecionada, o elo será simplesmente ignorado durante a apresentação do documento.

Assim como um switch de descritores é o suporte necessário para a definição de alternativas da forma de exibição de um mesmo conteúdo de mídia, um nó switch é o suporte necessário para a definição de alternativas de conteúdos.

Terminando este capítulo, a Figura 2.5 mostra como entidades do modelo NCM estão relacionadas a elementos da linguagem NCL. No perfil NCL 3.0 para TV digital, nem todas as facilidades providas pelo modelo NCM são refletidas na linguagem, mas todas as facilidades apresentadas nesta seção são refletidas.

O leitor interessado pode obter uma descrição mais detalhada do modelo conceitual NCM no Apêndice C.

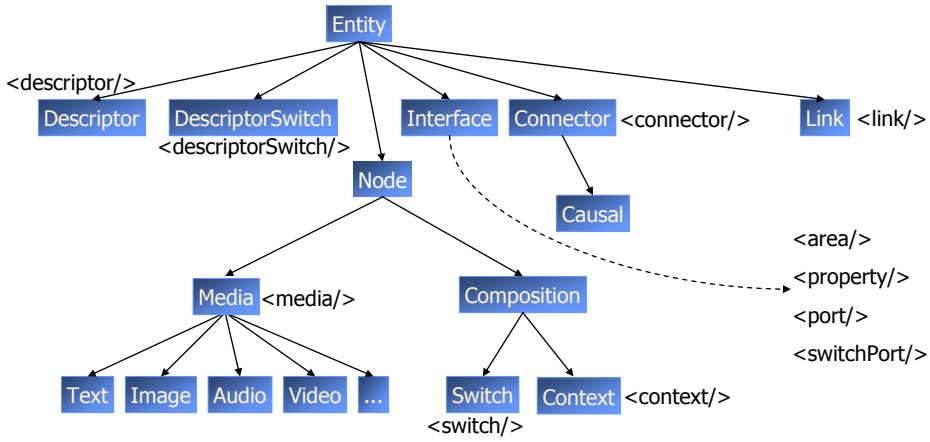


Figura 2.5 Entidades NCM e elementos da linguagem NCL.

IETF RFC 2396 (1998). "Uniform Resource Identifiers (URI): Generic Syntax." *RFC 2396*.

Soares, L.F.S. e Rodrigues, R.F. (2005). "Nested Context Model 3.0 Part 1 — NCM Core." Monografias em Ciência da Computação do Departamento de Informática, PUC-Rio, N.º 18/05. Rio de Janeiro. Maio. ISSN 0103-9741.

W3C REC-CSS2-19980512 (1998). World Wide Web Consortium, "Cascading Style Sheets, level 2", W3C Recommendation *CSS2-19980512*.

W3C REC-DOM-Level-2-Core-20001113 (2000). World Wide Web Consortium, "Document Object Model (DOM) Level 2 Core Specification", W3C Recommendation *DOM-Level-2-Core-20001113*.

W3C REC-xhtml1-20020801 (2002). World Wide Web Consortium, "XHTML™ 1.0 The Extensible HyperText Markup Language", *W3C Recommendation xhtml1-20020801*.

Capítulo 3

Introdução à Linguagem NCL

A NCL (*Nested Context Language*) é uma linguagem declarativa, uma aplicação XML. Baseada no modelo conceitual NCM, a NCL traz uma separação clara entre os conteúdos de mídia e a estrutura de uma aplicação. Um documento NCL apenas define como os objetos de mídia são estruturados e relacionados, no tempo e no espaço. Como uma linguagem de cola, ela não restringe nem prescreve os tipos de conteúdo dos objetos de mídia de uma aplicação.

Este capítulo traz uma introdução passo a passo de diversos elementos que compõem o perfil NCL 3.0 para TV digital. Nesta introdução, não existe a preocupação de definir cada elemento da linguagem e cada um de seus atributos, o que será feito na Parte 2 do livro. O intuito aqui é já fornecer ao leitor a habilidade de desenvolver programas NCL simples e capacitá-lo a melhor entender e exercitar os demais conceitos apresentados na Parte 2. A introdução será feita através de um único exemplo, construído paulatinamente.[1]

[1] Os conteúdos dos exemplos deste capítulo estão disponíveis sob a Licença Creative Commons Atribuição — Uso Não-Comercial — Compartilhamento pela mesma Licença 3.0 Unported, conforme publicado em clube.ncl.org.br.

3.1 O Primeiro João

Para introduzir a programação em NCL, vamos utilizar um único exemplo, *O Primeiro João*, que construiremos passo a passo.

O Primeiro João é baseado em um vídeo, uma animação de mesmo nome produzida por André Castelão, que por sua vez foi baseado nas crônicas de Mané Garrincha, escritas por Gerson Soares. A animação conta por que Garrincha passou a chamar de João todos os seus marcadores. A história se passa em um campo de pelada, onde Mané, ainda pré-adolescente, pobre, já fazia sua história. Ao ser marcado impiedosamente, Garrincha arrasa, com dribles desconcertantes, o time adversário e, principalmente, seu marcador, o pobre primeiro João; dribles esses que seriam repetidos na gloriosa carreira do ídolo brasileiro.

Vamos dividir nossa aplicação em 12 partes. Ao final deste capítulo teremos nosso programa não-linear interativo completo.

3.2 Sincronismo de Mídia sem Interatividade e com Reúso Apenas de Relação

Durante o vídeo da animação *O Primeiro João*, Garrincha dá o seu famoso drible em que leva os marcadores para a ponta, finge que vai avançar e retorna, por diversas vezes, até que finalmente realiza o drible e passa. Esse drible foi aplicado pelo Mané por diversas vezes em jogos oficiais, inclusive na Copa do Mundo pela seleção brasileira. Em outro trecho do vídeo, mais à frente, aparece o Garrincha e seu marcador, caído no chão, após um drible desconcertante. Cena idêntica aconteceu em um jogo oficial do Botafogo e Vasco, anos depois.

Nossa primeira aplicação, bem simples, ilustra como é fácil, em NCL, introduzir vários objetos de mídia sincronizados no tempo. Vamos acrescentar:

1. uma música de fundo (um chorinho), que deverá começar assim que terminar a apresentação inicial do vídeo e começar a animação propriamente dita;
2. um outro objeto de vídeo, que deverá ser exibido em paralelo e sincronizado com o famoso “drible do vaivém” do Mané, retratado na animação; e ainda
3. uma outra imagem, uma foto, que deverá ser exibida junto com a cena do marcador caído no chão.

O novo vídeo acrescentado é uma réplica, uma clonagem do drible real aplicado por Garrincha, realizado por garotos da Comunidade Beira Rio, no Rio de Janeiro, produzido pelo Ponto de Cultura CIDS-VG, daquela comunidade. Ele mostra garotos descalços em um campo de várzea, como era Garrincha na época em que aplicou o drible no primeiro João. O mesmo Ponto de Cultura é também responsável pela foto que retrata exatamente a mesma situação ocorrida na animação, quando o marcador do Garrincha vai ao chão, foto representada pelos mesmos garotos atores.

A Figura 3.1 ilustra a visão temporal da aplicação.

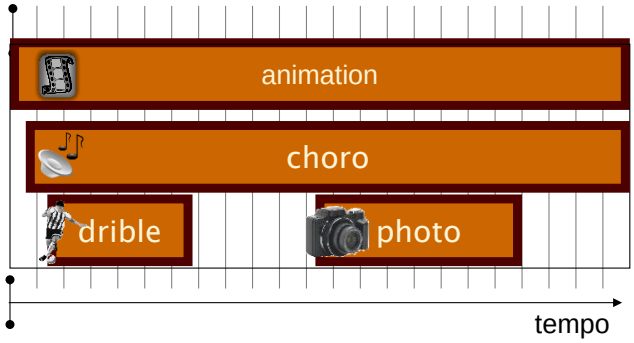


Figura 3.1 Visão temporal.

Para definir todos esses objetos de mídia, a NCL faz uso de seu elemento <media>, como ilustrado na Listagem 3.1.

```
<media id="animation" src="../media/animGar.mp4"/>
<media id="choro" src="../media/choro.mp3"/>
<media id="drible" src="../media/drible.mp4"/>
<media id="photo" src="../media/photo.png"/>
```

Listagem 3.1 Elementos <media>.

Todo elemento <media> tem um identificador, definido pelo atributo *id*, para que possa ser posteriormente referenciado. Ele tem também um atributo denominado *src*, que contém um URI para a localização do conteúdo. Em todos os casos de nosso exemplo, os conteúdos estão no diretório "/media", filho do mesmo diretório onde está definida a aplicação. É usual, em aplicações para TV digital, o atributo *src* referenciar um fluxo elementar MPEG System[2]. Não faremos isso no nosso exemplo para facilitar sua

[2] Como veremos no Capítulo 8, se referenciarmos um fluxo elementar MPEG System, a URI usada será: "sbtvd-ts://program_number.component_tag. Essa URI identifica o serviço dentro de um canal sintonizado da TV, por meio do *program_number;* e identifica um fluxo elementar, um vídeo por exemplo, dentro do serviço especificado, por meio do *component_tag*.

execução nos dispositivos mais usuais disponíveis para o leitor. Voltaremos a esse ponto no Capítulo 8.

Note que todo elemento NCL deve começar com o caractere "<". Quando o elemento não tem nenhum conteúdo nem elementos filhos, ele deve terminar sempre com "/>", como na Listagem 3.1. Em caso contrário, a definição dos atributos do elemento termina com o caractere ">". Após a definição de seu conteúdo ou elementos filhos, o elemento termina repetindo seu nome envolvido com os caracteres "</" na frente e o caractere ">" atrás, como visto a seguir, na redefinição do elemento <media id="animation".../> da Listagem 3.1 que, na Listagem 3.2 representa o vídeo da animação com dois elementos filhos aninhados, denominados <area>.

```
<media id="animation" src="../media/animGar.mp4"
                                              descriptor="screenDesc">
       <area id="segDrible" begin="12s"/>
       <area id="segPhoto" begin="41s"/>
</media>
```

Listagem 3.2 Elementos <area>.

Os elementos <area> delimitam trechos (no tempo ou espaço) do conteúdo do seu objeto de mídia pai. No caso, são delimitados o instante em que, na animação, o Garrincha inicia o drible de vaivém e o instante, após outro drible, em que o marcador aparece caído no chão. O primeiro instante é delimitado por meio do atributo *begin* igual a 12 segundos, e o segundo pelo atributo *begin* igual a 41 segundos. Como não foi especificado o valor do atributo *end*, ele é assumido, por default, como tendo o mesmo valor do final do vídeo da animação, em ambos os casos. Note que todo elemento <area> também possui um identificador (atributo *id*).

Nosso próximo passo no projeto da aplicação é definir em que posição da tela os vários objetos de mídia serão exibidos. A Figura 3.2 apresenta a visão de leiaute desejada. São definidas regiões para exibição do vídeo da animação ("screenReg"), em tela cheia, e uma região para exibição do vídeo do drible e da foto ("frameReg").

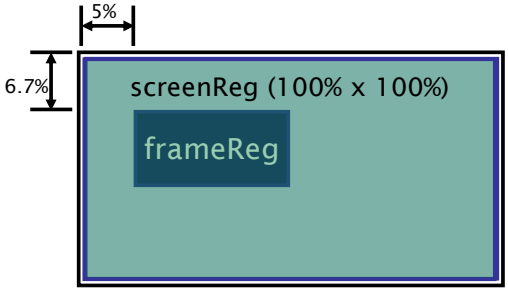


Figura 3.2 Visão de leiaute.

A definição dos espaços de exibição pode ser realizada por meio de elementos <property>, filhos dos elementos <media> que representam cada objeto de mídia da aplicação, como ilustrado na Listagem 3.3.

```
<media id="animation" src="../media/animGar.mp4">
        <area id="segDrible" begin="11.5s"/>
        <area id="segPhoto" begin="41s"/>
        <property name="width" value="100%"/>
        <property name="heigth" value="100%"/>
        <property name="zIndex" value="2"/>
</media>
<media id="choro" src="../media/choro.mp3"/>
<media id="drible" src="../media/drible.mp4">
        <property name="left" value="5%"/>
        <property name="top" value="6.7%"/>
        <property name="width" value="18.5%"/>
        <property name="heigth" value="18.5%"/>
        <property name="zIndex" value="3"/>
</media>
<media id="photo" src="../media/photo.png">
        <property name="left" value="5%"/>
        <property name="top" value="6.7%"/>
        <property name="width" value="18.5%"/>
        <property name="heigth" value="18.5%"/>
        <property name="zIndex" value="3"/>
        <property name="explicitDur" value="5s"/>
</media>
```

Listagem 3.3 Elementos <property>.

Todo elemento <property> possui um identificador válido no escopo do objeto de mídia, definidos por seu atributo *name*. Por exemplo, “left”, “top” “width”, “height”, “right”, e “bottom” são propriedades que definem a área de apresentação em relação à tela total do dispositivo de exibição (os valores defaults para esses atributos, bem como o que acontece em caso de inconsistência de valores, são discutidos na Parte II do livro). Essas propriedades podem ser definidas de forma absoluta ou como uma porcentagem, sempre com relação à tela total do dispositivo de exibição. A propriedade “zIndex” especifica como fica a sobreposição de regiões. Uma região de maior valor para *zIndex* se sobrepõe àquela de menor valor. Assim, no exemplo, são definidas uma região para exibição do vídeo da animação em tela cheia, e uma região para exibição do vídeo do drible e da foto se sobrepondo ao vídeo da animação.

Elementos <property> podem ser usados para a definição do valor de qualquer propriedade de um objeto de mídia, e não apenas aquelas que definem espaços de exibição. Por exemplo, para o objeto de mídia representando a foto, uma duração de 5s pode ser associada ao elemento <media> correspondente, por meio de sua propriedade "explicitDur", como ilustra a Listagem 3.3.

Finalmente, podemos agora definir os relacionamentos de sincronização entre os vários objetos de mídia. Em NCL, o elemento <body> agrupa todos os objetos de um documento e seus relacionamentos.

O elemento <body> é único em um documento NCL e pode ter um ou mais elementos <port> como filho. O elemento <port> indica por onde pode começar uma exibição do documento. No nosso exemplo, como ilustrado na Listagem 3.4, a exibição começa sempre pela apresentação do vídeo da animação.

```
<body>
    <port id="entry" component="animation"/>

    <!—definição dos diversos elementos de <media> do documento -->

    <!—definição dos diversos relacionamentos, elementos <link> do
                                                    documento -->
</body>
```

Listagem 3.4 Elementos <body> e <port>.

Relacionamentos são definidos por elementos <link>, como ilustrado na Listagem 3.5, para nosso exemplo de *O Primeiro João*. O relacionamento ilustrado define que, ao ser iniciada a exibição do vídeo da animação, o áudio com o chorinho também deve ser iniciado, mas 5 segundos depois (como determinamos no enunciado do exemplo, esse objeto de mídia é apresentado após os créditos iniciais do vídeo da animação, que duram 5 s).

```
<link  id="lMusic" xconnector="onBeginStart_delay">
       <bind role="onBegin" component="animation"/>
       <bind role="start" component="choro">
              <bindParam name="delay" value="5s"/>
       </bind>
</link>
```

Listagem 3.5 Elementos <link> e <bind>.

Um relacionamento em NCL é especificado referindo-se a uma relação, dada pelo valor URI do atributo *xconnector* do elemento <link>, e pelos

atores que exercerão os papéis da relação, dados pelos elementos <bind>. Um elemento <bind> especifica o papel da relação, através de seu atributo <role>, e a interface que exercerá o papel, dada pelos atributos *component*, que seleciona um objeto a ser relacionado, e *interface*, que seleciona uma interface desse objeto. Por enquanto, as únicas interfaces que definimos foram por meio de elementos <area>. Quando não especificada, o valor default para a *interface* assume uma área contendo todo o conteúdo do objeto.

No relacionamento definido pelo elemento <link> "lMusic" da Listagem 3.5, a relação referenciada é definida pelo elemento <causalConnector>, filho do elemento <connectorBase>, conforme ilustra a Listagem 3.6. Na relação (Listagem 3.5), o papel "onBegin" do conector é associado ao componente "animation", que, como vimos na Listagem 3.2, é associado ao vídeo da animação, definindo que, ao começar a exibição do vídeo, deve-se dar partida (*role* = "start") à exibição do chorinho, mas 5 segundos a partir do início do vídeo.

```
<connectorBase>
   <causalConnector id="onBeginStart_delay">
      <connectorParam name="delay"/>
      <simpleCondition role="onBegin"/>
      <simpleAction role="start" delay="$delay" max="unbounded"
                                                    qualifier="par"/>
   </causalConnector>
</connectorBase>
```

Listagem 3.6 Elemento <causalConnector> e seus elementos filhos.

A relação é causal, onde a condição é dada pelo papel "onBegin", e a ação é "start", com o parâmetro "delay" (retardo) a ser definido pelo relacionamento (o elemento <link>), quando da sua especificação. O atributo *max*="unbounded" definido na ação especifica que um número ilimitado de atores pode exercer esse papel. Quando existe mais de um ator para o papel, o atributo *qualifier* define a forma como as ações devem ser executadas, se em paralelo ou em sequência.

A Figura 3.3 apresenta a visão estrutural da aplicação com todos os relacionamentos entre os vários objetos de mídia definidos.

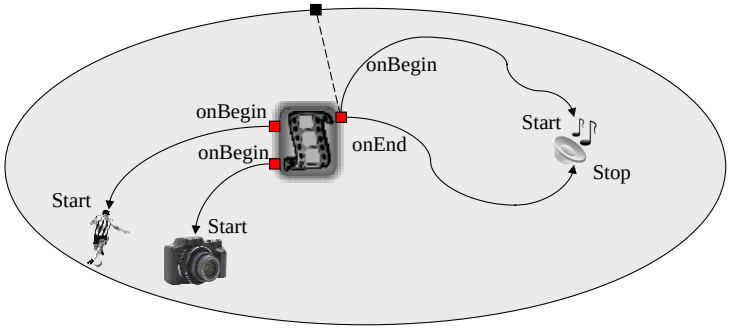


Figura 3.3 Visão estrutural.

Bases de conectores são definidas como elemento filho do elemento <head>, que define, como veremos, as partes reusáveis de uma aplicação. No caso, um mesmo conector pode ser usado por mais de um elemento <link>. A Listagem 3.7 apresenta a definição completa do elemento <head> da aplicação exemplo.

```
<head>
   <connectorBase>
      <causalConnector id="onBeginStart_delay">
        <connectorParam name="delay"/>
        <simpleCondition role="onBegin"/>
        <simpleAction role="start" delay="$delay" max="unbounded"
                                                    qualifier="par"/>
      </causalConnector>
      <causalConnector id="onBeginStart">
        <simpleCondition role="onBegin"/>
        <simpleAction role="start" max="unbounded" qualifier="par"/>
      </causalConnector>
      <causalConnector id="onEndStop">
        <simpleCondition role="onEnd"/>
        <simpleAction role="stop" max="unbounded" qualifier="par"/>
      </causalConnector>
   </connectorBase>
</head>
```

Listagem 3.7 Elemento <head> e seus elementos filhos.

Podemos agora definir todo o elemento <body> e seus filhos, para o exemplo, apresentados na Listagem 3.8.

```
<body>
   <port id="entry" component="animation"/>
   <media id="animation" src="../media/animGar.mp4">
```

```
      <area id="segDrible" begin="11.5s"/>
      <area id="segPhoto" begin="41s"/>
      <property name="width" value="100%"/>
      <property name="heigth" value="100%"/>
      <property name="zIndex" value="2"/>
   </media>
   <media id="choro" src="../media/choro.mp3"/>
   <media id="drible" src="../media/drible.mp4">
      <property name="left" value="5%"/>
      <property name="top" value="6.7%"/>
      <property name="width" value="18.5%"/>
      <property name="heigth" value="18.5%"/>
      <property name="zIndex" value="3"/>
   </media>
   <media id="photo" src="../media/photo.png">
      <property name="left" value="5%"/>
      <property name="top" value="6.7%"/>
      <property name="width" value="18.5%"/>
      <property name="heigth" value="18.5%"/>
      <property name="zIndex" value="3"/>
      <property name="explicitDur" value="5s"/>
   </media>
   <link  id="lMusic" xconnector="onBeginStart_delay">
      <bind role="onBegin" component="animation"/>
      <bind role="start" component="choro">
        <bindParam name="delay" value="5s"/>
      </bind>
   </link>
   <link  id="lDrible" xconnector="onBeginStart">
      <bind role="onBegin" component="animation"
                                          interface="segDrible"/>
      <bind role="start" component="drible"/>
   </link>
   <link  id="lPhoto" xconnector="onBeginStart">
      <bind role="onBegin" component="animation" interface="segPhoto"/>
      <bind role="start" component="photo"/>
   </link>
   <link  id="lEnd" xconnector="onEndStop">
      <bind role="onEnd" component="animation"/>
      <bind role="stop" component="choro"/>
   </link>
</body>
```

Listagem 3.8 Elemento <body> e seus elementos filhos.

O elemento <link> “lDrible” define que, ao começar (*role* “onBegin”) o trecho do vídeo da animação correspondente ao drible de vaivém do Garrincha (*component* “animation” *interface* “segDrible”), o vídeo do drible deve ser iniciado (*role* “start”).

De forma análoga, o elemento <link> “lPhoto” define que, ao começar (*role* “onBegin”) o trecho do vídeo da animação correspondente ao momento em que o marcador cai no chão (*component* “animation” *interface* “segPhoto”), a foto do marcador deve ser exibida (*role* “start”).

Finalmente, o elemento <link> “lEnd” define que, ao findar (“onEnd”) o vídeo da animação, o chorinho deve parar (“stop”) de tocar.

Os elementos <head> e <body> devem ser declarados como filhos do elemento <ncl>, completando a definição do documento: a aplicação declarativa NCL.

A Listagem 3.9 ilustra a aplicação completa. Note que toda aplicação NCL começa com a linha:
`<?xml version="1.0" encoding="ISO-8859-1"?>`
seguida da especificação do elemento <ncl>, onde o atributo *id* define um identificador para a aplicação e o atributo *xmlns* define o espaço de nomes (namespace) onde estão definidos os esquemas do perfil NCL EDTV, para verificação, pelo parser XML, da validade da aplicação.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Exemplo de sincronismo sem a interacao do usuario e com reúso
apenas de relações-->
<ncl id="syncEx" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
   <head>
     <connectorBase>
       <causalConnector id="onBeginStart_delay">
         <connectorParam name="delay"/>
         <simpleCondition role="onBegin"/>
         <simpleAction role="start" delay="$delay" max="unbounded"
                                                    qualifier="par"/>
       </causalConnector>
       <causalConnector id="onBeginStart">
         <simpleCondition role="onBegin"/>
         <simpleAction role="start" max="unbounded" qualifier="par"/>
       </causalConnector>
       <causalConnector id="onEndStop">
         <simpleCondition role="onEnd"/>
         <simpleAction role="stop" max="unbounded" qualifier="par"/>
       </causalConnector>
     </connectorBase>
```

```
  </head>

  <body>
    <port id="entry" component="animation"/>
    <media id="animation" src="../media/animGar.mp4">
      <area id="segDrible" begin="11.5s"/>
      <area id="segPhoto" begin="41s"/>
      <property name="width" value="100%"/>
      <property name="heigth" value="100%"/>
      <property name="zIndex" value="2"/>
    </media>
    <media id="choro" src="../media/choro.mp3"/>
    <media id="drible" src="../media/drible.mp4">
      <property name="left" value="5%"/>
      <property name="top" value="6.7%"/>
      <property name="width" value="18.5%"/>
      <property name="heigth" value="18.5%"/>
      <property name="zIndex" value="3"/>
    </media>
    <media id="photo" src="../media/photo.png">
      <property name="left" value="5%"/>
      <property name="top" value="6.7%"/>
      <property name="width" value="18.5%"/>
      <property name="heigth" value="18.5%"/>
      <property name="zIndex" value="3"/>
      <property name="explicitDur" value="5s"/>
    </media>
    <link  id="lMusic" xconnector="onBeginStart_delay">
      <bind role="onBegin" component="animation"/>
      <bind role="start" component="choro">
        <bindParam name="delay" value="5s"/>
      </bind>
    </link>
    <link  id="lDrible" xconnector="onBeginStart">
      <bind role="onBegin" component="animation"
                                          interface="segDrible"/>
      <bind role="start" component="drible"/>
    </link>
    <link  id="lPhoto" xconnector="onBeginStart">
      <bind role="onBegin" component="animation"
                                          interface="segPhoto"/>
      <bind role="start" component="photo"/>
    </link>
    <link  id="lEnd" xconnector="onEndStop">
```

```
        <bind role="onEnd" component="animation"/>
        <bind role="stop" component="choro"/>
      </link>
    </body>
</ncl>
```

Listagem 3.9 Documento NCL.

A Figura 3.4 ilustra dois momentos da exibição, o início do drible do Garrincha e o início da apresentação da foto, obtidos durante a execução da aplicação na implementação de referência do middleware Ginga-NCL.

Figura 3.4 Cenas da aplicação *O Primeiro João*.

3.3 Sincronismo de Mídia sem Interatividade, com Reuso de Características de Apresentação e Importação de Base

Nossa segunda aplicação é idêntica à primeira, mas agora especificada com a reutilização das características de exibição dos vários objetos de mídia e com a definição da base de conectores externa ao documento NCL. Ou seja, fora a nova sintaxe do documento NCL, nada mais foi acrescentado.

Nossa primeira alteração diz respeito à definição das regiões onde os diversos objetos de mídia serão posicionados, que não será mais especificada por meio dos elementos <property>, filhos dos elementos <media> correspondentes. Para permitir o reuso das regiões, elas serão definidas à parte, por meio dos elementos <region>. O conjunto de elementos <region> é definido como filho do elemento <regionBase>, como ilustrado na Listagem 3.10.

```
<regionBase>
        <region id="screenReg" width="100%" height="100%"
                                                    zIndex="2">
                <region id="frameReg" left="5%" top="6.7%"
                        width="18.5%" height="18.5%" zIndex="3"/>
        </region>
</regionBase>
```

Listagem 3.10 Elementos <region> e <regionBase>.

Todo elemento <region> possui um identificador e atributos homônimos aos definidos pelos elementos <property> (*left*, *top width*, *height*, *right*, *bottom* e *ZIndex*) que definem sua área de exibição em relação à região pai (os valores defaults para esses atributos, como já mencionamos, bem como o que acontece em caso de inconsistência de valores, são discutidos na Parte II do livro). Note que esses atributos podem ser definidos de forma absoluta ou como uma porcentagem, mas agora sempre com relação aos mesmos atributos da região pai. Quando o elemento não possui uma região pai, o posicionamento é realizado com relação à tela total do dispositivo de exibição. Por exemplo, na Listagem 3.10, a região "frameReg" é definida como filha da região "screenReg", ou seja, todos os valores definidos para seu posicionamento são relativos à região "screenReg". Note ainda que essa região será reusada tanto para a definição do posicionamento da foto quanto do posicionamento do vídeo do drible.

Avançando em nosso projeto da aplicação, temos agora de especificar como as regiões definidas (elementos <region>) são associadas aos vários objetos de mídia definidos (elementos <media>). Isso se dá através dos elementos <descriptor>. O conjunto de todos os descritores é definido como filho do elemento <descriptorBase>, como ilustrado na Listagem 3.11.

```
<descriptorBase>
    <descriptor id="screenDesc" region="screenReg"/>
    <descriptor id="photoDesc" region="frameReg" explicitDur="5s"/>
    <descriptor id="audioDesc"/>
    <descriptor id="dribleDesc" region="frameReg"/>
</descriptorBase>
```

Listagem 3.11 Elementos <descriptor> e <descriptorBase>.

O elemento <descriptor> especifica as características iniciais para a exibição de um objeto de mídia, inclusive seu posicionamento, dado por seu atributo *region*, que é usado para referenciar um elemento <region>. Um descritor pode ter outros atributos adicionais, como o atributo *explicitDur*, que determina o tempo de duração explícita de uma mídia. Por exemplo, o

elemento <descriptor> “photoDesc” especifica que a mídia deve ser exibida por 5 segundos. Note assim que o elemento <descriptor> pode definir outras propriedades de um objeto de mídia que não as de posicionamento, no caso exemplo, a propriedade *explicitDur*, definida na Seção 3.2 por meio do elemento <property>. Propriedades definidas pelo elemento <descriptor> podem ser reusadas na definição de vários objetos de mídia.

É muito importante ressaltar que os elementos <region> e <descriptor> não definem novas propriedades, mas valores iniciais para propriedades reservadas de um objeto de mídia. Várias dessas propriedades serão apresentadas à medida que avançamos com nosso exemplo. Uma lista dessas propriedades é apresentada no Capítulo 9.

Como mencionamos, um elemento <descriptor> é ligado a um objeto de mídia pelo atributo *descriptor* do elemento <media> que representa o objeto, como ilustra a Listagem 3.12 na definição de todos os elementos <media>, agora sem os elementos <property> filhos.

```
<media id="animation" src="../media/animGar.mp4"
                                              descriptor="screenDesc">
       <area id="segDrible" begin="11.5s"/>
       <area id="segPhoto" begin="41s"/>
</media>
<media id="choro" src="../media/choro.mp3" descriptor="audioDesc"/>
<media id="drible" src="../media/drible.mp4"
                                       descriptor="dribleDesc"/>
<media id="photo" src="../media/photo.png" descriptor="photoDesc"/>
```

Listagem 3.12 Elementos <media> redefinidos.

Voltemos nossa atenção agora para os conectores. Em relacionamentos definidos por elementos <link>, a relação referenciada pode ser definida em um documento externo, que deve ser importado para o documento em questão. Isso é especificado pelo elemento <importBase>, filho do elemento <connectorBase>, conforme ilustra a Listagem 3.13. Na listagem, conectores estão definidos em um arquivo, com o nome “causalConnBase.ncl”, no mesmo diretório onde se encontra o diretório com nosso documento NCL de exemplo. Esse arquivo de conectores, usados em todos os exemplos que se seguem neste capítulo, será conhecido pelo apelido “conEx”, como definido pelo atributo *alias*. O conteúdo desse arquivo é apresentado no Apêndice D.

```
<connectorBase>
    <importBase documentURI="../causalConnBase.ncl" alias="conEx"/>
</connectorBase>
```

Listagem 3.13 Elementos <importBase> e <connectorBase>.

Ao referenciar um conector definido externamente ao documento, um elemento <link> deve concatenar o valor do atributo *alias* com o caractere especial # seguido do identificador do conector. Os mesmos conectores definidos na Seção 3.2, quando definidos externamente no arquivo "causalConnBase.ncl", devem ser referenciados como ilustra a Listagem 3.14, que apresenta a redefinição de todos os elementos <link> do exemplo O Primeiro João.

```
<body>


        <link  id="lMusic" xconnector="conEx#onBeginStart_delay">
                <bind role="onBegin" component="animation"/>
                <bind role="start" component="choro">
                        <bindParam name="delay" value="5s"/>
                </bind>
        </link>
        <link id="lDrible" xconnector="conEx#onBeginStart">
                <bind role="onBegin" component="animation"
                                                interface="segDrible"/>
                <bind role="start" component="drible"/>
        </link>
        <link id="lPhoto" xconnector="conEx#onBeginStart">
                <bind role="onBegin" component="animation"
                                                interface="segPhoto"/>
                <bind role="start" component="photo"/>
        </link>
        <link id="lEnd" xconnector="conEx#onEndStop">
                <bind role="onEnd" component="animation"/>
                <bind role="stop" component="choro"/>
        </link>
</body>
```

Listagem 3.14 Redefinição dos elementos <link>.

Todas as bases que definimos, <regionBase>, <descriptorBase> e <connectorBase>, devem ser filhas do elemento <head> da NCL, como ilustrado na Listagem 3.15.

```
<head>
    <regionBase>
       <region id="screenReg" width="100%" height="100%"
                                                    zIndex="2">
          <region id="frameReg" left="5%" top="6.7%" width="18.5%"
                                    height="18.5%" zIndex="3"/>
       </region>
    </regionBase>
    <descriptorBase>
       <descriptor id="screenDesc" region="screenReg"/>
       <descriptor id="photoDesc" region="frameReg"
                                              explicitDur="5s"/>
       <descriptor id="audioDesc"/>
       <descriptor id="dribleDesc" region="frameReg"/>
    </descriptorBase>
    <connectorBase>
       <importBase documentURI="../causalConnBase.ncl"
                                                    alias="conEx"/>
    </connectorBase>
</head>
```

Listagem 3.15 Elemento <head> e seus elementos filhos.

A Listagem 3.16 apresenta a aplicação completa.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Exemplo de sincronismo sem a interacao do usuário, com reuso das
características de exibição e importação de base -->
<ncl id="sync" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
   <head>
      <regionBase>
         <region id="screenReg" width="100%" height="100%"
                                                    zIndex="2">
            <region id="frameReg" left="5%" top="6.7%" width="18.5%"
                                    height="18.5%" zIndex="3"/>
         </region>
      </regionBase>
      <descriptorBase>
         <descriptor id="screenDesc" region="screenReg"/>
         <descriptor id="photoDesc" region="frameReg"
                                              explicitDur="5s"/>
         <descriptor id="audioDesc"/>
         <descriptor id="dribleDesc" region="frameReg"/>
      </descriptorBase>
```

```
      <connectorBase>
         <importBase documentURI="../causalConnBase.ncl"
                                                    alias="conEx"/>
      </connectorBase>
   </head>
   <body>
      <port id="entry" component="animation"/>
      <media id="animation" src="../media/animGar.mp4"
                                    descriptor="screenDesc">
         <area id="segDrible" begin="12s"/>
         <area id="segPhoto" begin="41s"/>
      </media>
      <media id="choro" src="../media/choro.mp3"
                                    descriptor="audioDesc"/>
      <media id="drible" src="../media/drible.mp4"
                                    descriptor="dribleDesc"/>
      <media id="photo" src="../media/photo.png"
                                    descriptor="photoDesc"/>
      <link id="lMusic" xconnector="conEx#onBeginStart_delay">
         <bind role="onBegin" component="animation"/>
         <bind role="start" component="choro">
            <bindParam name="delay" value="5s"/>
         </bind>
      </link>
      <link  id="lDrible" xconnector="conEx#onBeginStart">
         <bind role="onBegin" component="animation"
                                          interface="segDrible"/>
         <bind role="start" component="drible"/>
      </link>
      <link id="lPhoto" xconnector="conEx#onBeginStart">
         <bind role="onBegin" component="animation"
                                          interface="segPhoto"/>
         <bind role="start" component="photo"/>
      </link>
      <link id="lEnd" xconnector="conEx#onEndStop">
         <bind role="onEnd" component="animation"/>
         <bind role="stop" component="choro"/>
      </link>
   </body>
</ncl>
```

Listagem 3.16 Documento NCL com reuso e importação.

3.4 Adicionando Sincronismo com Interatividade

Vamos agora incrementar nossa aplicação introduzindo um exemplo de sincronismo temporal com a interação do usuário.

Em certo trecho do vídeo da animação, ao cair no chão, o adversário do nosso Garrincha mostra sua chuteira. Nesse instante, faremos aparecer o ícone de uma chuteira que, se selecionado pelo telespectador através do botão vermelho do controle remoto, fará aparecer um vídeo de propaganda da chuteira. Lembramos que, na animação, Garrincha era uma criança pobre, ele jogava descalço. Assim, no vídeo de propaganda, também realizado pelo Ponto de Cultura CIDS-VG, aparece uma criança, a mesma que representou Garrincha no vídeo do drible e na foto, sonhando em possuir uma chuteira. Durante a exibição do vídeo de propaganda, o vídeo da animação deverá ser redimensionado, possibilitando a exibição simultânea dos dois vídeos sem sobreposição, mas se sobrepondo a uma imagem de fundo introduzida no exemplo.

A Figura 3.5 ilustra a visão estrutural da nova versão da aplicação. Note o acréscimo de três objetos de mídia, de uma nova âncora para o vídeo animação, de três novos relacionamentos e na modificação (extensão) dos relacionamentos que davam partida ao áudio e comandavam seu fim.

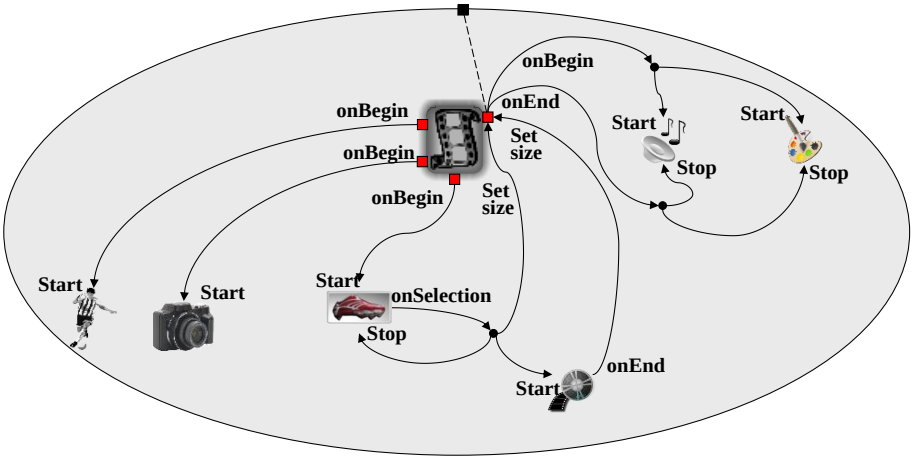


Figura 3.5 Visão estrutural.

Temos, então, de modificar a definição do elemento <media> que representa a animação, introduzindo uma nova âncora, ou seja, um novo elemento <area>, correspondendo ao tempo em que o ícone da chuteira deverá aparecer. Como o vídeo da animação deve ser redimensionado, caso o ícone seja selecionado, é necessário definir as propriedades de leiaute que serão modificadas. As propriedades de um objeto de mídia passíveis de manipulação através de relacionamentos devem ser explicitadas por meio de

elementos <property>, mesmo que seus valores iniciais sejam definidos usando elementos <descriptor> ou <region>. No caso, queremos manipular as variáveis de posicionamento e dimensão (“left”, “top”, “width”, “height”), representadas pelo valor “bounds”, como ilustrado pela Listagem 3.17, onde também é ilustrada a inclusão dos três novos objetos de mídia no conjunto de objetos de mídia anteriormente definido: a imagem de fundo (“background”), o ícone (“icon”) e o vídeo de propaganda (“shoes”).

```
<media id="background" src="../media/background.png"
                                    descriptor="backgroundDesc"/>
<media id="animation" src="../media/animGar.mp4"
                                    descriptor="screenDesc">
       <area id="segDrible" begin="12s"/>
       <area id="segPhoto" begin="41s"/>
       <area id="segIcon" begin="45s" end="51s"/>
       <property name="bounds"/>
</media>
<media id="choro" src="../media/choro.mp3" descriptor="audioDesc"/>
<media id="drible" src="../media/drible.mp4"
                                    descriptor="dribleDesc"/>
<media id="photo" src="../media/photo.png" descriptor="photoDesc"/>
<media id="icon" src="../media/icon.png" descriptor="iconDesc"/>
<media id="shoes" src="../media/shoes.mp4" descriptor="shoesDesc"/>
```

Listagem 3.17 Conjunto de elementos <media> da nova versão da aplicação.

Temos agora de acrescentar os descritores e as regiões para a apresentação dos novos objetos de mídia, como ilustra a Listagem 3.18. Note que as novas regiões para exibição do ícone e do vídeo de propaganda foram definidas como filhas do elemento <region> “screenReg”, ou seja, relativas a essa região previamente definida.

```
<regionBase>
...
      <region id="screenReg" width="100%" height="100%" zIndex="2">
        ...
         <region id="iconReg" left="87.5%" top="11.7%" width="8.45%"
                                    height="6.7%" zIndex="3"/>
         <region id="shoesReg" left="15%" top="60%" width="25%"
                                    height="25%" zIndex="3"/>
      </region>
</regionBase>
<descriptorBase>
...
      <descriptor id="iconDesc" region="iconReg" explicitDur="6s"/>
```

```
      <descriptor id="shoesDesc" region="shoesReg"/>
</descriptorBase>
```

Listagem 3.18 Definição dos novos elementos <region> e dos novos elementos <descriptor>.

Resta agora definir os três novos relacionamentos introduzidos, e incorporar aos relacionamentos "lMusic" e "lEnd" as ações para iniciar e terminar a apresentação da imagem de fundo, como ilustra a Listagem 3.19.

```
<link id="lMusic" xconnector="conEx#onBeginStart_delay">
      <bind role="onBegin" component="animation"/>
      <bind role="start" component="background">
         <bindParam name="delay" value="5s"/>
      </bind>
      <bind role="start" component="choro">
         <bindParam name="delay" value="5s"/>
      </bind>
</link>

<link id="lEnd" xconnector="conEx#onEndStop">
      <bind role="onEnd" component="animation"/>
      <bind role="stop" component="background"/>
      <bind role="stop" component="choro"/>
</link>

<link id="lIcon" xconnector="conEx#onBeginStart">
      <bind role="onBegin" component="animation"
                                          interface="segIcon"/>
      <bind role="start" component="icon"/>
</link>

<link id="lAdvert" xconnector="conEx#onKeySelectionStopSet_varStart">
      <bind role="onSelection" component="icon">
         <bindParam name="keyCode" value="RED"/>
      </bind>
      <bind role="set" component="animation" interface="bounds">
         <bindParam name="var" value="5%,6.67%,45%,45%"/>
      </bind>
      <bind role="start" component="shoes"/>
      <bind role="stop" component="icon"/>
</link>

<link id="lEndAdvert" xconnector="conEx#onEndSet">
```

```
      <bind role="onEnd" component="shoes"/>
      <bind role="set" component="animation" interface="bounds">
         <bindParam name="varSet" value="0,0,222.22%,222.22%"/>
      </bind>
</link>
```

Listagem 3.19 Definição dos novos relacionamentos.

Os dois primeiros elementos <link> (“lMusic” e “lEnd”) determinam o início e o fim da exibição da imagem de fundo como coincidindo com o início e fim da exibição do áudio chorinho, respectivamente.

O terceiro elemento <link> (“lIcon”) define que, ao começar (condição “onBegin”) a apresentação do trecho da animação quando o ícone deve aparecer, ele tenha sua apresentação iniciada (ação “start”).

O quarto elemento <link> (“lAdvert”) é mais complexo. Ele define que, se o ícone da chuteira for selecionado (condição “onSelection”) pela tecla vermelha do controle remoto (passada como parâmetro do papel correspondente, *keycode*=“RED”), o vídeo da animação deve ser redimensionado (ação “set” na propriedade “bounds” definida no elemento “animation”) para os valores passados como parâmetros (left=“5%”, top=“6,67%”, width e height =“45%”, todos relativos aos valores iniciais). O relacionamento também determina que a propaganda da chuteira deve ser iniciada (ação “start” no elemento “shoes”) e que o ícone deve parar sua exibição (ação “stop” no elemento “icon”).

O quinto elemento <link> (“lEndAdvert”) especifica que, quando o vídeo correspondente à propaganda da chuteira terminar (condição “onEnd”), o vídeo da animação deve retomar suas dimensões originais (ação “set” na propriedade “bounds” definida no elemento “animation”).

A Listagem 3.20 apresenta o programa NCL correspondente à nova versão da aplicação.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Exemplo de sincronismo sem interacao do usuario e com interacao
do usuario -->
<ncl id="syncInt" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
   <head>
      <regionBase>
        <region id="backgroundReg" width="100%" height="100%"
                                                    zIndex="1"/>
        <region id="screenReg" width="100%" height="100%"
                                                    zIndex="2">
           <region id="frameReg" left="5%" top="6.7%" width="18.5%"
                                      height="18.5%" zIndex="3"/>
```

```
        <region id="iconReg" left="87.5%" top="11.7%"
                    width="8.45%" height="6.7%" zIndex="3"/>
        <region id="shoesReg" left="15%" top="60%" width="25%"
                                    height="25%" zIndex="3"/>
      </region>
    </regionBase>
    <descriptorBase>
      <descriptor id="backgroundDesc" region="backgroundReg"/>
      <descriptor id="screenDesc" region="screenReg"/>
      <descriptor id="photoDesc" region="frameReg"
                                            explicitDur="5s"/>
      <descriptor id="audioDesc"/>
      <descriptor id="dribleDesc" region="frameReg"/>
      <descriptor id="iconDesc" region="iconReg"
                                            explicitDur="6s"/>
      <descriptor id="shoesDesc" region="shoesReg"/>
    </descriptorBase>
    <connectorBase>
      <importBase documentURI="../causalConnBase.ncl"
                                                    alias="conEx"/>
    </connectorBase>
  </head>

  <body>
    <port id="entry" component="animation"/>
    <media id="background" src="../media/background.png"
                                    descriptor="backgroundDesc"/>
    <media id="animation" src="../media/animGar.mp4"
                                    descriptor="screenDesc">
      <area id="segDrible" begin="12s"/>
      <area id="segPhoto" begin="41s"/>
      <area id="segIcon" begin="45s" end="51s"/>
      <property name="bounds"/>
    </media>
    <media id="choro" src="../media/choro.mp3"
                                    descriptor="audioDesc"/>
    <media id="drible" src="../media/drible.mp4"
                                    descriptor="dribleDesc"/>
    <media id="photo" src="../media/photo.png"
                                    descriptor="photoDesc"/>
    <media id="icon" src="../media/icon.png"
                                    descriptor="iconDesc"/>
    <media id="shoes" src="../media/shoes.mp4"
                                    descriptor="shoesDesc"/>
```

```
    <link id="lMusic" xconnector="conEx#onBeginStart_delay">
      <bind role="onBegin" component="animation"/>
      <bind role="start" component="background">
         <bindParam name="delay" value="5s"/>
      </bind>
      <bind role="start" component="choro">
         <bindParam name="delay" value="5s"/>
      </bind>
    </link>
    <link id="lDrible" xconnector="conEx#onBeginStart">
      <bind role="onBegin" component="animation"
                                          interface="segDrible"/>
      <bind role="start" component="drible"/>
    </link>
    <link id="lPhoto" xconnector="conEx#onBeginStart">
      <bind role="onBegin" component="animation"
                                          interface="segPhoto"/>
      <bind role="start" component="photo"/>
    </link>
    <link id="lEnd" xconnector="conEx#onEndStop">
      <bind role="onEnd" component="animation"/>
      <bind role="stop" component="background"/>
      <bind role="stop" component="choro"/>
    </link>
    <link id="lIcon" xconnector="conEx#onBeginStart">
      <bind role="onBegin" component="animation"
                                          interface="segIcon"/>
      <bind role="start" component="icon"/>
    </link>
    <link id="lAdvert"

    xconnector="conEx#onKeySelectionStopSet_varStart">
      <bind role="onSelection" component="icon">
         <bindParam name="keyCode" value="RED"/>
      </bind>
      <bind role="set" component="animation" interface="bounds">
         <bindParam name="var" value="5%,6.67%,45%,45%"/>
      </bind>
      <bind role="start" component="shoes"/>
      <bind role="stop" component="icon"/>
    </link>
    <link id="lEndAdvert" xconnector="conEx#onEndSet">
      <bind role="onEnd" component="shoes"/>
      <bind role="set" component="animation" interface="bounds">
```

```
            <bindParam name="varSet" value="0,0,222.22%,222.22%"/>
        </bind>
      </link>
   </body>
</ncl>
```

Listagem 3.20 *O Primeiro João* com sincronismo por interação.

A Figura 3.6 ilustra mais dois momentos da exibição, o início do aparecimento do ícone e o momento da apresentação da propaganda, obtidos durante a execução da aplicação na implementação de referência do middleware Ginga-NCL.

Figura 3.6 Cenas da aplicação *O Primeiro João* com interatividade.

3.5 Adicionando o Uso de Contextos

Em NCL, elementos <context> podem ser usados para estruturar uma aplicação. Nesta seção vamos exemplificar o uso de contextos, agrupando todos os elementos da propaganda da chuteira. Isso possibilitaria, além de maior estruturação do programa, o reúso de toda a estrutura. Vamos tratar o reúso na próxima seção.

A Figura 3.7 ilustra a visão estrutural da nova versão da aplicação.

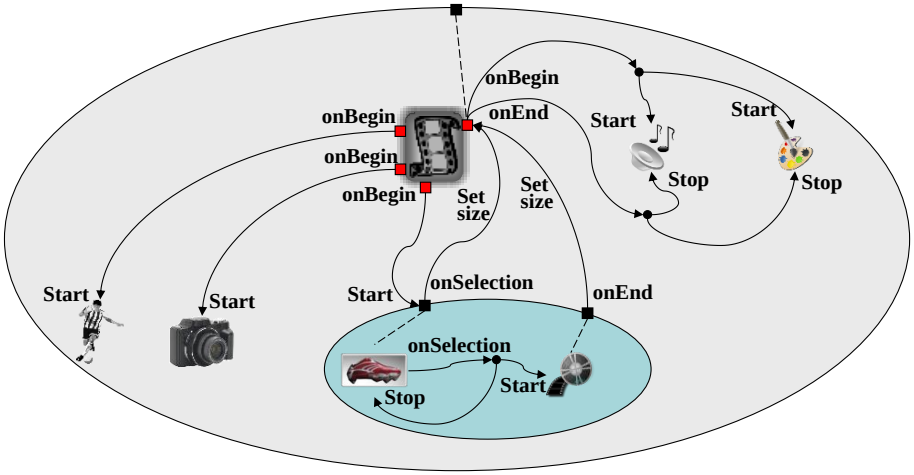


Figura 3.7 Cenas da aplicação *O Primeiro João* usando contextos.

Note, na Figura 3.7, que não é possível acessar diretamente objetos de mídia dentro de um contexto. Como visto no Capítulo 2, o acesso deve ser feito passando pelas portas, que são responsáveis por exportar o que é visível dentro de um contexto por relacionamentos externos. Portanto, nós devemos não só incluir os elementos <media> "icon" e "shoes" dentro de um elemento <context>, como também redefinir os três relacionamentos definidos na seção anterior. Mais ainda, note que um dos relacionamentos pode ser definido dentro do próprio contexto, uma vez que só envolve elementos que são seus filhos. Assim, a definição do contexto fica como ilustrado na Listagem 3.21.

```
<context id="advert">
    <port id="pIcon" component="icon"/>
    <port id="pShoes" component="shoes"/>
    <media id="icon" src="../media/icon.png"
                                    descriptor="iconDesc"/>
    <media id="shoes" src="../media/shoes.mp4"
                                    descriptor="shoesDesc"/>
    <link id="lBeginShoes"
                    xconnector="conEx#onKeySelectionStopStart">
       <bind role="onSelection" component="icon">
          <bindParam name="keyCode" value="RED"/>
       </bind>
       <bind role="start" component="shoes"/>
       <bind role="stop" component="icon"/>
    </link>
</context>
```

Listagem 3.21 Uso do elemento <context> na definição da propaganda da chuteira.

Note que o contexto oferece duas portas para o mundo exterior: uma para seu componente “icon”, que será usada como entrada no comando de início da apresentação do ícone e como saída no comando para o redimensionamento do vídeo da animação; e uma porta para o componente “shoes”, que será usada como saída no comando para restaurar o vídeo da animação em suas dimensões originais.

Os três relacionamentos definidos na seção anterior foram desmembrados em quatro, para que um relacionamento seja definido dentro do contexto. Esse relacionamento, identificado como “lBeginShoes” na Listagem 3.21, especifica que ao ser selecionado (condição “onSelection”) o ícone da chuteira, por meio do botão vermelho do controle remoto, a propaganda deve iniciar (ação “start” sobre o componente “shoes”) e a figura do ícone deve findar sua exibição (ação “stop” sobre o componente “icon”).

Os outros três relacionamentos restantes são especificados como filhos do elemento <body>, substituindo os três relacionamentos definidos na seção anterior, como ilustra a Listagem 3.22, que apresenta o novo programa para essa nova versão. Note os novos relacionamentos “lIcon”, “lAdvert”, “lEndAdvert”.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Exemplo de uso de contexto -->
<ncl id="contextExample"
               xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
   <head>
      <regionBase>
         <region id="backgroundReg" width="100%" height="100%"
                                                    zIndex="1"/>
         <region id="screenReg" width="100%" height="100%"
                                                    zIndex="2">
           <region id="frameReg" left="5%" top="6.7%" width="18.5%"
                                    height="18.5%" zIndex="3"/>
           <region id="iconReg" left="87.5%" top="11.7%"
                        width="8.45%" height="6.7%" zIndex="3"/>
           <region id="shoesReg" left="15%" top="60%" width="25%"
                                    height="25%" zIndex="3"/>
         </region>
      </regionBase>
      <descriptorBase>
         <descriptor id="backgroundDesc" region="backgroundReg"/>
         <descriptor id="screenDesc" region="screenReg"/>
         <descriptor id="photoDesc" region="frameReg"
                                              explicitDur="5s">
           <descriptorParam name="transparency" value="0.6"/>
```

```
        </descriptor>
        <descriptor id="audioDesc"/>
        <descriptor id="dribleDesc" region="frameReg"/>
        <descriptor id="iconDesc" region="iconReg"
                                              explicitDur="6s"/>
        <descriptor id="shoesDesc" region="shoesReg"/>
     </descriptorBase>
     <connectorBase>
        <importBase documentURI="../causalConnBase.ncl"
                                              alias="conEx"/>
     </connectorBase>
  </head>

  <body>
     <port id="entry" component="animation"/>
     <media id="background" src="../media/background.png"
                                    descriptor="backgroundDesc"/>
     <media id="animation" src="../media/animGar.mp4"
                                    descriptor="screenDesc">
        <area id="segDrible" begin="12s"/>
        <area id="segPhoto" begin="41s"/>
        <area id="segIcon" begin="45s" end="51s"/>
        <property name="bounds"/>
     </media>
     <media id="choro" src="../media/choro.mp3"
                                    descriptor="audioDesc"/>
     <media id="drible" src="../media/drible.mp4"
                                    descriptor="dribleDesc"/>
     <media id="photo" src="../media/photo.png"
                                    descriptor="photoDesc"/>
     <context id="advert">
        <port id="pIcon" component="icon"/>
        <port id="pShoes" component="shoes"/>
        <media id="icon" src="../media/icon.png"
                                    descriptor="iconDesc"/>
        <media id="shoes" src="../media/shoes.mp4"
                                    descriptor="shoesDesc"/>
        <link id="lBeginShoes"
                     xconnector="conEx#onKeySelectionStopStart">
          <bind role="onSelection" component="icon">
             <bindParam name="keyCode" value="RED"/>
          </bind>
          <bind role="start" component="shoes"/>
          <bind role="stop" component="icon"/>
```

```
    </link>
  </context>
  <link id="lMusic" xconnector="conEx#onBeginStart_delay">
    <bind role="onBegin" component="animation"/>
    <bind role="start" component="background">
      <bindParam name="delay" value="5s"/>
    </bind>
    <bind role="start" component="choro">
      <bindParam name="delay" value="5s"/>
    </bind>
  </link>
  <link id="lDrible" xconnector="conEx#onBeginStart">
    <bind role="onBegin" component="animation"
                                        interface="segDrible"/>
    <bind role="start" component="drible"/>
  </link>
  <link id="lPhoto" xconnector="conEx#onBeginStart">
    <bind role="onBegin" component="animation"
                                        interface="segPhoto"/>
    <bind role="start" component="photo"/>
  </link>
  <link id="lEnd" xconnector="conEx#onEndStop">
    <bind role="onEnd" component="animation"/>
    <bind role="stop" component="background"/>
    <bind role="stop" component="choro"/>
  </link>
  <link id="lIcon" xconnector="conEx#onBeginStart">
    <bind role="onBegin" component="animation"
                                        interface="segIcon"/>
    <bind role="start" component="advert" interface="pIcon"/>
  </link>
  <link id="lAdvert" xconnector="conEx#onKeySelectionSet_var">
    <bind role="onSelection" component="advert"
                                        interface="pIcon">
      <bindParam name="keyCode" value="RED"/>
    </bind>
    <bind role="set" component="animation" interface="bounds">
      <bindParam name="var" value="5%,6.67%,45%,45%"/>
    </bind>
  </link>
  <link id="lEndAdvert" xconnector="conEx#onEndSet_var">
    <bind role="onEnd" component="advert" interface="pShoes"/>
    <bind role="set" component="animation" interface="bounds">
      <bindParam name="var" value="0,0,222.22%,222.22%"/>
```

```
        </bind>
      </link>
   </body>
</ncl>
```

Listagem 3.22 *O Primeiro João* com uso de contextos.

3.6 Adicionando Reúso de Objetos de Mídia

O reúso de objetos de mídia e de contexto torna uma aplicação mais limpa, mais fácil de manter e menos sujeita a erros. Mesmo correndo o risco de tornar o contexto definido na seção anterior menos passível de reúso em outras partes do vídeo da animação (ou mesmo em outro vídeo), vamos utilizar esse contexto para introduzir o reúso de objetos de mídia, com o intuito apenas de tornar mais simples a definição de relacionamentos.

O que faremos é criar um novo objeto de mídia dentro do elemento <context> "advert", que herdará toda a definição do elemento <media> "animation" e, mais ainda, representará o mesmo objeto de mídia: o vídeo da animação. Como discutido na Seção 2, o modelo NCL permite que um mesmo nó esteja contido em mais de um contexto, e isso é feito através do reúso.

A Figura 3.8 ilustra o reúso do elemento <media> "animation". Note a simplificação da definição dos elos ligados à propaganda da chuteira, uma vez que agora eles são internos ao contexto e não precisam mais utilizar portas para o acesso ao vídeo da animação.

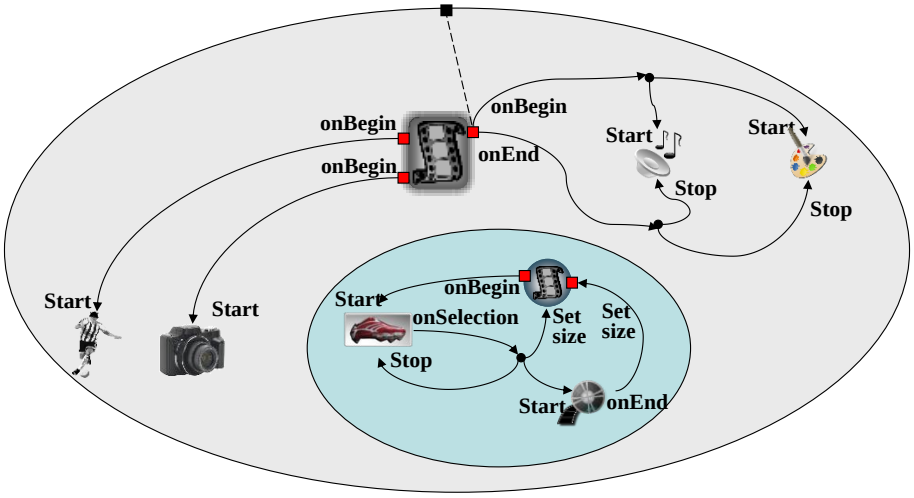


Figura 3.8 Reúso.

O novo elemento <media> definido dentro do contexto tem a definição dada pela Listagem 3.23. Note que o uso do atributo *refer* indica que o elemento herdará toda a definição do elemento referido, no caso o "animation". O elemento referenciado e o elemento que o referencia também devem ser considerados o mesmo nó em relação à apresentação se o atributo *instance* receber um valor "instSame", como no caso presente.

```
<media id="reusedAnimation" refer="animation" instance="instSame">
       <property name="bounds"/>
</media>
```

Listagem 3.23 Reúso de objeto de mídia.

Note também que, no novo elemento, novas interfaces (elementos <area> e <property>) podem ser criadas. No caso, para exemplificar, retiraremos a definição da propriedade "bounds" do elemento "animation" e a reincluiremos no reuso do elemento, como indicado na Listagem 3.23.

Nessa nova versão, os quatro relacionamentos introduzidos na seção anterior devem ser modificados e redefinidos como pertencendo ao contexto. Os quatro relacionamentos voltam a ser apenas três, e agora devem referenciar o elemento <media> "reusedAnimation", como ilustra a Listagem 3.24, contendo todo o novo programa NCL.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Exemplo de Reuso -->
<ncl id="reuse" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
   <head>
      <regionBase>
         <region id="backgroundReg" width="100%" height="100%"
                                                    zIndex="1"/>
         <region id="screenReg" width="100%" height="100%"
                                                    zIndex="2">
            <region id="frameReg" left="5%" top="6.7%" width="18.5%"
                                    height="18.5%" zIndex="3"/>
            <region id="iconReg" left="87.5%" top="11.7%"
                      width="8.45%" height="6.7%" zIndex="3"/>
            <region id="shoesReg" left="15%" top="60%" width="25%"
                                    height="25%" zIndex="3"/>
         </region>
      </regionBase>
      <descriptorBase>
         <descriptor id="backgroundDesc" region="backgroundReg"/>
         <descriptor id="screenDesc" region="screenReg"/>
         <descriptor id="photoDesc" region="frameReg"
                                             explicitDur="5s"/>
```

```
        <descriptor id="audioDesc"/>
        <descriptor id="dribleDesc" region="frameReg"/>
        <descriptor id="iconDesc" region="iconReg"
                                              explicitDur="6s"/>
        <descriptor id="shoesDesc" region="shoesReg"/>
     </descriptorBase>
     <connectorBase>
      <importBase documentURI="../causalConnBase.ncl" alias="conEx"/>
     </connectorBase>
  </head>
  <body>
     <port id="entry" component="animation"/>
     <media id="background" src="../media/background.png"
                                    descriptor="backgroundDesc"/>
     <media id="animation" src="../media/animGar.mp4"
                                    descriptor="screenDesc">
        <area id="segDrible" begin="12s"/>
        <area id="segPhoto" begin="41s"/>
        <area id="segIcon" begin="45s" end="51s"/>
     </media>
     <media id="choro" src="../media/choro.mp3"
                                    descriptor="audioDesc"/>
     <media id="drible" src="../media/drible.mp4"
                                    descriptor="dribleDesc"/>
     <media id="photo" src="../media/photo.png"
                                    descriptor="photoDesc"/>
     <context id="advert">
        <media id="reusedAnimation" refer="animation"
                                              instance="instSame">
           <property name="bounds"/>
        </media>
        <media id="icon" src="../media/icon.png"
                                    descriptor="iconDesc"/>
        <media id="shoes" src="../media/shoes.mp4"
                                    descriptor="shoesDesc"/>
        <link id="lIcon" xconnector="conEx#onBeginStart">
           <bind role="onBegin" component="reusedAnimation"
                                              interface="segIcon"/>
           <bind role="start" component="icon"/>
        </link>
        <link id="lBeginShoes"
              xconnector="conEx#onKeySelectionStopSet_varStart">
           <bind role="onSelection" component="icon">
            <bindParam name="keyCode" value="RED"/>
```

```
            </bind>
            <bind role="start" component="shoes"/>
            <bind role="set" component="reusedAnimation"
                                              interface="bounds">
              <bindParam name="var" value="5%,6.67%,45%,45%"/>
            </bind>
            <bind role="stop" component="icon"/>
         </link>
         <link id="lEndShoes" xconnector="conEx#onEndSet_var">
            <bind role="onEnd" component="shoes"/>
            <bind role="set" component="reusedAnimation"
                                              interface="bounds">
              <bindParam name="var" value="0,0,222.22%,222.22%"/>
            </bind>
         </link>
      </context>
      <link id="lMusic" xconnector="conEx#onBeginStart_delay">
         <bind role="onBegin" component="animation"/>
         <bind role="start" component="background">
            <bindParam name="delay" value="5s"/>
         </bind>
         <bind role="start" component="choro">
            <bindParam name="delay" value="5s"/>
         </bind>
      </link>
      <link id="lDrible" xconnector="conEx#onBeginStart">
         <bind role="onBegin" component="animation"
                                       interface="segDrible"/>
         <bind role="start" component="drible"/>
      </link>
      <link id="lPhoto" xconnector="conEx#onBeginStart">
         <bind role="onBegin" component="animation"
                                       interface="segPhoto"/>
         <bind role="start" component="photo"/>
      </link>
      <link id="lEnd" xconnector="conEx#onEndStop">
         <bind role="onEnd" component="animation"/>
         <bind role="stop" component="choro"/>
         <bind role="stop" component="background"/>
      </link>
   </body>
</ncl>
```

Listagem 3.24 *O Primeiro João* com reúso.

3.7 Usando o Canal de Interatividade

Vamos agora acrescentar em nosso exemplo o uso do canal de interatividade. Nessa nova versão do programa NCL, caso o ícone da chuteira seja selecionado, vamos não só apresentar a propaganda da chuteira, mas também um formulário HTML. O formulário, se devidamente preenchido e enviado à loja da propaganda por meio do canal de interatividade, trará como resposta a confirmação da compra. A loja, tendo recebido o formulário, pode depois providenciar a entrega do material adquirido.

A Figura 3.9 ilustra a visão estrutural da nova versão da aplicação. Note que as únicas modificações foram a introdução de um objeto de mídia representando o formulário no contexto da propaganda, o acréscimo de mais um ator no relacionamento disparado pela seleção do ícone da chuteira, que permitirá a exibição do formulário, e o fato de que agora não é o final da exibição da propaganda da chuteira que volta o vídeo da animação ao seu tamanho original, mas sim o final do preenchimento do formulário ou o final do tempo máximo para seu preenchimento.

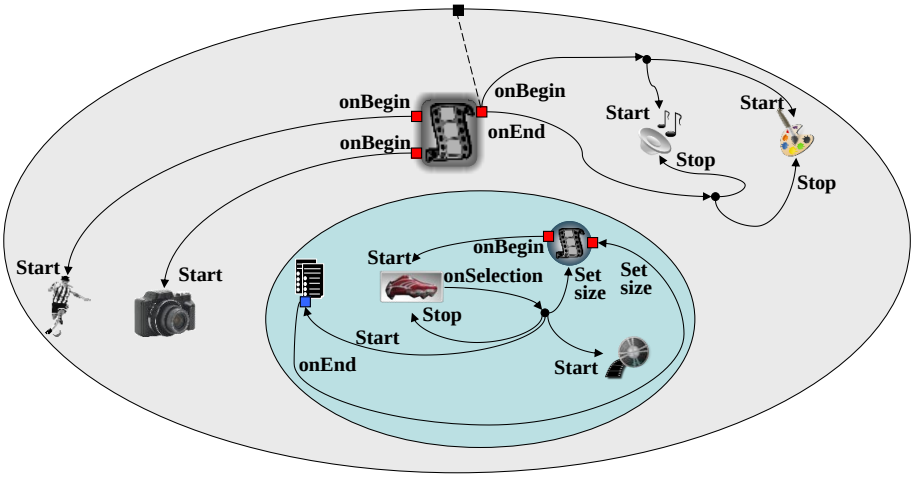


Figura 3.9 Visão estrutural da versão com o uso do canal de interatividade.

Ao termos de introduzir um novo elemento de <media>, temos também de introduzir o elemento <region> especificando onde ele será apresentado, e o elemento <descriptor>, fazendo a ligação do elemento <media> com o elemento <region>. A Listagem 3.25 ilustra os novos elementos inseridos. Note, mais uma vez, que a região definida para a exibição do formulário é filha da região definida para a exibição do vídeo da animação, ou seja, o posicionamento da primeira é realizado relativo à segunda. Note também que no descritor foi especificado um tempo máximo para o preenchimento do formulário, no caso igual a 45 segundos.

```
<head>
   <regionBase>
     ...
      <region id="screenReg" width="100%" height="100%" zIndex="2">
         ...
         <region id="formReg" left="56.25%" top="8.33%"
                       width="38.75%" height="71.7%" zIndex="3"/>
      </region>
   </regionBase>
   <descriptorBase>
      ...
      <descriptor id="formDesc" region="formReg" focusIndex="1"
                                                 explicitDur="45s"/>
   </descriptorBase>
   ...
</head>

<body>
     ...
   <context id="advert">
      ...
      <media id="ptForm" src="../media/ptForm.htm"
                                       descriptor="formDesc"/>
      ...
   </context>
   ...
</body>
```

Listagem 3.25 Novos elementos para a apresentação do formulário.

Ainda na Listagem 3.25, note que no elemento <descriptor> “formDesc” um novo atributo foi definido: *focusIndex*. Esse atributo define o elemento em foco para navegação por setas do controle remoto. No caso só há um elemento, mas quando incrementarmos mais nosso exemplo, na Seção 3.12, outros elementos focáveis serão definidos. Lá explicaremos melhor esse atributo. Um elemento em foco, caso seja pressionada a tecla “ENTER”, passa a receber (controlar) toda a navegação pelo controle remoto até que a tecla “BACK” seja pressionada, retornando então o controle ao formatador NCL. No caso em questão, após ser pressionada a tecla “ENTER”, o formulário HTML pode ser preenchido.

Devemos agora alterar alguns relacionamentos da versão anterior para espelhar a nova versão. O relacionamento “lBeginShoes” deve acrescentar um papel ação para dar início à apresentação do formulário. O relacionamento

“lEndShoes” deve ser substituído pelo relacionamento “lEndForm”, uma vez que agora é o término da apresentação do formulário que volta o vídeo da animação para seus valores de posicionamento iniciais, e não mais o término do vídeo de propaganda da chuteira. Os relacionamentos modificados são apresentados na Listagem 3.26.

```
<link id="lBeginShoes"
      xconnector="conEx#onKeySelectionStopSet_varStart">
        <bind role="onSelection" component="icon">
           <bindParam name="keyCode" value="RED"/>
        </bind>
        <bind role="start" component="shoes"/>
        <bind role="start" component="ptForm"/>
        <bind role="set" component="reusedAnimation"
                                              interface="bounds">
           <bindParam name="var" value="5%,6.67%,45%,45%"/>
        </bind>
        <bind role="stop" component="icon"/>
</link>
<link id="lEndForm" xconnector="conEx#onEndSet_var">
        <bind role="onEnd" component="ptForm"/>
        <bind role="set" component="reusedAnimation"
                                              interface="bounds">
           <bindParam name="var" value="0,0,222.22%,222.22%"/>
        </bind>
</link>
```

Listagem 3.26 Relacionamentos modificados para a nova versão.

A especificação completa da nova versão com uso do canal de interatividade é ilustrada na Listagem 3.27.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Exemplo de uso do canal de interatividade -->
<ncl id="return" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
   <head>
      <regionBase>
         <region id="backgroundReg" width="100%" height="100%"
                                                   zIndex="1"/>
         <region id="screenReg" width="100%" height="100%"
                                                   zIndex="2">
           <region id="frameReg" left="5%" top="6.7%" width="18.5%"
                                      height="18.5%" zIndex="3"/>
           <region id="iconReg" left="87.5%" top="11.7%"
```

```
                    width="8.45%" height="6.7%" zIndex="3"/>
         <region id="shoesReg" left="15%" top="60%" width="25%"
                                  height="25%" zIndex="3"/>
         <region id="formReg" left="56.25%" top="8.33%"
                    width="38.75%" height="71.7%" zIndex="3"/>
      </region>
   </regionBase>
   <descriptorBase>
      <descriptor id="backgroundDesc" region="backgroundReg"/>
      <descriptor id="screenDesc" region="screenReg"/>
      <descriptor id="photoDesc" region="frameReg"
                                          explicitDur="5s"/>
      <descriptor id="audioDesc"/>
      <descriptor id="dribleDesc" region="frameReg"/>
      <descriptor id="iconDesc" region="iconReg"
                                          explicitDur="6s"/>
      <descriptor id="shoesDesc" region="shoesReg"/>
      <descriptor id="formDesc" region="formReg" focusIndex="1"
                                          explicitDur="15s"/>
   </descriptorBase>
   <connectorBase>
      <importBase documentURI="../causalConnBase.ncl"
                                                  alias="conEx"/>
   </connectorBase>
</head>
<body>
   <port id="entry" component="animation"/>
   <media id="background" src="../media/background.png"
                           descriptor="backgroundDesc"/>
   <media id="animation" src="../media/animGar.mp4"
                           descriptor="screenDesc">
      <area id="segDrible" begin="12s"/>
      <area id="segPhoto" begin="41s"/>
      <area id="segIcon" begin="45s" end="51s"/>
   </media>
   <media id="choro" src="../media/choro.mp3"
                           descriptor="audioDesc"/>
   <media id="drible" src="../media/drible.mp4"
                           descriptor="dribleDesc"/>
   <media id="photo" src="../media/photo.png"
                           descriptor="photoDesc"/>
   <context id="advert">
      <media id="reusedAnimation" refer="animation"
                                    instance="instSame">
```

```
        <property name="bounds"/>
      </media>
      <media id="icon" src="../media/icon.png"
                        descriptor="iconDesc"/>
      <media id="shoes" src="../media/shoes.mp4"
                        descriptor="shoesDesc"/>
      <media id="ptForm" src="../media/ptForm.htm"
                        descriptor="formDesc"/>
      <link id="lIcon" xconnector="conEx#onBeginStart">
        <bind role="onBegin" component="reusedAnimation"
                                  interface="segIcon"/>
        <bind role="start" component="icon"/>
      </link>
      <link id="lBeginShoes"
            xconnector="conEx#onKeySelectionStopSet_varStart">
        <bind role="onSelection" component="icon">
            <bindParam name="keyCode" value="RED"/>
        </bind>
        <bind role="start" component="shoes"/>
        <bind role="start" component="ptForm"/>
        <bind role="set" component="reusedAnimation"
                                        interface="bounds">
            <bindParam name="var" value="5%,6.67%,45%,45%"/>
        </bind>
        <bind role="stop" component="icon"/>
      </link>
      <link id="lEndForm" xconnector="conEx#onEndSet_var">
        <bind role="onEnd" component="ptForm"/>
        <bind role="set" component="reusedAnimation"
                                        interface="bounds">
            <bindParam name="var" value="0,0,222.22%,222.22%"/>
        </bind>
      </link>
    </context>
    <link id="lMusic" xconnector="conEx#onBeginStart_delay">
      <bind role="onBegin" component="animation"/>
      <bind role="start" component="background">
        <bindParam name="delay" value="5s"/>
      </bind>
      <bind role="start" component="choro">
        <bindParam name="delay" value="5s"/>
      </bind>
    </link>
    <link id="lDrible" xconnector="conEx#onBeginStart">
```

```
         <bind role="onBegin" component="animation"
                                      interface="segDrible"/>
         <bind role="start" component="drible"/>
      </link>
      <link id="lPhoto" xconnector="conEx#onBeginStart">
         <bind role="onBegin" component="animation"
                                      interface="segPhoto"/>
         <bind role="start" component="photo"/>
      </link>
      <link id="lEnd" xconnector="conEx#onEndStop">
         <bind role="onEnd" component="animation"/>
         <bind role="stop" component="background"/>
         <bind role="stop" component="choro"/>
      </link>
   </body>
</ncl>
```

Listagem 3.27 *O Primeiro João* com uso do canal de interatividade.

A Figura 3.10 ilustra o momento da apresentação da propaganda da chuteira, obtido durante a execução da aplicação na implementação de referência do middleware Ginga-NCL.

Figura 3.10 Cenas da aplicação *O Primeiro João* com uso do canal de interatividade.

3.8 Uso de Múltiplos Dispositivos de Exibição

Podemos agora alterar a versão anterior da aplicação para trabalhar com múltiplos dispositivos de exibição.

Vamos, nessa nova versão, exibir toda a propaganda interativa, ou seja, o vídeo da propaganda e o formulário de compra, em outro dispositivo diferente do usado para a apresentação das demais mídias.

Quando fazemos uso de mais de um dispositivo de exibição, devemos para cada base de regiões (elemento <regionBase>) especificar a que dispositivo as regiões definidas se referem. Isso é feito no atributo *device* do elemento <regionBase>, como ilustra a Listagem 3.28; quando não declarado, assume um valor default.

```
<regionBase>
        <region id="screenReg" width="100%" height="100%"
                                                    zIndex="1">
                <region id="frameReg" left="5%" top="6.7%"
                        width="18.5%" height="18.5%" zIndex="2"/>
                <region id="iconReg" left="87.5%" top="11.7%"
                        width="8.45%" height="6.7%" zIndex="2"/>
        </region>
</regionBase>

<regionBase device="systemScreen(1)">
        <region id="backgroundReg" width="100%" height="100%"
                                                    zIndex="1">
                <region id="shoesReg" left="0" top="25%" width="50%"
                                    height="50%" zIndex="2"/>
                <region id="formReg" left="50%" top="0" width="50%"
                                    height="100%" zIndex="2"/>
        </region>
</regionBase>
```

Listagem 3.28 Uso de múltiplos dispositivos de exibição.

Na Listagem 3.28, o vídeo da animação, o vídeo do drible, a foto do marcador caído e o ícone da chuteira aparecerão na tela do dispositivo-base da apresentação, declarado por default. No caso do Sistema Brasileiro de TV Digital Terrestre, por default o dispositivo-base é a tela da TV. Já a figura de fundo, a propaganda da chuteira e o formulário a preencher aparecerão em um segundo dispositivo de exibição; por exemplo, a tela de um celular usado para interação.

O programa NCL deve agora ser modificado porque, como não haverá mais superposição de informações, não é necessário redimensionar o vídeo da animação. Fica por conta do leitor fazer essas modificações, visto que não prosseguiremos mais com esse exemplo até o Capítulo 15 A Figura 3.11 ilustra o momento da apresentação da propaganda, obtido durante a execução da aplicação na implementação de referência do middleware Ginga-NCL.

Figura 3.11 Cenas da aplicação *O Primeiro João* com o uso de múltiplos dispositivos.

3.9 Adaptação de Conteúdo

Vamos agora voltar à versão da Seção 3.7 e nela acrescentar a facilidade de adaptação de conteúdo. Para tanto, vamos exibir o formulário em diferentes línguas, dependendo do perfil do telespectador. Se o usuário falar a língua inglesa, vamos apresentar o formulário em inglês; caso contrário, apresentaremos o formulário em português.

A visão estrutural para essa versão do programa NCL é ilustrada na Figura 3.12.

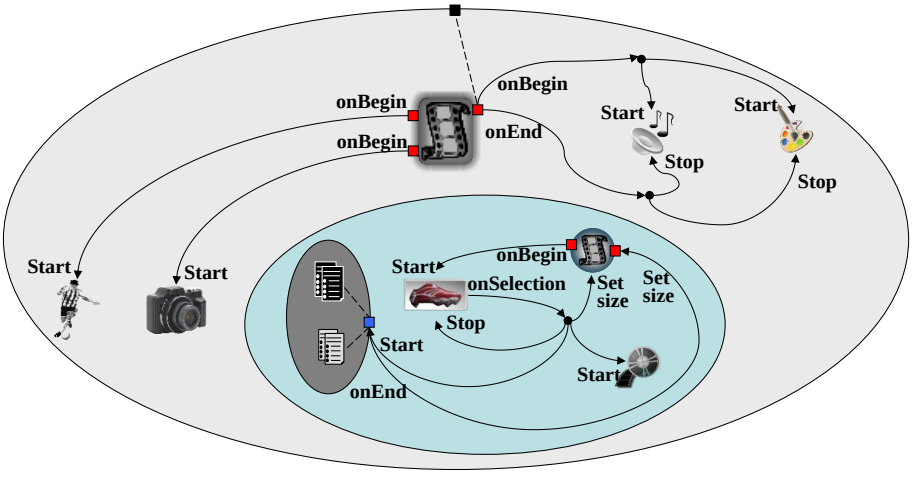


Figura 3.12 Visão estrutural da versão com adaptação de conteúdo.

O leitor deve notar, comparando a Figura 3.9 e a Figura 3.12, que a única diferença ocorre no objeto de mídia que representa o formulário. Na Figura 3.12, esse objeto é substituído por um elemento <switch>, representado em cinza dentro do contexto da propaganda. O elemento <switch>, por sua vez, contém tanto o objeto de mídia representando o formulário em português quanto o objeto de mídia representando o formulário em inglês. Todos os relacionamentos que tinham como ator o objeto de mídia formulário ("ptForm"), na versão da Seção 3.7, devem agora se referenciar ao elemento <switch> na nova versão do programa NCL.

O elemento <switch> escolhe um dos seus componentes filhos para apresentar, baseado em um conjunto de regras. As regras devem ser especificadas como filhas do elemento <ruleBase>, que por sua vez é filho do elemento <head>, conforme ilustrado para o exemplo em questão na Listagem 3.29.

```
<head>
   <ruleBase>
        <rule id="en" var="system.language" value="en"
                                                comparator="eq"/>
   </ruleBase>
   ...
</head>
```

Listagem 3.29 Elementos <ruleBase> e <rule>.

Um elemento <rule> tem um identificador, um atributo especificando uma variável a ser testada (*var*), um atributo especificando o valor sobre o qual a variável deve ser testada (*value*) e um atributo especificando o operador de comparação. No caso da Listagem 3.29, a regra "en" é especificada para testar se a variável "system.language" tem seu valor igual a "en". A variável "system.language" é uma variável controlada pelo sistema receptor, que atribui um valor a ela dependendo do idioma do usuário telespectador. Várias são as variáveis controladas pelo sistema. O Capítulo 9 discute o uso e a manutenção dessas variáveis.

A adaptação do conteúdo é especificada pelo elemento <switch>, conforme ilustrado na Listagem 3.30 para a nova versão de *O Primeiro João*.

```
<switch id="form">
       <switchPort id="spForm">
              <mapping component="enForm"/>
              <mapping component="ptForm"/>
       </switchPort>
       <bindRule constituent="enForm" rule="en"/>
       <defaultComponent component="ptForm"/>
```

```
        <media id="ptForm" src="../media/ptForm.htm"
                                        descriptor="formDesc"/>
        <media id="enForm" src="../media/enForm.htm"
                                        descriptor="formDesc"/>
</switch>
```

Listagem 3.30 Elementos <switch> e seus elementos filhos.

Um relacionamento não pode referenciar diretamente um elemento filho do <switch>, mas pode fazê-lo indiretamente por meio de portas do switch. Um elemento <switchPort> define uma porta que pode ser usada para referenciar qualquer interface de um elemento filho do <switch> para o qual um mapeamento (elemento <mapping>) com a porta foi estabelecido. Em outras palavras, quando o relacionamento é estabelecido com uma <switchPort>, a interface do elemento selecionado pelo switch e que tem mapeamento com a porta é que participa como ator do relacionamento.

O elemento <bindRule> determina qual elemento filho do <switch> é escolhido, dependendo da regra definida pelo elemento <rule>, por nós já discutida. Por exemplo, na Listagem 3.30, se a regra "en" for atendida (lembre-se de que essa regra especificava que o idioma do usuário telespectador era o inglês), o elemento <media> "enForm" deverá ser escolhido.

As regras são avaliadas em sequência, e a primeira regra que atende às exigências tem seu componente selecionado. Caso nenhuma regra seja satisfeita, o componente definido no elemento <defaultComponent> deve ser escolhido. Um elemento <switch> pode ter como elementos filhos selecionáveis elementos <media> ou <context>.

Como mencionamos anteriormente, essa nova versão do programa NCL deve modificar seus relacionamentos para referenciarem o elemento <switch>, definido na Listagem 3.30, e não mais diretamente o elemento <media> "ptForm", conforme ilustra a Listagem 3.31. Note a referência ao elemento <switch> "form" e à sua porta (<switchPort>) "spForm".

```
<link id="lBeginShoes"
               xconnector="conEx#onKeySelectionStopSet_varStart">
        <bind role="onSelection" component="icon">
               <bindParam name="keyCode" value="RED"/>
        </bind>
        <bind role="start" component="shoes"/>
        <bind role="start" component="form" interface="spForm"/>
        <bind role="set" component="reusedAnimation"
                                              interface="bounds">
               <bindParam name="var" value="5%,6.67%,45%,45%"/>
```

```
        </bind>
        <bind role="stop" component="icon"/>
</link>

<link id="lEndForm" xconnector="conEx#onEndSet_var">
        <bind role="onEnd" component="form" interface="spForm"/>
        <bind role="set" component="reusedAnimation"
                                              interface="bounds">
                <bindParam name="var" value="0,0,222.22%,222.22%"/>
        </bind>
</link>
```

Listagem 3.31 Redefinição dos relacionamentos para referência ao elemento <switch>.

A especificação completa da nova versão do programa NCL é ilustrada na Listagem 3.32.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Exemplo de switch e regras -->
<ncl id="switch" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
   <head>
      <ruleBase>
         <rule id="en" var="system.language" value="en"
                                              comparator="eq"/>
         <rule id="int" var="service.interactivity" value="true"
                                              comparator="eq"/>
        </ruleBase>
      <regionBase>
         <region id="backgroundReg" width="100%" height="100%"
                                                 zIndex="1"/>
         <region id="screenReg" width="100%" height="100%"
                                                 zIndex="2">
            <region id="frameReg" left="5%" top="6.7%" width="18.5%"
                                    height="18.5%" zIndex="3"/>
            <region id="iconReg" left="87.5%" top="11.7%"
                      width="8.45%" height="6.7%" zIndex="3"/>
            <region id="shoesReg" left="15%" top="60%" width="25%"
                                    height="25%" zIndex="3"/>
            <region id="formReg" left="56.25%" top="8.33%"
                      width="38.75%" height="71.7%" zIndex="3"/>
         </region>
      </regionBase>
      <descriptorBase>
         <descriptor id="backgroundDesc" region="backgroundReg"/>
```

```
        <descriptor id="screenDesc" region="screenReg"/>
        <descriptor id="photoDesc" region="frameReg"
                                              explicitDur="5s"/>
        <descriptor id="audioDesc"/>
        <descriptor id="dribleDesc" region="frameReg"/>
        <descriptor id="iconDesc" region="iconReg"
                                              explicitDur="6s"/>
        <descriptor id="shoesDesc" region="shoesReg"/>
        <descriptor id="formDesc" region="formReg" focusIndex="1"
                                              explicitDur="15s"/>
    </descriptorBase>
    <connectorBase>
        <importBase documentURI="../causalConnBase.ncl"
                                                    alias="conEx"/>
    </connectorBase>
</head>

<body>
    <port id="entry" component="animation"/>
    <media id="background" src="../media/background.png"
                                    descriptor="backgroundDesc"/>
    <media id="animation" src="../media/animGar.mp4"
                                    descriptor="screenDesc">
        <area id="segDrible" begin="12s"/>
        <area id="segPhoto" begin="41s"/>
        <area id="segIcon" begin="45s" end="51s"/>
    </media>
    <media id="choro" src="../media/choro.mp3"
                                    descriptor="audioDesc"/>
    <media id="drible" src="../media/drible.mp4"
                                    descriptor="dribleDesc"/>
    <media id="photo" src="../media/photo.png"
                                    descriptor="photoDesc"/>
    <context id="advert">
        <media id="reusedAnimation" refer="animation"
                                              instance="instSame">
            <property name="bounds"/>
        </media>
        <media id="icon" src="../media/icon.png"
                                    descriptor="iconDesc"/>
        <media id="shoes" src="../media/shoes.mp4"
                                    descriptor="shoesDesc"/>
        <switch id="form">
            <switchPort id="spForm">
```

```
            <mapping component="enForm"/>
            <mapping component="ptForm"/>
         </switchPort>
         <bindRule constituent="enForm" rule="en"/>
         <defaultComponent component="ptForm"/>
         <media id="ptForm" src="../media/ptForm.htm"
                                  descriptor="formDesc"/>
         <media id="enForm" src="../media/enForm.htm"
                                  descriptor="formDesc"/>
      </switch>
      <link id="lIcon" xconnector="conEx#onBeginStart">
         <bind role="onBegin" component="reusedAnimation"
                                        interface="segIcon"/>
         <bind role="start" component="icon"/>
      </link>
      <link id="lBeginShoes"
            xconnector="conEx#onKeySelectionStopSet_varStart">
         <bind role="onSelection" component="icon">
          <bindParam name="keyCode" value="RED"/>
         </bind>
         <bind role="start" component="shoes"/>
         <bind role="start" component="form" interface="spForm"/>
         <bind role="set" component="reusedAnimation"
                                        interface="bounds">
          <bindParam name="var" value="5%,6.67%,45%,45%"/>
         </bind>
         <bind role="stop" component="icon"/>
      </link>
      <link id="lEndForm" xconnector="conEx#onEndSet_var">
         <bind role="onEnd" component="form" interface="spForm"/>
         <bind role="set" component="reusedAnimation"
                                        interface="bounds">
          <bindParam name="var" value="0,0,222.22%,222.22%"/>
         </bind>
      </link>
   </context>
   <link id="lMusic" xconnector="conEx#onBeginStart_delay">
      <bind role="onBegin" component="animation"/>
      <bind role="start" component="background">
         <bindParam name="delay" value="5s"/>
      </bind>
      <bind role="start" component="choro">
         <bindParam name="delay" value="5s"/>
      </bind>
```

```
      </link>
      <link id="lDrible" xconnector="conEx#onBeginStart">
         <bind role="onBegin" component="animation"
                                    interface="segDrible"/>
         <bind role="start" component="drible"/>
      </link>
      <link id="lPhoto" xconnector="conEx#onBeginStart">
         <bind role="onBegin" component="animation"
                                    interface="segPhoto"/>
         <bind role="start" component="photo"/>
      </link>
      <link id="lEnd" xconnector="conEx#onEndStop">
         <bind role="onEnd" component="animation"/>
         <bind role="stop" component="background"/>
         <bind role="stop" component="choro"/>
      </link>
   </body>
</ncl>
```

Listagem 3.32 *O Primeiro João* com adaptação de conteúdo.

3.10 O Uso do Nó Settings

Vamos agora permitir em nossa aplicação *O Primeiro João* que todas as propagandas interativas (e só elas) sejam habilitadas ou inibidas pelo telespectador. Se inibidas, por exemplo, o ícone da chuteira nem aparecerá, impossibilitando o acionamento da propaganda correspondente.

Para fazer o controle da interatividade, vamos definir uma variável global do programa (serviço) denominada "service.interactivity". Conforme o valor dessa variável, as propagandas interativas serão inibidas (service.interactivity = "false") ou habilitadas (service.interactivity = "true"). Na verdade, o valor da variável será usado (testado nos relacionamentos) para permitir ou não o aparecimento dos ícones que permitem a propaganda interativa (no caso da nossa aplicação, o ícone da chuteira).

Em uma aplicação NCL, variáveis globais são tratadas como propriedades do nó *settings*. Em NCL, esse nó é representado pelo elemento <media>, com atributo *type* igual a "application/x-ncl-settings". Propriedades cujos valores podem ser manipulados por uma aplicação NCL devem ser declaradas no elemento <property> filho do elemento <media> representando o nó settings (exceto quando a manipulação for pelas regras, como discutiremos no Capítulo 11). Assim, para nossa nova versão da aplicação, o

nó settings deve ser declarado e seu valor iniciado, como ilustra a Listagem 3.33.

```
<media id="globalVar" type="application/x-ncl-settings">
        <property name="service.interactivity" value="true"/>
</media>
```

Listagem 3.33 O elemento <media> do tipo "application/x-ncl-settings".

Para o controle das propagandas interativas, vamos definir um contexto de interatividade, pois assim será possível seu reúso em outras aplicações, o que tornará também nosso programa mais bem estruturado. O elemento <context> "interactivity" conterá o nó settings, um ícone para avisar o usuário telespectador que a interatividade está ativa e outro para alertar que ela está inibida. Conforme o usuário selecione um ícone, ele é substituído pelo outro, permitindo assim ao usuário habilitar e desabilitar a interatividade. Cada vez que o ícone é trocado, a variável "service.interactivity" muda de valor.

Como, no início da aplicação, o ícone informando que a interatividade está habilitada (iniciação do procedimento) deve ser exibido e a variável colocada em "true", vamos, para facilitar a estruturação, colocar um elemento <media> no contexto de interatividade representando a mesma instância de apresentação do filme da animação, que dará partida ao procedimento de iniciação. A colocação desse elemento de fato dificulta o reúso do contexto em outra aplicação, mas vamos usar o procedimento mesmo assim para mais uma vez exemplificar o mecanismo de reúso.

A Figura 3.13 ilustra a visão estrutural do contexto de interatividade.

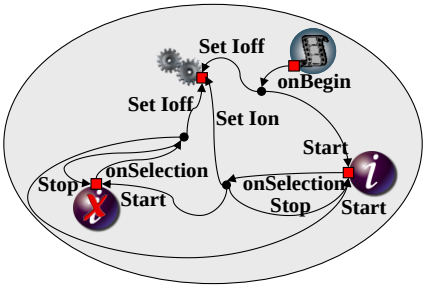


Figura 3.13 Visão estrutural do contexto de interatividade.

O leitor deve notar a existência de três relacionamentos. O primeiro inicia a variável "service.interactivity" e a exibição do ícone "intOn". O segundo, quando da seleção do ícone "intOn" pela tecla INFO, para sua exibição, inicia a apresentação do ícone "intOff" e troca o valor de service.interactivity". O terceiro é parecido com o segundo, quando da seleção do ícone "intOff" pela tecla INFO, para sua exibição, inicia a apresentação

do ícone “intOn” e troca o valor de service.interactivity”. A Listagem 3.34 apresenta o código NCL correspondente.

```
<context id="interactivity">
   <media id="globalVar" type="application/x-ncl-settings">
      <property name="service.interactivity" value="true"/>
   </media>
   <media id="anotherAnimation" refer="animation"
                                              instance="instSame"/>
   <media id="intOn" src="../media/intOn.png" descriptor="intDesc"/>
   <media id="intOff" src="../media/intOff.png" descriptor="intDesc"/>
   <link id="lInt" xconnector="conEx#onBeginSet_varStart">
      <bind role="onBegin" component="anotherAnimation"/>
      <bind role="start" component="intOn"/>
      <bind role="set" component="globalVar"
                              interface="service.interactivity">
         <bindParam name="var" value="true"/>
      </bind>
   </link>
   <link id="lOn" xconnector="conEx#onKeySelectionStopSet_varStart">
      <bind role="onSelection" component="intOn">
         <bindParam name="keyCode" value="INFO"/>
      </bind>
      <bind role="start" component="intOff"/>
      <bind role="stop" component="intOn"/>
      <bind role="set" component="globalVar"
                              interface="service.interactivity">
         <bindParam name="var" value="false"/>
      </bind>
   </link>
   <link id="lOff" xconnector="conEx#onKeySelectionStopSet_varStart">
      <bind role="onSelection" component="intOff">
         <bindParam name="keyCode" value="INFO"/>
      </bind>
      <bind role="start" component="intOn"/>
      <bind role="stop" component="intOff"/>
      <bind role="set" component="globalVar"
                              interface="service.interactivity">
         <bindParam name="var" value="true"/>
      </bind>
   </link>
</context>
```

Listagem 3.34 Contexto de interatividade.

Agora, que já temos a manipulação da variável "service.interactivity" definida, vamos usar seu valor para controlar o aparecimento ou não do ícone da chuteira, que habilita a propaganda interativa. Para tanto, temos de modificar o relacionamento que dá o início da apresentação do ícone, para que ele teste antes o valor da variável "service.interactivity". Como a variável é uma propriedade do nó settings, vamos reusá-lo (aqui, sim, um exemplo de reúso bem útil) dentro do contexto da propaganda como o ator (de fato sua propriedade "service.interactivity") do relacionamento que dispara a apresentação do ícone. A Figura 3.14 ilustra a visão estrutural de toda a nova versão de *O Primeiro João*. Note que um novo elemento <bind> foi adicionado ao elo que finaliza todos os objetos ao término do vídeo da animação. Esse novo elemento <bind> é ligado ao contexto de interatividade sem especificar nenhuma porta de entrada. Isso significa, em NCL, que a ação "stop" do elo será aplicada a *todos* os objetos do contexto que ainda estão sendo apresentados.

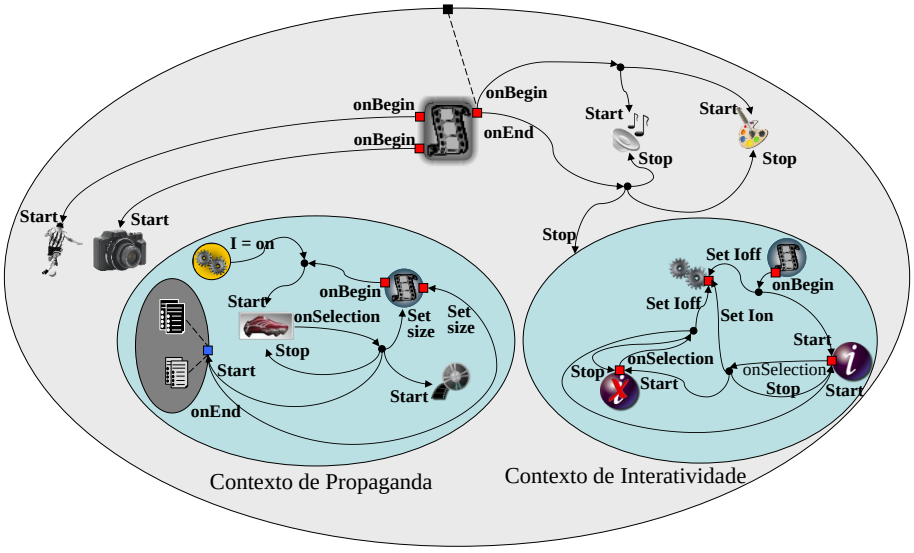


Figura 3.14 Visão estrutural da nova versão de *O Primeiro João*.

O leitor deve observar, pela comparação do contexto de propaganda da Figura 3.12 e da Figura 3.14, que apenas um relacionamento mudou. Todo o resto permaneceu igual. O novo relacionamento deve testar o valor da variável "service.interactivity" para iniciar ou não a apresentação do ícone da chuteira, conforme o valor dessa variável, conforme ilustra a Listagem 3.35.

```
<context id="advert">
...
    <media id="reusedGlobalVar" refer="globalVar"
                                          instance="instSame"/>
```

```
...
    <link id="lIcon" xconnector="conEx#onBeginVarStart">
       <bind role="onBegin" component="reusedAnimation"
                                              interface="segIcon"/>
       <bind role="var" component="reusedGlobalVar"
                              interface="service.interactivity"/>
       <bind role="start" component="icon"/>
    </link>
...
</context>
```

Listagem 3.35 Novo relacionamento para a exibição do ícone da chuteira (“icon”).

Vale a pena abrirmos aqui um parênteses para ver a definição do elemento <connector> “onBeginVarStart”, especificado externamente, como ilustra a Listagem 3.36.

```
<causalConnector id="onBeginVarStart">
   <compoundCondition operator="and">
      <simpleCondition role="onBegin"/>
      <assessmentStatement comparator="eq">
        <attributeAssessment role="var"
              attributeType="nodeProperty" eventType="attribution"/>
        <valueAssessment value="true"/>
      </assessmentStatement>
   </compoundCondition>
   <simpleAction role="start"/>
</causalConnector>
```

Listagem 3.36 Elemento <connector> “onBeginVarStart”.

Na Listagem 3.36, a condição de disparo da relação é composta. O disparo ocorre se for iniciada a apresentação de uma interface correspondente ao papel “onBegin” (no relacionamento da Listagem 3.35, quando for apresentado o instante do vídeo da animação especificado para o aparecimento do ícone) e se a variável correspondente ao papel “var” (no relacionamento da Listagem 3.35, a variável “service.interactivity”) tiver o valor atribuído igual a “true”. Valores de variáveis são testados a partir de elementos <assessmentStatement>, conforme veremos no Capítulo 10. Quando a condição composta for satisfeita, será disparada a ação que inicia a apresentação do papel “start” (no relacionamento da Listagem 3.35, associado ao ícone “icon”).

Finalmente, note que, para a apresentação do ícone, devem ser definidos o elemento <region> para seu posicionamento e o elemento <descriptor>

ligando o elemento <region> ao elemento <media> que o representa. A Listagem 3.37 apresenta o código NCL correspondente a essa nova versão de *O Primeiro João*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Exemplo de uso e reuso do no settings e link testando variavel do
settings -->
<ncl id="settings"
               xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
   <head>
     <ruleBase>
        <rule id="en" var="system.language" value="en"
                                              comparator="eq"/>
     </ruleBase>
     <regionBase>
        <region id="backgroundReg" width="100%" height="100%"
                                                    zIndex="1"/>
        <region id="screenReg" width="100%" height="100%"
                                                    zIndex="2">
           <region id="frameReg" left="5%" top="6.7%" width="18.5%"
                                       height="18.5%" zIndex="3"/>
           <region id="iconReg" left="87.5%" top="11.7%"
                        width="8.45%" height="6.7%" zIndex="3"/>
           <region id="shoesReg" left="15%" top="60%" width="25%"
                                       height="25%" zIndex="3"/>
           <region id="formReg" left="56.25%" top="8.33%"
                        width="38.75%" height="71.7%" zIndex="3"/>
           <region id="intReg" left="92.5%" top="91.7%"
                        width="5.07%" height="6.51%" zIndex="3"/>
        </region>
     </regionBase>
     <descriptorBase>
        <descriptor id="backgroundDesc" region="backgroundReg"/>
        <descriptor id="screenDesc" region="screenReg"/>
        <descriptor id="photoDesc" region="frameReg"
                                              explicitDur="5s"/>
        <descriptor id="audioDesc"/>
        <descriptor id="dribleDesc" region="frameReg"/>
        <descriptor id="iconDesc" region="iconReg"
                                              explicitDur="6s"/>
        <descriptor id="shoesDesc" region="shoesReg"/>
        <descriptor id="formDesc" region="formReg" focusIndex="1"
                                              explicitDur="15s"/>
        <descriptor id="intDesc" region="intReg"/>
```

```
    </descriptorBase>
    <connectorBase>
       <importBase documentURI="../causalConnBase.ncl"
                                                    alias="conEx"/>
    </connectorBase>
  </head>

  <body>
    <port id="entry" component="animation"/>
    <media id="background" src="../media/background.png"
                                    descriptor="backgroundDesc"/>
    <media id="animation" src="../media/animGar.mp4"
                                    descriptor="screenDesc">
       <area id="segDrible" begin="12s"/>
       <area id="segPhoto" begin="41s"/>
       <area id="segIcon" begin="45s" end="51s"/>
    </media>
    <media id="choro" src="../media/choro.mp3"
                                    descriptor="audioDesc"/>
    <media id="drible" src="../media/drible.mp4"
                                    descriptor="dribleDesc"/>
    <media id="photo" src="../media/photo.png"
                                    descriptor="photoDesc"/>
    <context id="interactivity">
       <media id="globalVar" type="application/x-ncl-settings">
          <property name="service.interactivity" value="true"/>
       </media>
       <media id="anotherAnimation" refer="animation"
                                          instance="instSame"/>
       <media id="intOn" src="../media/intOn.png"
                                          descriptor="intDesc"/>
       <media id="intOff" src="../media/intOff.png"
                                          descriptor="intDesc"/>
       <link id="lInt" xconnector="conEx#onBeginSet_varStart">
          <bind role="onBegin" component="anotherAnimation"/>
          <bind role="start" component="intOn"/>
          <bind role="set" component="globalVar"
                              interface="service.interactivity">
               <bindParam name="var" value="true"/>
          </bind>
       </link>
       <link id="lOn"
              xconnector="conEx#onKeySelectionStopSet_varStart">
          <bind role="onSelection" component="intOn">
```

```
            <bindParam name="keyCode" value="INFO"/>
        </bind>
        <bind role="start" component="intOff"/>
        <bind role="stop" component="intOn"/>
        <bind role="set" component="globalVar"
                        interface="service.interactivity">
            <bindParam name="var" value="false"/>
        </bind>
    </link>
    <link id="lOff"
            xconnector="conEx#onKeySelectionStopSet_varStart">
        <bind role="onSelection" component="intOff">
            <bindParam name="keyCode" value="INFO"/>
        </bind>
        <bind role="start" component="intOn"/>
        <bind role="stop" component="intOff"/>
        <bind role="set" component="globalVar"
                        interface="service.interactivity">
            <bindParam name="var" value="true"/>
        </bind>
    </link>
</context>
<context id="advert">
    <media id="reusedAnimation" refer="animation"
                                        instance="instSame">
        <property name="bounds"/>
    </media>
    <media id="reusedGlobalVar" refer="globalVar"
                                        instance="instSame"/>
    <media id="icon" src="../media/icon.png"
                                    descriptor="iconDesc"/>
    <media id="shoes" src="../media/shoes.mp4"
                                    descriptor="shoesDesc"/>
    <switch id="form">
        <switchPort id="spForm">
            <mapping component="enForm"/>
            <mapping component="ptForm"/>
        </switchPort>
        <bindRule constituent="enForm" rule="en"/>
        <defaultComponent component="ptForm"/>
        <media id="ptForm" src="../media/ptForm.htm"
                                    descriptor="formDesc"/>
        <media id="enForm" src="../media/enForm.htm"
                                    descriptor="formDesc"/>
```

```
        </switch>
        <link id="lIcon" xconnector="conEx#onBeginVarStart">
           <bind role="onBegin" component="reusedAnimation"
                                              interface="segIcon"/>
           <bind role="var" component="reusedGlobalVar"
                              interface="service.interactivity"/>
           <bind role="start" component="icon"/>
        </link>
        <link id="lBeginShoes"
              xconnector="conEx#onKeySelectionStopSet_varStart">
           <bind role="onSelection" component="icon">
              <bindParam name="keyCode" value="RED"/>
           </bind>
           <bind role="start" component="shoes"/>
           <bind role="start" component="form" interface="spForm"/>
           <bind role="set" component="reusedAnimation"
                                              interface="bounds">
              <bindParam name="var" value="5%,6.67%,45%,45%"/>
           </bind>
           <bind role="stop" component="icon"/>
        </link>
        <link id="lEndForm" xconnector="conEx#onEndSet_var">
           <bind role="onEnd" component="form" interface="spForm"/>
           <bind role="set" component="reusedAnimation"
                                              interface="bounds">
              <bindParam name="var" value="0,0,222.22%,222.22%"/>
           </bind>
        </link>
     </context>
     <link id="lMusic" xconnector="conEx#onBeginStart_delay">
        <bind role="onBegin" component="animation"/>
        <bind role="start" component="background">
           <bindParam name="delay" value="5s"/>
        </bind>
        <bind role="start" component="choro">
           <bindParam name="delay" value="5s"/>
        </bind>
     </link>
     <link id="lDrible" xconnector="conEx#onBeginStart">
        <bind role="onBegin" component="animation"
                                        interface="segDrible"/>
        <bind role="start" component="drible"/>
     </link>
     <link id="lPhoto" xconnector="conEx#onBeginStart">
```

```
        <bind role="onBegin" component="animation"
                                interface="segPhoto"/>
        <bind role="start" component="photo"/>
     </link>
     <link  id="lEnd" xconnector="conEx#onEndStop">
        <bind role="onEnd" component="animation"/>
        <bind role="stop" component="background"/>
        <bind role="stop" component="choro"/>
        <bind role="stop" component="interactivity"/>
     </link>
   </body>
</ncl>
```

Listagem 3.37 *O Primeiro João* com controle de propagandas interativas.

A Figura 3.15 ilustra o momento da apresentação do ícone de interatividade e um outro momento em que a interatividade foi inibida, obtidos durante a execução da aplicação na implementação de referência do middleware Ginga-NCL.

Figura 3.15 Cenas da aplicação *O Primeiro João* com controle de interatividade.

3.11 Efeitos de Transição e Animação

Vamos agora introduzir um efeito de transição no aparecimento do drible de vaivém do Garrincha (elemento <media> “drible”). O efeito de transição é uma característica de apresentação e, portanto, pode ser definido no elemento <descriptor> associado ao elemento <media> onde se quer a transição.

Antes de associarmos transições ao elemento <descriptor>, devemos criar uma base de transições para a aplicação NCL em questão. Isso é feito através do elemento <transitionBase>, também filho do elemento <head>. Uma base de transições contém, como diz seu próprio nome, transições

especificadas pelo elemento <transition>. A Listagem 3.38 ilustra dois tipos de transição que usaremos em nossa aplicação. A primeira transição, identificada como "trans1", especifica um desvanecimento de 2 segundos. A segunda transição, identificada como "trans2", especifica um efeito de varredura de barra vertical durante 1 segundo.

```
<head>
...
        <transitionBase>
                <transition id="trans1" type="fade" dur="2s"/>
                <transition id="trans2" type="barWipe" dur="1s"/>
        </transitionBase>
...
</head>
```

Listagem 3.38 Elementos <transitionBase> e <transition>.

Agora podemos associar as transições ao elemento <descriptor> associado ao vídeo "drible", conforme ilustrado na Listagem 3.39.

```
<descriptor    id="dribleDesc"    region="frameReg"    transIn="trans1"
                                                transOut="trans2"/>
```

Listagem 3.39 Efeito de transição.

Pela Listagem 3.39, uma transição de desvanecimento de entrada do vídeo "drible" e uma de varredura vertical ao final do vídeo foram definidas.

Ainda nessa mesma versão da aplicação, vamos acrescentar um efeito de animação à foto do marcador caído no chão (elemento <media> "photo") e também introduzir um efeito de transparência a essa foto. A Listagem 3.40 ilustra como o descritor associado à foto pode ser alterado para produzir o efeito de transparência (uma transparência de 60% foi definida).

```
<descriptor id="photoDesc" region="frameReg" explicitDur="5s">
        <descriptorParam name="transparency" value="0.6"/>
</descriptor>
```

Listagem 3.40 Efeito de transparência.

O efeito de animação que queremos é bem simples. Vamos apenas fazer com que a foto seja apresentada e depois se desloque vertical e continuamente. Para tanto, teremos de mexer no relacionamento que define o início da apresentação da foto. Iremos acrescentar ao relacionamento uma outra ação, uma ação de atribuição de valores, que irá modificar o valor do atributo *top* do elemento continuamente, até um certo valor final. A Figura

3.16 ilustra a nova visão estrutural, onde, por clareza, os contextos de interatividade e de propaganda são apresentados de forma resumida.

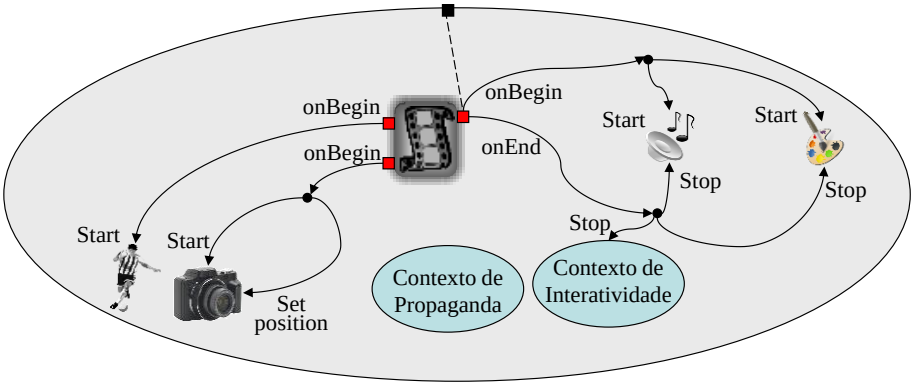


Figura 3.16 Visão estrutural de *O Primeiro João*, com efeito de animação e transição.

Como vamos modificar o atributo *top*, ele precisa ser explicitamente declarado no elemento <media> "photo", conforme ilustra a Listagem 3.41, que também traz a definição do novo relacionamento.

```
<body>
...
    <media id="photo" src="../media/photo.png"
                                    descriptor="photoDesc">
       <property name="top"/>
    </media>
...
    <link id="lPhoto"
              xconnector="conEx#onBeginStartSet_var_delay_duration">
       <bind role="onBegin" component="animation"
                                          interface="segPhoto"/>
       <bind role="start" component="photo"/>
       <bind role="set" component="photo" interface="top">
          <bindParam name="var" value="290"/>
          <bindParam name="delay" value="1s"/>
          <bindParam name="duration" value="3s"/>
       </bind>
    </link>
...
<body>
```

Listagem 3.41 Novo relacionamento com animação.

Na Listagem 3.41, note o novo papel introduzido (papel "set"). Ele estabelece que, depois de transcorrido 1 segundo do início da exibição da foto

(parâmetro “delay”), o seu valor de topo deve mudar para 290 pixels continuamente, em um intervalo de 3 segundos (parâmetro “duration”). O efeito que se terá é o da foto se deslocando verticalmente.

A especificação completa da nova versão do programa NCL é ilustrada na Listagem 3.42.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Exemplo de animacao -->
<ncl id="animationExample"
               xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
   <head>
     <ruleBase>
        <rule id="en" var="system.language" value="en"
                                              comparator="eq"/>
     </ruleBase>
     <transitionBase>
        <transition id="trans1" type="fade" dur="2s"/>
        <transition id="trans2" type="barWipe" dur="1s"/>
     </transitionBase>
     <regionBase>
        <region id="backgroundReg" width="100%" height="100%"
                                                    zIndex="1"/>
        <region id="screenReg" width="100%" height="100%" zIndex="2">
           <region id="frameReg" left="5%" top="6.7%" width="18.5%"
                                     height="18.5%" zIndex="3"/>
           <region id="iconReg" left="87.5%" top="11.7%"
                         width="8.45%" height="6.7%" zIndex="3"/>
           <region id="shoesReg" left="15%" top="60%" width="25%"
                                     height="25%" zIndex="3"/>
           <region id="formReg" left="56.25%" top="8.33%"
                         width="38.75%" height="71.7%" zIndex="3"/>
           <region id="intReg" left="92.5%" top="91.7%" width="5.07%"
                                     height="6.51%" zIndex="3"/>
        </region>
     </regionBase>
     <descriptorBase>
        <descriptor id="backgroundDesc" region="backgroundReg"/>
        <descriptor id="screenDesc" region="screenReg"/>
        <descriptor id="photoDesc" region="frameReg"
                                              explicitDur="5s">
           <descriptorParam name="transparency" value="0.6"/>
        </descriptor>
        <descriptor id="audioDesc"/>
```

```
      <descriptor id="dribleDesc" region="frameReg"
                              transIn="trans1" transOut="trans2"/>
      <descriptor id="iconDesc" region="iconReg" explicitDur="6s"/>
      <descriptor id="shoesDesc" region="shoesReg"/>
      <descriptor id="formDesc" region="formReg" focusIndex="1"
                                              explicitDur="15s"/>
      <descriptor id="intDesc" region="intReg"/>
    </descriptorBase>
    <connectorBase>
      <importBase documentURI="../causalConnBase.ncl"
                                                    alias="conEx"/>
    </connectorBase>
  </head>

  <body>
    <port id="entry" component="animation"/>
    <media id="background" src="../media/background.png"
                                     descriptor="backgroundDesc"/>
    <media id="animation" src="../media/animGar.mp4"
                                     descriptor="screenDesc">
      <area id="segDrible" begin="12s"/>
      <area id="segPhoto" begin="41s"/>
      <area id="segIcon" begin="45s" end="51s"/>
    </media>
    <media id="choro" src="../media/choro.mp3"
                                     descriptor="audioDesc"/>
    <media id="drible" src="../media/drible.mp4"
                                     descriptor="dribleDesc"/>
    <media id="photo" src="../media/photo.png"
                                     descriptor="photoDesc">
      <property name="top"/>
    </media>
    <context id="interactivity">
      <media id="globalVar" type="application/x-ncl-settings">
        <property name="service.interactivity" value="true"/>
      </media>
      <media id="anotherAnimation" refer="animation"
                                              instance="instSame"/>
      <media id="intOn" src="../media/intOn.png"
                                              descriptor="intDesc"/>
      <media id="intOff" src="../media/intOff.png"
                                              descriptor="intDesc"/>
      <link id="lInt" xconnector="conEx#onBeginSet_varStart">
        <bind role="onBegin" component="anotherAnimation"/>
```

```
         <bind role="start" component="intOn"/>
         <bind role="set" component="globalVar"
                              interface="service.interactivity">
            <bindParam name="var" value="true"/>
         </bind>
      </link>
            <bindParam name="varSet" value="false"/>
         </bind>
      </link>
      <link id="lOn"
              xconnector="conEx#onKeySelectionStopSet_varStart">
         <bind role="onSelection" component="intOn">
            <bindParam name="keyCode" value="INFO"/>
         </bind>
         <bind role="start" component="intOff"/>
         <bind role="stop" component="intOn"/>
         <bind role="set" component="globalVar"
                              interface="service.interactivity">
            <bindParam name="var" value="false"/>
         </bind>
      </link>
      <link id="lOff"
              xconnector="conEx#onKeySelectionStopSet_varStart">
         <bind role="onSelection" component="intOff">
            <bindParam name="keyCode" value="INFO"/>
         </bind>
         <bind role="start" component="intOn"/>
         <bind role="stop" component="intOff"/>
         <bind role="set" component="globalVar"
                              interface="service.interactivity">
            <bindParam name="var" value="true"/>
         </bind>
      </link>
   </context>
   <context id="advert">
      <media id="reusedAnimation" refer="animation"
                                             instance="instSame">
         <property name="bounds"/>
      </media>
      <media id="reusedGlobalVar" refer="globalVar"
                                             instance="instSame"/>
      <media id="icon" src="../media/icon.png"
                                             descriptor="iconDesc"/>
      <media id="shoes" src="../media/shoes.mp4"
```

```
                                    descriptor="shoesDesc"/>
      <switch id="form">
         <switchPort id="spForm">
            <mapping component="enForm"/>
            <mapping component="ptForm"/>
         </switchPort>
         <bindRule constituent="enForm" rule="en"/>
         <defaultComponent component="ptForm"/>
         <media id="ptForm" src="../media/ptForm.htm"
                     type="text/html" descriptor="formDesc"/>
         <media id="enForm" src="../media/enForm.htm"
                     type="text/html" descriptor="formDesc"/>
      </switch>
      <link id="lIcon" xconnector="conEx#onBeginVarStart">
         <bind role="onBegin" component="reusedAnimation"
                                          interface="segIcon"/>
         <bind role="var" component="reusedGlobalVar"
                             interface="service.interactivity"/>
         <bind role="start" component="icon"/>
      </link>
      <link id="lBeginShoes"
             xconnector="conEx#onKeySelectionStopSet_varStart">
         <bind role="onSelection" component="icon">
            <bindParam name="keyCode" value="RED"/>
         </bind>
         <bind role="start" component="shoes"/>
         <bind role="start" component="form" interface="spForm"/>
         <bind role="set" component="reusedAnimation"
                                          interface="bounds">
            <bindParam name="var" value="5%,6.67%,45%,45%"/>
         </bind>
         <bind role="stop" component="icon"/>
      </link>
      <link id="lEndForm" xconnector="conEx#onEndSet_var">
         <bind role="onEnd" component="form" interface="spForm"/>
         <bind role="set" component="reusedAnimation"
                                          interface="bounds">
            <bindParam name="var" value="0,0,222.22%,222.22%"/>
         </bind>
      </link>
   </context>
   <link id="lMusic" xconnector="conEx#onBeginStart_delay">
      <bind role="onBegin" component="animation"/>
      <bind role="start" component="background">
```

```
            <bindParam name="delay" value="5s"/>
         </bind>
         <bind role="start" component="choro">
            <bindParam name="delay" value="5s"/>
         </bind>
      </link>
      <link id="lDrible" xconnector="conEx#onBeginStart">
         <bind role="onBegin" component="animation"
                                       interface="segDrible"/>
         <bind role="start" component="drible"/>
      </link>
      <link id="lPhoto"
               xconnector="conEx#onBeginStartSet_var_delay_duration">
         <bind role="onBegin" component="animation"
                                       interface="segPhoto"/>
         <bind role="start" component="photo"/>
         <bind role="set" component="photo" interface="top">
            <bindParam name="var" value="290"/>
            <bindParam name="delay" value="1s"/>
            <bindParam name="duration" value="3s"/>
         </bind>
      </link>
      <link id="lEnd" xconnector="conEx#onEndStop">
         <bind role="onEnd" component="animation"/>
         <bind role="stop" component="background"/>
         <bind role="stop" component="choro"/>
         <bind role="stop" component="interactivity"/>
      </link>
   </body>
</ncl>
```

Listagem 3.42 *O Primeiro João* com efeitos de animação e transição.

A Figura 3. 17 ilustra o efeito de transição de saída, do tipo *bar wipe*, no vídeo do canto superior esquerdo, que se sobrepõe ao vídeo da animação.

Figura 3.17 Cenas da aplicação *O Primeiro João* com efeitos de transição.

3.12 Navegação por Teclas

A próxima funcionalidade que agregaremos em nossa aplicação *O Primeiro João* é a opção de escolher a música de fundo, ou seja, a substituição do chorinho pelo "rock", ou um "techno", ou um estilo "cartoon".

A escolha do ritmo se dará por meio de navegação sobre ícones (imagens) que os representam. A navegação se dará por meio das teclas CURSOR_RIGHT e CURSOR_LEFT do controle remoto. O ícone em foco, se selecionado pela tecla ENTER, efetuará a substituição da música.

Como os ícones estarão sempre visíveis, para não sobrepô-los ao vídeo da animação vamos redimensionar a região que define onde esse vídeo será apresentado. Teremos também de definir as regiões onde os ícones serão exibidos. Assim, o novo elemento <regionBase> pode ser definido conforme ilustrado na Listagem 3.43. Devemos notar, pela nova definição, que todas as regiões são definidas com relação à região da figura de fundo e que os ícones correspondentes aos vários ritmos da música de fundo serão apresentados na parte mais inferior da tela.

```
<regionBase>
   <region id="backgroundReg" width="100%" height="100%" zIndex="1">
      <region id="screenReg" width="100%" height="88%" zIndex="2"/>
      <region id="frameReg" left="5%" top="6.7%" width="18.5%"
                                    height="18.5%" zIndex="3"/>
      <region id="iconReg" left="87.5%" top="11.7%" width="8.45%"
                                    height="6.7%" zIndex="3"/>
      <region id="shoesReg" left="15%" top="60%" width="25%"
                                    height="25%" zIndex="3"/>
      <region id="formReg" left="56.25%" top="8.33%" width="38.75%"
                                    height="71.7%" zIndex="3"/>
      <region id="intReg" left="92.5%" top="91.7%" width="5.07%"
                                    height="6.51%" zIndex="3"/>
      <region id="chorinhoReg" left="2.5%" top="91.7%"
                        width="11.7%" height="6.51%" zIndex="3"/>
      <region id="rockReg" left="25%" top="91.7%" width="11.7%"
                                    height="6.51%" zIndex="3"/>
      <region id="technoReg" left="47.5%" top="91.7%" width="11.7%"
                                    height="6.51%" zIndex="3"/>
      <region id="cartoonReg" left="70%" top="91.7%" width="11.7%"
                                    height="6.51%" zIndex="3"/>
   </region>
</regionBase>
```

Listagem 3.43 Base de regiões da nova versão.

Como próximo passo, temos de definir o novo conjunto de descritores. Descritores são especificados para cada um dos quatro ícones, a serem apresentados para a seleção do ritmo.

Na definição de cada descritor, devemos informar seu índice para a navegação pelas teclas do controle remoto. Isso é feito pelo atributo *focusIndex* do elemento <descriptor>. Devemos também informar os próximos elementos a receberem o foco quando navegarmos por meio das teclas CURSOR_RIGHT e CURSOR_LEFT, indicando os próximos valores de *focusIndex* que deverão ser objeto de foco. Isso será feito por meio dos atributos *moveRight* e *moveLeft*, respectivamente. Assim, tomando como exemplo o elemento <descriptor> "chorinhoDesc" da Listagem 3.44, vemos que seu índice para foco é "2" e que, quando o elemento <media> que o referencia está com o foco e a tecla CURSOR_RIGHT do controle remoto é pressionada, o foco é movido para o elemento <media> que referencia o elemento <descriptor> com o atributo *focusIndex* igual a "3". Analogamente, quando o elemento <media> que referencia o elemento <descriptor> "chorinhoDesc" está com o foco e a tecla CURSOR_LEFT do controle remoto é pressionada, o foco é movido para o elemento <media> que referencia o elemento <descriptor> com o atributo *focusIndex* igual a "5". A Listagem 3.44 ilustra a definição dos novos descritores. Note que a navegação pelos ícones que representam os ritmos é circular.

```
<descriptorBase>

   <descriptor id="chorinhoDesc" region="chorinhoReg" focusIndex="2"
                                    moveRight="3" moveLeft="5"/>
   <descriptor id="rockDesc" region="rockReg" focusIndex="3"
                                    moveRight="4" moveLeft="2"/>
   <descriptor id="technoDesc" region="technoReg" focusIndex="4"
                                    moveRight="5" moveLeft="3"/>
   <descriptor id="cartoonDesc" region="cartoonReg" focusIndex="5"
                                    moveRight="2" moveLeft="4"/>
</descriptorBase>
```

Listagem 3.44 Novos descritores com a definição de atributos para navegação por teclas.

O leitor já deve ter percebido que, além dos quatro ícones mencionados, também teremos de acrescentar à aplicação mais três objetos de áudio, uma vez que o áudio do chorinho já estava presente na versão anterior da aplicação.

Vamos, então, estabelecer o seguinte cenário para nossa aplicação. Como anteriormente, o áudio chorinho começará 5 segundos após o início da aplicação. Juntamente com o início do "choro", vamos também iniciar a apresentação dos ícones (imagens) para choro, rock, techno e cartoon. Para

tanto, vamos colocar o elemento <media> “choro” do tipo áudio e os elementos <media> “imgChorinho”, “imgRock”, “imgTechno” e “imgCartoon”, do tipo imagem, dentro (como filhos) de um elemento <context> “menu”. Vamos também colocar cinco elementos <port>, cada um mapeando para cada um dos novos elementos <media> definidos. Como realizar uma ação “start” em um contexto sem especificar a porta significa que todas as portas devem receber a ação, bastará termos um relacionamento especificando o “start” do elemento <context> “menu” para que todos os ícones sejam apresentados e o chorinho iniciado, como queríamos.

Continuando com nosso cenário, vamos estabelecer que, em qualquer instante em que um dos ícones for acionado, com exceção do ícone do chorinho, o chorinho terá seu volume colocado em “zero” (mudo), o ritmo correspondente ao ícone selecionado iniciará sua apresentação e os demais ritmos serão abruptamente terminados. A qualquer momento que o ícone do chorinho for selecionado, os demais ritmos cessarão e o áudio do choro será restabelecido em seu valor inicial.

Para conseguirmos o efeito descrito no parágrafo anterior mais facilmente, vamos reunir os três áudios (elementos <media> “rock”, “techno” e “cartoon”) em um elemento <switch>, identificado como “musics”, e colocá-lo também como filho do elemento <context> “menu”. O elemento <switch> terá uma porta (elemento <switchPort>) mapeada para os três elementos de mídia representando os áudios, que serão escolhidos conforme o valor do atributo *focusIndex* do elemento <media> representando o ícone associado ao ritmo que foi selecionado pela tecla ENTER do controle remoto. Para tanto, a variável global “service.currentFocus” (uma propriedade do nó *settings* por nós já discutido) será consultada. Essa variável é mantida pelo sistema exibidor da aplicação NCL, contendo sempre o valor do atributo *focusIndex* corrente.

Quando quisermos parar *todos* os elementos filhos do elemento <context> “menu”, bastará realizar uma ação “stop” sobre o contexto, sem especificar nenhuma porta. Na nova versão da aplicação, vamos querer que isso aconteça quando o vídeo da animação atingir o ponto em que são apresentados os créditos dos autores (elemento <area> “segCred”). Vamos também aproveitar para trocar o término do contexto de interatividade para esse ponto do vídeo da animação. A Listagem 3.45 ilustra a definição do elemento <link> que permite esse procedimento.

```
<link xconnector="conEx#onEndStop">
   <bind role="onEnd" component="animation" interface="segCred"/>
   <bind role="stop" component="menu"/>
   <bind role="stop" component="interactivity"/>
</link>
```

Listagem 3.45 Finalização de todos os elementos em exibição do contexto “menu”.

Teremos, então, a visão estrutural ilustrada na Figura 3.18 para essa nova versão de *O Primeiro João*.

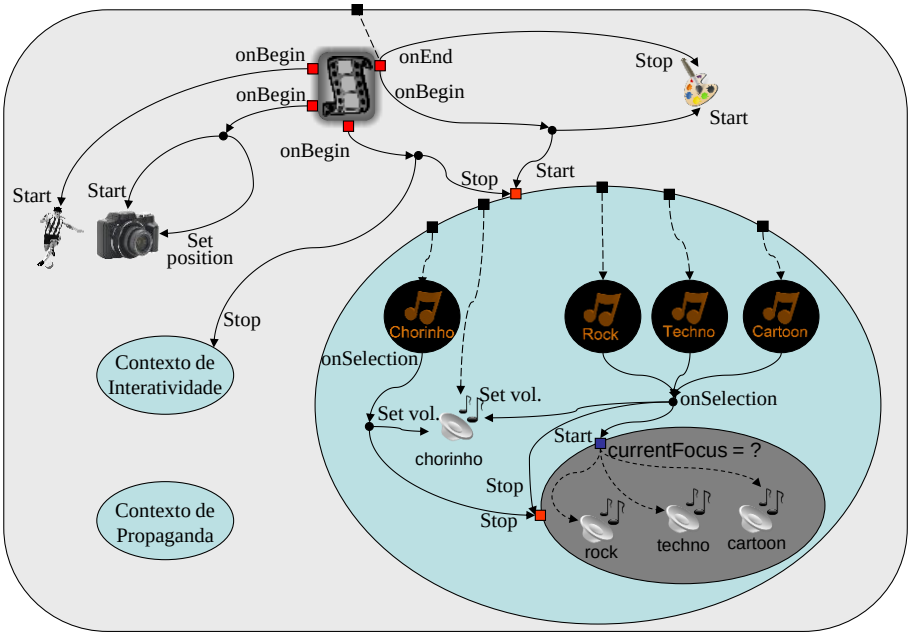


Figura 3.18 Visão estrutural de *O Primeiro João* com navegação por teclas.

Note pela Figura 3.18 que a seleção de qualquer um dos três ícones, “imgRock”, “imgTechno” ou “imgCartoon”, coloca o volume do chorinho em zero, para a exibição dos demais áudios e inicia a apresentação do áudio correspondente ao foco corrente, ou seja, ao áudio associado ao ícone selecionado.

Para definir o elemento <switch> “menu”, novas regras de seleção devem ser definidas no elemento <ruleBase> que, em sua nova versão, segue a especificação ilustrada na Listagem 3.46. Note que as novas regras introduzidas dizem respeito à variável “service.currentFocus”, que terá seu valor testado contra o valor do atributo *focusIndex* de cada um dos ícones associados aos elementos <media> representando os vários ritmos. A regra é considerada satisfeita se houver igualdade entre os valores.

```
<ruleBase>
   <rule id="en" var="system.language" value="en" comparator="eq"/>
   <rule id="int" var="service.interactivity" value="true"
                                              comparator="eq"/>
   <rule id="rRock" var="service.currentFocus" value="3"
                                              comparator="eq"/>
   <rule id="rTechno" var="service.currentFocus" value="4"
                                              comparator="eq"/>
   <rule id="rCartoon" var="service.currentFocus" value="5"
                                              comparator="eq"/>
</ruleBase>
```

Listagem 3.46 Redefinição do elemento <ruleBase> e seus novos elementos filhos.

Agora podemos definir o elemento <context> "menu", incluindo entre seus filhos o elemento <switch> "musics" para a escolha dos ritmos e os demais elementos discutidos nesta seção, conforme ilustra a Listagem 3.47.

```
<context id="menu">
   <port id="pChoro" component="choro"/>
   <port id="pChorinho" component="imgChorinho"/>
   <port id="pRock" component="imgRock"/>
   <port id="pTechno" component="imgTechno"/>
   <port id="pCartoon" component="imgCartoon"/>
   <media id="imgChorinho" src="../media/chorinho.png"
                                     descriptor="chorinhoDesc"/>
   <media id="imgRock" src="../media/rock.png"
                                     descriptor="rockDesc"/>
   <media id="imgTechno" src="../media/techno.png"
                                     descriptor="technoDesc"/>
   <media id="imgCartoon" src="../media/cartoon.png"
                                     descriptor="cartoonDesc"/>
   <media id="choro" src="../media/choro.mp3" descriptor="audioDesc">
      <property name="soundLevel" value="1"/>
   </media>
   <switch id="musics">
      <bindRule constituent="rock" rule="rRock"/>
      <bindRule constituent="techno" rule="rTechno"/>
      <bindRule constituent="cartoon" rule="rCartoon"/>
      <media id="rock" src="../media/rock.mp3"/>
      <media id="techno" src="../media/techno.mp3"/>
      <media id="cartoon" src="../media/cartoon.mp3"/>
   </switch>
   <link id="lChoro" xconnector="conEx#onSelectionSet_varStop">
```

```
      <bind role="onSelection" component="imgChorinho"/>
      <bind role="set" component="choro" interface="soundLevel">
         <bindParam name="var" value="1"/>
      </bind>
      <bind role="stop" component="musics"/>
   </link>
   <link id="lOthers"
               xconnector="conEx#onSelection_orSet_varStopStart">
      <bind role="onSelection" component="imgRock"/>
      <bind role="onSelection" component="imgTechno"/>
      <bind role="onSelection" component="imgCartoon"/>
      <bind role="set" component="choro" interface="soundLevel">
         <bindParam name="var" value="0"/>
      </bind>
      <bind role="stop" component="musics"/>
      <bind role="start" component="musics"/>
   </link>
</context>
```

Listagem 3.47 Elemento <context> “menu”.

Note, na Listagem 3.47, a definição das portas para todos os ícones que representam os diferentes ritmos e para o áudio do chorinho, conforme discutimos anteriormente. Note também a definição do elemento <switch> “musics” e, ainda, a definição dos elementos <link>. O primeiro relacionamento (“lChoro”) especifica que, ao ser selecionada (condição “onSelection”) a imagem correspondente ao ícone do chorinho, o som do áudio do choro deve ser colocado em seu volume original (ação “set” colocando o valor de *soundLevel* em “1”), e todos os outros ritmos devem ser encerrados (ação de “stop” no elemento <switch> “musics”). O segundo relacionamento (“lOthers”) especifica que, ao ser selecionada (condição “onSelection”) qualquer das imagens correspondentes aos outros ritmos (rock, techno e cartoon), o volume do áudio do choro deve ser abaixado totalmente (ação “set” colocando o valor de *soundLevel* em “0”), qualquer outro ritmo que estiver sendo tocado deve parar (ação de “stop” no elemento <switch> “musics”), e o áudio correspondente ao ícone selecionado (cujo valor de *focusIndex* é então atribuído ao *currentFocus*) deve ser iniciado (ação de “start” no elemento <switch> “musics”, com a seleção do componente segundo o valor da variável “service.currentFocus”).

Finalmente, podemos realizar as duas últimas alterações na versão anterior da aplicação. A primeira, alterando o elemento <link> “lMusic”, para que ele agora inicie a apresentação do elemento <context> “menu” a partir dos 5 segundos do início do vídeo da animação, isto é, inicie a apresentação

do chorinho e dos ícones correspondentes a todos os ritmos, conforme ilustra a Listagem 3.48.

```
<link id="lMusic" xconnector="conEx#onBeginStart_delay">
        <bind role="onBegin" component="animation"/>
        <bind role="start" component="background">
                <bindParam name="delay" value="5s"/>
        </bind>
        <bind role="start" component="menu">
                <bindParam name="delay" value="5s"/>
        </bind>
</link>
```

Listagem 3.48 Redefinição do elemento <link> "lMusic".

A segunda alteração é bem sutil. Quando da apresentação do formulário para compra da chuteira, devemos obrigatoriamente colocar o formulário com o foco, independentemente de onde o foco esteja (em relação aos ícones representando os diversos ritmos). Essa ação tornará possível o preenchimento do formulário. Caso não façamos assim, o foco ficará circulando entre os ícones e nunca será dado ao formulário. Note que, quando o formulário ganhar o foco, ele só será novamente passado aos ícones no final de sua apresentação. A Listagem 3.49 ilustra o elemento <link> responsável por determinar o foco no formulário, assim que se inicia sua apresentação. Note que, para estabelecer o foco, basta colocar a variável global "service.currentFocus" com o valor do atributo *focusIndex* do formulário (no caso, o valor "1").

```
<link id="lFormFocus" xconnector="conEx#onBeginSet_var">
        <bind role="onBegin" component="form"/>
        <bind role="set" component="reusedGlobalVar"
                                interface="service.currentFocus">
                <bindParam name="var" value="1"/>
        </bind>
</link>
```

Listagem 3.49 Passando o foco ao formulário assim que inicia sua apresentação.

A especificação completa da nova versão do programa NCL é ilustrada na Listagem 3.50.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Exemplo de navegacao por menu -->
<ncl id="menuEx" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
   <head>
```

```
<ruleBase>
   <rule id="en" var="system.language" value="en"
                                        comparator="eq"/>
   <rule id="int" var="service.interactivity" value="true"
                                        comparator="eq"/>
   <rule    id="rRock"    var="service.currentFocus"    value="3"
                                        comparator="eq"/>
   <rule id="rTechno" var="service.currentFocus" value="4"
                                        comparator="eq"/>
   <rule id="rCartoon" var="service.currentFocus" value="5"
                                        comparator="eq"/>
</ruleBase>
<transitionBase>
   <transition id="trans1" type="fade" dur="2s"/>
   <transition id="trans2" type="barWipe" dur="1s"/>
</transitionBase>
<regionBase>
   <region id="backgroundReg" width="100%" height="100%"
                                              zIndex="1">
      <region id="screenReg" width="100%" height="88%"
                                              zIndex="2"/>
      <region id="frameReg" left="5%" top="6.7%" width="18.5%"
                                 height="18.5%" zIndex="3"/>
      <region id="iconReg" left="87.5%" top="11.7%"
                 width="8.45%" height="6.7%" zIndex="3"/>
      <region id="shoesReg" left="15%" top="60%" width="25%"
                                 height="25%" zIndex="3"/>
      <region id="formReg" left="56.25%" top="8.33%"
                 width="38.75%" height="71.7%" zIndex="3"/>
      <region id="intReg" left="92.5%" top="91.7%"
                 width="5.07%" height="6.51%" zIndex="3"/>
      <region id="chorinhoReg" left="2.5%" top="91.7%"
                 width="11.7%" height="6.51%" zIndex="3"/>
      <region id="rockReg" left="25%" top="91.7%"
                 width="11.7%" height="6.51%" zIndex="3"/>
      <region id="technoReg" left="47.5%" top="91.7%"
                 width="11.7%" height="6.51%" zIndex="3"/>
      <region id="cartoonReg" left="70%" top="91.7%"
                 width="11.7%" height="6.51%" zIndex="3"/>
   </region>
</regionBase>

<descriptorBase>
   <descriptor id="backgroundDesc" region="backgroundReg"/>
```

```
        <descriptor id="screenDesc" region="screenReg"/>
        <descriptor id="photoDesc" region="frameReg"
                                              explicitDur="5s">
           <descriptorParam name="transparency" value="0.6"/>
        </descriptor>
        <descriptor id="audioDesc"/>
        <descriptor id="dribleDesc" region="frameReg"
                              transIn="trans1" transOut="trans2"/>
        <descriptor id="iconDesc" region="iconReg"
                                              explicitDur="6s"/>
        <descriptor id="shoesDesc" region="shoesReg"/>
        <descriptor id="formDesc" region="formReg" focusIndex="1"
                                              explicitDur="15s"/>
        <descriptor id="intDesc" region="intReg"/>
        <descriptor id="chorinhoDesc" region="chorinhoReg"
                     focusIndex="2" moveRight="3" moveLeft="5"/>
        <descriptor id="rockDesc" region="rockReg" focusIndex="3"
                                       moveRight="4" moveLeft="2"/>
        <descriptor id="technoDesc" region="technoReg"
                     focusIndex="4" moveRight="5" moveLeft="3"/>
        <descriptor id="cartoonDesc" region="cartoonReg"
                     focusIndex="5" moveRight="2" moveLeft="4"/>
     </descriptorBase>
     <connectorBase>
        <importBase documentURI="../causalConnBase.ncl"
                                                    alias="conEx"/>
     </connectorBase>
  </head>
  <body>
     <port id="entry" component="animation"/>
     <media id="background" src="../media/background.png"
                                       descriptor="backgroundDesc"/>
     <media id="animation" src="../media/animGar.mp4"
                                       descriptor="screenDesc">
        <area id="segDrible" begin="12s"/>
        <area id="segPhoto" begin="41s"/>
        <area id="segIcon" begin="45s" end="51s"/>
        <area id="segCred" end="64s"/>
     </media>
     <media id="drible" src="../media/drible.mp4"
                                       descriptor="dribleDesc"/>
     <media id="photo" src="../media/photo.png"
                                       descriptor="photoDesc">
        <property name="top"/>
```

```
    </media>
    <context id="interactivity">
       <media id="globalVar" type="application/x-ncl-settings">
          <property name="service.interactivity" value="true"/>
          <property name="service.currentFocus"/>
       </media>
       <media id="anotherAnimation" refer="animation"
                                             instance="instSame"/>
       <media id="intOn" src="../media/intOn.png"
                                             descriptor="intDesc"/>
       <media id="intOff" src="../media/intOff.png"
                                             descriptor="intDesc"/>
       <link id="lInt" xconnector="conEx#onBeginSet_varStart">
          <bind role="onBegin" component="anotherAnimation"/>
          <bind role="start" component="intOn"/>
          <bind role="set" component="globalVar"
                            interface="service.interactivity">
            <bindParam name="var" value="true"/>
          </bind>
       </link>
       <link id="lOn"
             xconnector="conEx#onKeySelectionStopSet_varStart">
          <bind role="onSelection" component="intOn">
            <bindParam name="keyCode" value="INFO"/>
          </bind>
          <bind role="start" component="intOff"/>
          <bind role="stop" component="intOn"/>
          <bind role="set" component="globalVar"
                            interface="service.interactivity">
            <bindParam name="var" value="false"/>
          </bind>
       </link>
       <link id="lOff"
             xconnector="conEx#onKeySelectionStopSet_varStart">
          <bind role="onSelection" component="intOff">
            <bindParam name="keyCode" value="INFO"/>
          </bind>
          <bind role="start" component="intOn"/>
          <bind role="stop" component="intOff"/>
          <bind role="set" component="globalVar"
                            interface="service.interactivity">
            <bindParam name="var" value="true"/>
          </bind>
       </link>
```

```
    </context>
    <context id="advert">
       <media id="reusedAnimation" refer="animation"
                                              instance="instSame">
          <property name="bounds"/>
       </media>
       <media id="reusedGlobalVar" refer="globalVar"
                                              instance="instSame"/>
       <media id="icon" src="../media/icon.png"
                                    descriptor="iconDesc"/>
       <media id="shoes" src="../media/shoes.mp4"
                                    descriptor="shoesDesc"/>
       <switch id="form">
          <switchPort id="spForm">
            <mapping component="enForm"/>
            <mapping component="ptForm"/>
          </switchPort>
          <bindRule constituent="enForm" rule="en"/>
          <defaultComponent component="ptForm"/>
          <media id="ptForm" src="../media/ptForm.htm"
                     type="text/html" descriptor="formDesc"/>
          <media id="enForm" src="../media/enForm.htm"
                     type="text/html" descriptor="formDesc"/>
       </switch>
       <link id="lIcon" xconnector="conEx#onBeginVarStart">
          <bind role="onBegin" component="reusedAnimation"
                                              interface="segIcon"/>
          <bind role="var" component="reusedGlobalVar"
                             interface="service.interactivity"/>
          <bind role="start" component="icon"/>
       </link>
       <link id="lBeginShoes"
             xconnector="conEx#onKeySelectionStopSet_varStart">
          <bind role="onSelection" component="icon">
            <bindParam name="keyCode" value="RED"/>
          </bind>
          <bind role="start" component="shoes"/>
          <bind role="start" component="form" interface="spForm"/>
          <bind role="set" component="reusedAnimation"
                                              interface="bounds">
            <bindParam name="var" value="5%,6.67%,45%,45%"/>
          </bind>
          <bind role="stop" component="icon"/>
       </link>
```

```
        <link id="lFormFocus" xconnector="conEx#onBeginSet">
           <bind role="onBegin" component="form"/>
           <bind role="set" component="reusedGlobalVar"
                            interface="service.currentFocus">
            <bindParam name="var" value="1"/>
           </bind>
        </link>
        <link id="lEndForm" xconnector="conEx#onEndSet_var">
           <bind role="onEnd" component="form" interface="spForm"/>
           <bind role="set" component="reusedAnimation"
                                              interface="bounds">
            <bindParam name="var" value="0,0,222.22%,222.22%"/>
           </bind>
        </link>
     </context>
     <context id="menu">
        <port id="pChoro" component="choro"/>
        <port id="pChorinho" component="imgChorinho"/>
        <port id="pRock" component="imgRock"/>
        <port id="pTechno" component="imgTechno"/>
        <port id="pCartoon" component="imgCartoon"/>
        <media id="imgChorinho" src="../media/chorinho.png"
                                   descriptor="chorinhoDesc"/>
        <media id="imgRock" src="../media/rock.png"
                                   descriptor="rockDesc"/>
        <media id="imgTechno" src="../media/techno.png"
                                   descriptor="technoDesc"/>
        <media id="imgCartoon" src="../media/cartoon.png"
                                   descriptor="cartoonDesc"/>
        <media id="choro" src="../media/choro.mp3"
                                   descriptor="audioDesc">
           <property name="soundLevel" value="1"/>
        </media>
        <switch id="musics">
           <bindRule constituent="rock" rule="rRock"/>
           <bindRule constituent="techno" rule="rTechno"/>
           <bindRule constituent="cartoon" rule="rCartoon"/>
           <media id="rock" src="../media/rock.mp3"/>
           <media id="techno" src="../media/techno.mp3"/>
           <media id="cartoon" src="../media/cartoon.mp3"/>
        </switch>
        <link id="lChoro" xconnector="conEx#onSelectionSet_varStop">
           <bind role="onSelection" component="imgChorinho"/>
           <bind role="set" component="choro"
```

```
                                    interface="soundLevel">
            <bindParam name="var" value="1"/>
         </bind>
         <bind role="stop" component="musics"/>
      </link>
      <link id="lOthers"
            xconnector="conEx#onSelection_orSet_varStopStart">
         <bind role="onSelection" component="imgRock"/>
         <bind role="onSelection" component="imgTechno"/>
         <bind role="onSelection" component="imgCartoon"/>
         <bind role="set" component="choro"
                                    interface="soundLevel">
            <bindParam name="var" value="0"/>
         </bind>
         <bind role="stop" component="musics"/>
         <bind role="start" component="musics"/>
      </link>
   </context>
   <link id="lMusic" xconnector="conEx#onBeginStart_delay">
      <bind role="onBegin" component="animation"/>
      <bind role="start" component="background">
         <bindParam name="delay" value="5s"/>
      </bind>
      <bind role="start" component="menu">
         <bindParam name="delay" value="5s"/>
      </bind>
   </link>
   <link id="lDrible" xconnector="conEx#onBeginStart">
      <bind role="onBegin" component="animation"
                                    interface="segDrible"/>
      <bind role="start" component="drible"/>
   </link>
   <link id="lPhoto"
         xconnector="conEx#onBeginStartSet_var_delay_duration">
      <bind role="onBegin" component="animation"
                                          interface="segPhoto"/>
      <bind role="start" component="photo"/>
      <bind role="set" component="photo" interface="top">
         <bindParam name="var" value="290"/>
         <bindParam name="delay" value="1s"/>
         <bindParam name="duration" value="3s"/>
      </bind>
   </link>
   <link xconnector="conEx#onEndStop">
```

```
         <bind role="onEnd" component="animation"
                                              interface="segCred"/>
         <bind role="stop" component="menu"/>
         <bind role="stop" component="interactivity"/>
      </link>
      <link id="lEnd" xconnector="conEx#onEndStop">
         <bind role="onEnd" component="animation"/>
         <bind role="stop" component="background"/>
      </link>
   </body>
</ncl>
```

Listagem 3.50 *O Primeiro João* com navegação no menu por teclas.

A Figura 3.19 ilustra a navegação por teclas na implementação de referência do middleware Ginga-NCL. Na figura à esquerda, o foco está sob o ícone de seleção do choro. Já na figura à direita, o foco se deslocou para o ícone que, se selecionado, troca a música para techno.

Figura 3.19 Cenas da aplicação *O Primeiro João* com navegação no menu por teclas.

3.13 Acrescentando um Objeto NCLua

A última funcionalidade que agregaremos ao nosso exemplo será um nó com código Lua, bem simples. Nosso objetivo é implementar um contador que armazenará quantas vezes o usuário de nosso exemplo *O Primeiro João* trocou de ritmo musical. Em outras palavras, cada vez que um novo ritmo é escolhido, o contador é incrementado. No final da apresentação do vídeo da animação, o resultado será apresentado na tela: "Número de vezes que você trocou de ritmo: *x*".

Para realizar essa nova computação, vamos criar um objeto NCLua (elemento <media> do tipo "application/x-ncl-NCLua), que identificaremos

por “changes”, contendo um script Lua que, ao ser iniciado, colocará sua variável interna *counter* com o valor “0”. Vamos também definir uma propriedade para esse objeto de mídia, denominada “add” (*name*=“add”) e uma âncora de conteúdo identificada como “print” (*id*=“print”), definindo uma interface identificada como “fim” (*label*=“fim”). Cada vez que um valor numérico for atribuído à propriedade “add”, por meio de um elemento <link>, o objeto NCLua deverá somar esse valor à variável *counter*. Se o objeto NCLua tiver sua interface “print” acionada, ele deverá imprimir na tela o valor da variável *conter*, compondo a frase “Número de vezes que você trocou de ritmo: *valor da variável counter*”.

A especificação do objeto de mídia NCLua é dada pelo elemento <media> “changes” na Listagem 3.51, e tem seu conteúdo armazenado no arquivo “counter.lua”. O leitor deve notar que o tipo do objeto (atributo *type*) não precisou ser declarado, porque o formatador NCL vai inferí-lo a partir da extensão “.lua” do arquivo imposto ao atributo *source* do elemento.

Na Listagem 3.51, nós definimos o elemento <media> “changes” como filho do elemento <context> “menu”, onde estão também todos os ícones para seleção dos ritmos. A partir de agora, quando o elemento <context> “menu” for iniciado, não só todos os ícones correspondentes aos ritmos serão exibidos juntamente com o áudio do chorinho, como também o objeto NCLua será iniciado, conforme especificado por todas as portas do elemento <contex> “menu” que estão mapeadas para âncoras de nós filhos do contexto.

```
<context id="menu">
        <port id="pChoro" component="choro"/>
        <port id="pChorinho" component="imgChorinho"/>
        <port id="pRock" component="imgRock"/>
        <port id="pTechno" component="imgTechno"/>
        <port id="pCartoon" component="imgCartoon"/>
        <port id="pNCLua" component="changes"/>
...
        <media id="changes" src="../script/counter.lua"
                                    descriptor="changesDesc">
            <area id="print" label="fim"/>
            <property name="add"/>
        </media>
...
</context>
```

Listagem 3.51 Objeto de mídia NCLua.

O resultado da computação do script será apresentado em uma região da tela referenciada no elemento <descriptor> e definida no elemento <region>, conforme ilustra a Listagem 3.52.

```
<regionBase>
   <region id="backgroundReg" width="100%" height="100%" zIndex="1">
...
       <region id="changesReg" left="0%" top="90%" width="100%"
                                    height="10%" zIndex="4"/>
   </region>
</regionBase>
<descriptorBase>
...
   <descriptor id="changesDesc" region="changesReg"/>
</descriptorBase>
```

Listagem 3.52 Definição da região de apresentação do objeto NCLua.

Para incrementarmos a variável *counter* do script Lua, vamos, a cada vez que um ritmo é trocado, atribuir o valor "1" à propriedade "add" do objeto NCLua. Faremos isso acrescentando um elemento <bind> ao elo de seleção dos ritmos "rock", "techno" e "cartoon", e outro ao elo de seleção do ícone chorinho. Esses elos, pertencentes ao elemento <context> "menu", ficam como ilustrado na Listagem 3.53.

```
<context id="menu">
...
       <link id="lChoro" xconnector="conEx#onSelectionSet_varStop">
          <bind role="onSelection" component="imgChorinho"/>
          <bind role="set" component="choro" interface="soundLevel">
               <bindParam name="var" value="1"/>
          </bind>
          <bind role="set" component="changes" interface="add">
               <bindParam name="var" value="1"/>
          </bind>
          <bind role="stop" component="musics"/>
       </link>
       <link id="lOthers"
               xconnector="conEx#onSelection_orSet_varStopStart">
          <bind role="onSelection" component="imgRock"/>
```

```
        <bind role="onSelection" component="imgTechno"/>
        <bind role="onSelection" component="imgCartoon"/>
        <bind role="set" component="choro" interface="soundLevel">
            <bindParam name="var" value="0"/>
        </bind>
        <bind role="set" component="changes" interface="add">
            <bindParam name="var" value="1"/>
        </bind>
        <bind role="stop" component="musics"/>
        <bind role="start" component="musics"/>
    </link>
</context>
```

Listagem 3.53 Elos para incremento da variável counter.

Como já mencionamos, o objeto NCLua será iniciado quando o elemento <context> "menu" for iniciado. Seu término se dá pelo término do mesmo contexto, como anteriormente, mas que agora também inclui o objeto NCLua. Para chamar o tratador do objeto NCLua que imprime o resultado, vamos incluir o objeto "animation" dentro do contexto "menu", com o nome "newAnimation", reusando toda a especificação definida anteriormente e incluindo uma nova âncora ("segLua"), cujo término ocorre 3 segundo antes de se iniciarem os créditos ("61s"). Vamos usar essa nova interface para acionar a impressão do resultado, com a introdução do novo elo apresentado na Listagem 3.54.

```
<media id="newAnimation" refer="animation"
                                        instance="instSame">
    <area id="segLua" end="61s"/>
</media>

...

<link xconnector="conEx#onEndStart">
    <bind role="onEnd" component="newAnimation" interface="segLua"/>
    <bind role="start" component="changes" interface="print"/>
</link>
```

Listagem 3.54 Elo para exibição do resultado final das mudanças de ritmo.

A Figura 3.20 apresenta a visão estrutural dessa versão final do exemplo *O Primeiro João*.

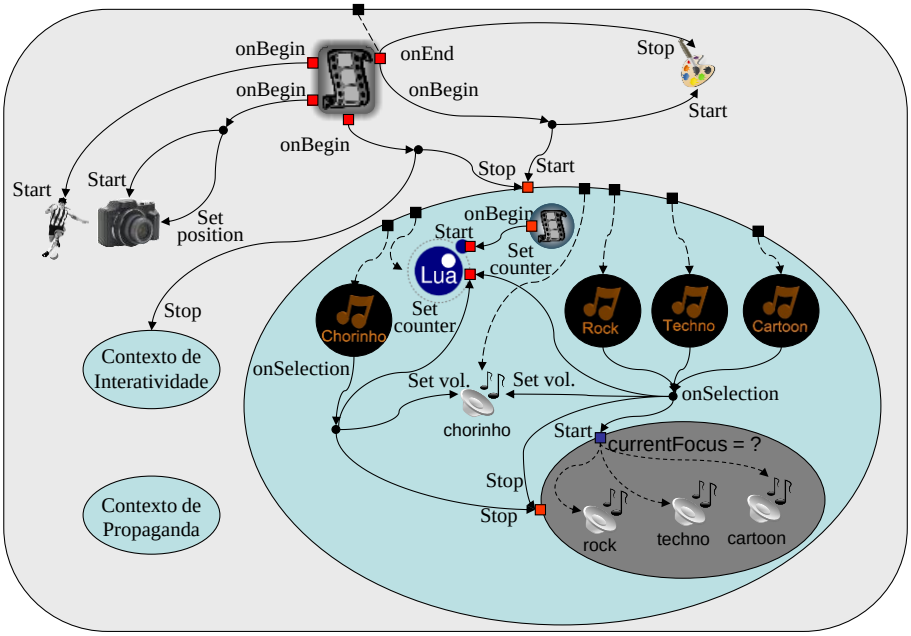


Figura 3.20 Visão estrutural da versão final do exemplo *O Primeiro João*.

Vamos agora ver o conteúdo do objeto NCLua. Ao iniciarmos um objeto com código Lua, não importa por qual de suas âncoras, ele primeiro executa uma rotina de iniciação, quando são cadastradas todas as funções tratadoras de eventos disparados interna ou externamente ao objeto. No caso de nosso exemplo, queremos cadastrar a função que tratará eventos que mudam o valor da propriedade "add" do elemento <media> "changes", e a função que tratará o evento que imprime o resultado, chamada a partir do elemento <area id="print" ...\> do mesmo elemento <media> "changes". A Listagem 3.55 exibe o script Lua de nosso exemplo.

Na Listagem 3.55, a linha 1 inicia o contador (variável local *counter*) em zero. A linha 2 define as variáveis cujos valores conterão a largura e a altura do *canvas* onde será apresentado o resultado. Esses valores são obtidos pela função `canvas:attrSize()`, que os obtém do elemento <descriptor> do objeto NCLua: no nosso caso (veja a Listagem 3.52) "100%" e "10%" da tela de exibição, respectivamente. Da linha 3 à linha 14, temos a definição da função que tratará do evento de atribuição de valores à propriedade "add" do objeto NCLua. A linha 29 registra essa função, ou seja, deixa o objeto pronto para receber eventos que dispararão a função tratadora. Da linha 15 à linha 28, temos a definição da função que tratará do evento de apresentação da interface "print" do objeto NCLua. A linha 30 registra essa função. Todo o procedimento descrito neste parágrafo é executado quando o objeto NCLua é iniciado (instanciado) pela ação "start" sobre o elemento <context> "menu".

```
1  local counter = 0
2  local dx, dy = canvas:attrSize()         -- dimensoes do canvas

3  function handler1 (evt)
4      if evt.class=='ncl' and evt.type=='attribution' and
                    evt.action=='start' and evt.name=='add' then
5          counter = counter + evt.value

6          event.post {
7             class   = 'ncl',
8             type    = 'attribution',
9             name    = 'add',
10            action  = 'stop',
11            value   = counter,
12         }
13     end
14 end

15 function handler2 (evt)
16         canvas:attrColor ('black')
17         canvas:drawRect('fill',0,0,dx,dy)
18         canvas:attrColor ('yellow')
19         canvas:attrFont ('vera', 24, 'bold')
20         canvas:drawText (10,10, 'O número de vezes que você trocou
                                        de ritmo foi: '..counter)
21         canvas:flush()

22         event.post {
23            class   = 'ncl',
24            type    = 'presentation',
25            label   = 'fim',
26            action  = 'stop',
27         }
28 end

29 event.register(handler1)
30 event.register(handler2,'ncl','presentation','fim','start')
```

Listagem 3.55 Script Lua do elemento <media> “changes”.

Como indicado na linha 4 da Listagem 3.55, a função tratadora de evento é chamada toda vez que acontece o início (note bem que é o início, pois retornaremos a esse ponto logo à frente neste texto) de atribuição ao elemento

<propriedade> cujo atributo *name*="add", e essa ação de atribuição parte de um elo do documento NCL (evt.class= = 'ncl').

As linhas 5 indica que o valor da variável *counter* deve ser adicionado do novo valor atribuído. Como, no nosso caso, a atribuição é sempre "1" (veja a Listagem 3.53), o valor da variável *counter* é incrementado.

A ação de "start" sobre a propriedade "add", proveniente de um elo do documento NCL, apenas inicia a atribuição de um valor a essa propriedade. Independentemente do valor que for atribuído, o final da atribuição deve ser sinalizada pelo objeto NCLua, tão logo ele termine a realização das tarefas chamadas pelo ínicio do evento de atribuição. Isso é realizado pelas linhas 6 a 12, na Listagem 3.55. Essas linhas comandam a geração de um evento (`event.post`) de fim de atribuição na propriedade "add", deixando como valor final dessa propriedade o valor da variável local *counter*. Note, assim, que a atribuição começa atribuindo o valor "1" (que pode ser visto como parâmetro de entrada para a função tratadora) e termina com o valor da variável *counter* (que pode ser visto como parâmetro de saída da função tratadora).

As linhas 15 a 28 indicam o que acontece quando a interface "print" tem sua apresentação iniciada. Nesse caso, queremos imprimir na tela o valor final da variável *counter*. Para tanto, as linhas 16 e 17 comandam o preenchimento de todo o canvas com a cor preta. As linhas 18 e 19 estabelecem a cor e a fonte para o texto da mensagem. A linha 20 comanda a impressão da mensagem "Número de vezes que você trocou de ritmo:", concatenado com o valor da variável *counter*. A linha 21 comanda a atualização (refrescamento) do canvas.

De forma análoga ao evento de atribuição sobre a propriedade "add", ação de "start" sobre a interface "print", proveniente de um elo do documento NCL, apenas inicia a apresentação. O final da apresentação deve ser sinalizado pelo objeto NCLua, tão logo ele termine a realização da tarefa. Isso é realizado pelas linhas 22 a 27, na Listagem 3.55. Essas linhas comandam a geração de um evento (`event.post`) de fim de apresentação da âncora "print".

Note como os tratadores foram registrados nas linhas 29 e 30 da Listagem 3.55. O primeiro tratador deve receber qualquer evento. A filtragem é feita no seu código (linha 4 na listagem). Já para o segundo tratador, a filtragem é feita pelo exibidor Lua antes de chamar o tratador, como mostra a linha 30 da listagem.

A especificação completa da nova versão do programa NCL é ilustrada na Listagem 3.56.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Exemplo de uso de objeto NCLua -->
<ncl id="nclua" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
   <head>
      <ruleBase>
         <rule id="en" var="system.language" value="en"
                                                    comparator="eq"/>
         <rule id="int" var="service.interactivity" value="true"
                                                    comparator="eq"/>
         <rule id="rRock" var="service.currentFocus" value="3"
                                                    comparator="eq"/>
         <rule id="rTechno" var="service.currentFocus" value="4"
                                                    comparator="eq"/>
         <rule id="rCartoon" var="service.currentFocus" value="5"
                                                    comparator="eq"/>
      </ruleBase>
      <transitionBase>
         <transition id="trans1" type="fade" dur="2s"/>
         <transition id="trans2" type="barWipe" dur="1s"/>
      </transitionBase>
      <regionBase>
         <region id="backgroundReg" width="100%" height="100%"
                                                    zIndex="1">
           <region id="screenReg" width="100%" height="88%"
                                                    zIndex="2"/>
           <region id="frameReg" left="5%" top="6.7%" width="18.5%"
                                    height="18.5%" zIndex="3"/>
           <region id="iconReg" left="87.5%" top="11.7%" width="8.45%"
                                    height="6.7%" zIndex="3"/>
           <region id="shoesReg" left="15%" top="60%" width="25%"
                                    height="25%" zIndex="3"/>
           <region id="formReg" left="56.25%" top="8.33%"
                       width="38.75%" height="71.7%" zIndex="3"/>
           <region id="intReg" left="92.5%" top="91.7%" width="5.07%"
                                    height="6.51%" zIndex="3"/>
           <region id="chorinhoReg" left="2.5%" top="91.7%"
                       width="11.7%" height="6.51%" zIndex="3"/>
           <region id="rockReg" left="25%" top="91.7%" width="11.7%"
                                    height="6.51%" zIndex="3"/>
           <region id="technoReg" left="47.5%" top="91.7%"
                       width="11.7%" height="6.51%" zIndex="3"/>
           <region id="cartoonReg" left="70%" top="91.7%" width="11.7%"
                                    height="6.51%" zIndex="3"/>
           <region id="changesReg" left="0%" top="90%" width="100%"
```

```
                                    height="10%" zIndex="4"/>
      </region>
   </regionBase>
   <descriptorBase>
      <descriptor id="backgroundDesc" region="backgroundReg"/>
      <descriptor id="screenDesc" region="screenReg"/>
      <descriptor id="photoDesc" region="frameReg" explicitDur="5s">
        <descriptorParam name="transparency" value="0.6"/>
      </descriptor>
      <descriptor id="audioDesc"/>
      <descriptor id="dribleDesc" region="frameReg" transIn="trans1"
                                          transOut="trans2"/>
      <descriptor id="iconDesc" region="iconReg" explicitDur="6s"/>
      <descriptor id="shoesDesc" region="shoesReg"/>
      <descriptor id="formDesc" region="formReg" focusIndex="1"
                                          explicitDur="15s"/>
      <descriptor id="intDesc" region="intReg"/>
      <descriptor id="chorinhoDesc" region="chorinhoReg"
                   focusIndex="2" moveRight="3" moveLeft="5"/>
      <descriptor id="rockDesc" region="rockReg" focusIndex="3"
                                    moveRight="4" moveLeft="2"/>
      <descriptor id="technoDesc" region="technoReg" focusIndex="4"
                                    moveRight="5" moveLeft="3"/>
      <descriptor id="cartoonDesc" region="cartoonReg"
                   focusIndex="5" moveRight="2" moveLeft="4"/>
      <descriptor id="changesDesc" region="changesReg"/>
   </descriptorBase>
   <connectorBase>
      <importBase documentURI="../causalConnBase.ncl"
                                                  alias="conEx"/>
   </connectorBase>
</head>

<body>
   <port id="entry" component="animation"/>
   <media id="background" src="../media/background.png"
                                    descriptor="backgroundDesc"/>
   <media id="animation" src="../media/animGar.mp4"
                                    descriptor="screenDesc">
      <area id="segDrible" begin="12s"/>
      <area id="segPhoto" begin="41s"/>
      <area id="segIcon" begin="45s" end="51s"/>
      <area id="segCred" end="62s"/>
   </media>
```

```
<media id="drible" src="../media/drible.mp4"
                                descriptor="dribleDesc"/>
<media id="photo" src="../media/photo.png"
                                descriptor="photoDesc">
   <property name="top"/>
</media>
<context id="interactivity">
   <media id="globalVar" type="application/x-ncl-settings">
     <property name="service.interactivity" value="true"/>
     <property name="service.currentFocus"/>
   </media>
   <media id="anotherAnimation" refer="animation"
                                       instance="instSame"/>
   <media id="intOn" src="../media/intOn.png"
                                       descriptor="intDesc"/>
   <media id="intOff" src="../media/intOff.png"
                                       descriptor="intDesc"/>
   <link id="lInt" xconnector="conEx#onBeginSet_varStart">
     <bind role="onBegin" component="anotherAnimation"/>
     <bind role="start" component="intOn"/>
     <bind role="set" component="globalVar"
                         interface="service.interactivity">
         <bindParam name="var" value="true"/>
     </bind>
   </link>
   <link id="lOn"
         xconnector="conEx#onKeySelectionStopSet_varStart">
     <bind role="onSelection" component="intOn">
         <bindParam name="keyCode" value="INFO"/>
     </bind>
     <bind role="start" component="intOff"/>
     <bind role="stop" component="intOn"/>
     <bind role="set" component="globalVar"
                         interface="service.interactivity">
         <bindParam name="var" value="false"/>
     </bind>
   </link>
   <link id="lOff"
         xconnector="conEx#onKeySelectionStopSet_varStart">
     <bind role="onSelection" component="intOff">
         <bindParam name="keyCode" value="INFO"/>
     </bind>
     <bind role="start" component="intOn"/>
     <bind role="stop" component="intOff"/>
```

```
        <bind role="set" component="globalVar"
                              interface="service.interactivity">
            <bindParam name="var" value="true"/>
        </bind>
      </link>
    </context>
    <context id="advert">
      <media id="reusedAnimation" refer="animation"
                                              instance="instSame">
        <property name="bounds"/>
      </media>
      <media id="reusedGlobalVar" refer="globalVar"
                                              instance="instSame"/>
      <media id="icon" src="../media/icon.png"
                                              descriptor="iconDesc"/>
      <media id="shoes" src="../media/shoes.mp4"
                                              descriptor="shoesDesc"/>
      <switch id="form">
        <switchPort id="spForm">
            <mapping component="enForm"/>
            <mapping component="ptForm"/>
        </switchPort>
        <bindRule constituent="enForm" rule="en"/>
        <defaultComponent component="ptForm"/>
        <media id="ptForm" src="../media/ptForm.htm"
                              type="text/html" descriptor="formDesc"/>
        <media id="enForm" src="../media/enForm.htm"
                              type="text/html" descriptor="formDesc"/>
      </switch>
      <link  id="lIcon" xconnector="conEx#onBeginVarStart">
        <bind role="onBegin" component="reusedAnimation"
                                              interface="segIcon"/>
        <bind role="var" component="reusedGlobalVar"
                              interface="service.interactivity"/>
        <bind role="start" component="icon"/>
      </link>
      <link  id="lBeginShoes"
            xconnector="conEx#onKeySelectionStopSet_varStart">
        <bind role="onSelection" component="icon">
            <bindParam name="keyCode" value="RED"/>
        </bind>
        <bind role="start" component="shoes"/>
        <bind role="start" component="form" interface="spForm"/>
        <bind role="set" component="reusedAnimation"
```

```
                                        interface="bounds">
            <bindParam name="var" value="5%,6.67%,45%,45%"/>
        </bind>
        <bind role="stop" component="icon"/>
      </link>
      <link id="lFormFocus" xconnector="conEx#onBeginSet_var">
        <bind role="onBegin" component="form"/>
        <bind role="set" component="reusedGlobalVar"
                                    interface="service.currentFocus">
            <bindParam name="var" value="1"/>
        </bind>
      </link>
      <link id="lEndForm" xconnector="conEx#onEndSet_var">
        <bind role="onEnd" component="form" interface="spForm"/>
        <bind role="set" component="reusedAnimation"
                                        interface="bounds">
            <bindParam name="var" value="0,0,222.22%,222.22%"/>
        </bind>
      </link>
    </context>
    <context id="menu">
      <port id="pChoro" component="choro"/>
      <port id="pChorinho" component="imgChorinho"/>
      <port id="pRock" component="imgRock"/>
      <port id="pTechno" component="imgTechno"/>
      <port id="pCartoon" component="imgCartoon"/>
      <port id="pNCLua" component="changes"/>
      <media id="changes" src="../script/counter.lua"
                                    descriptor="changesDesc">
        <area id="print" label="fim"/>
        <property name="add"/>
      </media>
      <media id="imgChorinho" src="../media/chorinho.png"
                                    descriptor="chorinhoDesc"/>
      <media id="imgRock" src="../media/rock.png"
                                    descriptor="rockDesc"/>
      <media id="imgTechno" src="../media/techno.png"
                                    descriptor="technoDesc"/>
      <media id="imgCartoon" src="../media/cartoon.png"
                                    descriptor="cartoonDesc"/>
      <media id="choro" src="../media/choro.mp3"
                                    descriptor="audioDesc">
        <property name="soundLevel" value="1"/>
      </media>
```

```
        <media id="newAnimation" refer="animation"
                                              instance="instSame">
          <area id="segLua" end="61s"/>
        </media>
        <switch id="musics">
          <bindRule constituent="rock" rule="rRock"/>
          <bindRule constituent="techno" rule="rTechno"/>
          <bindRule constituent="cartoon" rule="rCartoon"/>
          <media id="rock" src="../media/rock.mp3"/>
          <media id="techno" src="../media/techno.mp3"/>
          <media id="cartoon" src="../media/cartoon.mp3"/>
        </switch>
        <link id="lChoro" xconnector="conEx#onSelectionSet_varStop">
          <bind role="onSelection" component="imgChorinho"/>
          <bind role="set" component="choro" interface="soundLevel">
              <bindParam name="var" value="1"/>
          </bind>
          <bind role="set" component="changes" interface="add">
              <bindParam name="var" value="1"/>
          </bind>
          <bind role="stop" component="musics"/>
        </link>
        <link id="lOthers"
              xconnector="conEx#onSelection_orSet_varStopStart">
          <bind role="onSelection" component="imgRock"/>
          <bind role="onSelection" component="imgTechno"/>
          <bind role="onSelection" component="imgCartoon"/>
          <bind role="set" component="choro" interface="soundLevel">
              <bindParam name="var" value="0"/>
          </bind>
          <bind role="set" component="changes" interface="add">
              <bindParam name="var" value="1"/>
          </bind>
          <bind role="stop" component="musics"/>
          <bind role="start" component="musics"/>
        </link>
        <link xconnector="conEx#onEndStart">
          <bind role="onEnd" component="newAnimation"
                                              interface="segLua"/>
          <bind role="start" component="changes" interface="print"/>
        </link>
      </context>
      <link  id="lMusic" xconnector="conEx#onBeginStart_delay">
        <bind role="onBegin" component="animation"/>
```

```
         <bind role="start" component="background">
           <bindParam name="delay" value="5s"/>
         </bind>
         <bind role="start" component="menu">
           <bindParam name="delay" value="5s"/>
         </bind>
      </link>
      <link  id="lDrible" xconnector="conEx#onBeginStart">
         <bind role="onBegin" component="animation"
                                              interface="segDrible"/>
         <bind role="start" component="drible"/>
      </link>
      <link  id="lPhoto"
               xconnector="conEx#onBeginStartSet_var_delay_duration">
         <bind role="onBegin" component="animation"
                                              interface="segPhoto"/>
         <bind role="start" component="photo"/>
         <bind role="set" component="photo" interface="top">
           <bindParam name="var" value="290"/>
           <bindParam name="delay" value="1s"/>
           <bindParam name="duration" value="3s"/>
         </bind>
      </link>
      <link xconnector="conEx#onEndStop">
         <bind role="onEnd" component="animation"
                                              interface="segCred"/>
         <bind role="stop" component="menu"/>
         <bind role="stop" component="interactivity"/>
      </link>
      <link  id="lEnd" xconnector="conEx#onEndStop">
         <bind role="onEnd" component="animation"/>
         <bind role="stop" component="background"/>
      </link>
   </body>
</ncl>
```

Listagem 3.56 *O Primeiro João* com objeto NCLua.

A Figura 3.21 ilustra o uso de objetos NCLua na implementação de referência do middleware Ginga-NCL. Note a mensagem sobre quantas vezes o ritmo foi trocado.

Figura 3.21 Cenas da aplicação *O Primeiro João* com objeto NCLua.

Bibliografia

ABNT NBR 15606-2 (2011). Associação Brasileira de Normas Técnicas, "Televisão digital terrestre — codificação de dados e especificações de transmissão para radiodifusão digital, Parte 2: Ginga-NCL para receptores fixos e móveis — Linguagem de aplicação XML para codificação de aplicações", Sistema Brasileiro de TV Digital Terrestre, *NBR 15606-2*.

ITU-T H.761 (2011). "Nested Context Language (NCL) and Ginga-NCL for IPTV Services." Recommendation H.761, Genebra.

W3C REC-SMIL2-20051213 (2008). World Wide Web Consortium, "Synchronized Multimedia Integration Language — SMIL 2.1 Specification", *W3C Recommendation SMIL2-20051213*.

W3C REC-xml-20060816 (2006). World Wide Web Consortium, "Extensible Markup Language (XML) 1.0", W3C Recommendation *xml-20060816*.

W3C REC-xml-names-20060816 (2006). World Wide Web Consortium, "Namespaces in XML (2.ª ed.)", W3C Recommendation *xml-names 20060816*.

Soares, L.F.S. e Rodrigues, R.F. (2005). "Nested Context Model 3.0 Part 1 — NCM Core." Monografias em Ciência da Computação do Departamento de Informática, PUC-Rio, N.º 18/05. Rio de Janeiro. Maio. ISSN 0103-9741.

Capítulo 4

Perfis NCL

Todos os elementos da linguagem NCL são oferecidos no perfil completo da linguagem. No entanto, a linguagem pode ser restrita a domínios específicos (por exemplo, TV Digital), e, para esses domínios, perfis específicos da linguagem podem ser definidos.

Este capítulo introduz, de forma genérica, os vários perfis e módulos da linguagem NCL. Toda a Parte II deste livro é dedicada à apresentação do perfil EDTV (*Enhanced Digital TV*), definido para sistemas de TV digital.[1]

[1] Este capítulo se baseia em Soares *et al*. (2006). O uso do material foi gentilmente cedido pelo Departamento de Informática da PUC-Rio.

4.1 Introdução

A abordagem modular tem sido utilizada em várias linguagens XML [W3C REC-xml-20060816, 2006] recomendadas pelo W3C.

Módulos são coleções de elementos, atributos e valores de atributos XML semanticamente relacionados que representam uma unidade de funcionalidade. Os módulos são definidos em conjuntos coerentes. Essa coerência é expressa por meio da associação de um mesmo *namespace* [W3C REC-xml-names-20060816, 2006] aos elementos desses módulos.

Um *perfil de linguagem* é uma combinação de módulos. Os módulos são atômicos, isto é, não podem ser subdivididos quando incluídos em um perfil de linguagem. Além disso, a especificação de um módulo pode incluir um conjunto de requisitos para integração, com o qual os perfis de linguagem, que incluem o módulo, devem obrigatoriamente ser compatíveis.

A NCL foi especificada de forma modular, permitindo a combinação de seus módulos em perfis de linguagem. Cada perfil pode agrupar um subconjunto de módulos NCL, permitindo a criação de linguagens voltadas para as necessidades específicas dos usuários. Além disso, os módulos e perfis NCL podem ser combinados com módulos definidos em outras linguagens, permitindo a incorporação de características da NCL naquelas linguagens e vice-versa.

Normalmente, há um perfil de linguagem que incorpora quase todos os módulos associados a um único *namespace*. Esse é o caso do perfil "Linguagem NCL".

Um outro perfil da linguagem, com a mesma expressividade do perfil "Linguagem NCL", é definido contendo as facilidades mínimas de reuso da linguagem. Nesse perfil, denominado "Raw", módulos que definem elementos apenas para facilitar o reuso são evitados. É importante salientar que uma aplicação que segue o perfil "Linguagem NCL" sempre poderá ser convertida para o perfil "Raw". Usualmente, o formatador (player) para o perfil "Raw" é mais fácil de ser implementado do que aquele para o perfil "Linguagem NCL". Por outro lado, desenvolver aplicações seguindo o perfil "Raw" pode ser difícil e trabalhoso, ao contrário do perfil "Linguagem NCL" que possui entidades de mais elevado nível de abstração. Pode-se então dizer que o perfil "Raw" privilegia o desenvolvimento de formatadores NCL, enquanto o perfil "Linguagem NCL" privilegia o desenvolvimento de aplicações.

Outros perfis de linguagem podem ser especificados como subconjuntos de um perfil maior ou incorporar uma combinação de módulos associados a diferentes *namespaces*. Exemplos do primeiro caso são os perfis TVD Básico ("perfil BDTV") e TVD Avançado ("perfil EDTV") da NCL [Soares *et al.*, 2006; ABNT NBR 15606-2, 2011; ITU-T H.761, 2011]. Esses perfis foram

definidos para ajustar a linguagem às características do ambiente de televisão digital, com seus vários dispositivos de apresentação: aparelho de televisão, dispositivos móveis, dispositivos portáteis etc.

O principal objetivo da conformidade com perfis de linguagem é aumentar a interoperabilidade. Os módulos obrigatórios são definidos de forma que qualquer documento, especificado em conformidade com um perfil de linguagem, resulta em uma apresentação razoável quando apresentado em um perfil distinto daquele para o qual foi especificado. O formatador de documentos, suportando o conjunto de módulos obrigatórios, ignoraria todos os outros elementos e atributos desconhecidos.

4.2 Módulos NCL

Como mencionamos na seção anterior, módulos são coleções de elementos, atributos e valores de atributos XML semanticamente relacionados que representam uma unidade de funcionalidade.

A versão NCL 3.0 é dividida em 15 áreas funcionais, que são novamente divididas em módulos. Das 15 áreas funcionais, 14 são utilizadas para definir os perfis TVD Avançado e TVD Básico [Soares *et al.*, 2006]. As 14 áreas funcionais e seus módulos correspondentes usados nos perfis para TV digital são apresentados na Tabela 4.1.

Tabela 4.1 Áreas funcionais da NCL 3.0

Áreas Funcionais	Módulos	Elementos
Structure	Structure	ncl
		head
		body
Layout	Layout	regionBase
		region
Components	Media	media
	Context	context
Interfaces	MediaContentAnchor	area
	CompositeNodeInterface	port
	PropertyAnchor	property
	SwitchInterface	switchPort
		mapping
Presentation Specification	Descriptor	descriptor
		descriptorParam

		descriptorBase
Linking	Linking	bind
		bindParam
		linkParam
		link
Connectors	CausalConnectorFunctionality (agrupa funcionalidades dos módulos: ConnectorCausalExpression; ConnectorCommonPart; ConnectorAssessmentExpression; CausalConnector)	causalConnector
		connectorParam
		simpleCondition
		compoundCondition
		simpleAction
		compoundAction
		assessmentStatement
		attributeAssessment
		valueAssessment
		compoundStatement
	ConnectorBase	connectorBase
Presentation Control	TestRule	ruleBase
		rule
		compositeRule
	TestRuleUse	bindRule
	ContentControl	switch
		defaultComponent
	DescriptorControl	descriptorSwitch
		defaultDescriptor
Timing	Timing	
Reuse	Import	importBase
		importDocumentBase
		importNCL
	EntityReuse	
	ExtendedEntityReuse	
Navigational Key	KeyNavigation	
Animation	Animation	
Transition Effects	TransitionBase	transitionBase
	Transition	transition
Meta-Information	Metainformation	meta
		metadata

Os identificadores de *namespace* XML para o conjunto completo de módulos, elementos e atributos NCL 3.0 estão contidos no seguinte namespace: http://www.ncl.org.br/NCL3.0/.

Cada módulo NCL possui um identificador único a ele associado. Os identificadores dos módulos NCL 3.0 estão de acordo com aTabela 4.2.

Tabela 4.2 Identificadores dos módulos de NCL 3.0

Módulos	Identificadores
Animation	http://www.ncl.org.br/NCL3.0/Animation
CompositeNodeInterface	http://www.ncl.org.br/NCL3.0/CompositeNodeInterface
CausalConnector	http://www.ncl.org.br/NCL3.0/CausalConnector
CausalConnectorFunctionality	http://www.ncl.org.br/NCL3.0/CausalConnectorFunctionality
ConnectorCausalExpression	http://www.ncl.org.br/NCL3.0/ConnectorCausalExpression
ConnectorAssessmentExpression	http://www.ncl.org.br/NCL3.0/ConnectorAssessmentExpression
ConnectorBase	http://www.ncl.org.br/NCL3.0/ConnectorBase
ConnectorCommonPart	http://www.ncl.org.br/NCL3.0/ConnectorCommonPart
ContentControl	http://www.ncl.org.br/NCL3.0/ContentControl
Context	http://www.ncl.org.br/NCL3.0/Context
Descriptor	http://www.ncl.org.br/NCL3.0/Descriptor
DescriptorControl	http://www.ncl.org.br/NCL3.0/DescriptorControl
EntityReuse	http://www.ncl.org.br/NCL3.0/EntityReuse
ExtendedEntityReuse	http://www.ncl.org.br/NCL3.0/ExtendedEntityReuse
Import	http://www.ncl.org.br/NCL3.0/Import
Layout	http://www.ncl.org.br/NCL3.0/Layout
Linking	http://www.ncl.org.br/NCL3.0/Linking
Media	http://www.ncl.org.br/NCL3.0/Media
MediaContentAnchor	http://www.ncl.org.br/NCL3.0/MediaContentAnchor
KeyNavigation	http://www.ncl.org.br/NCL3.0/KeyNavigation
PropertyAnchor	http://www.ncl.org.br/NCL3.0/PropertyAnchor
Structure	http://www.ncl.org.br/NCL3.0/Structure
SwitchInterface	http://www.ncl.org.br/NCL3.0/SwitchInterface
TestRule	http://www.ncl.org.br/NCL3.0/TestRule
TestRuleUse	http://www.ncl.org.br/NCL3.0/TestRuleUse
Timing	http://www.ncl.org.br/NCL3.0/Timing
TransitionBase	http://www.ncl.org.br/NCL3.0/TransitionBase
Transition	http://www.ncl.org.br/NCL3.0/Transition
Metainformation	http://www.ncl.org.br/NCL3.0/MetaInformation

A Parte II deste livro é dedicada a uma discussão detalhada de cada área funcional e de cada um de seus módulos (seus elementos e atributos).

4.3 Perfis NCL

Como já mencionamos, cada perfil NCL pode agrupar um subconjunto de módulos NCL, permitindo a criação de linguagens de acordo com as necessidades dos usuários.

Qualquer documento em conformidade com os perfis NCL deve obrigatoriamente ter o elemento <ncl> como seu elemento-raiz.

O perfil NCL 3.0 completo, também chamado de perfil Linguagem NCL 3.0, é o "perfil completo" da linguagem NCL 3.0. Ele compreende todos os módulos NCL e fornece todas as facilidades para a autoria declarativa de documentos NCL.

O perfil NCL 3.0 Raw inclui os módulos: *Structure*, *Media*, *Context*, *MediaContentAnchor*, *CompositeNodeInterface*, *PropertyAnchor*, *Linking*, *CausalConnectorFunctionality*, *ConstraintConnectorFunctionality* (não discutido neste livro), *ConnectorBase*, *EntityReuse*, *ExtendedEntityReuse* e *Meta-Information*.

Como já mencionamos, o perfil NCL 3.0 Raw é tão expressivo quanto ao perfil completo da linguagem, mas sem os "açucares sintáticos" definidos para reúso. Embora possa ser usado na autoria de documentos, é tipicamente um perfil projetado para sintaxe de transferência. Por isso, não será discutido neste livro.

Os perfis definidos para uso em Sistemas de TV Digital [Soares *et al*., 2006] são:

NCL 3.0 DTV Avançado — EDTV

Inclui os módulos: *Structure*, *Layout*, *Media*, *Context*, *MediaContentAnchor*, *CompositeNodeInterface*, *PropertyAnchor*, *SwitchInterface*, *Descriptor*, *Linking*, *CausalConnectorFunctionality*, *ConnectorBase*, *TestRule*, *TestRuleUse*, *ContentControl*, *DescriptorControl*, *Timing*, *Import*, *EntityReuse*, *ExtendedEntityReuse*, *KeyNavigation*, *Animation*, *TransitionBase*, *Transition* e *Meta-Information*.

NCL 3.0 DTV Básico — BDTV

Inclui os módulos *Structure*, *Layout*, *Media*, *Context*, *MediaContentAnchor*, *CompositeNodeInterface*, *PropertyAnchor*, *SwitchInterface*, *Descriptor*, *Linking*, *CausalConnectorFunctionality*, *ConnectorBase*, *TestRule*, *TestRuleUse*, *ContentControl*, *DescriptorControl*, *Timing*, *Import*, *EntityReuse*, *ExtendedEntityReuse* e *KeyNavigation*.

NCL 3.0 CausalConnector

Inclui os módulos *Structure*, *CausalConnectorFunctionality* e *ConnectorBase*.

Ambos os perfis EDTV e BDTV são usados para criação de aplicações declarativas. A única diferença é que no perfil BDTV os efeitos de transição e animação não podem ser realizados de forma declarativa, e metainformações extras não podem ser incluídas.

O perfil NCL 3.0 CausalConnector permite a criação de conectores causais simples, alguns deles por nós utilizados no Capítulo 3.

Da mesma forma que seus módulos, cada perfil possui um identificador de *namespace* XML único a ele associado. Os identificadores dos perfis NCL 3.0 para TV digital devem estar de acordo com a Tabela 4.3.

Tabela 4.3 Identificadores dos perfis NCL 3.0

Perfis	Identificadores
EDTV	http://www.ncl.org.br/NCL3.0/EDTVProfile
BDTV	http://www.ncl.org.br/NCL3.0/BDTVProfile
CausalConnector	http://www.ncl.org.br/NCL3.0/CausalConnectorProfile

A nova versão da linguagem NCL (versão 4.0) inclui a possibilidade de manipulação de objetos 3D: como embutir objetos 3D em documentos NCL, como exibir objetos 2D em superfícies 3D, como controlar o comportamento de objetos 3D por meio de elementos <link> de NCL, como relacionar objetos 3D especificados em diferentes mundos, como relacionar objetos 2D e 3D etc. Além do suporte a objetos 3D, NCL 4.0 traz um melhor suporte ao uso de múltiplos dispositivos, ao uso de dispositivos de entrada multimodal e a aplicações NCL cientes de contexto. Para tanto, o uso de meta informações é também bastante aprimorado nessa nova versão da NCL.

4.3.1 Informações sobre Versões da NCL

As seguintes instruções de processamento devem ser incluídas em um documento NCL. Elas identificam documentos NCL que contenham apenas os elementos definidos na versão NCL com a qual o documento está de acordo.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="qualquer string" xmlns="http://www.ncl.org.br/NCL3.0/profileName">
```

O atributo *id* do elemento <ncl> pode receber qualquer cadeia de caracteres como valor.

O número de versão de uma especificação NCL consiste em um número principal e outro secundário, separados por um ponto. Os números são representados como uma cadeia de caracteres formada por números decimais, na qual os zeros à esquerda são suprimidos. O número de versão inicial do padrão para TV digital é 3.0.

Novas versões da NCL poderão ser publicadas, mas sempre de acordo com a seguinte política de versionamento:

- se os receptores compatíveis com versões mais antigas ainda puderem receber um documento com base na especificação revisada, com relação a correções de erro ou por motivos operacionais, a nova versão da NCL deve obrigatoriamente ser publicada com o número secundário atualizado;

- se os receptores compatíveis com versões mais antigas não puderem receber um documento baseado nas especificações revisadas, o número principal deve obrigatoriamente ser atualizado.

Uma versão específica é sempre definida sob o URI http://www.ncl.org.br/NCL3.0/profileName, onde o número da versão é escrito imediatamente após a sigla "NCL", seguido de "/" e o nome do perfil.

Bibliografia

ABNT NBR 15606-2 (2011). Associação Brasileira de Normas Técnicas, "Televisão digital terrestre — codificação de dados e especificações de transmissão para radiodifusão digital, Parte 2: Ginga-NCL para receptores fixos e móveis — linguagem de aplicação XML para codificação de aplicações", Sistema Brasileiro de TV Digital Terrestre, *NBR 15606-2*.

ITU-T H.761 (2011). "Nested Context Language (NCL) and Ginga-NCL for IPTV Services." Recommendation H.761, Genebra.

Soares, L.F.G. e Rodrigues, R.F. (2006) "Nested Context Model 3.0 Part 8 — NCL (Nested Context Language) Digital TV Profiles." Monografias em Ciência da Computação do Departamento de Informática, PUC-Rio, N.º 35/06. Rio de Janeiro. Outubro de 2006. ISSN 0103-9741.

W3C REC-SMIL2-20051213 (2008). World Wide Web Consortium, "Synchronized Multimedia Integration Language — SMIL 2.1 Specification", *W3C Recommendation SMIL2-20051213*.

W3C REC-xml-20060816 (2006). World Wide Web Consortium, "Extensible Markup Language (XML) 1.0", W3C Recommendation *xml-20060816*.

W3C REC-xml-names-20060816 (2006). World Wide Web Consortium, "Namespaces in XML (2.ª ed.)", W3C Recommendation *xml-names 20060816*.

PARTE II

Linguagem NCL
Perfil EDTV

Capítulo 5

Estrutura de Aplicações NCL

Neste capítulo, descrevemos a estrutura básica de aplicações NCL.[1] Ao final do capítulo, você será capaz de localizar, no código de uma aplicação NCL, os elementos correspondentes às diferentes características do documento, o que facilita a leitura e o entendimento de aplicações.

[1] Os elementos estruturais de documentos NCL e seus atributos são definidos no módulo **Structure** [ABNT, NBR 15606-2, 2007;ITU-T, H.761, 2011].

5.1 Introdução à Estrutura do Código NCL

Como descrito no capítulo anterior, assim como qualquer arquivo XML, toda aplicação NCL deve apresentar um cabeçalho XML como primeira linha do arquivo:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

A estrutura básica de uma aplicação NCL é formada pelo elemento <ncl>, e por seus elementos filhos <head> (cabeçalho) e <body> (corpo). O elemento <ncl> possui os atributos *id* e *xmlns*, que identificam a aplicação e o perfil de linguagem utilizado, respectivamente, conforme o seguinte formato:

```
<ncl id="qualquer_cadeia_de_caracteres"
                    xmlns="http://www.ncl.org.br/NCL3.0/nomePerfil">
```

O atributo *id* de um elemento <ncl> é obrigatório e pode ter como valor qualquer cadeia de caracteres que comece com uma letra ou sublinhado ("_") e que contenha apenas letras, dígitos, "." e "_".[2]

O nome do perfil, no caminho do URI do namespace, também é obrigatório e deve ser "EDTVProfile" ou "BDTVProfile", para indicar o perfil Enhanced DTV ou Basic DTV, respectivamente.[3]

O elemento <head> contém bases de elementos referenciados pelo corpo da aplicação NCL (definido no elemento <body>), como as regiões, os descritores, as transições, os conectores e as regras. Também é no elemento <head> que se definem os documentos que podem ser reutilizados pelo documento atual, bem como os metadados que auxiliam na descrição do documento como um todo.

O elemento <body> contém os elementos que definem o conteúdo da aplicação propriamente dita, tais como objetos de mídia, elos, contextos e objetos switch. Os elementos, atributos e conteúdos que definem a estrutura de documentos NCL no perfil EDTV estão sumarizados na Tabela 5.1.[4]

---

[2] Na verdade, o valor do atributo *id* de qualquer elemento da NCL deve seguir essa mesma regra de formação.

[3] Como veremos no Capítulo 10, o perfil CausalConnectorProfile é também usado, mas para a definição de uma base de relações e não na definição de uma aplicação.

[4] Como de praxe, ao definir um elemento filho utilizaremos a seguinte convenção: uma "interrogação" indica que o elemento é opcional (pode não existir ou ter uma ocorrência), um "asterisco" indica que o elemento pode ocorrer zero ou mais vezes, e um sinal de "mais" indica que o elemento deve ocorrer pelo menos uma vez, mas também pode ocorrer várias vezes. Os atributos de um elemento que têm sua declaração *obrigatória* são sublinhados, ao contrário dos demais.

Tabela 5.1 Elementos, Atributos e Conteúdo (Elementos Filhos) que Definem a Estrutura de Documentos NCL no Perfil EDTV

Elementos	Atributos	Conteúdo
ncl	*id, title, xmlns*	(head?, body?)
head		(importedDocumentBase?, ruleBase?, transitionBase?, regionBase*, descriptorBase?, connectorBase?, meta*, metadata*)
body	*id*	(port | property | media | context | switch | link | meta | metadata)*

É recomendado que os elementos filhos do elemento <head> sejam declarados na ordem indicada na tabela e ilustrada pela Listagem 5.1. Já os elementos do <body> podem ser definidos em qualquer ordem.

```
<head>
   <importedDocumentBase>
   <!-- apliações NCL importadas e referenciadas por esta -->
   </importedDocumentBase>
   <ruleBase>
   <!-- regras para adaptar o conteúdo da aplicação e sua forma de
apresentação -->
   </ruleBase>
   <transitionBase>
   <!-- efeitos de transição para apresentação dos objetos de mídia -->
   </transitionBase>
   <regionBase >
   <!-- áreas de exibição destinadas aos objetos de mídia -->
   </regionBase>
   <descriptorBase>
   <!-- configurações de apresentação dos objetos de mídias -->
   </descriptorBase>
   <connectorBase>
   <!-- comportamento dos elos de relacionamento entre objetos -->
   </connectorBase>
   <meta/>   <!-- dados descritivos simples -->
   <metadata>
   <!-- dados descritivos estruturados -->
   </metadata>
</head>
```

Listagem 5.1 Estrutura do elemento <head>, indicando a ordem recomendada para seus elementos filhos numa aplicação NCL.

Os elementos filhos dos elementos <head> e <body> são definidos em outros módulos e apresentados em diferentes capítulos deste livro, conforme indicado na Tabela 5.2. Os módulos são indicados aqui para facilitar a consulta a outros documentos de especificação da NCL (p. ex., ABNT, NBR, 15606-2, 2011 e ITU-T, H.761, 2011).

Tabela 5.2 Módulos que Definem os Elementos da NCL no Perfil EDTV, Filhos dos Elementos <head> e <body>, e os Respectivos Capítulos que os Descrevem

Pai	Elemento (Lista parcial)	Módulo	Capítulo ou seção
head	importedDocumentBase	Import	13
	ruleBase	TestRule	11.1
	transitionBase	TransitionBase	7.3
	regionBase region	Layout	6
	descriptorBase descriptor	Descriptor	7
	descriptorSwitch	DescriptorControl	11.3
	connectorBase	ConnectorBase (e outros)	10
	meta metadata	Metainformation	12
body	port	CompositeNodeInterface	8.3
	media	Media	8.1
	area	MediaContentAnchor	9.1
	property	PropertyAnchor	9.2
	context	Context	8.2
	switch	ContentControl	11.2
	switchPort	SwitchInterface	11.2
	link	Linking	10

ABNT NBR 15606-2 2011. Associação Brasileira de Normas Técnicas, "Televisão digital terrestre — Codificação de dados e especificações de transmissão para radiodifusão digital, Parte 2: Ginga-NCL para receptores fixos e móveis — Linguagem de aplicação XML para codificação de aplicações", Sistema Brasileiro de TV Digital Terrestre, *NBR 15606-2*.

[ITU-T H.761 2011. Nested Context Language (NCL) and Ginga-NCL for IPTV. ITU-T Rec. H.761. Genebra.

Capítulo 6

Leiaute da Apresentação: Regiões

Neste capítulo, descrevemos os elementos e atributos para definição de regiões de apresentação em classes de dispositivos de exibição.[1] Ao final deste capítulo, você será capaz de definir onde os objetos de mídia de uma aplicação NCL podem ser inicialmente apresentados.

[1] Regiões e seus atributos são definidos no módulo **Layout** [ABNT, NBR, 15606-2, 2011; ITU-T, H.761, 2011].

6.1 Introdução

A NCL permite definir onde cada objeto de mídia será apresentado, isto é, em que classe de dispositivos e em qual região de apresentação de cada classe de dispositivos.

As classes de dispositivos e suas regiões podem ser definidas por meio de propriedades dos objetos de mídia (veja Capítulo 9), por meio de elementos descritores (veja Capítulo 7), ou por meio de elementos <regionBase> e <region>, respectivamente, discutidos neste capítulo.

Para cada classe de dispositivos de saída podemos definir, no cabeçalho do documento (dentro do elemento `<head>`), uma base de regiões utilizando o elemento `<regionBase>`.

Dentro de uma base de regiões, definimos as regiões através de elementos `<region>`. Os elementos <region> definem as áreas, nos dispositivos de saída, onde os objetos de mídia poderão ser exibidos *inicialmente*.[2] Para organizar o leiaute das diversas partes do documento hipermídia, as regiões podem ser aninhadas. Em outras palavras, uma região pode ser definida com relação à área total associada ao dispositivo correspondente a uma base de regiões ou aninhada em outra região. A definição de base de regiões com diversas regiões aninhadas é ilustrada pela Listagem 6.1.

Vale relembrar que, como uma aplicação NCL é descrita em um documento XML, um elemento, que pode ou não conter outros (p. ex., `<region>`), possui duas formas de definição:

- elemento atômico, sem outros elementos aninhados:

  `<region id="rgPDAmenu1" />`

- elemento com outros elementos aninhados e com o fechamento explícito ao final do bloco:

```
<region id="rgPDAmenu" >
      <region id="rgPDAmenu1" />
      <region id="rgPDAmenu2" />
      <region id="rgPDAmenu3" />
</region>
```

[2] Como visto na Seção 10.3, durante a apresentação de um objeto de mídia, a aplicação NCL pode alterar os atributos que definem onde ele é exibido.

```
<head>
      <!-- uma base para cada dispositivo -->
      <regionBase id="rbTV"> <!-- dispositivo: TV (default) -->
         <region id="rgTVtelaInteira"> <!-- tela da TV -->
            <region id="rgTVmenu" > <!-- menu na TV -->
               <region id="rgTVmenu1" />
               <region id="rgTVmenu2" />
               <region id="rgTVmenu3" />
            </region>
         </region>
      </regionBase>
      <regionBase id="rbPDA" device="systemScreen(1)"> <!-- disp:PDA-->
         <region id="rgPDAtelaInteira"> <!-- tela do PDA -->
            <region id="rgPDAmenu" > <!-- menu no PDA -->
               <region id="rgPDAmenu1" />
               <region id="rgPDAmenu2" />
               <region id="rgPDAmenu3" />
            </region>
         </region>
      </regionBase>
      <!-- continuação do cabeçalho da aplicação -->
</head>
```

TV

PDA

Listagem 6.1 Definição das bases de regiões para dois dispositivos e suas regiões filhas, onde serão exibidas as mídias.

6.1.1 Importação de Base de Regiões

Além de elementos <region>, uma base de regiões também pode conter elementos <importBase>, para importar as regiões definidas na base de regiões de um outro documento NCL. O elemento <importBase> possui os seguintes atributos[3]:

- *alias*: "apelido" do arquivo importado, ou seja, o nome que será utilizado como prefixo para se referir aos elementos importados, no formato "apelido#id_do_elemento_importado";
- *documentURI*: a localização e o nome do arquivo que contém a base a ser importada;
- *region*: no caso de o arquivo importado conter uma base de regiões, o atributo define qual região da aplicação conterá as regiões importadas. Se

[3] Como sempre, neste capítulo, os atributos sublinhados são obrigatórios.

o atributo não for especificado, assume-se que seja toda a área de exibição do dispositivo.

A Listagem 6.2 apresenta o código correspondente à importação e uso de uma base de regiões de um documento NCL externo. As regiões definidas na base de regiões do arquivo "baseRegMenu.ncl" são importadas como filhos de <regionBase> do documento atual, ao passo que as regiões definidas na base de regiões do arquivo "baseRegMenu.ncl" são importadas como filhos da região "rgVideoAux".

```
<regionBase>
       <importBase alias="regMenu" documentURI="baseRegMenu.ncl" />
       <importBase alias="regDoc"
documentURI="baseRegDocumentario.ncl"
                              region="rgVideoAux" />

       <region id="rgTV">
              <region id="rgVideoAux" top="5%" left="5%"
                                    width="50%" height="50%" />
              ...
       </region>
</regionBase>
...
<descriptorBase>
       <descriptor id="dLegenda" region="regDoc#rgLegenda" />
       ...
</descriptorBase>
```

Listagem 6.2 Importação de uma base de regiões e uso de região importada, assumindo que no arquivo "baseRegDocumentario.ncl" haja uma região com identificador "rgLegenda".

6.1.2 Atributos de Base de Regiões

Uma base de regiões <regionBase> possui os seguintes atributos:

- *id*: identificador unívoco da base de regiões. Esse atributo não é obrigatório para o elemento <regionBase>, mas quando especificado deve seguir a mesma regra de formação para o atributo *id* definida no Capítulo 5;
- *device*: classe de dispositivos à qual os filhos do elemento <regionBase> se referem. Pode conter valores como "systemScreen(i)", "systemAudio(i)", onde *i* é um inteiro maior que zero, conforme as classes de dispositivos disponíveis.

O atributo *device* também é opcional. Se não for definido, ele assume como *default* a classe "systemScreen(0)". Nessa classe, só um dispositivo é

registrado, também por *default*: o dispositivo de exibição associado ao dispositivo onde está sendo processado o documento NCL. Neste capítulo, vamos supor que temos apenas um dispositivo de exibição e que, portanto, será declarado por *default*. Esse dispositivo pode ser uma TV ligada ao conversor digital (*set-top box*) onde a aplicação NCL está sendo executada, a tela de um computador onde a aplicação NCL está sendo processada etc. No Capítulo 15 trataremos da programação para múltiplos dispositivos de exibição.

Vale observar que, ao associar uma classe de dispositivos a uma base de regiões `<regionBase>`, o dispositivo escolhido define variáveis globais de ambiente, tais como:

- "system.screenSize(*i*)": tamanho de tela da classe de dispositivos *i*, expresso em (linhas, pixels/linha);
- "system.screenGraphicSize(*i*)": resolução do plano gráfico da classe de dispositivos *i*, expresso em (linhas, pixels/linha);
- "system.audioType(*i*)": tipo de áudio da classe de dispositivos *i*, que pode assumir um dos seguintes valores: "mono", "stereo" ou "5.1".

Essas e outras variáveis de ambiente são definidas na Seção 9.3.1.

6.1.3 Atributos de Região

Como já mencionamos, uma região serve para iniciar a posição dos objetos de mídia num local específico. Para apresentar um vídeo no centro da tela e com uma margem de 10%, por exemplo, podemos definir duas regiões: "rgTV", que ocupa toda a área disponível do dispositivo, e "rgTVcentro", para que um objeto mídia seja apresentado no centro da tela, conforme a Listagem 6.3.

```
<regionBase id="rbTV" device="systemScreen(0)">
      <region id="rgTV">
         <region id="rgTVcentro" left="10%" top="10%"
                                 width="80%" height="80%" />
      </region>
</regionBase>
```

Listagem 6.3 Exemplo de definição de região centralizada em outra.

Essa definição corresponde ao leiaute apresentado na Figura 6.1.

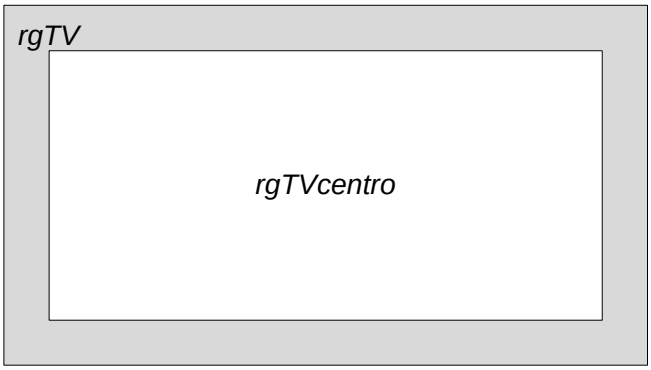


Figura 6.1 Leiaute de exemplo com uma região (“rgCentro”) centralizada sobre uma região pai (“rgTV”).

A NCL define os seguintes atributos de região:

- *id*: identificador único, utilizado nas referências à região (por exemplo, pelos descritores dos objetos de mídia que serão apresentados na região), que segue a mesma regra de formação para o atributo *id* definida no Capítulo 5;
- *left* (esquerda): coordenada *x* do lado esquerdo da região, com relação à coordenada do lado esquerdo da região pai (ou da área total associada ao dispositivo, no caso de a região não estar aninhada a nenhuma outra);
- *top* (topo): coordenada *y* do lado superior da região, com relação à coordenada do lado superior da região pai (ou da área total associada ao dispositivo, no caso de a região não estar aninhada a nenhuma outra);
- *right* (direita): coordenada *x* do lado direito da região, com relação à coordenada do lado direito da região pai (ou da área total associada ao dispositivo, no caso de a região não estar aninhada a nenhuma outra);
- *bottom* (base): coordenada *y* do lado inferior da região, com relação à coordenada do lado inferior da região pai (ou da área total associada ao dispositivo, no caso de a região não estar aninhada a nenhuma outra);
- *width* (largura) e *height* (altura): dimensões horizontal e vertical da região;
- *zIndex*: número entre 0 e 255 que define a “camada” da região ou posição da região no eixo “z”, utilizado geralmente para indicar, no

caso de regiões sobrepostas, quais regiões aparecem sobre quais outras. Camadas com *zIndex* maior são apresentadas no topo de camadas com *zIndex* menor (isto é, sobrepondo essas últimas). Quando não especificado, assume o valor *default* "0";

- *title* (título): título da região, cujo uso depende da implementação do formatador.

Todos os atributos de um elemento <region> são opcionais, exceto o atributo *id*.

Os atributos *left*, *top*, *right*, *bottom*, *width* e *height* podem ser representados em pixels (p.ex., "50px" ou simplesmente "50") ou em porcentagem (p. ex., "25%"). O percentual é sempre relativo à largura do pai, no caso das definições dos atributos *right*, *left* e *width*, e à altura do pai, para as definições dos atributos *bottom*, *top* e *height*. Uma restrição é que as regiões filhas não podem ficar fora da área estabelecida por suas regiões pais. Quando qualquer um desses atributos não for especificado e não puder ter seu valor calculado a partir de outros atributos, seu valor deve ser herdado do valor correspondente definido no pai do elemento <region> que define o atributo.

O autor pode escolher especificar as dimensões de uma região conforme sua conveniência. Por exemplo, em certos casos, pode ser melhor definir os atributos *right*, *bottom*, *width* e *height*. Em outros casos, pode ser mais apropriado especificar os atributos *top*, *left*, *width* e *height*. É importante observar que, caso todos os atributos de posicionamento e dimensão sejam especificados, os atributos *left* e *width* têm precedência sobre o atributo *right*, assim como os atributos *top* e *height* têm precedência sobre o atributo *bottom*.

Quando duas ou mais regiões possuírem o mesmo valor de *zIndex*, e em cada região for apresentada uma mídia, existem duas possibilidades: caso uma mídia seja apresentada antes da outra, a mídia que for iniciada depois vai se sobrepor à que foi iniciada antes (ordem temporal). Caso as duas mídias sejam iniciadas ao mesmo tempo, a ordem é escolhida arbitrariamente pelo formatador.

A Figura 6.2 ilustra os atributos *top*, *left*, *right*, *bottom*, *width*, *height* e *zIndex*.

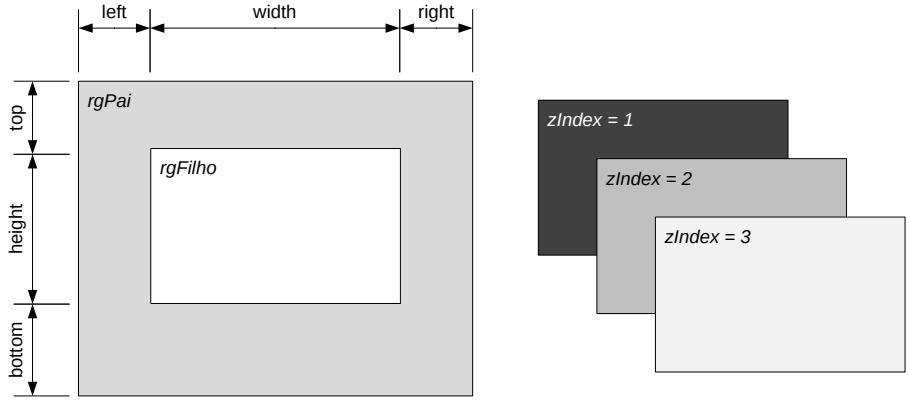


Figura 6.2 Atributos de posicionamento e dimensão de uma região.

Vale observar que uma região herda por *default* os atributos da sua região pai. Vamos considerar um menu vertical de quatro opções. Em vez de definirmos a região correspondente a cada item de menu com posição relativa à tela do dispositivo, podemos aninhá-las a uma região que simboliza toda a área de tela (*bounding box*) ocupada pelo menu (Figura 6.3).

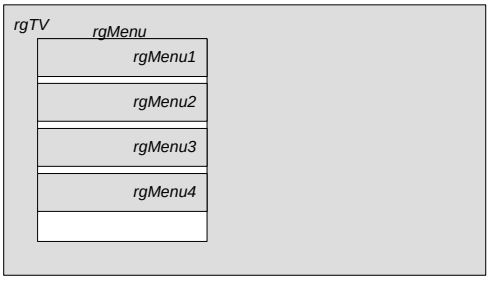


Figura 6.3 Leiaute de exemplo com diversas regiões (“rgMenu1”, “rgMenu2”, “rgMenu3”, “rgMenu4”) posicionadas de forma relativa a uma região pai (“rgMenu”).

Observamos na Listagem 6.4 que a região “rgMenu1” começa a uma distância de 50 pixels das bordas superior e esquerda da tela e possui largura de 200 pixels, pois herda os valores desses atributos da região “rgMenu”. Por outro lado, a altura de “rgMenu1” é de 50 pixels, conforme definido no atributo do próprio elemento.

```
<regionBase>
        <region id="rgTV">
                <region id="rgMenu" left="50" top="50"
                            width="200" height="240">
                        <region id="rgMenu1" height="50" />
                        <region id="rgMenu2" top="60" height="50" />
                        <region id="rgMenu3" top="120" height="50" />
                        <region id="rgMenu4" top="180" height="50" />
                </region>
        </region>
</regionBase>
```

Listagem 6.4 Definição de regiões para um menu vertical com quatro opções.

Para exemplificar a utilidade desse aninhamento de regiões, suponha que agora se decida que o menu deve ficar no lado direito da tela, em vez de no lado esquerdo. A partir da definição anterior, a única modificação que precisa ser feita é na especificação dos atributos da região "rgMenu": em vez de situar o menu a 50 pixels do lado esquerdo da região pai "rgTV" (posição definida através do atributo *left*), definimos agora o atributo *right*, situando o menu a 50 pixels do lado direito da região pai "rgTV", conforme o trecho de código apresentado na Listagem 6.5. As demais regiões permanecem inalteradas.

```
<regionBase>
        <region id="rgTV">
                <region id="rgMenu" right="50" top="50"
                            width="200" height="240">
                        <region id="rgMenu1" height="50" />
                        <region id="rgMenu2" top="60" height="50" />
                        <region id="rgMenu3" top="120" height="50" />
                        <region id="rgMenu4" top="180" height="50" />
                </region>
        </region>
</regionBase>
```

Listagem 6.5 Alteração de atributo de uma região pai.

A Figura 6.4 ilustra essa nova disposição das regiões do menu.

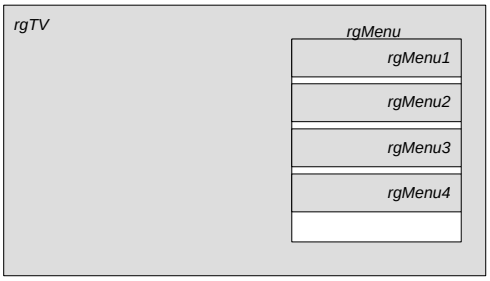


Figura 6.4 Leiaute de exemplo com diversas regiões (“rgMenu1”, “rgMenu2”, “rgMenu3”, “rgMenu4”) posicionadas de forma relativa a uma região pai (“rgMenu”), próximas ao lado direito da região “*rgTV*”.

Exemplo 6.1. — Reproduzindo um Vídeo em Tela Inteira

Como exemplo, esta seção apresenta uma aplicação NCL simples, que apenas reproduz um vídeo em tela inteira. Apesar de não se tratar de um documento hipermídia típico (pois não há elos ligando objetos), o exemplo tem por objetivo apenas explorar alguns atributos de região e também relembrar como as regiões são associadas através de elementos <descritores> a elementos <media>, sem entrar em detalhes sobre esses elementos, que são assuntos de outros capítulos.

A Figura 6.5 apresenta a visão estrutural desse exemplo. Observamos que o contexto <body> contém apenas um nó de mídia, “videoPrincipal”, mapeado pela porta “pVideoPrincipal”.

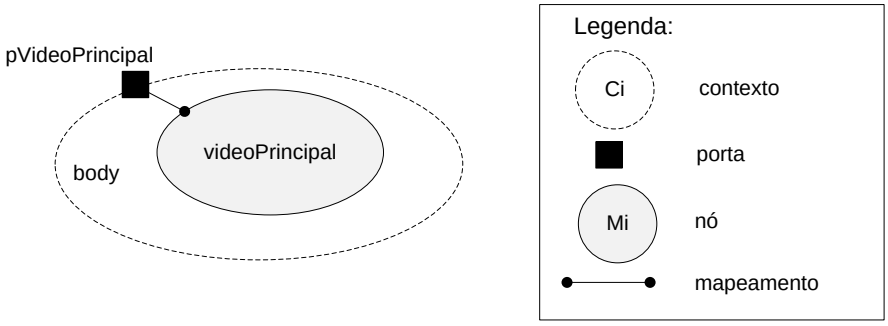


Figura 6.5 Visão estrutural do exemplo para reprodução de um único vídeo, sem sincronismo ou interação com o usuário.

A Figura 6.6 ilustra a visão de leiaute, com a região “rgTVtelaInteira” ocupando a tela inteira do dispositivo, representada aqui pela região “rgTV”.

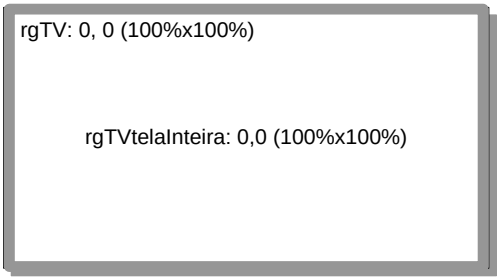


Figura 6.6 Visão de leiaute do exemplo.

A Figura 6.7 ilustra as visões temporal e espacial do documento. A visão temporal é bem simples: envolve apenas a exibição integral da mídia “videoPrincipal”. A visão espacial reflete quais mídias são exibidas para o espectador num determinado instante de tempo. Nesse exemplo em especial, apenas a mídia “videoPrincipal*”* é exibida durante todo o tempo de execução do documento.

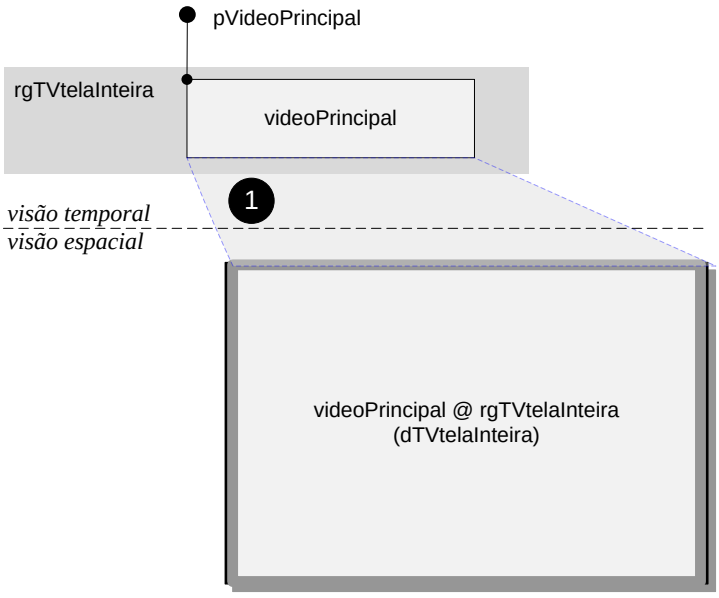


Figura 6.7 Visões temporal e espacial do exemplo.

Passo a passo

Para construir uma aplicação NCL, podemos partir do esqueleto básico apresentado na Listagem 6.6, no qual os trechos destacados indicam as lacunas que devemos preencher.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="exemplo" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
   <head>
      <regionBase>
        <!-- regiões onde as mídias serão apresentadas -->
      </regionBase>
      <descriptorBase>
       <!-- definição de como as mídias serão apresentadas -->
      </descriptorBase>
   </head>
   <body >
      <port id="nomeDaPorta" component="umNoDesteContexto" />
      <!--  contextos, nós de mídia e suas âncoras,
      elos e outros elementos -->
   </body>
</ncl>
```

Listagem 6.6 Esqueleto básico de uma aplicação NCL.

Para construir uma aplicação em NCL que reproduz um vídeo, criamos os seguintes elementos:

1. o dispositivo onde a aplicação será exibida;
2. a região da tela onde o vídeo será exibido;
3. o descritor que determina a forma como o vídeo será exibido e em que região. Nesse caso, o vídeo associado a esse descritor será exibido na região "rgTVtelaInteira", com as propriedades *default* de exibição de vídeo;
4. o nó de mídia "videoPrincipal", o vídeo propriamente dito;
5. a porta que define o ponto de entrada do documento hipermídia. Nesse caso, a porta "pVideoPrincipal", que mapeia para o único elemento de mídia, "videoPrincipal".

Observação: Esse exemplo assume que há um arquivo de vídeo chamado "principal.mpg" num subdiretório "media/" do documento atual.

Passos 1 e 2: Definindo Regiões de Tela

Como visto anteriormente, as regiões definem *onde* as mídias poderão ser apresentadas. São definidas por elementos <region> de uma base de regiões <regionBase> no cabeçalho <head> do documento. As regiões podem ser aninhadas umas nas outras, para facilitar o seu posicionamento relativo.

Nesse exemplo, temos apenas uma região que será associada à mídia de vídeo ("rgTVtelaInteira"), aninhada à região associada à tela da TV ("rgTV") e ocupando todo o seu espaço:

```
<head>
      ...
      <regionBase>
         <region id="rgTV" left="0" top="0"
                           width="100%" height="100%">
            <region id="rgTVtelaInteira" left="0" top="0"
                           width="100%" height="100%" />
         </region>
      </regionBase>
      ...
</head>
```

Vale observar que a posição *default* é *left*="0" e *top*="0", e que a largura e altura *defaults* são "100%". Sendo assim, as regiões desse exemplo podem ser definidas de forma mais concisa:

```
<head>
      ...
      <regionBase>
         <region id="rgTV">
            <region id="rgTVtelaInteira" />
         </region>
      </regionBase>
      ...
</head>
```

Lembramos ainda que os elementos da NCL (região, descritor, objeto de mídia, porta etc.) devem possuir identificadores únicos, representados pelo atributo *id*. Por exemplo, uma mídia não pode ter o mesmo *id* de uma região ou de qualquer outro elemento.

Passo 3: Definindo o Descritor que Determina como o Vídeo será Exibido

Tendo especificado as regiões, definimos agora o descritor que determinará *como* o vídeo será exibido. Nesse exemplo, o descritor é utilizado apenas para associar uma mídia a uma região. O código NCL correspondente ao descritor criado é o seguinte:

```
<head>
...
        <descriptorBase>
            <descriptor id="dTVtelaInteira"
                                                region="rgTVtelaInteira"
            />
        </descriptorBase>
...
</head>
```

Esse descritor "dTVtelaInteira" é extremamente simples, pois faz somente a associação com uma região, "rgTVtelaInteira".

Passo 4: Definindo a Mídia Propriamente Dita

Tendo definido as regiões e os descritores, o próximo passo é criar os objetos de mídia <media> que compõem o documento. Lembrando o esqueleto de uma aplicação NCL, a definição dos nós do documento é feita na seção <body>, que corresponde ao contexto do documento como um todo, conforme ilustrado pela Listagem 6.7.

```
<body>
      <media .../>
      <media .../>
      <!-- continuação do núcleo da aplicação -->
</body>
```

Listagem 6.7 Definição dos objetos de mídia no núcleo do documento.

Em sua definição mais abreviada, uma mídia deve definir, além do seu identificador, o arquivo de mídia propriamente dito e o descritor que define como a mídia será apresentada inicialmente. No exemplo, há apenas uma mídia de vídeo, identificada como "videoPrincipal", que deve ser apresentada através do descritor "dTVtelaInteira". O arquivo de vídeo que deve ser reproduzido quando esse objeto for iniciado é o arquivo "principal.mpg", localizado num subdiretório "media/" do diretório onde se encontra o arquivo NCL, conforme a Listagem 6.8.

```
<body>
      ...
      <media id="videoPrincipal" src="media/principal.mpg"
                     descriptor="dTVtelaInteira" />
      ...
</body>
```

Listagem 6.8 Definição dos objetos de mídia no núcleo do documento.

Passo 5: Definindo a Porta do Contexto <body> **que Determina o Início da aplicação (Quais Mídias Devem ser Apresentadas Inicialmente)**

Para concluir a criação desse exemplo, é necessário definir pelo menos uma porta do contexto <body> que mapeie para um nó de mídia, cuja apresentação será iniciada quando a aplicação for executada. No exemplo, o código NCL correspondente à porta do <body> é o seguinte:

```
<body>
       <port id="pVideoPrincipal" component="videoPrincipal" />
       ...
</body>
```

A Listagem 6.9 apresenta o código completo do exemplo, que reproduz o vídeo armazenado em "media/principal.mpg" em tela inteira.

```
1:  <?xml version="1.0" encoding="ISO-8859-1"?>
2:  <ncl id="exemplo" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
3:     <head>
4:        <regionBase>
5:           <region id="rgTV">
6:              <region id="rgTVtelaInteira" zIndex="1" />
7:           </region>
8:        </regionBase>
9:        <descriptorBase>
10:          <descriptor                              id="dTVtelaInteira"
                    region="rgTVtelaInteira" />
11:       </descriptorBase>
12:    </head>
13:    <body>
14:       <port id="pVideoPrincipal" component="videoPrincipal" />
15:       <media      id="videoPrincipal"      src="media/principal.mpg"
               descriptor=" dTVtelaInteira" />
16:    </body>
17: </ncl>
```

Listagem 6.9 Listagem completa do exemplo.

6.2 Referência Rápida – Regiões

A Tabela 6.1 sumariza os elementos e atributos do módulo *Layout*.

Tabela 6.1 Elementos, Atributos e Conteúdo (Elementos Filhos) Definidos pelo Módulo **Layout** do Perfil EDTV

Elementos	Atributos	Conteúdo
regionBase	*id, device*	(importBase|region)+
region	*id, left, right, top, bottom, height, width, zIndex, title*	(region)*

Bibliografia

ABNT NBR 15606-2 2011. Associação Brasileira de Normas Técnicas, "Televisão digital terrestre — Codificação de dados e especificações de transmissão para radiodifusão digital, Parte 2: Ginga-NCL para receptores fixos e móveis — Linguagem de aplicação XML para codificação de aplicações", Sistema Brasileiro de TV Digital Terrestre, *NBR 15606-2*.

ITU-T H.761 2011. Nested Context Language (NCL) and Ginga-NCL for IPTV. ITU-T Rec. H.761. Genebra.

Capítulo 7

Apresentação de Objetos de Mídia: Descritores

Neste capítulo, descrevemos elementos e atributos de descritores,[1] responsáveis pela definição dos parâmetros de exibição de uma mídia em dada região, ou seja, pela configuração de *como* os objetos de mídia deverão ser apresentados. Os descritores descrevem também como pode ser feita a navegação entre objetos de mídia através das teclas (setas) de navegação. Ainda neste capítulo, descrevemos os efeitos de transição disponíveis em NCL e como podem ser utilizados pelos descritores.

Ao final deste capítulo, você será capaz de definir como um nó é apresentado e como o usuário poderá navegar pelos objetos de mídia através das teclas (setas) de navegação.

[1] Descritores e seus atributos são definidos no módulo **Descriptor** [ABNT NBR, 15606-2 ,2011; ITU-T, H.761, 2011].

7.1 Introdução a Descritores

Os descritores (elemento <descriptor>) especificam como os objetos de mídia a eles associados serão exibidos *inicialmente*. Assim como as regiões, os descritores são definidos no cabeçalho do documento (dentro do elemento <head>), em uma base de descritores definida pelo elemento <descriptorBase>. No entanto, há duas diferenças importantes com relação à definição de regiões:

- uma aplicação NCL não pode possuir mais de uma base de descritores <descriptorBase>;
- um descritor <descriptor> não pode estar aninhado a outro descritor.

O tipo mais simples de descritor apenas faz a associação com uma região, conforme o exemplo da Listagem 7.1.

```
<head>
      <!-- ... -->
      <descriptorBase>
         <descriptor id="dTVtelaInteira"
                                    region="rgTVtelaInteira" />
      </descriptorBase>
      <!-- ... -->
</head>
```

Listagem 7.1 Definição de descritor simples que apenas faz associação com uma região.

Nesse caso, toda mídia associada ao descritor "dTVtelaInteira" será apresentada inicialmente na região "rgTVtelaInteira", com os valores *defaults* de exibição (volume de som, dimensionamento na região etc.).

Um descritor pode, no entanto, definir parâmetros adicionais de exibição de mídia, através de elementos filhos <descriptorParam>, conforme ilustra a Listagem 7.2.

```
<head>
      <!-- ... -->
      <!-- base de descritores -->
      <descriptorBase>
         <descriptor id="dTVtelaInteira" region="rgTVtelaInteira" />
         <descriptor id="dTVmenu" region="rgTVmenu">
            <descriptorParam name="transparency" value="30%" />
         </descriptor>
         <descriptor id="dTVmenu1" region="rgTVmenu1" />
         <descriptor id="dTVmenu2" region="rgTVmenu2" />
         <descriptor id="dTVmenu3" region="rgTVmenu3" />
      </descriptorBase>
      <!-- ... -->
</head>
```

Listagem 7.2 Definição de uma base de descritores.

Elementos <descriptorParam> definem valores para propriedades dos objetos de mídia que referem o descritor e pode incluir entre eles valores para os atributos que definem uma região: *left*, *right*, *width*, *right, zIndex* etc. Nesse caso, esses valores sobrescrevem aqueles definidos pelos elementos <region>.

O elemento <descriptorBase> possui apenas um atributo, *id*, que é opcional. Nesse elemento, o *id* serve para possibilitar o reúso da base de descritores em uma outra aplicação NCL, como é apresentado na Seção 13. Como todo atributo *id*, ele deve ser único por todo o documento NCL e segue a mesma regra de formação para o atributo *id* definida no Capítulo 5.

7.1.1 Importação de Bases de Descritores

Assim como ocorre com a base de regiões, uma base de descritores também pode conter, além de elementos <descriptor>, elementos <importBase> para importar os descritores definidos na base de descritores de um outro documento NCL. O elemento <importBase> possui os seguintes atributos:

- *alias*: "apelido" do arquivo importado, ou seja, o nome que será utilizado como prefixo para se referir aos elementos importados, no formato "apelido#id_do_elemento_importado";
- *documentURI*: a localização e o nome do arquivo que contém a base a ser importada;

- *region*: no caso de o arquivo importado conter uma base de regiões, define qual região da aplicação conterá as regiões importadas.

Quando a base de descritores de um documento NCL é importada, além dos descritores contidos nessa base, as regiões e as regras das bases no documento importado também são importadas para as bases correspondentes no documento NCL atual (importador).

7.1.2 Atributos Básicos de Descritor

A NCL define os seguintes atributos de descritor:

- *id:* identificador único, utilizado nas referências ao descritor (por exemplo, nos objetos de mídia apresentados pelo descritor). Esse é o único atributo obrigatório de um elemento <descriptor>, e segue a mesma regra de formação para o atributo *id* definida no Capítulo 5;

- *region:* identificador da região associada ao descritor. Todo objeto de mídia que utilize esse descritor será exibido inicialmente nessa região. Esse atributo só faz sentido para objetos que possuem um conteúdo visual a ser exibido;

- *explicitDur*:[2] define a duração do objeto de mídia associado ao descritor. O valor desse atributo pode ser expresso em segundos, no formato "9.9s" (valor numérico inteiro ou real seguido da letra "s"). O valor pode também ser expresso como "horas:minutos:segundos.fração", onde "horas" é um inteiro no intervalo [0,23]; "minutos" é um inteiro no intervalo [0,59]; "segundos" é um inteiro no intervalo [0,59]; e "fração" é um inteiro positivo. Quando *explicitDur* não for especificado, o objeto que estiver associado ao descritor será exibido com sua duração *default*. Objetos de mídia sem duração intrínseca, como textos e imagens estáticas, possuem duração *default* infinita, ou seja, para suas exibições terminarem, precisamos definir esse atributo ou encerrá-las explicitamente por um elo de sincronismo. Por exemplo, para que um vídeo seja apresentado por apenas três segundos, podemos associá-lo a um descritor especificado como a seguir:

```
<descriptor id="dTVtelaInteira" region="rgTVtelaInteira"
                explicitDur="3s" />
```

[2] Os atributos *explicitDur* e *freeze* são definidos pelo módulo **Timing** [ABNT, NBR 15606-2, 2011; ITU-T, H.761, 2011].

- *freeze:* identifica o que acontece ao final da apresentação do objeto de mídia associado ao descritor. O valor *default* do atributo é "false". Em um vídeo, o valor "true" indica que, ao término natural do vídeo, o último quadro deve ser congelado indefinidamente, até que algum elo termine sua exibição. Por exemplo, caso se queira manter o último quadro do vídeo congelado, seu descritor deve ser definido da seguinte maneira:

```
<descriptor id="dTVtelaInteira" region="rgTVtelaInteira"
                freeze="true" />
```

- *player:* identifica a ferramenta de apresentação a ser utilizada para exibir o objeto de mídia associado ao descritor. Quando esse atributo é omitido, o formatador NCL deve levar em conta o atributo *type* do elemento <media> correspondente ao objeto de mídia a ser apresentado. Se esse atributo também não for especificado, o formatador NCL deve considerar a extensão do arquivo especificado no atributo *src* do elemento <media>.

Exemplo 7.1 — Reproduzindo uma Imagem sobre um Vídeo

Esta seção apresenta uma modificação sobre a aplicação NCL do exemplo do capítulo anterior. O novo exemplo reproduz um vídeo em tela inteira e uma imagem que aparece no canto superior direito para indicar uma oportunidade de interação. Essa imagem é utilizada em exemplos posteriores para iniciar a interatividade, através do acionamento de uma tecla do controle remoto.

Vamos iniciar a exibição da imagem junto com o vídeo. Para fazer isso, não basta apenas acrescentar o novo objeto de mídia, o descritor e a região correspondente, pois o nó não seria apresentado jamais (Figura 7.1).

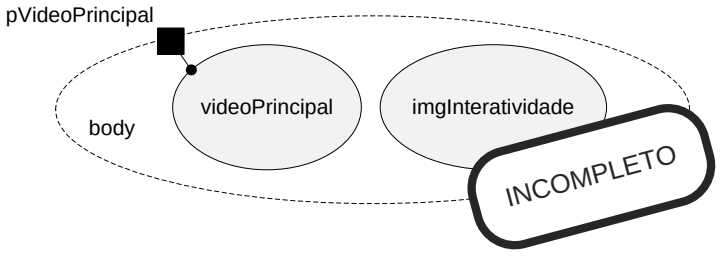


Figura 7.1 Visão estrutural ilustrando nó inatingível.

Para que a exibição da imagem ocorra no início da aplicação, junto com o vídeo, é suficiente declararmos outra porta (“pImgInteratividade”), mapeando para a imagem (“imgInteratividade”), conforme a visão estrutural ilustrada na Figura 7.2.

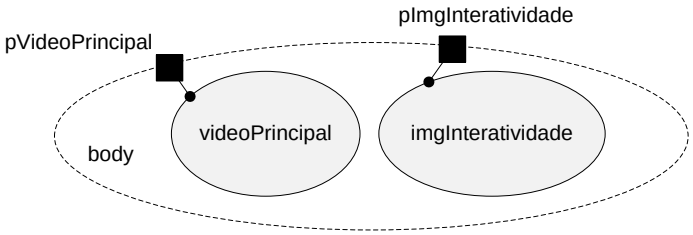


Figura 7.2 Visão estrutural do Exemplo 7.1.

A visão de leiaute é apresentada na Figura 7.3. Observamos que foi introduzida uma nova região (“rgInteratividade”), onde será apresentado o novo objeto de mídia.

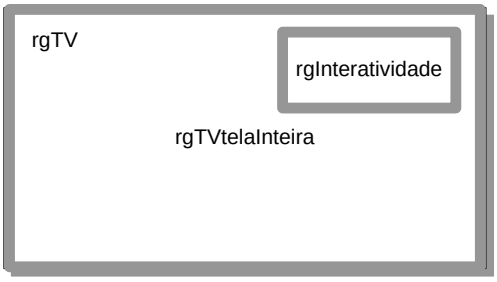


Figura 7.3 Visão de leiaute do Exemplo 7.1.

Surge um problema, no entanto. Como uma imagem não tem duração intrínseca, ela é, *por default*, exibida indefinidamente, e, nesse caso, a aplicação não terminará jamais. Para resolver esse problema, no exemplo a imagem é definida como tendo a duração explícita de quatro segundos. Uma solução melhor é apresentada no Capítulo 10, dedicado a elos e conectores. As visões temporal e espacial desse exemplo são ilustradas na Figura 7.4.

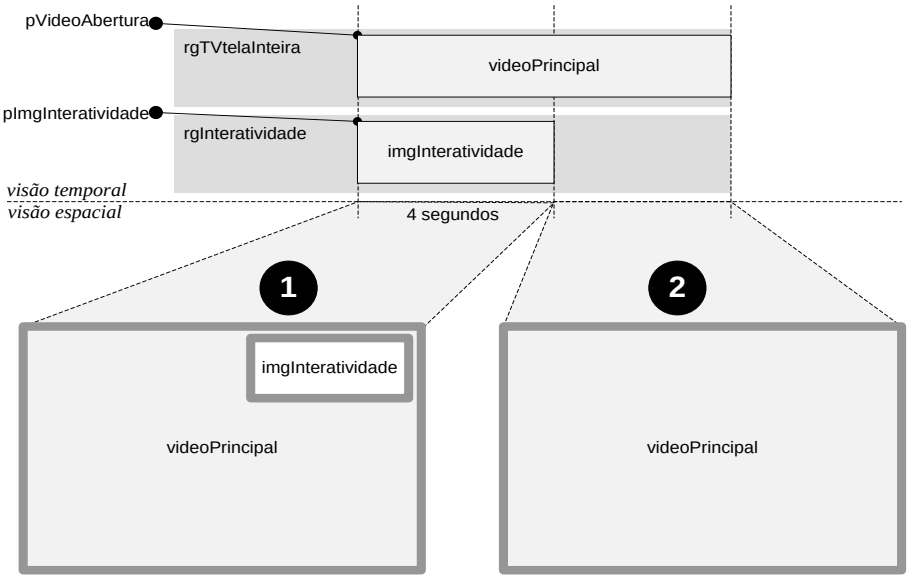


Figura 7.4 Visões temporal e espacial do Exemplo 7.1.

Passo a passo

Tomando como base o exemplo do capítulo anterior, para construir o novo exemplo, é necessário criar os seguintes elementos:

1. a região da tela onde a imagem será exibida, no caso "rgInteratividade";
2. o descritor que determina a forma como a imagem será exibida, em que região e durante quanto tempo ("dInteratividade"). Nesse caso, o vídeo associado a esse descritor será exibido na região "rgInteratividade", durante quatro segundos;
3. o nó de mídia "imgInteratividade", a imagem propriamente dita;
4. a porta que mapeia para a imagem, no caso "pImgInteratividade".

Observação: Para executar o exemplo, são necessários um arquivo de vídeo chamado "principal.mpg" e um arquivo de imagem chamado "vermelhoI.png".

Passo 1: Definindo a região onde exibir a imagem do "botão"

Assim como no exemplo do capítulo anterior, devemos definir, na seção `<regionBase>` do cabeçalho do documento, uma região para a apresentação do nó de mídia da imagem. Para garantir que a imagem será apresentada numa camada superior à camada de vídeo, utilizamos o atributo de região *zIndex*.

```
<head>
...
       <regionBase>
         <region id="rgTV">
            <region id="rgTVtelaInteira" />
             <region id="rgInteratividade" right="5%" bottom="5%"
                     width="45" height="45" zIndex="3" />
         </region>
      </regionBase>
...
</head>
```

Passo 2: Definindo o descritor que determina como a imagem do "botão" será exibida

Assim como no exemplo do capítulo anterior, devemos definir, na seção <descriptorBase> do cabeçalho do documento, um descritor para a apresentação do nó de mídia de imagem na região "rgInteratividade". Devemos definir também o atributo *explicitDur* para restringir a duração da imagem a quatro segundos:

```
<head>
...
    <descriptorBase>
       <descriptor id="dTVelaInteira" region="rgTVtelaInteira" />
       <descriptor id="dInteratividade" region="rgInteratividade"
                                              explicitDur="4s" />
    </descriptorBase>
...
</head>
```

Passo 3: Definindo a mídia correspondente à imagem do "botão"

Para criar o nó de mídia para a imagem, devemos editar a seção <body> e acrescentar o elemento de mídia correspondente, indicando qual descritor deve ser utilizado para iniciar sua apresentação:

```
<body>
...
   <media id="videoPrincipal" src="media/principal.mpg"
                                    descriptor="dTVtelaInteira" />
   <media id="imgInteratividade" src="media/vermelhoI.png"
                                    descriptor="dInteratividade" />
...
</body>
```

Passo 4: Definindo a porta adicional para a nova imagem

Para que a imagem seja iniciada junto com o início da aplicação, devemos acrescentar, também na seção <body>, uma nova porta mapeando para o novo elemento de mídia:

```
<body>
       <port id="pVideoPrincipal" component="videoPrincipal" />
       <port id="pInteratividade" component="imgInteratividade" />
...
</body>
```

Quando a aplicação iniciar, as mídias mapeadas em todas as portas do contexto <body> são iniciadas; nesse caso, "videoPrincipal" e "imgInteratividade". A Listagem 7.3 apresenta o código completo do Exemplo 7.1.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="exemplo02" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
   <head>
      <regionBase>
         <region id="rgTV">
            <region id="rgTVtelaInteira" zIndex="1" />
            <region id="rgInteratividade" right="5%" bottom="5%"
                              width="45" height="45" zIndex="3" />
         </region>
      </regionBase>
      <descriptorBase>
         <descriptor id="dTVtelaInteira"
            region="rgTVtelaInteira" />
         <descriptor id="dInteratividade" region="rgInteratividade"
                                              explicitDur="4s" />
      </descriptorBase>
   </head>
   <body>
      <port id="pVideoPrincipal" component="videoPrincipal" />
      <port id="pInteratividade" component="imgInteratividade" />
      <media id="videoPrincipal" src="media/principal.mpg"
         descriptor=" dTVtelaInteira " />
      <media id="imgInteratividade" src="media/vermelhoI.png"
                                     descriptor="dInteratividade" />
   </body>
</ncl>
```

Listagem 7.3 Listagem completa do Exemplo 7.1.

Nesse exemplo, a imagem do "botão" de interatividade é exibida por um período de tempo pré-definido (quatro segundos). No Capítulo 10, mostramos como fazer para sincronizar a imagem com o vídeo, iniciando e terminando a exibição da imagem junto com o vídeo.

7.1.3 Parâmetros de Descritor

A NCL define, ainda, o seguinte elemento opcional contido num elemento de descritor:

- <descriptorParam>: define um parâmetro do descritor como um par (propriedade, valor). As propriedades e seus respectivos valores dependem da ferramenta de exibição da mídia associada ao descritor.

Cada descritor pode conter diversos elementos <descriptorParam>, definidos no formato:

```
<descritor ...>
       <descriptorParam name="nomeParam" value="valorParam" />
       <descriptorParam name="nomeParam" value="valorParam" />
       ...
</descritor>
```

Por exemplo, um descritor pode definir um parâmetro adicional **soundLevel**, com valor "0.9" ou 90%, para indicar que a mídia correspondente deve ser reproduzida com o volume a 90% do seu volume de gravação:

```
<descritor id="dTVtelaInteira" region="rgTVtelaInteira">
       <descriptorParam name="soundLevel" value="0.9" />
</descritor>
```

ou, ainda,

```
<descritor id="dTVtelaInteira" region="rgTVtelaInteira">
       <descriptorParam name="soundLevel" value="90%" />
</descritor>
```

O uso de parâmetros de descritor promove alto grau de flexibilidade. Cabe a cada programa de exibição do objeto de mídia (ferramenta de exibição) interpretar essas propriedades da forma adequada.

Existem parâmetros de descritor reservados para diversos tipos de mídia: objetos de mídia com áudio, objetos de mídia visual e objetos de mídia textual. Na verdade, os nomes reservados são para propriedades de objetos de

mídia que podem ter seu valor inicial especificados pelo elemento <descriptorParam>.

Alguns parâmetros parâmetros de descritor reservados para objetos de mídia com áudio:

- **"soundLevel", "trebleLevel", "bassLevel"**: valores entre "0" e "1", ou entre "0%" e "100%". No caso de **soundLevel**, "0" = mute; "0.5" ou "50%" = volume pela metade; e "1" ou "100%" = volume máximo).
- **"balanceLevel":** valor entre "-1" e "1", ou entre "100%" e "100%".

Alguns parâmetros de descritor reservados para objetos de mídia visual:

- **"top", "left", "bottom", "right", "width", "height":** posição e dimensões do objeto de mídia, conforme definido pelos atributos análogos do elemento <region>;
- **"location":** posição do objeto de mídia, formatada como dois números separados por vírgula, na ordem "left, top" num dos seguintes formatos:
- números reais entre 0 e 100, seguidos do sinal de porcentagem; ou
- números inteiros que especifiquem um valor em pixels;
- **"size":** dimensões do objeto de mídia, formatada como dois números separados por vírgula, na ordem "width, height", num dos seguintes formatos:
- números reais entre 0 e 100, seguidos do sinal de porcentagem; ou
- números inteiros não-negativos que especifiquem um valor em pixels;
- **"bounds":** posição e dimensões do objeto de mídia, formatada como quatro números separados por vírgula, na ordem "left, top, width, height", num dos seguintes formatos:
- números reais entre 0 e 100, seguidos do sinal de porcentagem; ou
- números inteiros que especifiquem um valor em pixels;
- **"zIndex":** posição da região no eixo "z", utilizado geralmente para indicar sobreposições de regiões, conforme definido pelo atributo análogo do elemento <region>;

- **"background":** cor de fundo exibida quando a mídia não couber na região (dependendo do valor do atributo *fit*). Deve ser utilizado um dos seguintes nomes de cores reservados: "white", "black", "silver", "gray", "red", "maroon", "fuchsia", "purple", "lime", "green", "yellow", "olive", "blue", "navy", "aqua" ou "teal". Também pode ser "transparent";

- **visible:** "true" (a mídia deve ser apresentada na tela do dispositivo) ou "false" (a mídia não deve ser apresentada na tela do dispositivo);

- **transparency:** número real entre 0 e 1 (ou entre 0 e 100%) indicando transparência: "0" significa totalmente opaco e "1" ou "100%" significa totalmente transparente;

- **fit:** um dos seguintes valores: "fill", "hidden", "meet", "meetBest" ou "slice", cujos efeitos são descritos a seguir:

- "fill": redimensiona o conteúdo do objeto de mídia para que toque todas as bordas da região, distorcendo-o caso necessário, conforme ilustrado pela Figura 7.5;

Figura 7.5 Ilustração do parâmetro de descritor **fit** com o valor "fill".

- "hidden": se a altura intrínseca ao conteúdo da mídia for menor que o atributo *height*, o objeto precisa ser renderizado a partir do topo e ter sua altura restante preenchida com a cor de *background*; caso seja maior, o restante deve ser cortado. Idem para a largura esquerda, conforme ilustrado pela Figura 7.6, para os dois casos: mídia menor que a região e mídia maior que a região;

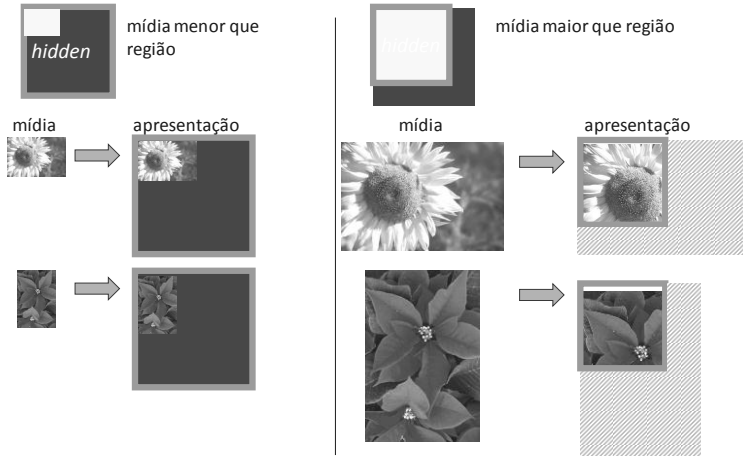


Figura 7.6 Ilustração do parâmetro de descritor **fit** com o valor "hidden".

- "meet": redimensiona o conteúdo do objeto de mídia mantendo suas proporções até atingir uma das bordas da região. Caso haja um espaço vazio à direita ou na parte de baixo, deve ser preenchido com a cor de *background*, conforme ilustrado pela Figura 7.7;

Figura 7.7 Ilustração do parâmetro de descritor **fit** com o valor "meet".

- "meetBest": semelhante ao "meet", mas o objeto de mídia não é ampliado em mais do que o dobro das dimensões originais, conforme ilustrado pela Figura 7.8;

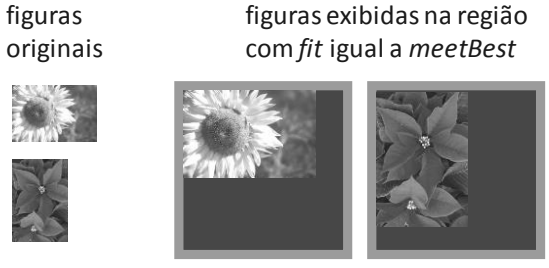


Figura 7.8 Ilustração do parâmetro de descritor **fit** com o valor "meetBest".

- "slice": redimensiona o conteúdo do objeto de mídia mantendo suas proporções até que toda a região seja preenchida. Parte do conteúdo pode ser cortado à direita ou na parte de baixo do conteúdo, conforme ilustrado pela Figura 7.9.

Figura 7.9 Ilustração do parâmetro de descritor **fit** com o valor "slice".

Alguns parâmetros de descritor reservados para objetos de mídia textual;

- **scroll:** um dos seguintes valores: "none", "horizontal", "vertical", "both" ou "automatic". Se o valor for "automatic", as barras de rolamento só aparecem caso necessário, ou seja, caso o texto não caiba na região à qual está associado;
- **style:** localização de um arquivo de folha de estilo;
- **fontColor:** a cor da fonte ("white", "black", "silver", "gray", "red", "maroon", "fuchsia", "purple", "lime", "green", "yellow", "olive", "blue", "navy", "aqua" ou "teal");
- **fontFamily:** lista priorizada com nomes de fontes específicas ou genéricas. Especificação obrigatória;
- **fontSize:** tamanho da fonte, em pontos. Especificação obrigatória;
- **fontVariant:** texto "normal" ou em "small-caps" (letras maiúsculas pequenas);
- **fontWeight:** "normal" ou "bold" (negrito).

Alguns parâmetros de descritor reservados para qualquer tipo de objeto de mídia:

- **reusePlayer:** define se um player já instanciado e se pode ou não ser reutilizado. Pode assumir os valores "false" ou "true";
- **playerLife:** define o que acontece com o player quando termina de tocar a mídia associada ao descritor: "keep" ou "close".

No Capítulo 9, valores *default* para propriedades, definidas ou não por meio de elementos <descriptorParam>, são estabelecidos.

Exemplo 7.2 — Reproduzindo uma Imagem com Transparência sobre um Vídeo

Este exemplo apresenta uma modificação sobre a aplicação NCL do exemplo anterior, de modo a exibir a imagem de botão com transparência. Para isso, é necessário apenas acrescentar um parâmetro (através do elemento <descriptorParam>) ao descritor "dInteratividade", conforme ilustrado no código da Listagem 7.4. As visões estrutural, de layout e temporal não sofrem alterações.

```
<descriptor id="dInteratividade" region="rgInteratividade"
              explicitDur="4s">
       <descriptorParam name="transparency" value="60%" />
</descriptor>
```

Listagem 7.4 Exemplo de descritor com transparência.

7.2 Navegação por Teclas

Para permitir a navegação por itens de um menu utilizando teclas, utilizamos os atributos *focusIndex, moveLeft, moveRight, moveUp, moveDown, focusBorderColor, focusBorderTransparency, focusBorderWidth, focusSrc, focusSelSrc* e *selBorderColor* do elemento <descriptor>.[3] Como sempre, esses atributos definem, de fato, valores para propriedades de objetos de mídia associados ao descritor.

Definimos a navegação por teclas nos descritores utilizados pelos nós que podem ter o foco da interação. Para cada descritor, definimos um atributo

[3] Atributos de descritor relacionados à navegação estão descritos no módulo **KeyNavigation** [ABNT, NBR 15606-2, 2011; ITU-T, H.761, 2011].

focusIndex, cujo valor corresponde a um índice que o objeto de mídia associado àquele descritor receberá. Utilizamos o valor desse atributo nos demais descritores para indicar o destino do foco quando o usuário pressionar a seta para cima (atributo *moveUp*), para baixo (atributo *moveDown*), para a esquerda (atributo *moveLeft*) e para a direita (atributo *moveRight*). A ausência de um desses atributos significa que não há navegação com a tecla de seta correspondente do controle remoto.

Por exemplo, num menu vertical de seis itens, faz sentido definir como valores para o atributo *focusIndex* os valores de "1" a "6". Os atributos *moveDown* e *moveUp* de cada descritor indicam para qual opção deve mudar o foco quando as teclas DOWN e UP forem pressionadas, respectivamente. Por exemplo, no descritor de *focusIndex* "1", definimos o atributo *moveDown* como "2" para significar que, quando o foco estiver no elemento associado ao descritor com *focusIndex* "1", se o usuário pressionar a tecla de seta para baixo o foco passa para o elemento associado ao descritor com *focusIndex* "2". É possível, ainda, definir um menu circular, no qual o atributo moveUp do primeiro descritor do primeiro item tem como valor o focusIndex do descritor do último item. A Figura 7.10 ilustra esses atributos para a versão simples e para a versão circular.

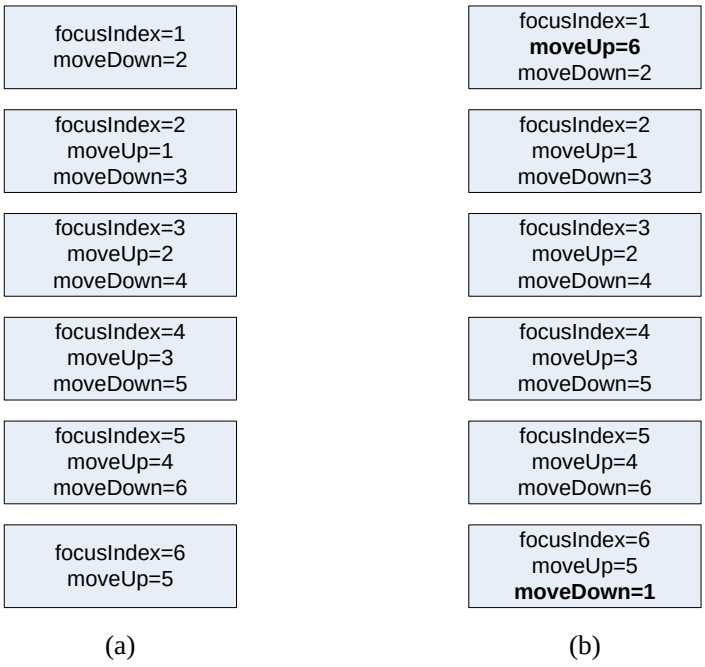


Figura 7.10 Atributos de descritor relacionados à navegação em um menu vertical de seis itens: (a) não-circular e (b) circular.

De forma análoga, para menus horizontais, os atributos *moveLeft* e *moveRight* de cada descritor indicam para qual opção deve mudar o foco quando as teclas LEFT e RIGHT forem pressionadas, respectivamente.

Nesse tipo de menu, é imprescindível que o usuário seja mantido informado sobre qual é a seleção atual. Para isso, este exemplo utiliza o conceito de foco introduzido em NCL 3.0. Podemos definir cor, transparência e largura de moldura do elemento em foco pelos atributos *focusBorderColor*, *focusBorderTransparency* e *focusBorderWidth*. Um valor negativo (p. ex., "–2") para a largura indica que a moldura deve ser exibida ocupando alguns (no caso, "2") pixels "dentro" da região em que o botão aparece, enquanto um valor positivo projeta a moldura para fora dessa região, circunscrevendo o nó, conforme ilustrado pela Figura 7.11.

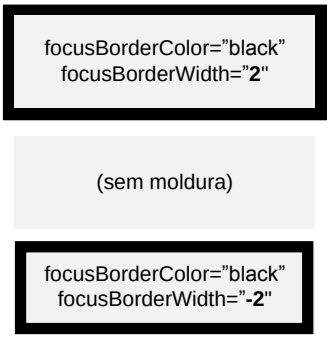


Figura 7.11 Ilustração de diferentes valores de *focusBorderWidth* para traçar uma moldura ao redor do elemento em foco.

Além de traçar uma moldura, a NCL permite que o próprio objeto de mídia seja modificado quando o descritor ganha o foco, através do atributo *focusSrc*. Seu valor é um URI, semelhante ao atributo *src* do elemento <media>, que apresentamos no próximo capítulo. Quando o usuário pressiona o botão OK para selecionar o elemento em foco, também é possível mudar a cor da moldura e a mídia, através dos atributos *selBorderColor* e *focusSelSrc*.

Resumindo, os atributos do elemento <descriptor> relacionados à navegação por teclas são os seguintes:

- *focusIndex*: define um índice de navegação para o objeto de mídia associado ao descritor. Caso não seja definido um valor para esse atributo, o objeto de mídia associado ao descritor não poderá receber o foco da navegação. O foco inicial estará no objeto com *focusIndex*

menor. Vale observar que cada valor de *focusIndex* deve ser único por todo o documento. Caso dois ou mais descritores diferentes possuam o mesmo valor de *focusIndex*, um deles será ignorado; *focusBorderColor*: define a cor da moldura que destaca o elemento em foco, ou seja, o retângulo que deve aparecer sobrescrito à região desse descritor quando o objeto a ele associado receber o foco. Pode ser um dos seguintes valores: (“white”, “black”, “silver”, “gray”, “red”, “maroon”, “fuchsia”, “purple”, “lime”, “green”, “yellow”, “olive”, “blue”, “navy”, “aqua” ou “teal”). Esse atributo assume o valor *default* especificado pela propriedade *defaultSelBorderColor* do nó *settings* do documento;

- *focusBorderWidth*: define a espessura, em *pixels*, da moldura desse descritor quando o elemento a ele associado receber o foco. Caso seja igual a 0, nenhuma borda será apresentada. Um valor positivo indica que a borda estará fora do conteúdo do objeto, ao passo que um valor negativo indica que a borda será desenhada sobre o conteúdo do objeto. Esse atributo assume o valor *default* especificado pela propriedade *defaultFocusBorderWidth* do nó *settings* do documento;

- *focusBorderTransparency*: define a porcentagem de transparência da moldura. Deve ser um valor real entre 0 e 1 (ou um valor em porcentagem, entre 0 e 100%), onde 0 significa totalmente opaco e 1 significa totalmente transparente. Esse atributo assume o valor *default* especificado pela propriedade *defaultFocusBorderTransparency* do nó *settings* do documento;

- *focusSrc*: define o caminho de uma mídia alternativa que deve ser exibida quando o elemento associado a esse descritor estiver com o foco. Segue o mesmo formato do atributo *src* do elemento <media>, conforme apresentado no próximo capítulo;

- *focusSelSrc*: define o caminho de uma mídia alternativa que deve ser exibida quando for pressionada a tecla de ativação (OK, Enter ou equivalente) enquanto o elemento associado a esse descritor estiver com o foco. Segue o mesmo formato do atributo *src* do elemento <media>, conforme apresentado no próximo capítulo;

- *selBorderColor*: define uma cor de moldura que deve ser exibida quando for pressionada a tecla de ativação (OK, Enter ou equivalente) enquanto o elemento associado a esse descritor estiver com o foco. Esse atributo assume o valor *default* especificado pela propriedade *defaultFocusBorderColor* do nó *settings* do documento;

- *moveLeft*: identifica o índice de navegação do elemento *E* que deve receber o foco caso seja pressionada a seta para a esquerda enquanto o elemento associado a esse descritor estiver com o foco (definido pelo atributo *focusIndex* do elemento *E*). Caso o elemento que receberia o foco esteja com a propriedade *visible* igual a "false" ou não esteja sendo apresentado, o foco não se desloca;

- *moveRight*: identifica o índice de navegação do elemento *E* que deve receber o foco caso seja pressionada a seta para a direita enquanto o elemento associado a esse descritor estiver com o foco (definido pelo atributo *focusIndex* do elemento *E*). Caso o elemento que receberia o foco esteja com a propriedade *visible* igual a "false" ou não esteja sendo apresentado, o foco não se desloca;

- *moveUp*: identifica o índice de navegação do elemento *E* que deve receber o foco caso seja pressionada a seta para cima enquanto o elemento associado a esse descritor estiver com o foco (definido pelo atributo *focusIndex* do elemento *E*). Caso o elemento que receberia o foco esteja com *visible* igual a "false" ou não esteja sendo apresentado, o foco não se desloca;

- *moveDown*: identifica o índice de navegação do elemento *E* que deve receber o foco caso seja pressionada a seta para baixo enquanto o elemento associado a esse descritor estiver com o foco (definido pelo atributo *focusIndex* do elemento *E*). Caso o elemento que receberia o foco esteja com *visible* igual a "false" ou não esteja sendo apresentado, o foco não se desloca.

Vale observar que se, a qualquer momento da apresentação, o foco for perdido (p. ex., se a mídia associada ao descritor em foco terminar de ser exibida), um novo foco será definido para o elemento cujo descritor tiver o menor valor para *focusIndex*. Além disso, caso o foco deva ser deslocado para um elemento cuja propriedade *visible* está com o valor "false" ou para um elemento que não esteja sendo apresentado, o foco atual não se desloca, como já mencionado.

Quando um elemento em foco é selecionado pressionando-se a tecla de ativação (ENTER, OK), o controle é passado para o *player* do elemento <media> correspondente. A partir de então, o *player* pode seguir suas próprias regras de navegação (p. ex., uso de setas para fazer o rolamento em uma página HTML). O controle de foco deve ser passado de volta ao formatador NCL quando a tecla BACK (volta) for pressionada. Nesse caso, o foco se desloca para o elemento identificado pelo atributo *service.currentFocus* do nó *settings*, variável de ambiente vista na Seção

9.3.1. O controle de foco também pode ser deslocado por programação. Para isso, devemos atribuir valores à propriedade *service.currentKeyMaster* do nó *settings*, através de um elo[4] ou de um comando de um objeto imperativo (NCLua ou NCLet)[5] ou, ainda, pelo *player* (ferramenta de exibição) do nó que está com o controle de foco atual.

A Listagem 7.5 apresenta a definição dos descritores de um menu vertical de quatro itens, cujas imagens e molduras são modificadas quando cada item ganha o foco.

```
<descriptorBase>
        <descriptor id="dMenuItem1" region="rgMenuItem1"
                focusIndex="1" moveUp="4" moveDown="2"
                focusSelSrc="media/menu1ON.gif"
                focusBorderWidth="3" focusBorderColor="white"
selBorderColor="white" />
        <descriptor id="dMenuItem2" region="rgMenuItem2"
                focusIndex="2" moveUp="1" moveDown="3"
                focusSelSrc="media/menu2ON.gif"
                focusBorderWidth="3" focusBorderColor="white"
selBorderColor="white" />
        <descriptor id="dMenuItem3" region="rgMenuItem3"
                focusIndex="3" moveUp="2" moveDown="4"
                focusSelSrc="media/menu3ON.gif"
                focusBorderWidth="3" focusBorderColor="white"
selBorderColor="white" />
        <descriptor id="dMenuItem4" region="rgMenuItem4"
                focusIndex="4" moveUp="3" moveDown="1"
                focusSelSrc="media/menu4ON.gif"
                focusBorderWidth="3" focusBorderColor="white"
selBorderColor="white" />
        ...
</descriptorBase>
```

Listagem 7.5 Definição de descritores de um menu vertical de quatro itens.

7.3 Efeitos de Transição

Para exibir mídias com efeitos de transição, utilizamos os atributos *transIn* e *transOut* do elemento <descriptor>, que fazem referência a elementos

[4] Elos que manipulam propriedades são apresentados na Seção 10.8.

[5] Objetos imperativos são apresentados nos Capítulos 17 e 18.

<transition> de uma base de transições <transitionBase>.[6] Os efeitos de transição não são obrigatórios no perfil BDTV da linguagem NCL 3.0, mas apenas no perfil EDTV. Antes de examinar os atributos *transIn* e *transOut*, descrevemos a base de transições, seus elementos e atributos.

7.3.1 Base de Transições

As transições (elemento <transition>) especificam efeitos de transição que os descritores podem utilizar na exibição de objetos de mídia em uma aplicação NCL. Assim como os descritores, as transições são definidas no cabeçalho do documento (dentro do elemento <head>), em uma base de transições definida pelo elemento <transitionBase>:

```
<transitionBase>
       <transition ... />
</transitionBase>
```

Assim como a base de descritores, uma aplicação NCL pode possuir apenas uma base de transições. Assim como os descritores, transições também podem ser importadas de documentos NCL externos, através do elemento <importBase> e seus atributos *alias* e *documentURI*. A Tabela 7.1 sumariza os atributos e conteúdo do elemento <transitionBase>.

Tabela 7.1 Elementos, Atributos e Conteúdo (Elementos Filhos) de uma Base de Transições

Elementos	Atributos	Conteúdo
transitionBase	*id*	(importBase, transition)+

7.3.2 Transições

A NCL utiliza os tipos de transição e seus respectivos subtipos apresentados na Tabela 7.2. Cada tipo de transição descreve um grupo de transições que são intimamente relacionadas. Dentro desse tipo, cada uma das transições individuais é associada a um subtipo que enfatiza as características distintas da transição. O atributo *type* de um elemento <transition> é obrigatório e utilizado para especificar a transição geral. O atributo *subtype* fornece o controle específico para a transição. Esse atributo é opcional e, se especificado, deve ser um dos subtipos de transição apropriados para o tipo

[6] A especificação das transições encontra-se nos módulos **TransitionBase** e **Transition** [ABNT, NBR 15606-2, 2011; ITU-T, H.761, 2011].

correspondente. Se esse atributo não for especificado, a transição deve ser revertida para o subtipo *default* do tipo especificado. Vale notar que apenas os subtipos *default* são obrigatoriamente implementados no middlewares Ginga para TV digital terrestre. Outros tipos e subtipos são opcionais. Caso um tipo ou subtipo não esteja definido na implementação em um determinado dispositivo, o efeito de transição correspondente será ignorado, mas a aplicação será executada sem erros.

Tabela 7.2 Tipos e Subtipos de Transição

Tipo	Subtipos (Valor *Default* Sublinhado)
"barWipe"	"leftToRight", "topToBottom"
"irisWipe"	"rectangle, "diamond"
"clockWipe"	"clockwiseTwelve", "clockwiseThree", "clockwiseSix", "clockwiseNine"
"snakeWipe"	"topLeftHorizontal", "topLeftVertical", "topLeftDiagonal", "topRightDiagonal", "bottomRightDiagonal", "bottomLeftDiagonal"
"fade"	"crossfade", "fadeToColor", "fadeFromColor"

Os tipos de transição podem ser descritos como:

- "barWipe": varredura simples, horizontal ou vertical;
- "irisWipe": varredura a partir do centro de um objeto de mídia, seguindo uma figura geométrica;
- "clockWipe": varredura como um ponteiro de relógio;
- "snakeWipe": varredura em espiral;
- "fade": mudança gradual de cores.

Os atributos do elemento <transition> são os seguintes:

- *id*: atributo obrigatório que identifica a transição, que será utilizado como valor para os atributos *transIn* e *transOut* dos descritores, e que segue a mesma regra de formação para o atributo *id* definida no Capítulo 5;

- *type*: atributo obrigatório que especifica o tipo da transição, conforme a Tabela 7.2;
- *subtype*: subtipo da transição, conforme o tipo (Tabela 7.2);
- *direction*: especifica a direção em que ocorrerá a transição. Os valores permitidos são "forward" e "reverse". O valor *default* é "forward". Nem todas as transições terão interpretações reversas significantes. Por exemplo, um *crossfade* não é uma transição geométrica e, portanto, não tem interpretação de direção reversa. As transições que não têm interpretação reversa devem ter o atributo *direction* ignorado e o valor *default* "forward" assumido;
- *dur*: duração da transição, em segundos. O *default* é "1s";
- *startProgress*: especifica o quanto de progresso para a transição deve ser assumido ao iniciar a execução, entre [0.0,1.0]. Um valor de "0.5" para uma transição de entrada *type* "barWipe", *subtype* "leftToRight", significa que, no início da transição, metade da transição já terá ocorrido, ou seja, metade da imagem do objeto de mídia já aparece de imediato, e a transição prossegue revelando o restante da imagem, conforme ilustrado pela Figura 7.12;

Figura 7.12 Ilustração de uma transição de entrada *type* "barWipe", *subtype*="leftToRight", *startProgress*="0.5", que apresenta a imagem inicialmente já a 50% do progresso total da transição.

- *endProgress*: especifica até quanto de progresso para a transição deve ser realizado até terminar a execução entre [0.0,1.0]. Ao atingir esse valor, toda a mídia é revelada;
- *fadeColor*: cor utilizada pelo efeito de transição do tipo "fade", com subtipo "fadeToColor" ou "fadeFromColor". O valor *default* para o atributo é "black". Esse atributo é ignorado para outros tipos e subtipos de transição;

- *horRepeat*: quantas vezes a transição deve ser realizada no eixo horizontal. O valor *default* é "1";
- *verRepeat*: quantas vezes a transição deve ser realizada no eixo vertical. O valor *default* é "1";
- *borderWidth*: largura da moldura gerada ao longo da borda da transição. Se for "0" (o *default*), não é gerada nenhuma moldura ao longo da borda da transição;
- *borderColor*: cor da moldura gerada ao longo da borda da transição. Se for "blend", a moldura gerada ao longo da borda será de uma cor aditiva às cores das mídias de origem. O valor *default* para esse atributo é "black".

A Tabela 7.3 sumariza os atributos e conteúdo do elemento <transition>.

Tabela 7.3 Elementos, Atributos e Conteúdo (Elementos Filhos) das Transições

Elementos	Atributos	Conteúdo
transition	*id, type, subtype, dur, startProgress, endProgress, direction, fadeColor, horRepeat, vertRepeat, borderWidth, borderColor*	—

7.3.3 Atributos de Descritor para Transições

Os atributos *transIn* e *transOut* do elemento <descriptor> são definidos do seguinte modo:

- *transIn*: identificador da transição que será executada ao iniciar a apresentação do objeto de mídia, tal como definida pelo elemento <transition> na base de transições <transitionBase>. Podem ser definidas diversas transições, separadas por ponto-e-vírgula, para o caso de a transição desejada prioritariamente não estar disponível;
- *transOut*: identificador da transição que será executada ao terminar a apresentação do objeto de mídia, tal como definida pelo elemento <transition> na base de transições <transitionBase>. Podem ser definidas diversas transições, separadas por ponto-e-vírgula, para o caso de a transição desejada prioritariamente não estar disponível.

Exemplo 7.3 — Reproduzindo uma Imagem com Transição de Saída

O novo exemplo apresenta uma modificação sobre a aplicação NCL do exemplo anterior, de modo a terminar a exibição da imagem de interatividade com uma transição do tipo **fade**. Para tanto, é necessário definir uma base de transições e acrescentar o atributo *transOut* ao descritor correspondente, conforme ilustrado na Listagem 7.6.

```
<head>
      ...
      <transitionBase>
        <transition id="tFade" type="fade"
                              subtype="crossfade" dur="0.5s" />
      </transitionBase>
      <descriptorBase>
        <descriptor id="dInteratividade" region="rgInteratividade"
                      explicitDur="4s" transOut="tFade">
              <descriptorParam name="transparency" value="60%" />
        </descriptor>
      </descriptorBase>
      ...
</head>
```

Listagem 7.6 Definição de efeito de transição do tipo **fade**.

7.4 Referência Rápida — Descritores

A Tabela 7.4 sumariza os elementos e atributos que definem os descritores. Como sempre, os atributos obrigatórios são sublinhados.

Tabela 7.4 Elementos, Atributos e Conteúdo (Elementos Filhos) que Definem Descritores para o Perfil EDTV

Elementos	Atributos	Conteúdo
descriptor	*id*, *player*, *explicitDur*, *region*, *freeze*, *transIn*, *transOut*, *moveLeft*, *moveRight*, *moveUp*, *moveDown*, *focusIndex*, *focusBorderColor*, *focusBorderWidth*, *focusBorderTransparency*, *focusSrc*, *focusSelSrc*, *selBorderColor*	(descriptorParam)*
descriptorParam	*name*, *value*	—
descriptorBase	*id*	(importBase | descriptor | descriptorSwitch)+

Bibliografia

ABNT NBR 15606-2 2011. Associação Brasileira de Normas Técnicas, "Televisão digital terrestre — Codificação de dados e especificações de transmissão para radiodifusão digital, Parte 2: Ginga-NCL para receptores fixos e móveis — Linguagem de aplicação XML para codificação de aplicações", Sistema Brasileiro de TV Digital Terrestre, *NBR 15606-2*.

ITU-T H.761 2011. Nested Context Language (NCL) and Ginga-NCL for IPTV. ITU-T Rec. H.761. Genebra.

Capítulo 8

Objetos de Mídia e Contexto

Neste capítulo, descrevemos os elementos e atributos dos objetos de mídia[1] e contextos.[2] Apresentamos os tipos de objetos de mídia permitidos em NCL, incluindo alguns tipos de objetos especiais. Na medida em que as aplicações NCL se tornam maiores e mais complexas, seu código pode ser estruturado em contextos para aumentar a legibilidade e as oportunidades de reúso do código. Ao final deste capítulo, você será capaz de estruturar os nós e os elos de uma aplicação em contextos para obter esses benefícios.

8.1 Objetos de Mídia e seus Atributos

Cada objeto de mídia geralmente possui, além de seu identificador, os atributos *src*, que define um URI do conteúdo do objeto (isto é, onde o

[1] Objetos de mídia e seus atributos são definidos no módulo **Media** [ABNT, NBR 15606-2, 2011; ITU-T, H.761, 2011].

[2] Contextos e seus atributos são definidos no módulo **Context** [ABNT, NBR 15606-2, 2011; ITU-T, H.761, 2011].

conteúdo do objeto se encontra), *type*, que define o tipo de objeto (vídeo, áudio, texto etc.) e *descriptor*, que referencia o descritor que define como a mídia será apresentada.

Um objeto (nó) de mídia ou de conteúdo é representado pelo elemento <media>. Cada definição de objeto de mídia pode apresentar, além do arquivo de origem da mídia, o descritor que regulará a apresentação daquele objeto de mídia. No exemplo a seguir, são definidos objetos de mídia do tipo vídeo e imagem. Note que o atributo *type* é opcional. No caso, ele pode ser inferido pela extensão do arquivo que define o conteúdo do objeto. O atributo *descriptor* também é opcional. Se não for especificado, todas as definições que regulam a apresentação do objeto devem ser inferidas dos elementos <property> do objeto de mídia em questão, conforme discutido no Capítulo 9.

```
<media id="videoPrincipal" src="media/principal.mpg"
                              descriptor="dTVtelaInteira" />
<media id="imgInteratividade" src="media/vermelhoI.png"
                              descriptor="dInteratividade" />
```

8.1.1 Atributos de Objeto de Mídia

Um objeto de mídia possui os seguintes atributos:

- *id*: identificador único, obrigatório, do objeto de mídia, utilizado nas referências ao objeto (por exemplo, nas portas dos contextos que levam à mídia), e que segue a mesma regra de formação para o atributo *id* definida no Capítulo 5;
- *src*: fonte do objeto de mídia, ou seja, a localização ou caminho do objeto de mídia, conforme apresentado na Seção 8.1.2;
- *type*: atributo opcional que define o tipo de mídia, especificado como tipo MIME, conforme apresentado na Seção 8.1.3;
- *descriptor*: identificador (*id*) do descritor que controla a apresentação do objeto de mídia;
- *refer*:[3] referência a um outro nó de mídia previamente definido, como forma de reúso de nó (utiliza os atributos do nó de mídia referenciado, exceto o *id*). Esse atributo é descrito na Seção 12.1;
- *instance*:[4] utilizado apenas quando o atributo *refer* é definido. Pode assumir os valores "new", "instSame" e "gradSame", conforme visto na Seção 12.2.

[3] O atributo *refer* é definido no módulo **EntityReuse** [ABNT. NBR 15606-2. 2011; ITU-T, H.761, 2011].

8.1.2 O Atributo *src*

O atributo *src* define um URI do conteúdo do objeto. Um URI é composto de um *esquema* e uma *parte específica* do esquema, como na Figura 8.1.

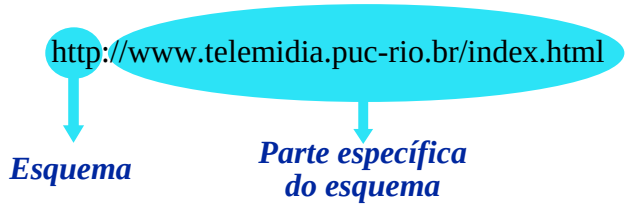


Figura 8.1 Sintaxe de um URI

O *esquema* especifica o protocolo a ser usado para a transferência do conteúdo, e sua *parte específica* define o caminho para o conteúdo em um dado servidor.

```
"//" servidor [":"port] "/" url-caminho / "#"fragmento
```

Um URI absoluto contém todas as informações necessárias para localizar seu recurso. A NCL, no entanto, também permite o uso de URI relativo. URIs relativos são endereços incompletos aplicados a um URI-base para completar a localização. As partes omitidas são o esquema do URI, o servidor e, também, em alguns casos, parte do caminho.

O benefício principal da utilização de URIs relativos é a possibilidade de mover ou copiar para outros locais os documentos e diretórios contidos no URI, sem exigir a troca dos valores dos atributos referindo-se ao URI dentro dos documentos. Apenas o URI-base mudaria de valor. Isso é especialmente interessante quando se transportam documentos de provedores de conteúdo para os receptores. Os caminhos relativos do URI são tipicamente utilizados como um meio rápido de localização de arquivos de mídia armazenados no mesmo diretório do documento NCL atual ou em um diretório próximo a ele. Quando o arquivo está localizado no mesmo diretório do documento NCL atual, o caminho relativo consiste apenas no nome do arquivo (incluindo, opcionalmente, um identificador de fragmento dentro do arquivo). Um caminho relativo também pode trazer um caminho de diretórios antes do nome do arquivo, especificando o caminhar desde o diretório do documento NCL atual até o arquivo em questão que se deseja localizar.

[4] O atributo *instance* é definido no módulo **ExtendedEntityReuse** [ABNT, NBR 15606-2, 2011; ITU-T, H.761, 2011].

Os seguintes URIs são definidos pelo Sistema Brasileiro de TV Digital terrestre:

- arquivos locais:
  "file:///caminho_arquivo/#id_fragmento"
- arquivos remotos baixados do canal de interatividade utilizando o protocolo http:
  "http://id_servidor/caminho_arquivo/#id_fragmento"
- arquivos remotos baixados do canal de interatividade utilizando o protocolo https:
  "https://id_servidor/caminho_arquivo/#id_fragmento"
- *streams* baixados do canal de interatividade utilizando o protocolo rtsp:
  "rtsp://id_servidor/caminho_arquivo/#id_fragmento"
- *streams* baixados do canal de interatividade utilizando o protocolo rtp:
  "rtp://id_servidor/caminho_arquivo/#id_fragmento"
- streams recebidos como fluxos elementares no fluxo de transporte;[5]
  "sbtvd-ts://program_number.component_tag
- fluxo de conteúdo idêntico a um fluxo que esteja em apresentação por um outro objeto de mídia:
  "ncl-mirror://identificador (*id*) do objeto de mídia

  Note que, nesse último caso, o objeto que referencia terá conteúdo idêntico e será apresentado sempre no mesmo ponto de exibição do objeto referenciado.

Para objetos de mídia com o atributo *src*, cujo valor identifica o esquema "sbtvd-ts", a parte específica do esquema, mais precisamente, o "program_number.component_tag", pode ser substituído pelas seguintes palavras reservadas:

[5] *Streams* recebidos como fluxos elementares do fluxo de transporte TS podem vir entrelaçados. Por exemplo, o conteúdo de uma propaganda pode vir em um mesmo fluxo elementar entremeado com o fluxo de um filme. Em um mesmo fluxo elementar, um *stream* de um mesmo conteúdo de um objeto de mídia é identificado por seu *contentId* (ver Apêndice E). Todo objeto de mídia tem uma propriedade (propriedades são discutidas no Capítulo 9) que contém o valor do *contentID* de um stream. Esse valor é adquirido pelo sistema (por exemplo, o middleware Ginga-NCL) do fluxo de transporte, tão logo o objeto é instanciado para apresentação.

video	Correspondente ao vídeo ES principal que está sendo apresentado no plano de vídeo, conforme definido pela ABNT NBR 15604.
audio	Correspondente ao áudio ES principal, conforme definido pela ABNT NBR 15604
text	Correspondente ao texto ES principal, conforme definido pela ABNT NBR 15604
video(i)	Correspondente ao i-ésimo menor vídeo ES *component-tag* listado na PMT dos serviços sintonizados.
audio(i)	Correspondente ao i-ésimo menor áudio ES *component-tag* listado na PMT dos serviços sintonizados.
text(i)	Correspondente ao i-ésimo menor texto ES *component-tag* listado na PMT dos serviços sintonizados.

8.1.3 O Atributo *type*

Como mencionamos, o atributo *type* é opcional. Quando um objeto de mídia inicia sua apresentação, o formatador NCL escolhe a ferramenta de exibição conforme a propriedade *player* do objeto de mídia a ser exibido. Se esse atributo não for especificado, o formatador deve levar em conta o atributo *type* do elemento <media>. Se esse atributo também não for especificado, o formatador deve considerar a extensão do arquivo especificado no atributo *src* do elemento <media>.

Os valores permitidos para o atributo *type* dependem do perfil NCL e devem obrigatoriamente seguir o formato MIME *Media Types* (ou, simplesmente, *mimetypes*). Um mimetype é uma cadeia de caracteres que define a classe da mídia (áudio, vídeo, imagem, texto, aplicação) e um tipo de codificação de mídia (como jpeg, mpeg etc.). Os mimetypes podem ser registrados ou informais. Os mimetypes registrados são controlados pela IANA (Internet Assigned Numbers Authority). Os mimetypes informais não são registrados pela IANA, mas são definidos de comum acordo; eles normalmente têm "x" ou "vnd" antes do nome do tipo de mídia. A Tabela 8.1 apresenta alguns tipos MIMES. Cabe ao sistema de TV digital definir seus tipos obrigatórios e opcionais; em ABNT, NBR 15606-1, 2011 podem ser encontrados os tipos definidos pelo SBTVD.

Tabela 8.1 Alguns Tipos MIMES

Valor de type	Extensão de Arquivo do Atributo src
text/html	.htm, .html, .xhtml
text/css	.css
text/xml	.xml
text/plain	.txt
image/bmp	.bmp
image/png	.png
image/gif	.gif
image/jpeg	.jpg, .jpeg, .jpe
audio/basic	.ua
audio/x-wav	.wav
audio/mpeg	.mp1, .mp2, .mp3
audio/mpeg4	.mp4, .mpg4
video/mpeg	.mp2, .mpeg, .mpg, .mpe
video/mp4	.mp4, .mpg4
video/x-mng	.mng
video/quicktime	.qt, .mov
video/x-msvideo	.avi
application/x-ginga-ncl, ou application/x-ncl-NCL	.ncl
application/x-ginga-NCLua, ou application/x-ncl-NCLua	.lua
application/x-ginga-NCLet	.xlt, .xlet, .class
application/x-ginga-settings, ou application/x-ncl-settings[6]	Não tem arquivo associado (não se define o atributo *src*). Também chamado de nó ou objeto *settings*, é um objeto que define atributos globais para ser utilizado em regras e em relacionamentos
application/x-ginga-time, ou application/x-ncl-time	O atributo *src* define a hora de acordo com a Universal Time Coordinated (UTC)

[6] O nó do tipo settings é descrito em detalhes na Seção 9.3.

Pela Tabela 8.1, cinco tipos especiais são definidos para objetos de mídia. Objetos de mídia do tipo "application/x-ginga-NCL", ou "application/x-ncl-NCL", têm como conteúdo um documento especificado em NCL, ou seja, código NCL. Esses objetos são assunto do Capítulo 14. Objetos de mídia do tipo "application/x-ginga-NCLua", ou "application/x-ncl-NCLua", têm como conteúdo código Lua, assunto dos Capítulos 17 e 18; já objetos de mídia do tipo "application/x-ginga-NCLet", ou "application/x-ncl-NCLet", têm como conteúdo código imperativo Java. Objetos imperativos é assunto do Capítulo 17. O objeto de mídia do tipo "application/x-ginga-settings", ou "application/x-ncl-settings", único em um documento NCL, define atributos globais e é assunto da Seção 9.3.

O tipo "application/x-ginga-time", ou "application/x-ncl-time", é aplicado a um elemento <media> especial cujo conteúdo é o Universal Time Coordinated (UTC). Só pode existir um objeto de mídia com esse tipo em um documento NCL. Seu conteúdo segue a seguinte sintaxe:
Ano":"Mês":"Dia":"Hora":"Minuto":"Segundo"."Fração,
onde Ano é um inteiro, Mês é um inteiro no intervalo [1,12], Dia é um inteiro no intervalo [1,31], Hora é um inteiro no intervalo [0, 23], Minuto é um inteiro no intervalo [0,59], Segundo é um inteiro no intervalo [0,59] e Fração é um inteiro positivo. Chamamos a atenção para o fato desse objeto de mídia especificar um tempo absoluto, independentemente do início de exibição do objeto de mídia ou da aplicação NCL.

Podemos definir um cronômetro numa aplicação NCL utilizando um elemento <media> sem fonte, que define uma espécie de relógio relativo ao tempo de início desse elemento <media>.

8.2 Contextos

Um contexto agrupa objetos (de mídia, de contexto ou *switch*)[7] e elos. O elemento <body> de toda aplicação NCL é um caso particular de contexto que representa a aplicação como um todo.

Os demais contextos de uma aplicação NCL são definidos pelo elemento <context>. Um contexto pode aninhar outros contextos ou switches, mas existe uma restrição: um contexto não pode conter recursivamente a si mesmo. Os contextos podem ser aninhados, por exemplo, para refletir a estrutura do documento e ajudar o autor a organizar os segmentos da aplicação. Um contexto (elemento <context>) é definido conforme o modelo da Listagem 8.1.

```
<context id="ctxNome">
        <!-- portas -->
        <!-- mídias, contextos e switches -->
        <!-- elos -->
        ...
</context>
```

Listagem 8.1 Esquema de definição de contexto.

Suponha uma aplicação NCL instrucional que tenha, além do vídeo principal, um menu que dá acesso a uma entrevista com o diretor do vídeo, um portfólio com a biografia dos atores e um *quizz* sobre o conteúdo do vídeo. Essas diferentes partes da aplicação podem ser estruturadas em diferentes contextos, conforme a Listagem 8.2.

```
<body>
        ...
        <context id="menu"> ... </context>
        <context id="entrevista">
                <context id="protagonista"> ... </context>
                <context id="coadjuvante1"> ... </context>
        </context>
        <context id="biografias"> ... </context>
        <context id="quizz"> ... </context>
        ...
</body>
```

Listagem 8.2 Programa estruturado em contextos.

Os atributos de um contexto são:

- *id*: identificador único do contexto, que segue a mesma regra de formação para o atributo *id* definida no Capítulo 5;
- *refer*:[7] referência a um outro contexto previamente definido (utiliza os atributos e elementos filhos do contexto referenciado, exceto o atributo *id*); O atributo *refer* é descrito no Capítulo 13.

Um contexto encapsula os objetos e elos que contém. Sendo assim, para acessar um objeto dentro de um contexto, é necessário definir *portas* no contexto que mapeiem para os objetos que se deseja acessar, conforme descrito na próxima seção.

[7] O atributo *refer* é definido no módulo **EntityReuse** [ABNT, NBR 15606-2, 2007; ITU-T, H.761 ,2009].

8.3 Portas

Uma porta <port> é um ponto de interface de um contexto que oferece acesso externo ao conteúdo (objetos internos) do contexto.[8] Em outras palavras, para um elo ser ligado a um objeto interno ao contexto, esse contexto deve possuir uma porta que mapeie o objeto interno desejado, conforme ilustrado pela Figura 8.2.

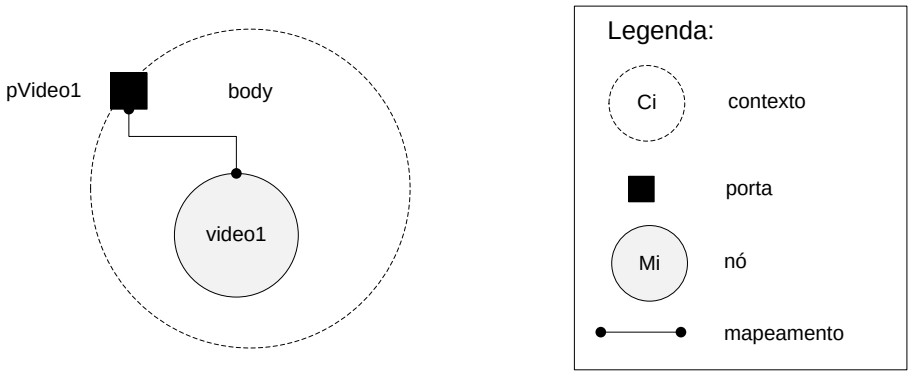


Figura 8.2 Porta "pVideo1" como ponto de entrada a um nó "video1" interno a um contexto.

Observe que, na Figura 8.2, a porta "pVideo1" do contexto <body> mapeia para o vídeo "video1". Isso significa que, ao iniciar a execução do documento, o formatador seguirá a porta "pVideo1", que leva ao nó de conteúdo "video1", que será então apresentado.

Uma porta possui os seguintes atributos:

- *id*: identificador único da porta, utilizado nas referências à porta (por exemplo, por elos), que segue a mesma regra de formação para o atributo *id* definida no Capítulo 5;
- *component*: objeto de mídia ou contexto ao qual a porta mapeia;
- *interface*: nome do ponto de interface de destino. Caso o valor de *component* seja um objeto de mídia, o valor de *interface* deve ser o identificador de uma âncora (de conteúdo ou propriedade) do objeto;[9] caso *component* seja um contexto, o valor de *interface* deve ser o identificador de uma porta do contexto (Figura 8.3). Caso não seja

[8] Portas e seus atributos são definidos no módulo **CompositeNodeInterface** [ABNT, NBR 15606-2, 2007; ITU-T, H.761, 2009].

[9] Âncoras são definidas no Capítulo 9.

especificada, a *interface* se refere ao objeto de destino como um todo (âncora de conteúdo total).

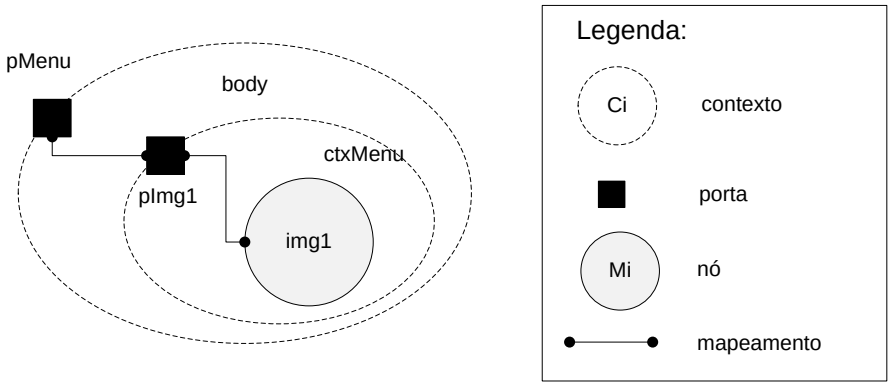


Figura 8.3 Exemplo de portas e contextos, definidos na Listagem 8.3.

```
<body>
        <port id="pMenu" component="ctxMenu" interface="pImg1" />
        <context id="ctxMenu">
                <port id= "pImg1" component= "img1" />
                ...
        </context>
        ...
</body>
```

Listagem 8.3 Exemplo de portas e contextos.

Em todo documento costuma haver pelo menos uma porta de entrada (elemento <port> dentro do <body>), indicando qual é o objeto de mídia ou contexto inicial. Caso se queira iniciar mais do que uma mídia no início da exibição de um contexto, podemos criar mais portas, tal como no exemplo a seguir:

```
<body>
        <port id="pVideoPrincipal" component="videoPrincipal" />
        <port id="pInteratividade" component="imgInteratividade" />
        ...
</body>
```

Observe que esse exemplo indica apenas que as mídias "videoPrincipal" e "imgInteratividade" serão *iniciadas* ao mesmo tempo com o início da exibição do documento, mas não implica que as mídias terminarão ao mesmo

tempo. Para sincronizar o término de apresentação de objetos, é necessário definir um ou mais elos, conforme visto no Capítulo 10.

8.4 Referência Rápida — Mídias, Contextos e Portas

A Tabela 8.2 sumariza os atributos e conteúdo dos elementos <media>, <context> e <port> definidos pelo perfil NCL EDTV. Como sempre, os atributos sublinhados são obrigatórios.

Tabela 8.2 Elementos, Atributos e Conteúdo que Definem Nós de Mídia, Contextos e Portas no Perfil EDTV

Elementos	Atributos	Conteúdo
media	*id*, *src*, *type*, *descriptor*, *refer*, *instance*	(area | property)*
context	*id*, *refer*	(port|property|media|context|link|switch|meta|metadata)*
port	*id*, *component*, *interface*	—

Bibliografia

ABNT NBR 15606-2 2011. Associação Brasileira de Normas Técnicas, "Televisão digital terrestre — Codificação de dados e especificações de transmissão para radiodifusão digital, Parte 2: Ginga-NCL para receptores fixos e móveis — Linguagem de aplicação XML para codificação de aplicações", Sistema Brasileiro de TV Digital Terrestre, *NBR 15606-2*.

ITU-T H.761 2011. Nested Context Language (NCL) and Ginga-NCL for IPTV. ITU-T Rec. H.761. Genebra.

Capítulo 9

Âncoras de Conteúdo e Propriedades

Este capítulo apresenta âncoras de conteúdo e propriedades.[1] As primeiras delimitam trechos de mídias que podem ser utilizados por elos para um sincronismo mais fino do que a mídia inteira. Já as propriedades permitem que os elos manipulem propriedades dos nós durante a execução da aplicação. Ao final deste capítulo, você será capaz de definir diferentes janelas de oportunidade para sincronismo e interação em diferentes momentos de apresentação de uma mesma mídia, bem como definir propriedades de objetos de mídia, que poderão ser alteradas por elos, como sua posição e suas dimensões, durante a execução da aplicação.

[1] Âncoras de conteúdo e propriedades, bem como seus atributos, são definidos nos módulos **MediaContentAnchor** e **PropertyAnchor,** respectivamente ABNT, NBR 15606-2, 2011 e ITU-T, H.761, 2011.

9.1 Âncoras de Conteúdo

Uma *âncora de conteúdo* define um trecho da mídia (intervalo de tempo, região no espaço ou ambos) como uma interface que poderá ser utilizada como ponto de ativação ou ação de elos. Um trecho de mídia é um conjunto de *unidades de informação* (*information units*) contíguas de um objeto. A definição dessas unidades de informação depende do tipo de mídia. As unidades de informação de um vídeo, por exemplo, podem ser os quadros (*frames)* do vídeo. Para definir um trecho de uma mídia dinâmica como um vídeo, por exemplo, uma âncora define geralmente o período de tempo, em segundos, que delimita um trecho do vídeo desejado. Alternativamente, pode definir um trecho do vídeo através de um intervalo de quadros.

As âncoras de conteúdo são utilizadas comumente para sincronizar mídias, bem como para delimitar uma janela de interação, que é, na verdade, uma sincronização, mas com a intervenção do usuário. Uma âncora de conteúdo é definida como um elemento <area> dentro do elemento <media>.

```
<media ...>
               <area ... />
               <area ... />
               ...
       ...
</media>
```

Na Listagem 9.1, são definidas três âncoras de conteúdo para um nó de vídeo.

```
<media id="video1" src="media/video1.mpg" descriptor="dVideo1">
       <!-- âncoras de conteúdo no vídeo -->
       <area id="a1" begin="5s" end="9s" />
       <area id="a2" begin="10s" end="14s" />
       <area id="a3" begin="15s" end="19s" />
</media>
```

Listagem 9.1 Definição de âncoras de conteúdo em uma mídia do tipo vídeo.

Na visão estrutural, as âncoras são ilustradas por pontos de interface em um objeto de mídia, conforme ilustrado pela Figura 9.1.

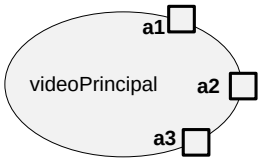


Figura 9.1 Âncoras de uma mídia na visão estrutural.

A NCL define os seguintes atributos de âncora de conteúdo:

- *id*: identificador único, obrigatório, da âncora, que segue a mesma regra de formação para o atributo *id* definida no Capítulo 5;
- *begin*: início da âncora de uma mídia contínua. Pode ser definido em segundos, no formato "99.9s", ou no formato Hora":"Minuto":"Segundo"."Fração, onde Hora é um inteiro no intervalo [0,23], Minuto é um inteiro no intervalo [0,59], Segundo é um inteiro no intervalo [0,59] e Fração é um inteiro positivo;
- *end*: término da âncora de uma mídia contínua. Pode ser definido em segundos, no formato "99.9s", ou no formato Hora":"Minuto":"Segundo"."Fração, onde Hora é um inteiro no intervalo [0,23], Minuto é um inteiro no intervalo [0,59], Segundo é um inteiro no intervalo [0,59] e Fração é um inteiro positivo;[2]
- *first*: início da âncora de uma mídia contínua. Pode ser definido em quadros, no formato "32f", em amostras, no formato "32s" ou em NPT (*Normal Play Time*), no formato "32npt";
- *last*: término da âncora de uma mídia contínua. Pode ser definido em quadros, no formato "32f", em amostras, no formato "32s" ou em NPT (*Normal Play Time*), no formato "32npt";
- *beginText, endText*: texto da âncora no arquivo de origem (atributo válido apenas para mídias de texto);
- *beginPosition, endPosition*: ocorrência do texto (especificado pelo atributo *beginText, ou endText*, respectivamente) da âncora no arquivo de origem (atributo válido apenas para mídias de texto);
- *coords*: coordenadas em pixels dos vértices do polígono que determina a âncora espacial (atributo válido para mídias visuais), no formato "$X_1,Y_1, X_2,Y_2,...,X_n,Y_n$";
- *label*: identificador da âncora no arquivo de origem, seguindo a interpretação dada pela ferramenta de exibição.
- *clip*: identificador de um trecho de uma cadeia temporal de um objeto hipermídia declarativo (apresentado no Capítulo 14), seguindo a interpretação dada pela ferramenta de exibição do objeto.

---

[2] Para o elemento <media> do tipo "application/x-ncl-time", os atributos *begin* e *end* devem ser especificados de acordo com a seguinte sintaxe:
Ano":"Mês":"Dia":"Hora":"Minutos":"Segundos"."Fração
de acordo com a zona do tempo do país. O formatador NCL é responsável por traduzir esse valor para um valor correspondente de UTC.

Alguns valores *default* para os atributos do elemento <area> que definem uma âncora de conteúdo são assumidos por um formatador NCL. Se o atributo *begin* for definido, mas o atributo *end* não for especificado, o final de toda a apresentação do conteúdo de mídia é considerado como encerramento da âncora. Por outro lado, se o atributo *end* for definido sem uma definição explícita de *begin*, o início de toda a apresentação do conteúdo de mídia é considerado como o início da âncora. Comportamento semelhante é esperado com relação aos atributos *first* e *last*. Comportamento semelhante também é esperado com relação aos atributos *beginText* e *endText*. No caso de um elemento <media> do tipo application/x-ncl-time, os atributos *begin* e *end* devem obrigatoriamente ser especificados.

No NCM (ver Capítulo 2), todo objeto de mídia ou de contexto tem uma âncora de conteúdo representando o conjunto de todas as unidades de informação que compõem o conteúdo do objeto. Essa âncora é chamada de *âncora de conteúdo total* (*whole-content anchor*) e é declarada por *default* nos documentos NCL. Toda vez que um componente NCL é referenciado sem a especificação de uma de suas âncoras, a âncora de conteúdo total deve ser assumida.

A Listagem 9.2 exemplifica trechos de código NCL com a definição de várias âncoras de conteúdo para vários objetos de mídia, algumas dessas definições omitindo atributos para os quais são assumidos valores *default* anteriormente mencionados.

```
<media id="film" type="video/mpeg" src="ginga.mp4" descriptor="vDesc">
        <area id="anchor1" begin="10s" end="40s" />
        <area id="anchor2" first="100f" end="200f" />
        <area id="anchor3" begin="00:05:00.0" />
        <area id="anchor4" end="1200f" />
</media>

<media id="music" type="audio/mpeg" src="../media/samba.mp3"V
                                              descriptor="audioDesc">
        <area id="anchor5" end="50s" />
        <area id="anchor6" first="8000s" end="80000s" />
</media>

<media id="lyrics" type="text/plain" src="../media/letra.txt"
                                              descriptor="textoDesc">
        <area    id="anchor7"    beginText="mulata"    beingPosition="2"
                                 endText="mulata" endPosition="2"/>
</media>

<media id="photo" type="image/jpg" src="noel.jpg" descriptor="iDesc">
        <area id="anchor8" coords="10,100,110,100,110,200,10,200" />
</media>

<media id="time" type="application/x-ncl-time">
        <area id="anchor9" begin="2009:05:21:15:00:00.0"
                                        end="2009:05:21:15:30:00.0" />
</media>

<media id="animation" type="application/x-ncl-nclua"
                        src="animacao.lua" descriptor="animacaoDesc">
        <area id="anchor10" label="danca" />
</media>
```

Listagem 9.2 Âncoras de conteúdo.

Para o objeto de mídia "film" foram definidas quatro âncoras. A âncora "anchor1" inicia a 10 segundos do início de exibição do objeto de mídia "film" e termina a 50 segundos desse início. A âncora "anchor2" é semelhante, e inicia em 100 quadros e termina em 200 quadros. Já a âncora "anchor3" inicia a 5 minutos do início do objeto de mídia "film" e termina apenas quando o objeto de mídia termina de exibir todo o seu conteúdo. E a âncora "anchor4", por sua vez, inicia com o início de exibição do objeto de mídia "film" e termina a 1.200 quadros do início dessa exibição.

O objeto de mídia "music" define duas âncoras: "anchor5", de 0 a 50 segundos, e "anchor6", de 8.000 a 80.000 amostras do arquivo de áudio.

Para o objeto de mídia "lyrics", a "anchor7" foi definida para a segunda ocorrência do texto "mulata". Por exemplo, a âncora seria definida para a palavra indicada em negrito no seguinte exemplo de conteúdo do objeto "lyrics".

"Ai, mulata assanhada

Que passa com graça

Fazendo pirraça

Fingindo inocente

Tirando o sossego da gente

Ai, **mulata** se eu pudesse

E se meu dinheiro desse

Eu te dava sem pensar

Essa terra, este céu, este mar

E ela finge que não sabe

Que tem feitiço no olhar"

No caso do objeto de mídia "photo", a âncora "anchor8" é definida por coordenadas na tela. Essa definição de âncora por coordenadas é útil para dispositivos com apontador ou tela de toque.

O objeto de mídia "time" define a âncora "anchor9" como iniciando às 15 horas do dia 21 de maio de 2009 e terminando meia hora depois. Podemos observar que, como ocorre em todos os objetos de mídia, a âncora só tem efeito caso o objeto de mídia tenha sido iniciado.

Finalmente, o objeto de mídia "animation" define a âncora "anchor10" como sendo identificada pelo rótulo "danca". Cabe ao objeto imperativo

"animacao.lua" tratar os eventos disparados sobre essa âncora, como será visto no Capítulo 17.

9.2 Propriedades

As *âncoras de propriedade* ou simplesmente *propriedades* definem propriedades ou grupos de propriedades de um objeto de mídia ou de contexto como interfaces que poderão ser manipuladas pelos elos. Diversas propriedades dos objetos de mídia podem ser manipuladas por elos (por exemplo: volume de áudio de um objeto de áudio, coordenadas e dimensões de exibição de um objeto de mídia visual, grau de transparência etc.). Algumas propriedades, no entanto, como o identificador do objeto de mídia, não podem ser alteradas.

A NCL define os seguintes atributos de âncora de propriedade:

- *name*: nome da propriedade ou grupo de propriedades;
- *value*: valor inicial atribuído à propriedade ou grupo de propriedades;
- *externable*: valor booleano que define se a propriedade é visível para relacionamentos ou não.

O atributo *value* de um elemento <property> é opcional e define um valor inicial para a propriedade declarada como *name*. Quando o valor não é especificado, a propriedade assume como valor inicial aquele definido nos atributos homônimos do descritor ou região associados ao objeto onde a propriedade foi definida. Quando o atributo *value* é especificado, ele tem prioridade sobre o valor definido nos atributos homônimos do descritor ou região associados ao nó.

Todas as propriedades (e seus valores iniciais) de um objeto NCL podem ser definidas apenas pelos seus elementos <property>. Os elementos <descriptor>, <descriptorParam> e <region> são apenas uma opção a mais (opção de reúso) para a definição dos valores iniciais das propriedades.

É possível ter exibidores de documentos NCL (formatadores) que definam algumas propriedades de nós implicitamente. Entretanto, em geral, é de boa prática definir explicitamente as interfaces a serem manipuladas, por segurança. Pela especificação do Ginga-NCL [ABNT, NBR 15606-2, 2011; ITU-T, H.761, 2011], todas as interfaces que forem manipuladas por elos devem ser explicitamente definidas por meio de elementos <property> com o atributo *externable* igual a "true". Quando uma propriedade é definida por meio dos elementos <descriptor>, <descriptorParam> e <region> o atributo *externable* recebe o valor "false" por *default*. Quando uma propriedade é

definida por meio do elemento <property> o atributo *externable* recebe o valor "true" por *default*, se não declarado.

Os elementos <body>, <context> e <media> podem ter várias propriedades embutidas (pré-definidas). Exemplos dessas propriedades são: *top, left, bottom, right, width, height, plan, explicitDur, background, transparency, visible, fit, scroll, style, soundLevel, balanceLevel, trebleLevel, bassLevel, fontColor, fontFamily, fontStyle, fontSize, fontVariant, fontWeight, reusePlayer, playerLife* etc..

Se as propriedades *left, right, top, bottom, width* ou *height* de um objeto de mídia tiverem seus valores definidos em percentagem (%) em elementos <property>, as percentagens se referem ao tamanho da tela do dispositivo onde o objeto de mídia será exibido.

Para propriedades relativas a objetos de mídia do tipo áudio (soundLevel, trebleLevel e bassLevel), os vaores definidos em elementos <property> devem ser entendidos como relativos ao volume gravado.

A propriedade *visible* também pode ser associada a um elemento <context> ou ao elemento <body>. Nesses casos, quando a propriedade tiver seu valor igual a "true", vale a especificação de *visible* de cada objeto filho da composição. Quando tiver o seu valor igual a "false", todos os elementos da composição são exibidos de forma invisível.[3]

Algumas propriedades têm o seu valor definido pelo próprio sistema (por exemplo, o middleware Ginga-NCL), como a propriedade *contentId*,[4] associada a um objeto de mídia contínua cujo conteúdo se refere a um fluxo elementar. Inicialmente, *contentId* tem o valor "nulo", mas assim que o objeto é iniciado *contentId* assume o valor do identificador (contido no campo também chamado *contentId*) transportado no descritor de referência NPT (ver Apêndice E). Outro exemplo de propriedade manipulada pelo sistema é a *standby*. Essa propriedade assume o valor "true" enquanto um objeto de mídia contínua, já iniciado e cujo conteúdo se refere a um fluxo elementar, estiver com seu conteúdo temporariamente interrompido por outro conteúdo entrelaçado no mesmo fluxo elementar. Lembramos que, mesmo quando uma propriedade é definida de forma "embutida", se for utilizada em um

---

[3] Em particular, quando um documento tem seu elemento <body> com a propriedade *visible* = "false" e seu evento de apresentação (ver Capítulo 10) no estado="paused", diz-se que a aplicação está em "*espera*" *(stand-by*). No middleware Ginga-NCL, quando uma aplicação entra em *stand-by*, o vídeo principal do serviço volta a ocupar 100% da dimensão da tela em que é exibido e o áudio principal a 100% de seu volume.

[4] No Capítulo 8 mencionamos, também como nota, essa propriedade, descrita em detalhes no Apêndice E. Relembrando o capítulo anterior, streams recebidos como fluxos elementares do fluxo de transporte TS podem vir entrelaçados. Por exemplo, o conteúdo de uma propaganda pode vir em um mesmo fluxo elementar entremeado com o fluxo de um filme. Em um mesmo fluxo elementar, um stream de um mesmo conteúdo de um objeto de mídia é identificado por seu *contentId*.

relacionamento (elo), ela deve ser explicitamente declarada em um elemento <property>.

Um grupo de propriedades de um objeto também pode ser explicitamente declarado como uma única interface para o objeto, isto é, um único elemento <property>. Isso permite que os autores especifiquem o valor de várias propriedades com uma propriedade única. Os seguintes grupos são reconhecidos por qualquer formatador em conformidade com a NCL: *location*, agrupando (*left, top*), nessa ordem; *size*, agrupando (*width, height*), nessa ordem; e *bounds*, agrupando (*left, top, width, height*), nessa ordem. Quando um formatador trata uma alteração no valor de um grupo de propriedades, ele testa a consistência do processo apenas ao seu final.

As palavras *top, left, bottom, right, width, height, plan, baseDeviceRegion, deviceClass, explicitDur, background, transparency, visible, rgbCromakey, fit, scroll, style, soundLevel, balanceLevel, trebleLevel, bassLevel, zIndex, fontColor, fontFamily, fontStyle, fontSize, fontAlign, fontVariant, fontWeight, player, reusePlayer, playerLife, moveLeft, moveRight, moveUp, moveDown, focusIndex, focusBorderColor, selBorderColor, focusBorderWidth, focusBorderTransparency, focusSrc, focusSelSrc, freeze, transIn, transOut, location, size* e *bounds* são palavras reservadas para valores do atributo *name* do elemento <property>. Seus significados são definidos nos Capítulos 6 e 7, e seus valores default são dados na Tabela 9.1

Tabela 9.1 Alguns nomes reservados para propriedades e seus valores default

Propriedade	***Default***
top, left, bottom, right, width, height	Se qualquer valor dessas propriedades não for definido e não puder ser inferido das regras definidas pela especificação NCL, ele deve obrigatoriamente assumir o valor "0".
location	Veja primeira linha da tabela
size	Veja primeira linha da tabela
bounds	Veja primeira linha da tabela
plan	"video",para objeto de mídia com o atributo *src* referindo um PES de um fluxo TS, "graphic", para todos os outros casos.
baseDeviceRegion	Não existe default.
deviceClass	A mesma classe de dispositivo que executa o

	formatador NCL.
explicitDur	Para mídias contínuas deve assumer o valor da duração da apresentação natural do conteúdo, caso contrário deve assumir o valor “nill”
background	transparent
visible	true
transparency	0%
rgbChromakey	nill
fit	fill
scroll	none
style	nill
soundLevel, trebleLevel, bassLevel	1
balanceLevel	0
zIndex	0
fontColor	white
fontAlign	left
fontFamily	
fontStyle	normal
fontSize	Não existe default.
fontVariant	normal
fontWeight	normal
player	Não existe default.
reusePlayer	false
playerLife	close
moveLeft, moveRight, moveUp, moveDown, focusIndex	nill
focusBorderColor	O valor definido por *default.focusBorderColor*
selBorderColor	O valor definido por *default.selBorderColor*
focusBorderWidth	O valor definido por *default.focusBorderWidth*

focusBorderTransparency	O valor definido por *default.focusTransparency*
focusSrc, focusSelSrc	nill
freeze	false
transIn, transOut	string vazia

Note que o valor inicial de uma propriedade de um objeto de mídia pode ser definido em um elemento <region> associado ao objeto, em um elemento <descriptor> (ou em seu elemento filho <descriptorParam> associado ao objeto) ou em um elemento <property> definido como filho do elemento <media> que define esse objeto. No caso de o valor inicial ser atribuído por mais de uma forma, o valor definido no elemento <property> tem precedência sobre o valor definido pelo <descriptor> (ou pelo <descriptorParam>) que, por sua vez, tem precedência sobre o valor definido no elemento <region>.

A Listagem 9.3 define quatro propriedades para um nó de vídeo, além de uma âncora de conteúdo. As duas primeiras propriedades assumem como valores iniciais aqueles especificados pelos atributos do mesmo nome do elemento <region> ao qual o objeto "videoPrincipal" está associado. A terceira propriedade especifica o valor inicial de "200" pixels para o atributo *width* e "100" pixels para o atributo *height*, mesmo que tenham sido atribuídos outros valores iniciais por meio de um elemento <region> associado ao objeto "videoPrincipal" ou por meio de um elemento <descriptorParam> filho do elemento <descriptor> "dVideo1". A quarta propriedade define uma nova propriedade "rate" com o valor inicial igual a "15".

```
<media id="videoPrincipal" src="media/video1.mpg" descriptor="dVideo1">
       <!-- âncoras de propriedade manipuladas pelos elos -->
       <property name="top" />
       <property name="left" />
       <property name="size" value="200,100" />
       <property name="rate" value="15" />

       <!-- âncora de conteúdo no vídeo -->
       <area id="aVideo1Imagem1" begin="3s" end="8s" />
</media>
```

Listagem 9.3 Âncoras de propriedade e de conteúdo.

9.3 Objeto de Mídia do Tipo "application/x-ncl-settings"

Como vimos no Capítulo 9, existe um objeto de mídia especial definido com o tipo **application/x-ncl-settings**, também chamado de objeto *settings*. Esse tipo de objeto é utilizado para agrupar propriedades globais definidas pelo autor do documento e *variáveis de ambiente* (também chamadas de *variáveis de contexto*) reservadas.

A Listagem 9.4 ilustra a definição de um objeto *settings* com duas propriedades. A propriedade "system.language", mantida pelo formatador NCL, especifica o idioma para o qual a apresentação será feita. Já a propriedade "interaction" é definida pelo autor do documento, nesse caso para indicar se a interatividade é permitida (valor "true") ou não (valor "false"). Podemos observar que a interatividade é permitida inicialmente, conforme indicado pelo atributo *value*="true". A definição de um valor para o atributo *value* equivale a definir um valor *default* para a propriedade.

```
<media id="nodeSettings" type="application/x-ncl-settings">
        <property name="system.language" />
        <property name="interaction" value="true" />
</media>
```

Listagem 9.4 Nó *settings* com duas propriedades que podem ser manipuladas pela aplicação.

Devemos novamente observar que um documento NCL pode conter apenas um objeto do tipo *settings*.

9.4 Variáveis de Ambiente

O middleware Ginga define diversas variáveis de ambiente, que funcionam como propriedades com diferentes escopos e tipos de acesso. Alguns tipos de variáveis de ambiente podem ter seus valores alterados por aplicações NCL; outros, apenas lidos. As variáveis de ambiente são divididas em seis grupos:

- **system**: variáveis gerenciadas pelo sistema receptor. Elas podem ser lidas, mas não podem ter seus valores alterados por uma aplicação NCL, nem por um procedimento Lua nem por um procedimento Xlet. As aplicações nativas do receptor podem alterar os valores dessas variáveis. Os valores dessas variáveis devem persistir por todo o ciclo de vida do receptor. Essas variáveis têm o prefixo "**system.**", como por exemplo "**system.language**";

- **user**: variáveis gerenciadas pelo sistema receptor. Elas podem ser lidas, mas não podem ter seus valores alterados por uma aplicação NCL, nem por um procedimento Lua nem por um procedimento Xlet. As aplicações nativas do receptor podem alterar os valores dessas variáveis. Os valores dessas variáveis devem persistir por todo o ciclo de vida do receptor. Essas variáveis têm o prefixo "**user.**", como por exemplo "**user.location";**

- ***default***: variáveis gerenciadas pelo sistema receptor. Elas podem ser lidas e ter seus valores alterados por uma aplicação NCL, por um procedimento Lua ou por um procedimento Xlet. As aplicações nativas do receptor podem alterar os valores dessas variáveis. Os valores dessas variáveis devem persistir por todo o ciclo de vida do receptor. No entanto, a cada troca de canal, os valores dessas variáveis retornam aos seus valores iniciais. Essas variáveis têm o prefixo "**default.**", como por exemplo "**default.focusBorderColor";**

- **service**: variáveis gerenciadas pelo formatador NCL. Elas podem ser lidas e, em geral, ter seus valores alterados por uma aplicação NCL do mesmo serviço. Elas podem apenas ser lidas por um procedimento Lua ou por um procedimento Xlet do mesmo serviço. As modificações sobre os valores devem ser feitas através de comandos NCL. Os valores devem persistir por todo o ciclo de vida do serviço. Essas variáveis têm o prefixo "**service.**", como por exemplo "**service.currentFocus**";

- **SI**: variáveis gerenciadas pelo middleware Ginga. Elas podem ser lidas, mas não podem ter seus valores alterados por uma aplicação NCL, nem por um procedimento Lua nem por um procedimento Xlet. Os valores dessas variáveis persistem durante todo o tempo de sintonia de um canal. Essas variáveis têm o prefixo "**si.**", como por exemplo "**si.channelNumber**";

- **channel**: variáveis gerenciadas pelo formatador NCL. Elas podem ser lidas e ter seus valores alterados por uma aplicação NCL do mesmo canal. Elas podem apenas ser lidas por um procedimento Lua ou por um procedimento Xlet do mesmo canal. As modificações sobre os valores devem ser feitas através de comandos NCL. Os valores devem persistir até a próxima troca de canal. Essas variáveis têm o prefixo "**channel.**";

- **shared**: variáveis gerenciadas pelo formatador NCL. Elas podem ser lidas e ter seus valores alterados por uma aplicação NCL. Elas podem apenas ser lidas por um procedimento Lua ou por um procedimento Xlet do mesmo canal. As modificações sobre os valores devem ser feitas

através de comandos NCL. Os valores dessas variáveis devem persistir por todo o ciclo de vida do serviço que as definiu. Essas variáveis têm o prefixo "**shared.**".

A Tabela 9.2 descreve o significado e os valores possíveis das variáveis de ambiente do grupo `system` no middleware Ginga-NCL.

Tabela 9.2 Variáveis de Ambiente do Grupo `system`

Variável	Significado	Valores possíveis
system.language	Idioma do áudio	Valores definidos pela ISO 639-1
system.caption	Idioma do *closed caption*	Valores definidos pela ISO 639-1
system.subtitle	Idioma das legendas (*subtitle*)	Valores definidos pela ISO 639-1
system.returnBitRate(i)	Taxa do canal interativo (i) em Kbps	real
system.screenSize	Tamanho da tela do dispositivo de exibição, em linhas, pixels/linha, quando uma classe não é definida	(inteiro, inteiro)
system.screenGraphicSize	Resolução configurada para o plano gráfico da tela do dispositivo de exibição, em (linhas, pixels/linha), quando uma classe não é definida	(inteiro, inteiro)
system.audioType	Tipo de áudio do dispositivo de exibição, quando uma classe não é definida	"mono" | "stereo" | "5.1"
system.screenSize (i)	Tamanho da tela do dispositivo (i) em (linhas, pixels/linha)	(inteiro, inteiro)
system.screenGraphicSize(i)	Resolução definida para o plano gráfico do dispositivo (i) em (linhas, pixels/linha)	(inteiro, inteiro)
system.audioType(i)	Tipo de áudio do dispositivo (i)	"mono" | "stereo" | "5.1"
system.devNumber(i)	Número de dispositivos de exibição cadastrados na classe (i)	inteiro
system.classType(i)	Tipo da classe (i)	("passive" | "ative")

system.parentDeviceRegion(i)	Identifica o element <region> em uma outra <regionBase> associada ao dispositivo pai que cria o mapa de bits enviado à classe passiva (i); nesta região, o mapa de bits também deve ser exibido	Cinco números separados por vírgulas, cada um seguindo as regras de valores associados para as propriedades *left*, *top*, *width, height,* e *zIndex*, respectivamente,
system.info(i)	Lista de exibidores de mídia da classe (i) de dispositivos de exibição	string
system.classNumber	Número de classes de dispositivos de exibição definidas	inteiro
system.CPU	Desempenho da CPU em MIPS	real
system.memory	Espaço de memória em Mbytes	inteiro
system.operatingSystem	Tipo do sistema operacional	string
system.javaConfiguration	Tipo e versão da configuração Java suportada pelo receptor JVM	string (tipo imediatamente seguido da versão, como por exemplo: “CLDC1.1”)
system.javaProfile	Tipo e versão do perfil Java suportado pelo receptor JVM	string (tipo imediatamente seguido da versão, como por exemplo: “MIDP2.0”)
system.luaVersion	Versão da máquina Lua suportada pelo receptor	string
system.ncl.version	Versão da linguagem NCL	string
system.GingaNCL.version	Versão do ambiente Ginga-NCL	string
system.xxx	Qualquer variável com o prefixo “system” deve ser reservada para uso futuro	

A Tabela 9.3 descreve o significado e os valores possíveis das variáveis de ambiente do grupo user no middleware Ginga-NCL.

Tabela 9.3 Variáveis de Ambiente do Grupo user

Variável	Significado	Valores possíveis
user.age	Idade do usuário	inteiro
user.location	Localização do usuário (código de endereçamento postal)	string
user.genre	Sexo do usuário	“m” | “f”
user.xxx	Qualquer variável com o prefixo “user” deve ser reservada para uso futuro	

A Tabela 9.4 descreve o significado e os valores possíveis das variáveis de ambiente do grupo *default* no middleware Ginga-NCL.

Tabela 9.4 Variáveis de Ambiente do Grupo *default*

Variável	Significado	Valores possíveis
default.focusBorderColor	Cor *default* aplicada à moldura de um elemento em foco	“white” | “black” | “silver” | “gray” | “red” | “maroon” | “fuchsia” | “purple” | “lime” | “green” | “yellow” | “olive” | “blue” | “navy” | “aqua” | “teal”
default.selBorderColor	Cor *default* aplicada à moldura de um elemento em foco ao ser ativado	“white” | “black” | “silver” | “gray” | “red” | “maroon” | “fuchsia” | “purple” | “lime” | “green” | “yellow” | “olive” | “blue” | “navy” | “aqua” | “teal”
default.focusBorderWidth	Largura *default* (em pixels) aplicada à moldura de um elemento em foco	integer
default.focusBorderTransparency	Transparência *default* aplicada à moldura de um elemento em foco	valor real entre 0 e 1 (p. ex., “0.3”), ou valor real entre 0 e100 seguido do caractere “%” (p. ex., “30%”), onde “1” e “100%” significam transparência total e “0” e “0%” significam nenhuma transparência
default.xxx	Qualquer variável com o prefixo “*default*” deve ser reservada para uso futuro	

A Tabela 9.5 descreve o significado e os valores possíveis das variáveis de ambiente do grupo service no middleware Ginga-NCL.

Tabela 9.5 Variáveis de Ambiente do Grupo service

Variável	Significado	Valores possíveis
service.currentFocus	O valor de *focusIndex* do elemento <media> em foco	inteiro
service.currentKeyMaster	Identificador (*id*) do elemento <media> que controla as teclas de navegação; caso esse elemento <media> não esteja em pausa, o controle das navegações por teclas pertence ao formatador NCL	string
service.xxx	Qualquer variável com o prefixo “service” deve ser reservada para uso futuro	

A Tabela 9.6 descreve o significado e os valores possíveis das variáveis de ambiente do grupo si no middleware Ginga-NCL.

Tabela 9.6 Variáveis de Ambiente do Grupo SI

Variável	Significado	Valores possíveis
si.numberOfServices	Número de serviços disponíveis, no país, para o canal sintonizado	inteiro
si.numberOfPartialServices	Número de serviços 1-seg disponíveis, no país, para o canal sintonizado	inteiro
si.channelNumber	Número do canal sintonizado	inteiro
si.xxx	Qualquer variável com o prefixo “si” deve ser reservada para uso futuro	

A Tabela 9.7 descreve o significado e os valores possíveis das variáveis de ambiente do grupo channel no middleware Ginga-NCL.

Tabela 9.7 Variáveis de Ambiente do Grupo channel

Variável	Significado	Valores possíveis
channel.keyCapture	Requisição de teclas alfanuméricas por aplicações NCL	string
channel.virtualKeyboard	Requisição do teclado virtual por aplicações NCL	(true | false)
channel.keyboardBounds	Região de exibição do teclado virtual (left, top, width, height)	(inteiro, inteiro, inteiro, inteiro)
channel.xxx	Qualquer variável com o prefixo “channel” deve ser reservada para uso futuro	

9.5 Referência Rápida — Âncoras de Conteúdo e Propriedades

A Tabela 9.8 apresenta o elemento e os atributos que definem âncoras de conteúdo e propriedades. Como sempre, ao atributos sublinhados são obrigatórios.

Tabela 9.8 Elemento e Atributos que Definem Âncoras de Conteúdo e Propriedades no Perfil EDTV

Elementos	Atributos	Conteúdo
area	*id, coords, begin, end, beginText, beginPosition, endText, endPosition first, last, label, clip*	—
property	*name, value, externable*	—

Bibliografia

ABNT NBR 15606-2 2011. Associação Brasileira de Normas Técnicas, "Televisão digital terrestre — Codificação de dados e especificações de transmissão para radiodifusão digital, Parte 2: Ginga-NCL para receptores fixos e móveis — Linguagem de aplicação XML para codificação de aplicações", Sistema Brasileiro de TV Digital Terrestre, *NBR 15606-2*.

ITU-T H.761 2011. Nested Context Language (NCL) and Ginga-NCL for IPTV. ITU-T Rec. H.761. Genebra.

Capítulo 10

Sincronização: Conectores e Elos

Este capítulo apresenta os elos e conectores[1] que definem o sincronismo e, em particular, a interatividade entre os objetos de uma aplicação NCL. Ao final deste capítulo, você estará apto a criar aplicações com diversos comportamentos de sincronismo entre objetos, tais como iniciar um objeto quando outro terminar, iniciar um objeto junto com o início de outro, terminar um objeto junto com outro, exibir um objeto de mídia quando uma tecla é pressionada pelo usuário, redimensionar um objeto de mídia quando outro é iniciado, entre outros.

[1] Elos e seus atributos são definidos no módulo **Linking**, ao passo que conectores e seus atributos são definidos no módulo **ConnectorBase** [ABNT, NBR 15606-2, 2011; ITU-T, H.761, 2011].

10.1 Introdução

Um elo (elemento <link>) associa objetos por meio de um conector (elemento <causalConnector>), que define a semântica da associação. Os elos são definidos como descendentes do elemento <body>, ao passo que os conectores são definidos numa base de conectores (elemento <connectorBase>) filha do elemento <head> da aplicação NCL. Este capítulo detalha os elementos filhos e atributos de conectores de sincronismo de apresentação, de interatividade, de manipulação de propriedades, bem como os elementos e atributos de elos.

10.2 Base de Conectores

Os conectores são definidos em uma base de conectores (elemento <connectorBase>), que contém como único atributo o identificador (atributo *id*) da base. Uma base de conectores pode possuir os seguintes elementos filhos:

- <causalConnector>: define um conector propriamente dito, conforme descrito na próxima seção;
- <importBase>: permite importar uma base de conectores de um outro arquivo. Esse elemento é descrito no Capítulo 12, sobre reúso em aplicações NCL.

Sendo assim, uma base de conectores pode ser definida conforme a seguinte estrutura:

```
<connectorBase id=”meusConectores”>
        <importBase ... />
        <importBase ... />

        <causalConnector id=“onBeginStart”>
                ...
        </causalConnector>
        <causalConnector id=“___”>
                ...
        </causalConnector>
</connectorBase>
```

10.3 Conectores

Como apresentado na Parte I deste livro, em NCL o sincronismo não é feito por marcas de tempo (*timestamps*), mas sim por relações de causalidade

definidas nos conectores (*connectors*).[2] Um elemento <causalConnector> representa uma relação causal que pode ser utilizada por elementos <link> na definição de relacionamentos entre objetos. Em uma relação causal, uma *condição* deve ser satisfeita para que *ações* possam ser disparadas. Um <causalConnector> especifica, através de seus elementos filhos, um conjunto de pontos da interface, chamados *papéis* (*role*), conforme ilustrado pela Figura 10.1.

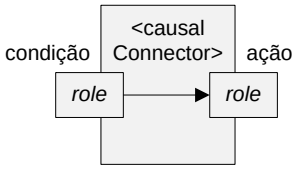


Figura 10.1 Ilustração de um conector causal (elemento <causalConnector>) com papéis (*role*) de condição e ação.

Um elemento <link> refere-se a um <causalConnector> e define um conjunto de mapeamentos (elementos <bind> filhos do elemento <link>), que associam interfaces de objetos de uma aplicação NCL a um papel do conector utilizado, conforme ilustrado pela Figura 10.2.

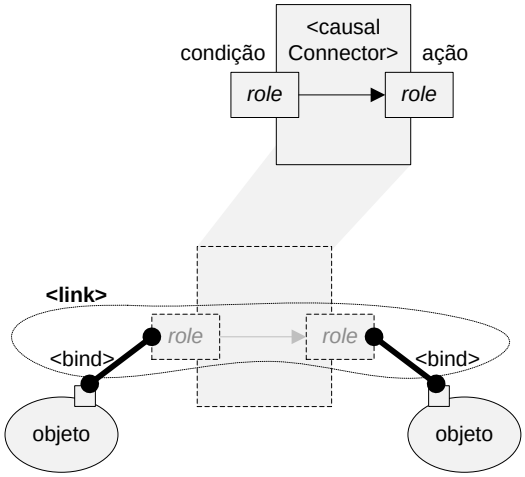


Figura 10.2 Ilustração de elo (elemento <link>).

[2] Bases de conectores, conectores e seus atributos são especificados nos módulos **ConnectorBase, ConnectorCommonPart, CausalConnector, CausalConnectorFunctionality, ConnectorCausalExpression, ConnectorAssessmentExpression** e **ConnectorTransitionAssessment** [ABNT, NBR 15606-2, 2011; ITU-T, H.761, 2011].

Os elementos filhos de um <causalConnector> são:

- <connectorParam>: define parâmetros de conector cujos valores deverão ser definidos pelos elos que utilizam o conector;
- <simpleCondition> e <compoundCondition>: definem as condições simples ou compostas de ativação do elo que utiliza o conector;
- <simpleAction> e <compoundAction>: definem as ações simples ou compostas que são realizadas quando o elo que utiliza o conector é ativado.

Podemos definir um conector bastante simples para iniciar uma mídia junto com outra, como o conector "onBeginStart" ilustrado na Figura 10.3 e definido pela Listagem 10.1.

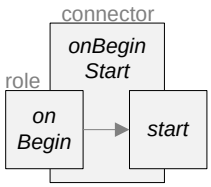


Figura 10.3 Conector "onBeginStart", que define o comportamento: quando a mídia associada ao papel **onBegin** iniciar, a mídia associada ao papel **start** também iniciará.

```
<causalConnector id="onBeginStart">
       <simpleCondition role="onBegin" />
       <simpleAction role="start" />
</causalConnector>
```

Listagem 10.1 Definição do conector "onBeginStart".

Na Listagem 10.1, o conector define a condição (<simpleCondition>) sob a qual o elo pode ser ativado e a ação (<simpleAction>) que será realizada quando o elo for ativado. Um conector deve possuir pelo menos uma condição e uma ação. Cada condição ou ação é associada a um papel (*role*).

Em geral, o comportamento de um conector pode ser lido do seguinte modo: "quando condição/evento, então ação". Por exemplo, no caso do conector "onBeginStart" da Listagem 10.1, lemos: 'quando <algum objeto ligado ao papel "onBegin"> iniciar, inicia também <algum objeto ligado ao papel "start">'. Frisando mais uma vez, o conector define os papéis de condição e ação (p. ex., iniciar, terminar) e seu comportamento; cabe aos elos ligar objetos a esses papéis.

Os nomes dos papéis “onBegin” e “start” fazem parte de um conjunto de papéis predefinidos, cujo comportamento é interpretado corretamente por um formatador NCL, como o implementado pelo middleware Ginga-NCL. Por conveniência, para cada condição ou ação que pode ser definida num conector pode existir um nome de papel predefinido reconhecido pelo formatador NCL. A Tabela 10.1 apresenta os papéis de condição predefinidos e a Tabela 10.2 apresenta os papéis predefinidos de ação que devem ser reconhecidos por qualquer formatador NCL.

Tabela 10.1 Papéis Predefinidos de **Condição**

Papel	Descrição (Quando o Elo Será Ativado)
onBegin	Quando a apresentação for iniciada...
onEnd	Quando a apresentação for terminada (naturalmente ou por uma ação **stop**)...
onAbort	Quando a apresentação for abortada...
onPause	Quando a apresentação for pausada...
onResume	Quando a apresentação for retomada após uma pausa...
onSelection ou onBeginSelection	Quando uma tecla (a ser especificada) for pressionada enquanto o objeto ligado a esse papel estiver sendo apresentado ou quando a tecla OK for pressionada enquanto o objeto ligado a esse papel estiver com o foco, ou quando um dispositivo apontador, por exemplo o mouse, selecionar o objeto em apresentação ligado a esse papel...
onEndSelection	Quando uma tecla (a ser especificada) terminar de ser pressionada enquanto o objeto ligado a esse papel estiver sendo apresentado ou quando a tecla OK terminar de ser pressionada enquanto o objeto ligado a esse papel estiver com o foco, ou quando um dispositivo apontador, por exemplo o mouse, terminar a seleção do objeto em apresentação ligado a esse papel...
onBeginAttribution	Logo antes que um valor (a ser especificado) seja atribuído a propriedades ligadas a esse papel...
onEndAttribution	Logo após um valor (a ser especificado) ter sido atribuído a propriedades ligadas a esse papel...
onAbortAttribution	Quando a atribuição for abortada...
onPauseAttribution	Quando a atribuição for pausada...
onResumeAttribution	Quando a atribuição for retomada após uma pausa...

Tabela 10.2 Papéis Predefinidos de **Ação**

Papel	Descrição (Ação a Ser Realizada Quando o Elo for Ativado)
start	... inicia a apresentação dos objetos associados a esse papel
stop	... termina a apresentação dos objetos associados a esse papel
abort	... aborta a apresentação dos objetos associados a esse papel
pause	... pausa a apresentação do objeto associados a esse papel
resume	... retoma a apresentação do objeto associados a esse papel (caso esteja em pausa)
set	... estabelece um valor (a ser especificado) às propriedades associadas a esse papel
startAttribution	... inicia a atribuição de um valor (a ser especificado) às propriedades associadas a esse papel
stopAttribution	... termina a atribuição
abortAttribution	... aborta a atribuição
pauseAttribution	... pausa a atribuição
resumeAttribution	... retoma a atribuição

Mais precisamente, as relações definidas por elementos <causalConnector> são baseadas em eventos. Um evento é uma ocorrência no tempo que pode ser instantânea ou ter duração mensurável.

A NCL, em sua versão 3.0, define os seguintes tipos de eventos:

- *evento de apresentação*: apresentação de um subconjunto das unidades de informação (âncora de conteúdo) de um objeto de mídia. Um caso particular é a âncora de conteúdo total (ver Capítulo 9). Eventos de apresentação também podem ser definidos sobre nós de composição (representados por um elemento <body>, <context> ou <switch>), representando a apresentação das unidades de informação de qualquer nó dentro do nó de composição;
- *evento de seleção*: seleção de um subconjunto das unidades de informação (âncora de conteúdo) de um objeto de mídia sendo apresentado e visível;

- *evento de atribuição*: atribuição de um valor a uma propriedade de um objeto, que deve ser declarada explicitamente em um elemento <property>, filho do objeto;
- *evento de composição*: apresentação da estrutura de um nó de composição (representado por um elemento <body>, <context> ou <switch>). Os eventos de composição são utilizados para apresentar o mapa da composição (organização da composição). Essa funcionalidade é opcional no perfil EDTV e BDTV.

Cada evento define uma máquina de estados controlada pelo formatador NCL, apresentado na Figura 10.4

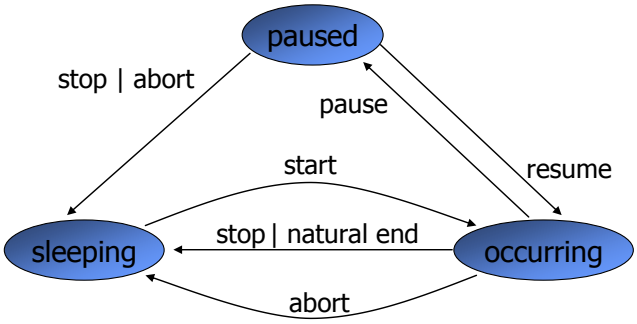


Figura 10.4 Máquina de estados de eventos.

Voltando aos papéis predefinidos: os papéis de condição **"onBegin", "onEnd", "onAbort", "onPause" e "onResume",** assim como os papéis de ação "start", "stop", "abort", "pause" e "resume" estão relacionados às possíveis transições de estados de eventos de apresentação de âncoras de conteúdo, conforme ilustrado na Figura 10.4.

Os papéis de condição "onSelection", "onBeginSelection" "onEndSelection" estão relacionados às possíveis transições de estados de eventos de seleção de âncoras de conteúdo. Eles são ligados à interatividade, realizada por meio de dispositivos de entrada, como o controle remoto da TV; vamos deixar sua discussão específica para a Seção 10.5. Já os papéis de condição "onBeginAttribution", "onEndAttribution", onAbortAttribution, onPauseAttribution e onResumeAttribution, bem como os papéis de ação "set", "startAttribution", "stopAttribution", "abortAttribution", "pauseAttribution" e "resumeAttribution"estão relacionados aos eventos de atribuição, isto é, à manipulação de valores de propriedades; vamos deixar sua discussão específica para a Seção 10.8.

O papel (atributo *role*) é apenas um dos atributos de uma condição, mas existem outros atributos. A NCL define os seguintes atributos do elemento <simpleCondition>:

- *role*: nome do papel. É atributo obrigatório e deve ser único dentro de um conector. Como dissemos, por conveniência, utilizamos com frequência um dos papéis de condição predefinidos, como os descritos na Tabela 10.1;
- *delay*: tempo decorrido entre a condição ser verdadeira e o elo ser ativado, em segundos, e no formato "9s", tendo como valor *default* "0s";
- *eventType*: tipo de evento associado ao papel da condição. Pode assumir o valor "presentation" (para eventos de apresentação), "selection" (para seleção, p. ex., através de teclas) ou "attribution" (para eventos de atribuição de valor). Caso o valor do atributo *role* seja um dos valores predefinidos na Tabela 10.1, esse atributo assume um valor *default*, como apresentado na Tabela 10.3.
- *transition*: transição na máquina de estados. Pode assumir os valores: "starts", "stops", "aborts", "pauses" ou "resumes", conforme a máquina de eventos de um objeto. Caso o valor do atributo *role* seja um dos valores predefinidos na Tabela 10.1, esse atributo assume um valor *default*, como apresentado na Tabela 10.3;

Tabela 10.3 Valores de atributos *eventType* e *transition* assumidos por *default* quando o atributo *role* usa palavras reservadas em uma condição

Valor de *role*	Valor de *transition*	Valor de *eventType*
onBegin	starts	presentation
onEnd	stops	presentation
onAbort	aborts	presentation
onPause	pauses	presentation
onResume	resumes	presentation
onSelection	starts	selection
onBeginSelection	starts	selection
onEndSelection	stops	selection
onBeginAttribution	starts	attribution
onEndAttribution	stops	attribution
onAbortAttribution	aborts	attribution
onPauseAttribution	pauses	attribution
onResumeAttribution	resumes	attribution

- *key*: código da tecla do controle remoto que ativa o elo, no caso de *eventType* "selection". Pode assumir um dos seguintes valores: "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z", "*", "#", "MENU", "INFO", "GUIDE", "CURSOR_DOWN", "CURSOR_LEFT", "CURSOR_RIGHT", "CURSOR_UP", "CHANNEL_DOWN", "CHANNEL_UP", "VOLUME_DOWN", "VOLUME_UP", "ENTER", "RED", "GREEN", "YELLOW", "BLUE", "BACK", "EXIT", "POWER", "REWIND", "STOP", "EJECT", "PLAY", "RECORD", "PAUSE";
- *min*: cardinalidade mínima do papel, ou seja, o número mínimo de interfaces[3] de objetos que devem ser associadas a esse papel no elo que o utilize. O default é "1";
- *max*: cardinalidade máxima do papel, ou seja, o número máximo de interfaces de objetos que devem ser associadas a esse papel no elo que o utilize. Caso não haja limite no número máximo de interfaces que podem ser associadas ao papel, deve assumir o valor "unbounded". O default é "1";
- *qualifier*: define se todas as condições devem ser satisfeitas por todas as interfaces ligadas ao papel (valor "and") ou se basta que qualquer uma delas seja satisfeita (valor "or"). Esse atributo só tem efeito quando a cardinalidade máxima do papel é maior do que 1. O default é "or".

Também para as ações, o papel é apenas um de seus atributos. A NCL define os seguintes atributos do elemento <simpleAction>:

- *role*: nome do papel. Deve ser único dentro de um conector. Como dissemos, por conveniência, utilizamos com frequência um dos papéis de ação predefinidos, como os descritos na Tabela 10.2;
- *delay*: tempo decorrido entre a ativação do elo e o disparo da ação, em segundos, e no formato "9s". O default é "0s";
- *eventType*: tipo de evento associado ao papel da ação. Pode assumir o valor "presentation" (para eventos de apresentação), "selection"

[3] Até agora definimos as âncoras de conteúdo e as propriedades como interfaces de um objeto de mídia, e as portas e propriedades como interfaces de contextos. Como veremos em outros capítulos, um contexto também pode ter âncoras de conteúdo como interface (esse conceito será extremamente útil quando, na Parte III, estudarmos objetos hipermídia declarativos). Como veremos no próximo capítulo, um objeto switch define ainda um outro tipo de interface, denominada *portSwitch*.

(para seleção, p. ex., através de teclas) ou "attribution" (para eventos de atribuição de valor). Caso o valor do atributo *role* seja um dos valores predefinidos na Tabela 10.2, esse atributo é opcional. Esse atributo assume valores por *default*, como apresentado na Tabela 10.4;

- *actionType*: ação que causa uma transição na máquina de estados do evento. Pode assumir os valores "start", "stop", "abort", "pause" ou "resume". Caso o valor do atributo *role* seja um dos valores predefinidos na Tabela 10.1, esse atributo assume valores por *default*, como apresentado na Tabela 10.4.

Tabela 10.4 Valores de atributos *eventType* assumidos por *default* quando o atributo *role* usa palavras reservadas em uma ação

Valor de *role*	Valor de *actionType*	Valor de *eventType*
start	*start*	*presentation*
stop	*stop*	*presentation*
abort	*abort*	*presentation*
pause	*pause*	*presentation*
resume	*resume*	*presentation*
set	*start*	*attribution*
startAttribution	*start*	*attribution*
stopAttribution	*stop*	*attribution*
abortAttribution	*abort*	*attribution*
pauseAttribution	*pause*	*attribution*
resumeAttribution	*resume*	*attribution*

- *value*: valor a ser atribuído às propriedades associadas ao papel, caso o valor de *eventType* seja "attribution";
- *min*: cardinalidade mínima do papel, ou seja, o número mínimo de interfaces de objetos que devem ser associados a esse papel no elo que o utilize. O default é "1";
- *max*: cardinalidade máxima do papel, ou seja, o número máximo de interfaces de objetos que devem ser associados a esse papel no elo que o utilize. Caso não haja um limite no número máximo de interfaces de objetos que podem ser associadas ao papel, deve assumir o valor "unbounded". O default é "1";

- *qualifier*: define se as ações, em cada interface de objeto ligada ao papel, devem ser disparadas em paralelo (valor "par") ou em sequência (valor "seq"). Esse atributo só tem efeito quando a cardinalidade máxima do papel é maior do que 1. O default é "par";
- *repeat*: esse atributo só é válido no caso de evento de apresentação e no caso de a ação ser "start". Ele define o número de vezes em que a apresentação da âncora de conteúdo deve se repetir, com o intervalo de *repeatDelay* segundos entre cada repetição. O valor *default* é "0";
- *repeatDelay*: tempo em segundos decorrido entre cada repetição da ação, no formato "9s". Esse atributo só tem efeito quando o valor de *repeat* é maior que 0;
- *duration*: tempo de duração de uma atribuição. O default é "0", ou seja, a atribuição é feita instantaneamente. Caso o valor seja maior que zero, o valor da propriedade é gradualmente modificado até chegar ao valor final, cujo incremento é definido pelo atributo *by*;
- *by*: incremento a cada passo de atribuição ao longo do tempo definido por *duration*. Caso seja "indefinite", a atribuição é feita linear e continuamente, conforme a implementação do formatador. Esse atributo só tem efeito quando o atributo *duration* assume um valor maior que zero.

Os atributos *duration* e *by* são utilizados para atingir efeitos de animação, por exemplo, para mover ou redimensionar um objeto na tela gradualmente, conforme detalhado na Seção 10.10.

Como mencionado, palavras reservadas para papéis facilitam a definição de conectores. No entanto, os papéis de um conector não estão restritos àqueles definidos por meio de palavras reservadas. Para definir um papel que não tenha sido predefinido, é necessário definir um valor para o atributo *eventType*.

No caso do elemento <*simpleCondition*>, é necessário definir também um valor para o atributo *transition*. No caso do elemento <*attributeAssessment*>, é necessário definir também um valor para o atributo *attributeType*. Já no caso do elemento <*simpleAction*>, é necessário definir também um valor para o atributo *actionType*.

10.4 Elos

Um elo define o relacionamento de sincronismo propriamente dito entre interfaces de objetos de uma aplicação NCL. Seu comportamento é definido pelo conector que o elo utiliza.

Para fazer a associação de interfaces de objetos com os papéis de um conector, um elo simplesmente utiliza elementos <bind> (ligação), conforme o seguinte esquema:

```
<link xconnector="id_do_conector">
   <bind role="id_de_papel_de_condicao"
                          component="id_de_um_objeto"
                          interface="id_de_uma_interface" />
   <bind role="id_de_papel_de_acao" component="id_de_um_objeto"
                          interface="id_de_uma_interface" />
</link>
```

Como exemplo, a Figura 10.5 apresenta a visão estrutural correspondente a um exemplo de sincronismo de início de apresentação de objetos de mídia, destacando o elo de sincronismo que deve iniciar a apresentação da mídia "imgInteratividade" assim que a apresentação da mídia "videoAbertura" começar.

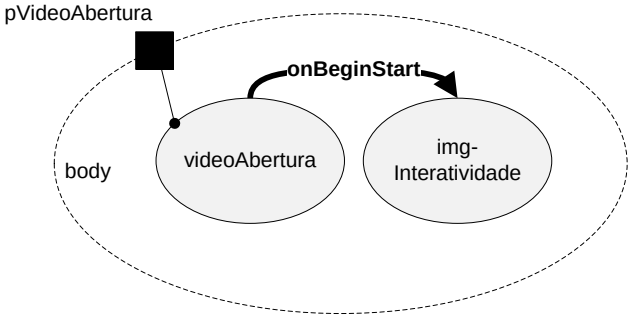


Figura 10.5 Visão estrutural do exemplo, destacando o elo de sincronismo entre as mídias.

Na Figura 10.5, ao utilizar o conector, o elo associa a mídia "videoPrincipal" (a âncora de conteúdo total é assumida por *default*) ao papel "onBegin", e a mídia "imgInteratividade" (novamente com a âncora de conteúdo total assumida por *default*) ao papel "start", conforme ilustrado na Figura 10.6 e pela Listagem 10.2. Como já mencionamos, os conectores são definidos numa base de conectores (elemento <connectorBase> dentro da seção <head>), ao passo que os elos são definidos no núcleo do documento (dentro da seção do elemento <body> ou de algum contexto interno a ela).

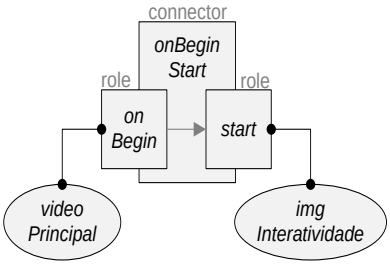


Figura 10.6 Elo que utiliza o conector "*onBeginStart*", ligando as mídias "*videoPrincipal*" ao papel "onBegin" e "*imgInteratividade*" ao papel "start" do conector, respectivamente.

```
<link id="lVideoPrincInicia" xconnector="onBeginStart">
        <bind role="onBegin" component="videoPrincipal" />
        <bind role="start" component="imgInteratividade" />
</link>
```

Listagem 10.2 Código para definição de um elo que utiliza o conector "onBeginStart" para sincronizar o início das mídias "videoPrincipal" e "imgInteratividade", utilizando o connector definido na Figura 10.1

No caso da Listagem 10.2, podemos ler o elo como: Quando iniciar ("onBegin") a apresentação do "videoPrincipal", inicia também ("start") a apresentação da "imgInteratividade".

Analogamente, para sincronizar o término de apresentação das mídias, podemos definir um conector "onEndStop", utilizado por um segundo elo, conforme indicado na Listagem 10.3.

```
<connectorBase>
        <causalConnector id="onEndStop">
                <simpleCondition role="onEnd" />
                <simpleAction role="stop" />
        </causalConnector>
</connectorBase>
                                                        ... trecho da seção <head>
```

```
                                                        ... trecho da seção <body>
<link id="lVideoPrincTermina" xconnector="onEndStop">
        <bind role="onEnd" component="videoPrincipal" />
        <bind role="stop" component="imgInteratividade" />
</link>
```

Listagem 10.3 Código para definição de um conector "*onEndStop*" e elo que o utiliza para sincronizar o término das mídias "*videoPrincipal*" e "*imgInteratividade*".

A NCL define os seguintes atributos de elo (elemento <link>):

- *id*: identificador único do elo, que deve seguir a mesma regra de formação para o atributo *id* definida no Capítulo 5. Note que esse atributo é opcional;
- *xconnector*: atributo obrigatório, identificador do conector associado ao elo, que também deve seguir a mesma regra de formação para o atributo *id* definida no Capítulo 5.

A NCL define os seguintes elementos como filhos de num elemento <link>:

- <bind>: indica uma ligação entre a interface (*interface*) do componente (*component*, objeto de mídia, de contexto ou switch) e seu papel (*role*) no elo, conforme a semântica do conector. Um elo pode conter diversos elementos <bind>, mas deve conter pelo menos um <bind> para cada papel definido no conector.
- <linkParam>: define um parâmetro do elo como um par [propriedade, valor]. As propriedades e seus respectivos valores dependem da definição do conector ao qual o elo está associado. Um elo pode conter diversos elementos <linkParam>.

A Listagem 10.4 apresenta o código completo de uma aplicação NCL que sincroniza o início e o término da apresentação das mídias "videoPrincipal" e "imgInteratividade".

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="exemplo" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
<head>
   <regionBase>
       <region id="rgTV">
              <region id="rgTVtelaInteira" />
              <region id="rgInteratividade" right="5%" bottom="5%"
                     width="200" height="45" zIndex="3" />
       </region>
   </regionBase>
   <descriptorBase>
       <descriptor id="dTVtelaInteira" region="rgTVtelaInteira" />
       <descriptor id="dInteratividade" region="rgInteratividade"
                     explicitDur="4s">
              <descriptorParam name="transparency" value="60%" />
       </descriptor>
   </descriptorBase>
   <connectorBase>
       <causalConnector id="onBeginStart">
              <simpleCondition role="onBegin" />
```

```
            <simpleAction role="start" />
        </causalConnector>
        <causalConnector id="onEndStop">
            <simpleCondition role="onEnd" />
            <simpleAction role="stop" />
        </causalConnector>
    </connectorBase>
</head>
<body>
    <port id="pVideoPrincipal" component="videoPrincipal" />
    <media id="videoPrincipal" src="media/principal.mpg"
                                    descriptor="dTVtelaInteira" />
    <media id="imgInteratividade" src="media/vermelhoI.png"
                                    descriptor="dInteratividade"/
    <link id="lVideoPrincInicia" xconnector="onBeginStart">
        <bind role="onBegin" component="videoPrincipal" />
        <bind role="start" component="imgInteratividade" />
    </link>
    <link id="lVideoPrincTermina" xconnector="onEndStop">
        <bind role="onEnd" component="videoPrincipal" />
        <bind role="stop" component="imgInteratividade" />
    </link>
</body>
</ncl>
```

Listagem 10.4 Conector e elo para sincronizar o início e o término de exibição de duas mídias.

A Figura 10.7 apresenta a visão estrutural correspondente à Listagem 10.4. Na figura, rotulamos o elo com o identificador do conector por ele utilizado, para evidenciar o comportamento do elo. Além disso, as mídias que estão ligadas a papéis de condição ficam na origem do elo, ao passo que as mídias ligadas a papéis de ação ficam no destino do elo, conforme indicado pela direção da seta.

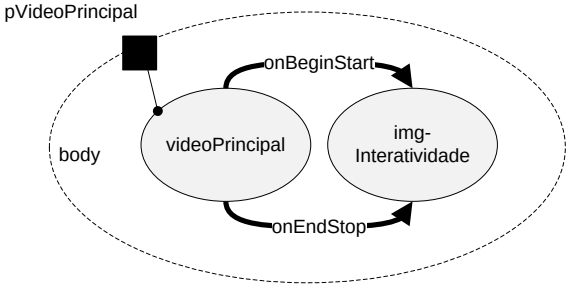


Figura 10.7 Visão estrutural correspondente à Listagem 10.4, com destaque nos elos de sincronismo de início e término de exibição das mídias.

A Figura 10.8 apresenta as visões temporal e espacial correspondentes à Listagem 10.4. Podemos observar na figura a indicação de que os elos de sincronismo é que garantem que a apresentação da mídia "imgInteratividade" será iniciada e concluída junto com a mídia "videoPrincipal".

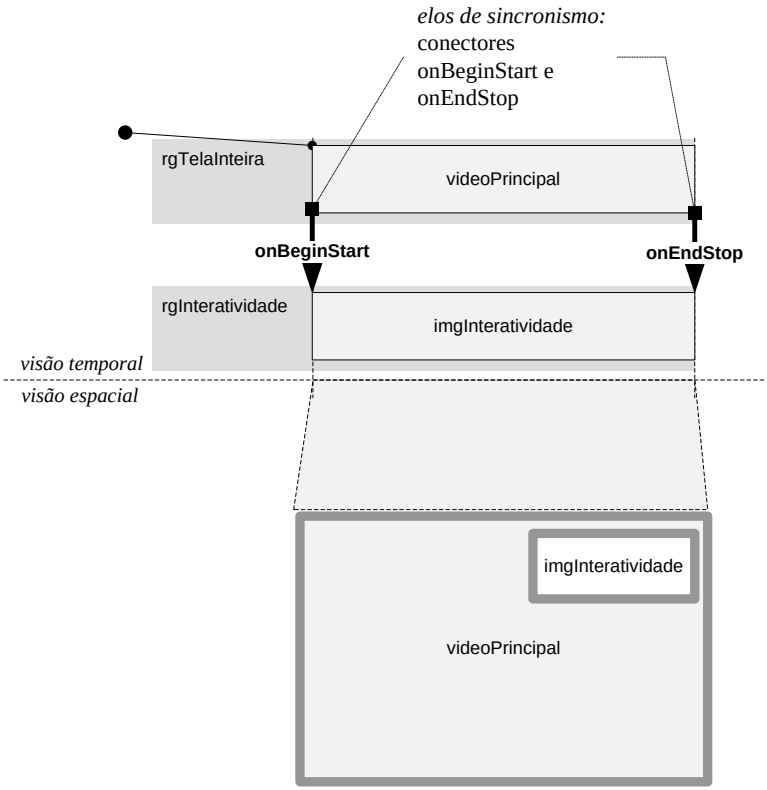


Figura 10.8 Visões temporal e espacial correspondentes à Listagem 10.4, que sincroniza o início e o término de exibição de duas mídias.

Damos a seguir mais alguns exemplos de definição de elos e conectores.

Exemplo 10.1 — Sincronizando o Início de Exibição de Objetos de Mídia com um Retardo

Continuando o exemplo anterior, suponha agora que a imagem "imgInteratividade" deva começar a ser exibida três segundos após o início do vídeo "videoPrincipal". Na definição do conector, é possível definir um retardo para a execução de uma ação, ou seja, um período de tempo em segundos que deve transcorrer após a ativação do elo para que aquela ação seja executada.

A Listagem 10.5 ilustra a definição de um conector com retardo fixo de três segundos, através do atributo *delay* do elemento <simpleAction>. Observe

que, no caso de uma duração de tempo, o valor do parâmetro deve ser seguido da letra "s".

```
<connectorBase>
        <causalConnector id="onBeginStartDelay3">
                <simpleCondition role="onBegin" />
                <simpleAction role="start" delay="3s" />
        </causalConnector>
</connectorBase>
```

Listagem 10.5 Definição de conector com retardo fixo de três segundos.

Apesar de esse conector produzir o efeito necessário, ele é específico apenas para o caso de retardos de três segundos. Para que o conector possa ser reaproveitado por mais elos, é importante que o valor do retardo seja informado pelo elo e não determinado pelo conector. Para contemplar esses casos, a NCL permite definir um parâmetro de conector (através do elemento <connectorParam>), conforme ilustra a Listagem 10.6. Note que, para utilizar o valor desse parâmetro no atributo *delay* do elemento <simpleAction>, ele deve ser precedido de um sinal de cifrão.

```
<connectorBase>
        <causalConnector id="onBeginStart_vDelayelay">
                <connectorParam name="vDelay" />
                <simpleCondition role="onBegin" />
                <simpleAction role="start" delay="$vDelay" />
        </causalConnector>
</connectorBase>
```

Listagem 10.6 Definição de conector com retardo parametrizado.

O elo que utilizar o conector com retardo parametrizado precisa informar qual é o valor desejado para o retardo. Isso é feito através do elemento <linkParam>, conforme ilustrado pela Listagem 10.7. O valor do atributo *name* deve ser o nome do parâmetro tal como tiver sido definido no conector através do atributo *name* do elemento <connectorParam>.

```
<link id="lVideoPrincipalInicia" xconnector="onBeginStart_vDelay">
        <linkParam name="vDelay" value="3s" />
        <bind role="onBegin" component="videoPrincipal" />
        <bind role="start" component="imgInteratividade" />
</link>
```

Listagem 10.7 Definição de elo que informa ao conector o valor do retardo desejado.

A Figura 10.9 apresenta a visão estrutural desse exemplo, semelhante à visão estrutural do exemplo anterior, modificando apenas o rótulo do elo que utiliza o conector com retardo.

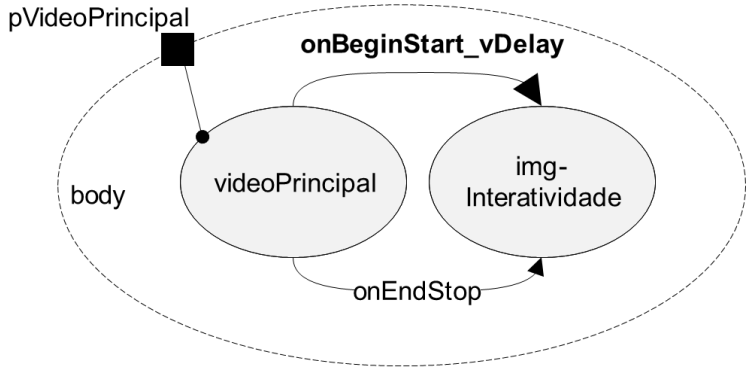


Figura 10.9 Visão estrutural de uma aplicação que utiliza um conector com retardo.

As visões temporal e espacial deste exemplo são apresentadas na Figura 10.10.

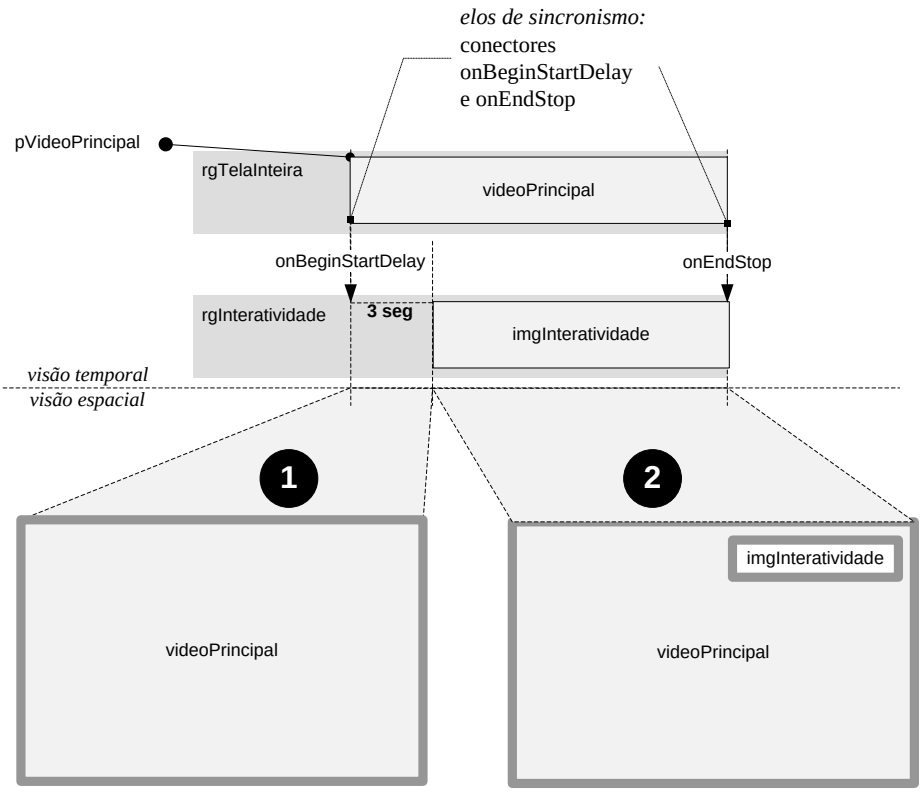


Figura 10.10 Visões temporal e espacial do exemplo de conector que permite a uma mídia ser iniciada com um retardo após uma outra.

Exemplo 10.2 — Ligando Diversos Objetos a um Mesmo Papel

Da forma como foi definido o conector do exemplo, o elo deve associar apenas um objeto a cada papel. Muitas vezes, no entanto, é necessário realizar um mesmo tipo de ação ou condição sobre diversos objetos. Suponha que haja um outro objeto de mídia definido com o *id* "imgImagem1" e ele deva aparecer junto com a "imgInteratividade". O conector deve ser modificado para permitir a ligação de vários objetos no papel de ação "start", e o elo deve criar mais uma ligação através do elemento <bind>, conforme ilustra a Listagem 10.8.

O atributo *max* indica o número máximo de ligações para aquele papel, cujo valor "unbounded" significa que o número de ligações permitidas é ilimitado. Já o atributo *qualifier* indica como devem ser acionados os objetos ligados àquele papel, em paralelo (valor "par") ou sequencialmente (valor "seq").

No elo são definidas as ligações para cada objeto que deve ser iniciado quando o elo for ativado. Observe que, por causa da definição do retardo, todos os objetos ligados ao papel "start" serão iniciados três segundos após a ativação do elo.

```
<connectorBase>
        <causalConnector id="onBeginStart_vDelay">
                <connectorParam name="vDelay" />
                <simpleCondition role="onBegin" />
                <simpleAction role="start" delay="$vDelay"
                                max="unbounded" qualifier="par" />
        </causalConnector>
</connectorBase>
                                                    ... trecho da seção <head>
```

```
                                                    ... trecho da seção <body>
<link id="lVideoPrincipalInicia" xconnector="onBeginStart_vDelay">
        <linkParam name="vDelay" value="3s" />
        <bind role="onBegin" component="videoPrincipal" />
        <bind role="start" component="imgInteratividade" />
        <bind role="start" component="imgImagem1" />
</link>
```

Listagem 10.8 Redefinição de conector e elo para permitir a ligação de mais de uma mídia num mesmo papel.

Exemplo 10.3 — Passando Parâmetros pelas Ligações do Elo

No exemplo anterior, o valor de retardo informado será utilizado por todos os objetos ligados ao papel "start". Muitas vezes, no entanto, é necessário definir diferentes valores de retardo para cada objeto. Isso pode ser feito através de parâmetros das ligações, definidos por elementos <bindParam> de cada ligação <bind>:

- <bindParam>: define um parâmetro específico do <bind> como um par [propriedade, valor]. As propriedades e seus respectivos valores dependem da definição do conector ao qual o elo está associado.

Suponha que o objeto de mídia definido pelo *id* "imgImagem1" deva ser iniciado após um retardo maior. Utilizando o elemento <bindParam>, podemos definir um retardo diferente do *default* do elo para esse objeto de mídia, conforme ilustrado pela Listagem 10.9.

```
<link id="lVideoPrincipalInicia" xconnector="onBeginStart_vDelay">
       <linkParam name="vDelay" value="3s" />
       <bind role="onBegin" component="videoPrincipal" />
       <bind role="start" component="imgInteratividade" />
       <bind role="start" component="imgImagem1">
              <bindParam name="vDelay" value="5s" />
       </bind>
</link>
```

Listagem 10.9 Diferentes valores de retardo informados por parâmetros de elo e de ligação.

Nesse exemplo, os objetos de mídia iniciam, por default, três segundos após a ativação do elo, conforme definido pelo parâmetro de elo <linkParam>. O objeto de mídia "imgImagem1", no entanto, é iniciado cinco segundos após a ativação do elo, conforme o valor de retardo definido através do parâmetro da ligação <bindParam>.

Observe que, caso um mesmo parâmetro seja definido tanto por uma ligação quanto por um elo, o valor definido pelo parâmetro da ligação (no elemento <bindParam>) tem prioridade sobre o parâmetro de elo (no elemento <linkParam>) homônimo.

A Tabela 10.5 sumariza os atributos e conteúdo dos elementos que definem elos. Como sempre, os atributos obrigatórios estão sublinhados.

Tabela 10.5 Elementos, atributos e conteúdo que definem elos

Elementos	Atributos	Conteúdo
link	id, xconnector	(linkParam*, bind+)
bind	role, component, interface, descriptor	(bindParam)*
bindParam	name, value	—
linkParam	name, value	—

A Listagem 10.10 apresenta um esqueleto de código de definição do elo, com todos os seus elementos filhos.

```
<body>
    ...
    <link xconnector="___">
      <linkParam name="___" value="___" />
      <linkParam name="___" value="___" />

      <bind role="___" component="___" interface="__">
            <bindParam name="___" value="___" />
            <bindParam name="___" value="___" />
      </bind>

      <bind role="___" component="___" interface="__">
            <bindParam name="___" value="___" />
      </bind>

      <bind role="___" component="___" interface="__" />
      <bind role="___" component="___" interface="__" />
    </link>
     ...
</body>
```

Listagem 10.10 Esqueleto de código de definição do elo.

10.5 Conectores e Elos de Interatividade

Os conectores apresentados até agora neste capítulo não envolveram interatividade, ou seja, a possibilidade de ação do usuário sobre a aplicação — principal característica de programas de TV digital interativa. Conforme

visto na Tabela 10.1, o papel predefinido "onSelection" é utilizado para esse fim. Ao definir uma condição que utiliza esse papel, o conector pode definir também o atributo *key*, que identifica o código da tecla utilizada para acionar os elos que utilizam o conector.

Exemplo 10.4 — Exibindo um Objeto Quando o Usuário Pressiona uma Tecla

Este exemplo define um conector e um elo para apresentar um objeto quando o usuário pressiona uma determinada tecla. Suponha que haja um objeto de mídia identificado por "imgMenu" (além de seus respectivos descritor e região) representando um menu que deve ser apresentado quando o telespectador pressionar a tecla vermelha ("RED") do controle remoto, enquanto o objeto "imgInteratividade" estiver sendo apresentado. Esse comportamento pode ser ilustrado pela visão estrutural apresentada na Figura 10.11, considerando o nome do conector como "onKeySelectionStart".

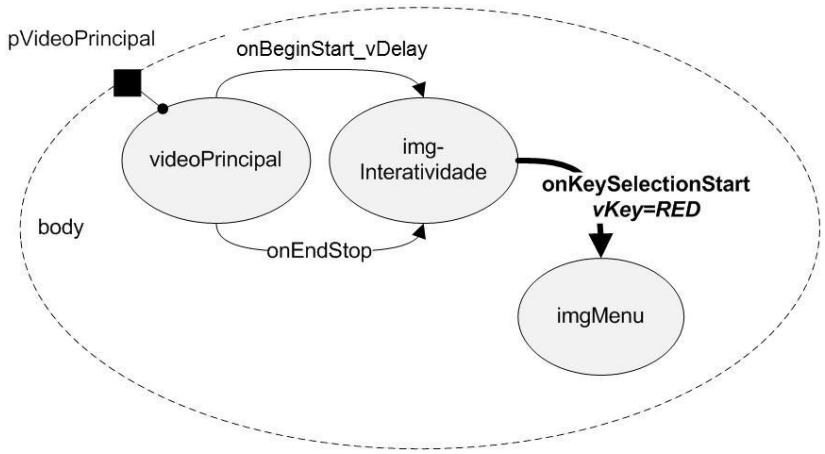


Figura 10.11 Visão estrutural do exemplo de conector de interatividade.

A Listagem 10.11 apresenta a definição do conector "onKeySelectionStart" e um elo que o utiliza para fins do exemplo. Observe que, assim como nos conectores que definimos anteriormente que disparam uma ação com retardo, esse conector recebe o código da tecla como parâmetro (denominado "vKey"), para propiciar seu reúso.

```
<causalConnector id="onKeySelectionStart">
        <connectorParam name="vKey" />
        <simpleCondition role="onSelection" key="$vKey" />
        <simpleAction role="start" max="unbounded" qualifier="par" />
</causalConnector>
                                                    ... trecho da seção <head>
```

```
                                                    ... trecho da seção <body>
<link xconnector="onKeySelectionStart">
        <bind role="onSelection" component="imgInteratividade">
                <bindParam name="vKey" value="RED" />
        </bind>
        <bind role="start" component="imgMenu" />
</link>
```

Listagem 10.11 Conector e elo que apresentam objetos quando uma determinada tecla do controle remoto é acionada.

O fato de o papel "onSelection" estar associado ao objeto de mídia "imgInterativIdade" significa que o elo só está disponível enquanto esse objeto estiver sendo apresentado. Caso o usuário pressione a tecla vermelha em outro momento, o elo não será ativado e o objeto "imgMenu" não será apresentado. Esse mecanismo permite contextualizar a janela de oportunidade de interação aos objetos do documento. A Figura 10.12 apresenta um *storyboard* para ilustrar o comportamento do exemplo.

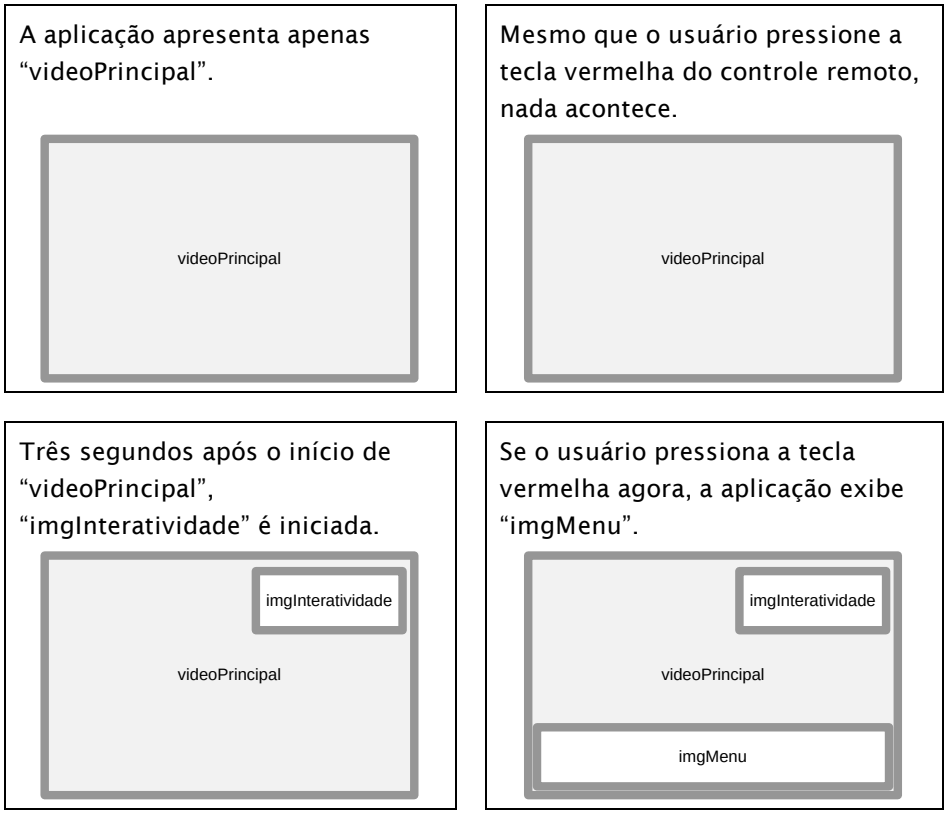


Figura 10.12 *Storyboard* do exemplo de utilização do conector "*onKeySelectionStart*".

A Tabela 10.6 apresenta os códigos de tecla definidos pela linguagem NCL e que podem ser utilizados como valores do atributo *key*.

Tabela 10.6 Códigos de teclas definidos para uso em aplicações NCL

Tipo de Tecla	Código
Numéricas	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, *, #
Alfabéticas	A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z
Guia de programação	MENU, INFO, GUIDE
Setas	CURSOR_LEFT, CURSOR_RIGHT, CURSOR_UP, CURSOR_DOWN
Acionamento	ENTER
Mudanças de canal	CHANNEL_UP, CHANNEL_DOWN
Mudanças de volume	VOLUME_UP, VOLUME_DOWN
Cores	RED, GREEN, YELLOW, BLUE
Controle	BACK, EXIT, POWER, REWIND, STOP, EJECT, PLAY, RECORD, PAUSE

A Listagem 10.12apresenta o código completo da aplicação NCL do Exemplo 10.4, com destaque para o conector e o elo de interatividade.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="exemplo" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
<head>
   <regionBase>
      <region id="rgTV">
        <region id="rgTVtelaInteira" />
        <region id="rgInteratividade" right="20" top="20"
                                width="40" height="40" zIndex="3" />
        <region id="rgMenu" zIndex="4" bottom="5" left="0"
                                       width="1920" height="75" />
      </region>
```

```
   </regionBase>
   <descriptorBase>
      <descriptor id="dTVtelaInteira" region="rgTVtelaInteira" />
      <descriptor id="dInteratividade" region="rgInteratividade"
                                                    explicitDur="4s">
        <descriptorParam name="transparency" value="60%" />
      </descriptor>
      <descriptor id="dMenu" region="rgMenu" />
   </descriptorBase>
   <connectorBase>
      <causalConnector id="onBeginStart_vDelay">
        <connectorParam name="vDelay" />
        <simpleCondition role="onBegin" />
        <simpleAction role="start" delay="$vDelay"
                           max="unbounded" qualifier="par"/>
      </causalConnector>

      <causalConnector id="onEndStop">
        <simpleCondition role="onEnd" />
        <simpleAction role="stop" max="unbounded" qualifier="par"/>
      </causalConnector>

      <causalConnector id="onKeySelectionStart">
        <connectorParam name="vKey" />
        <simpleCondition role="onSelection" key="$vKey" />
        <simpleAction role="start" max="unbounded" qualifier="par"/>
      </causalConnector>
   </connectorBase>
</head>
<body>
   <port id="pVideoPrincipal" component="videoPrincipal" />

   <media id="videoPrincipal" src="media/principal.mpg"
                                    descriptor="dTVtelaInteira" />
   <media id="imgInteratividade" src="media/vermelhoI.png"
                                    descriptor="dInteratividade" />
   <media id="imgMenu" src="media/menu.png" descriptor="dMenu" />

   <link id="lVideoPrincipalInicia" xconnector="onBeginStart_vDelay">
      <bind role="onBegin" component="videoPrincipal" />
      <bind role="start" component="imgInteratividade">
        <bindParam name="vDelay" value="3s" />
      </bind>
   </link>

   <link xconnector=" onKeySelectionStart ">
```

```
      <bind role="onSelection" component="imgInteratividade">
        <bindParam name="vKey" value="RED" />
      </bind>
      <bind role="start" component="imgMenu" />
   </link>

   <link xconnector="onEndStop">
      <bind role="onEnd" component="videoPrincipal" />
      <bind role="stop" component="imgInteratividade" />
      <bind role="stop" component="imgMenu" />
   </link>
</body>
</ncl>
```

Listagem 10.12 Código NCL de aplicação que exibe uma imagem de menu quando o usuário pressiona a tecla vermelha do controle remoto.

10.6 Conectores com Múltiplas Ações e Condições

Os conectores apresentados até aqui envolvem apenas uma condição e uma ação. No entanto, há ocasiões em que, para uma mesma condição, é necessário disparar dois ou mais tipos de ações diferentes, conforme ilustrado pela Figura 10.13.

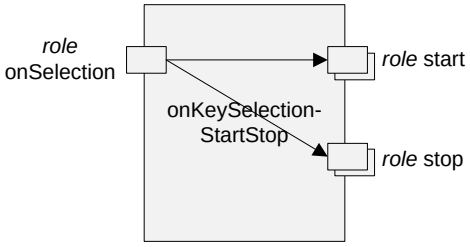


Figura 10.13 Conector com múltiplos papéis de ação, cada qual podendo ser associado a um número indefinido de objetos (*max*="unbounded").

A Figura 10.14 ilustra o uso desse tipo de conector. Trata-se de uma pequena alteração à visão estrutural da Figura 10.11. Ao pressionar a tecla vermelha ("RED") do controle remoto, o elo não apenas inicia a apresentação de "imgMenu", mas também encerra a apresentação de imgInteratividade".

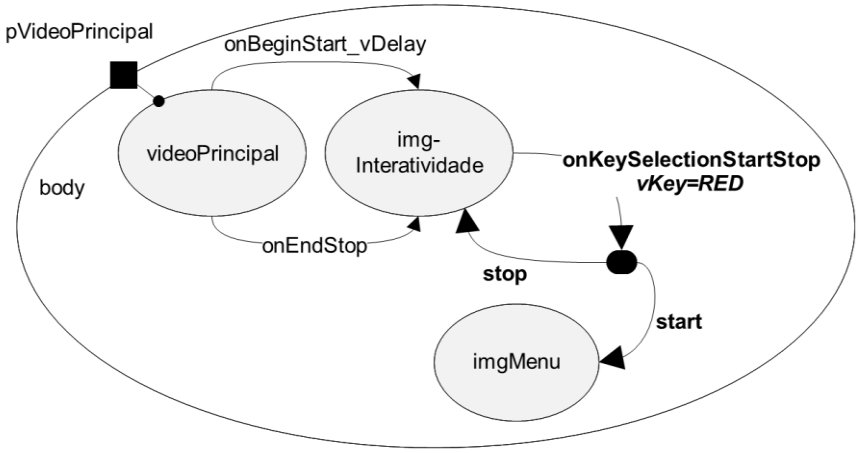


Figura 10.14 Visão estrutural com elo que utiliza um conector de ações compostas.

O elemento <compoundAction> permite definir mais do que uma ação num mesmo conector, conforme especificado na Listagem 10.13. Esse elemento possui um atributo *operator*, que define a ordem em que as ações que ele contém são disparadas: em paralelo ("par") ou em sequência ("seq"). No caso do conector deste exemplo, as ações são disparadas em paralelo. Além disso, diversos objetos podem ser ligados aos papéis "start" e "stop", devido à definição do atributo *max* com valor "unbounded" em cada ação.

```
<causalConnector id="onKeySelectionStartStop">
      <connectorParam name="aKey" />
      <simpleCondition role="onSelection" key="$aKey" />
      <compoundAction operator="par">
         <simpleAction role="start" max="unbounded" operator="par"/>
         <simpleAction role="stop" max="unbounded" operator="par"/>
      </compoundAction>
 </causalConnector>
```

... trecho da seção <head>

... trecho da seção <body>

```
<link xconnector="onKeySelectionStartStop">
      <bind role="onSelection" component="imgInteratividade">
        <bindParam name="vKey" value="RED" />
      </bind>
      <bind role="start" component="imgMenu" />
      <bind role="stop" component="imgInteratividade" />
</link>
```

Listagem 10.13 Conector "*onKeySelectionStartStop*" e elo correspondente, ilustrando a composição de ações.

A Listagem 10.14 apresenta o código completo da aplicação NCL que utiliza o conector "onKeySelectionStartStop", com destaque para as linhas de código relevantes ao exemplo.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="exemplo" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
<head>
   <regionBase>
      <region id="rgTV">
         <region id="rgTVtelaInteira" />
         <region id="rgInteratividade" right="20" top="20"
                              width="40" height="40" zIndex="3" />
         <region id="rgMenu" zIndex="4" bottom="5" left="0"
                                       width="1920" height="75" />
      </region>
   </regionBase>
   <descriptorBase>
      <descriptor id="dTVtelaInteira" region="rgTVtelaInteira" />
      <descriptor id="dInteratividade" region="rgInteratividade"
                                                    explicitDur="4s">
         <descriptorParam name="transparency" value="60%" />
      </descriptor>
      <descriptor id="dMenu" region="rgMenu" />
   </descriptorBase>
   <connectorBase>
      <causalConnector id="onBeginStart_vDelay">
         <connectorParam name="vDelay" />
         <simpleCondition role="onBegin" />
         <simpleAction role="start" delay="$vDelay"
                              max="unbounded" qualifier="par" />
      </causalConnector>

      <causalConnector id="onEndStop">
         <simpleCondition role="onEnd" />
         <simpleAction role="stop" max="unbounded" qualifier="par" />
      </causalConnector>

      <causalConnector id="onKeySelectionStartStop">
         <connectorParam name="vKey" />
         <simpleCondition role="onSelection" key="$vKey" />
         <compoundAction operator="par">
               <simpleAction role="start"
                              max="unbounded" qualifier="par" />
               <simpleAction role="stop"
                              max="unbounded" qualifier="par" />
         </compoundAction>
      </causalConnector>
```

```
   </connectorBase>
</head>
<body>
   <port id="pVideoPrincipal" component="videoPrincipal" />

   <media id="videoPrincipal" src="media/principal.mpg"
                                    descriptor="dTVtelaInteira" />
   <media id="imgInteratividade" src="media/vermelhoI.png"
                                    descriptor="dInteratividade" />
   <media id="imgMenu" src="media/menu.png" descriptor="dMenu" />

   <link id="lVideoPrincipalInicia" xconnector="onBeginStart_vDelay">
      <bind role="onBegin" component="videoPrincipal" />
      <bind role="start" component="imgInteratividade">
         <bindParam name="vDelay" value="3s" />
      </bind>
   </link>

   <link xconnector="onKeySelectionStartStop">
      <bind role="onSelection" component="imgInteratividade">
         <bindParam name="vKey" value="RED" />
      </bind>
      <bind role="start" component="imgMenu" />
      <bind role="stop" component="imgInteratividade" />
   </link>

   <link xconnector="onEndStop">
      <bind role="onEnd" component="videoPrincipal" />
      <bind role="stop" component="imgInteratividade" />
      <bind role="stop" component="imgMenu" />
   </link>
</body>
</ncl>
```

Listagem 10.14 Código NCL de aplicação que exibe uma imagem de menu e encerra a imagem de interatividade quando o usuário pressiona a tecla vermelha do controle remoto.

Do mesmo modo, um conector pode definir mais do que uma condição. As condições vistas até agora (“onBegin”, “onEnd”, “onSelection”) estão relacionadas com a transição de um evento de apresentação ou de interatividade. Além dessas, é possível construir condições que testem o estado de apresentação de um evento, atributos associados a eventos ou valores de propriedades, como definido no Capítulo 9. O elemento <assessmentStatement> é utilizado para fazer tais comparações. A Figura 10.15 ilustra um conector que faz uso de múltiplos papéis de condição.

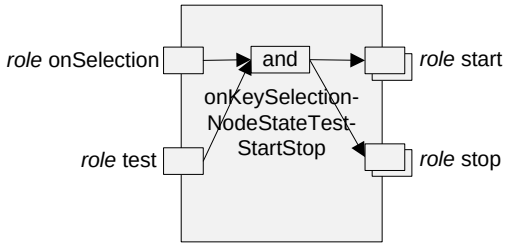


Figura 10.15 Conector com múltiplos papéis de condição, ligados pelo operador "and".

O elemento <assessmentStatement> possui dois tipos de elementos filhos: <attributeAssessment>, que define um atributo cujo valor será testado, e <valueAssessment>, que define um valor a ser testado. O elemento <assessmentStatement> possui ainda o atributo *comparator*, que define o operador de comparação, conforme definido na Tabela 10.7. O valor de um atributo definido pelo elemento <attributeAssessment> pode ser testado contra o valor de outro atributo definido por outro elemento <attributeAssessment> ou contra um valor definido pelo elemento <valueAssessment>.

Tabela 10.7 Operadores de comparação que podem ser utilizados em elementos <assessmentStatement>.

Valor	Significado
eq	igual a (*equal to*)
ne	diferente de (*not equal to*)
gt	maior que (*greater than*)
lt	menor que (*less than*)
gte	maior ou igual a (*greater than or equal to*)
lte	menor ou igual a (*less than or equal to*)

O elemento <attributeAssessment> possui um atributo *role* cujo valor deve ser único no conjunto de papéis do conector. Como normalmente ocorre, um *role* é um ponto de interface do conector, que é associado às interfaces dos objetos por um elemento <link> que referencia o conector. Um elemento <attributeAssessment> também define um tipo de evento (atributo *eventType*, podendo assumir os valores "presentation", "selection" ou "attribution"), que terá o valor de seu estado ("state") ou de seus atributos

(*occurrences* ou *repetitions*)[4] comparados, conforme especificado pelo atributo *attributeType* (que no caso pode ter como valor "state", "occurrences" ou "repetitions"). Se o *eventType* for "attribution", o *attributeType* é opcional e pode ter também o valor "nodeProperty" (*default*), indicando que uma propriedade do objeto deve ser avaliada. Se o valor de *eventType* for "selection" (seleção), é conveniente que o elemento também defina a qual dispositivo de entrada a seleção se refere (por exemplo, teclas de um teclado ou controle remoto), através do atributo *key*. Um valor de compensação (*offset*) pode ser adicionado a um elemento <attributeAssessment> antes da comparação (por exemplo, uma compensação pode ser adicionada a uma avaliação de atributo para especificar "a posição vertical da tela mais 50 pixels").

O elemento <valueAssessment> tem um atributo *value* que deve obrigatoriamente assumir um valor de estado de evento ("sleeping", "occurring" ou "paused") ou um valor qualquer a ser comparado com uma propriedade do objeto ou atributo de evento.

A Figura 10.16 ilustra a visão estrutural de uma aplicação que utiliza um elo com condição composta, definido na Listagem 10.15.

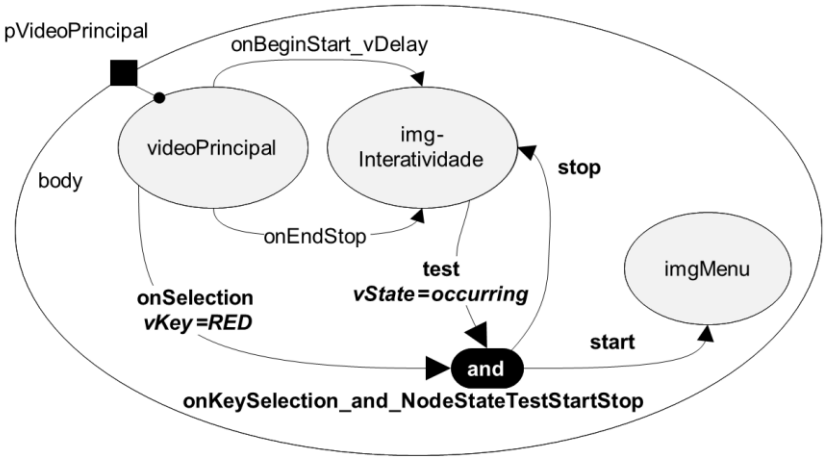


Figura 10.16 Visão estrutural com elo que utiliza um conector com condição composta.

O conector, definido na Listagem 10.15, é acionado apenas quando duas condições são satisfeitas: 1) uma determinada tecla (identificada pelo parâmetro "*vKey*") é pressionada e 2) o estado de apresentação do objeto é o

[4] Os atributos *occurrences* e *repetition* informam quantas vezes um evento já ocorreu e quantas vezes ele ainda deverá ocorrer imediatamente após o final da ocorrência atual, respectivamente. Para eventos de seleção, o atributo *repetitions* não é definido.

estado desejado (identificado pelo parâmetro "*vState*"). Observe que o atributo *operator* do elemento <compoundCondition> é que define se todas as condições devem ser satisfeitas (*operator*= "and") ou se apenas uma das condições precisa ser satisfeita para acionar o elo (*operator*="or").

Os estados de apresentação de um objeto que podem ser testados com esse conector são: "occurring" (em apresentação), "paused" (em pausa) ou "sleeping" (parado). O elo ilustrado na Listagem 10.15 apresenta o objeto de mídia "imgMenu" quando a tecla vermelha é selecionada, mas apenas se o objeto de mídia "imgInteratividade" estiver sendo apresentado. Sendo assim, o comportamento é semelhante ao do elo apresentado na Listagem 10.11. A solução ilustrada pela Listagem 10.11 é a preferencial. No entanto, pode haver casos em que um elo deve ser acionado apenas quando mais de uma mídia estiver sendo apresentada. Nesses casos, a definição de um <assessmentStatement> se faz necessária.

```
<causalConnector id="onKeySelection_and_NodeStateTestStartStop">
      <connectorParam name="vKey" />
      <connectorParam name="vState" />
      <compoundCondition operator="and">
         <simpleCondition role="onSelection" key="$vKey" />
         <assessmentStatement comparator="eq">
        <attributeAssessment role="test"
               eventType="presentation" attributeType="state"/>
        <valueAssessment value="$vState" />
          </assessmentStatement>
      </compoundCondition>
      <compoundAction operator="par">
         <simpleAction role="start" />
         <simpleAction role="stop" />
      </compoundAction>
</causalConnector>
```

... trecho da seção <head>

... trecho da seção <body>

```
      <link xconnector="onKeySelection_and_NodeStateTestStartStop">
         <bind role="onSelection" component="videoPrincipal">
               <bindParam name="vKey" value="RED" />
         </bind>
         <bind role="test" component="imgInteratividade">
               <bindParam name="vState" value="occurring" />
         </bind>
         <bind role="start" component="imgMenu" />
         <bind role="stop" component="imgInteratividade" />
      </link>
```

Listagem 10.15 Conector "*onKeySelection_and_NodeStateTestStartStop*" cujas duas condições precisam ser verdadeiras para o elo correspondente ser acionado.

Do mesmo modo como as condições podem ser compostas, também as cláusulas de comparação podem ser compostas, através do elemento <compoundStatement>, que pode conter diversos elementos <assessmentStatement> e também outros elementos <compoundStatement>.

Exemplo 10.5 — Exibindo um Vídeo em *Loop* até a Intervenção do Usuário

O objetivo deste exemplo é exibir um vídeo em *loop*, permitindo ao usuário interromper essa exibição pressionando a tecla azul ("BLUE") do controle remoto.

Para que um objeto de mídia contínua possa ser exibido em *loop*, pode-se utilizar um conector "onEndStart" com o mesmo nó de vídeo nos papéis "onEnd" e "start". Para parar esse vídeo e iniciar o próximo, no entanto, surge um problema. Caso seja utilizada a ação "stop", o elo associado ao conector "onEndStart" será acionado novamente, e a exibição do primeiro vídeo reiniciará. Para terminar a apresentação do vídeo sem acionar esse elo, é necessário interromper o vídeo em *loop* utilizando a ação "abort", que não aciona os elos com mídias no papel "onEnd".

Para que o usuário saiba qual é a tecla que pode pressionar a cada instante, é importante que um objeto de mídia seja exibido indicando as oportunidades de interação e o que será feito. Nesse exemplo, é exibida uma imagem indicando que a tecla azul do controle remoto serve para interromper o *loop* e parar o documento.

Para iniciar, terminar e abortar objetos quando uma tecla do controle remoto é selecionada, criamos um conector denominado "onKeySelectionAbortStop".

A Figura 10.17 apresenta a visão estrutural do exemplo e a Figura 10.18 apresenta a visão de leiaute.

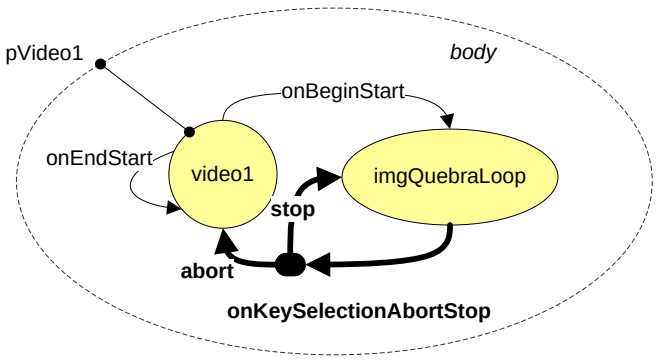


Figura 10.17 Visão estrutural do Exemplo 10.5.

Figura 10.18 Visão de leiaute do Exemplo 10.5.

As visões temporal e espacial do Exemplo 10.5 são apresentadas na Figura 10.19.

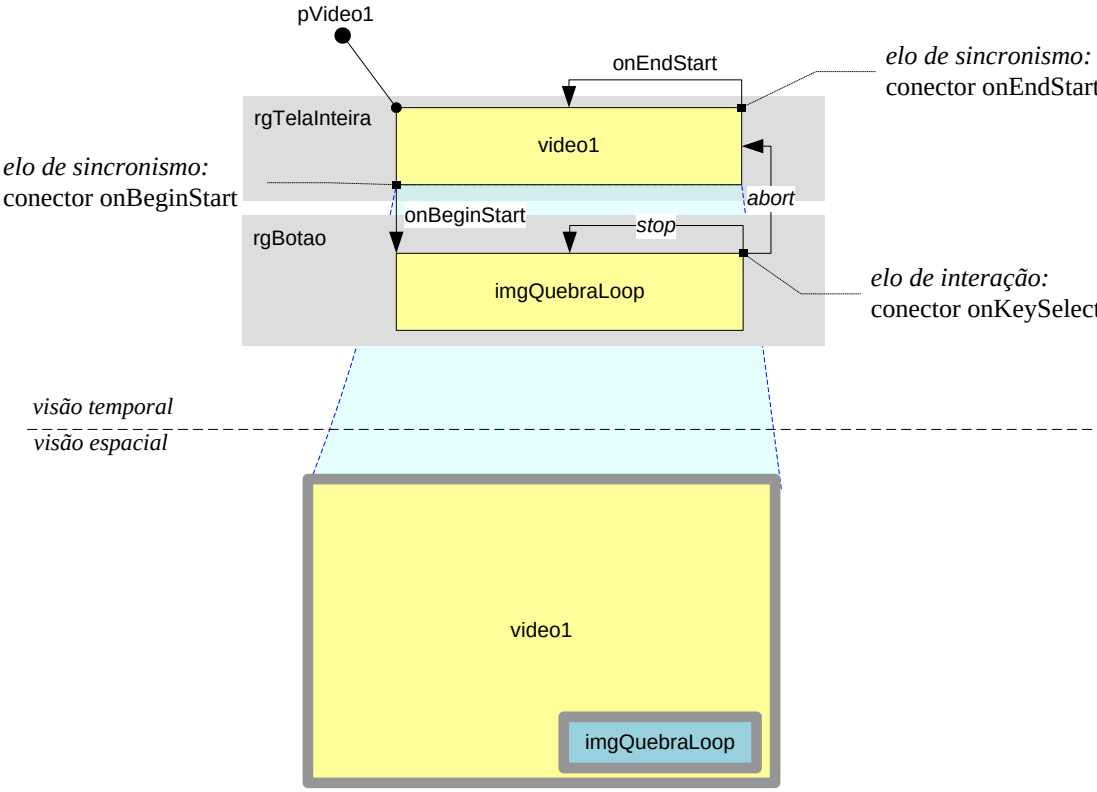


Figura 10.19 Visões temporal e espacial do Exemplo 10.5.

A Listagem 10.16 apresenta os conectores e elos do Exemplo 10.5, com destaque aos trechos relacionados com o uso da ação “abort”.

```
<causalConnector id="onEndStart">
        <simpleCondition role=”onEnd” />
        <simpleAction role=”start” />
</causalConnector>

<causalConnector id="onKeySelectionAbortStop">
   <connectorParam name="vKey" />
   <simpleCondition role="onSelection" key="$vKey" />
   <compoundAction operator="par">
      <simpleAction role="abort" />
       <simpleAction role="stop" />
   </compoundAction>
</causalConnector>
```

... trecho da seção <head>

... trecho da seção <body>

```
<link xconnector="onEndStart">
   <bind component="video1" role="onEnd" />
   <bind component="video1" role="start" />
</link>

<link xconnector="onKeySelectionAbortStop">
   <bind component="imgQuebraLoop" role="onSelection">
      <bindParam name="vKey" value="BLUE" />
   </bind>
   <bind component="video1" role="abort" />
   <bind component="prog3" role="stop" />
</link>
```

Listagem 10.16 Conectores “onEndStart” e “onKeySelectionAbortStop”, o elo que mantém o objeto “video1” em loop e o elo que interrompe o objeto em *loop*.

10.7 Conectores que Definem Novos Papéis

Como mencionamos na Seção 10.3, palavras reservadas para papéis facilitam a definição de conectores. No entanto, os papéis de um conector não estão restritos àqueles definidos por meio de palavras reservadas. Como vimos naquela seção, para definir um papel que não tenha sido predefinido, é necessário definir um valor para o atributo *eventType*.

No caso do elemento <simpleCondition>, é necessário definir também um valor para o atributo *transition*. No caso do elemento <attributeAssessment>, é necessário definir também um valor para o atributo *attributeType*. Já no caso do elemento <simpleAction>, é necessário definir também um valor para o atributo *actionType*.

10.8 Conectores e Elos que Manipulam Propriedades

Uma utilização comum de propriedades ocorre quando uma mídia precisa ser reposicionada ou redimensionada durante a execução da aplicação. Para alterar o valor de uma propriedade, definimos um conector que utiliza o papel predefinido "set".

De modo semelhante aos elos definidos com retardo ou tecla de ativação, para tornar o conector mais útil, o valor a ser atribuído é definido como parâmetro do conector. A Listagem 10.17 apresenta o código de um conector que apresenta o seguinte comportamento: quando uma determinada tecla é pressionada, um ou mais objetos são redimensionados e um ou mais objetos são apresentados.

```
<causalConnector id="onKeySelectionSet_vNewValueStartStop">
      <connectorParam name="vKey" />
      <connectorParam name="vNewValue" />
      <simpleCondition role="onSelection" key="$vKey" />
      <compoundAction operator="par">
        <simpleAction role="set" value="$vNewValue" max="unbounded"/>
        <simpleAction role="start" max="unbounded" />
        <simpleAction role="stop" max="unbounded" />
      </compoundAction>
</causalConnector>
```

Listagem 10.17 Conector que manipula uma propriedade de nó.

Para o elo fazer a ligação com uma propriedade de um nó, é necessário que o elemento <bind> defina, além dos atributos *role* e *component*, o atributo *interface*, para identificar a propriedade que se deseja alterar. Além disso, esse <bind> deve conter um parâmetro (<bindParam>), que define o novo valor a ser atribuído à propriedade correspondente. A Listagem 10.18 apresenta um elo que é acionado quando a tecla vermelha é pressionada e, quando isso acontece, redimensiona o objeto "videoPrincipal" e apresenta a imagem "imgFundo".

```
<link xconnector=" onKeySelectionSet_vNewValueStartStop">
      <bind role="onSelection" component="imgInteratividade">
        <bindParam name="vKey" value="RED" />
      </bind>
      <bind role="set" component="videoPrincipal" interface="left">
        <bindParam name="vNewValue" value="45%" />
      </bind>
```

```
      <bind role="set" component="videoPrincipal" interface="top">
        <bindParam name="vNewValue" value="30%" />
      </bind>
      <bind role="set" component="videoPrincipal" interface="width">
        <bindParam name="vNewValue" value="40%" />
      </bind>
      <bind role="set"component="videoPrincipal" interface="height">
        <bindParam name="vNewValue" value="40%" />
      </bind>
      <bind role="start" component="imgFundo" />
      <bind role="stop" component="imgInteratividade" />
</link>
```

Listagem 10.18 Elo que manipula propriedades de mídia.

A Figura 10.20 apresenta a visão estrutural que ilustra o conector e o elo definidos nas Listagens 10.17 e 10.18.

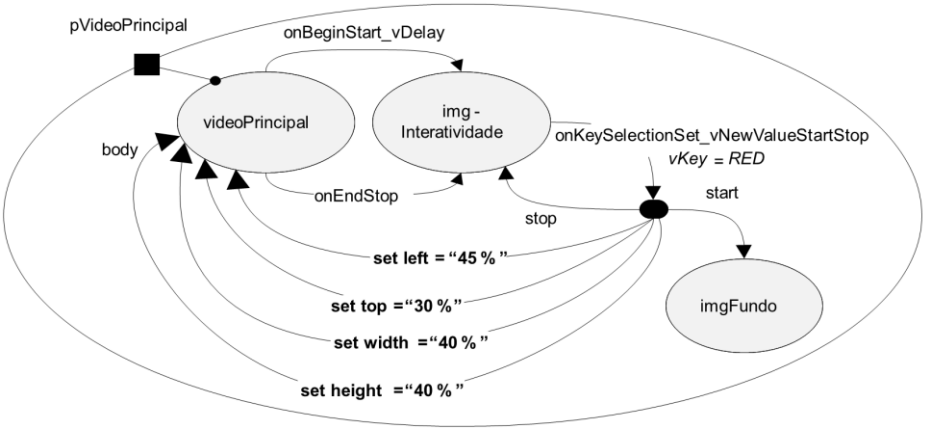


Figura 10.20 Visão estrutural de aplicação que manipula o valor de diversas propriedades.

Observe que, assim como é possível manipular cada propriedade de posição e dimensões separadamente, também é possível definir e manipular em conjunto todos esses valores, através do grupo de propriedades bounds (Listagem 10.19).

```
<media id="videoPrincipal" src="media/video1.mpg" descriptor="dVideo1">
      <property name="bounds" />
</media>
...
<link xconnector=" onKeySelectionSet_vNewValueStartStop">
      <bind role="onSelection" component="imgInteratividade">
         <bindParam name="vKey" value="RED" />
      </bind>
      <bind role="set" component="videoPrincipal" interface="bounds">
         <bindParam name="vNewValue" value="45%,30%,40%,40%" />
      </bind>
      <bind role="start" component="imgFundo" />
      <bind role="stop" component="imgInteratividade" />
</link>
```

Listagem 10.19 Elo que manipula posição e dimensões de uma mídia através da propriedade **bounds**, bem como a mídia que define a propriedade.

A Figura 10.21 apresenta a visão estrutural que ilustra o conector e o elo definidos nas Listagens 10.17 e 10.19.

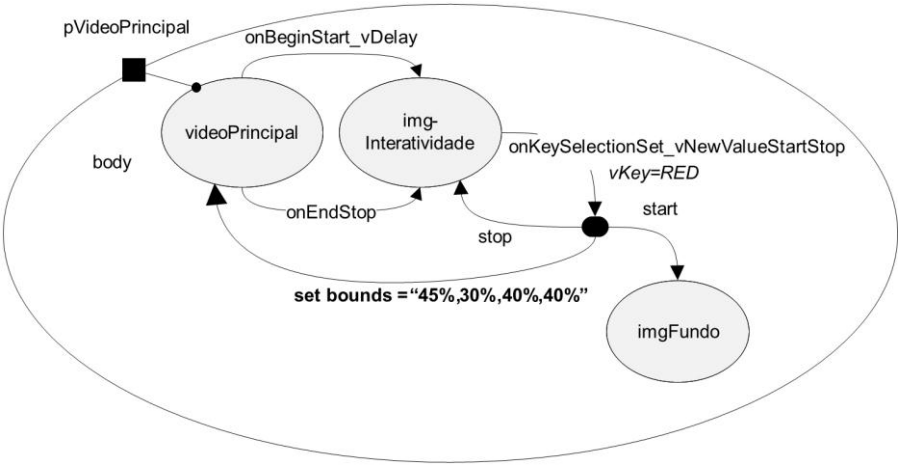


Figura 10.21 Visão estrutural de aplicação que manipula um valor de um grupo de propriedades.

Exemplo 10.6 — Alternando a Visibilidade de Mídias

Vamos considerar agora uma situação em que há dois vídeos pré-gravados e armazenados apresentando diferentes tomadas de uma mesma cena, como, por exemplo, arquivos de vídeo curto cujo conteúdo corresponde a diferentes câmeras em um jogo de futebol. Vamos considerar uma aplicação NCL que permita ao usuário explorar o *replay* de uma jogada de "melhores momentos" sob demanda, trazendo os arquivos de vídeo mencionados durante o jogo ou o intervalo. A jogada do *replay* é única, mas há os dois arquivos de vídeo, cada

qual correspondendo a um ângulo do jogo. Para o usuário alternar entre os diferentes vídeos, que correspondem às diferentes câmeras, não é possível utilizar as ações de apresentação (p. ex., "start"/"stop", "pause"/"resume") para mostrar um vídeo e ocultar o outro. A ação "start" inicia a apresentação de uma mídia armazenada desde seu início, e a ação "resume" inicia do ponto em que foi feita a pausa no vídeo, e não do ponto em que está o outro vídeo. Enfim, como não há nenhuma relação temporal entre os dois vídeos pré-armazenados, é necessário manter os dois vídeos "tocando em paralelo", mas um deles deve permanecer oculto e sem som até que seja feita a troca.

O objetivo deste exemplo é permitir ao usuário alternar entre dois vídeos, através da seleção das teclas vermelha ("RED") e verde ("GREEN") do controle remoto. Ambos os vídeos devem ser apresentados na mesma posição da tela.

Os dois vídeos devem ser iniciados de forma sincronizada, sendo que um deles deve iniciar invisível e sem som. Quando o usuário fizer uma seleção com as teclas do controle remoto, os valores das propriedades dos dois vídeos devem ser invertidos. Quando o primeiro vídeo ("*video1*") terminar, deve-se ocultar as mídias ilustrando os botões e o segundo vídeo ("*video2*").

Como sempre, é importante apresentar para o usuário mídias que indiquem as oportunidades de interação com o programa. Além disso, apresentar para o usuário uma opção de interatividade que de fato não produz efeito não é uma boa prática de design, com relação à usabilidade do programa. Sendo assim, somente o botão que trocar o vídeo atual deve ser exibido.

As Figuras 10.22 a 10.26 ilustram a construção passo a passo da visão estrutural do Exemplo 10.6.

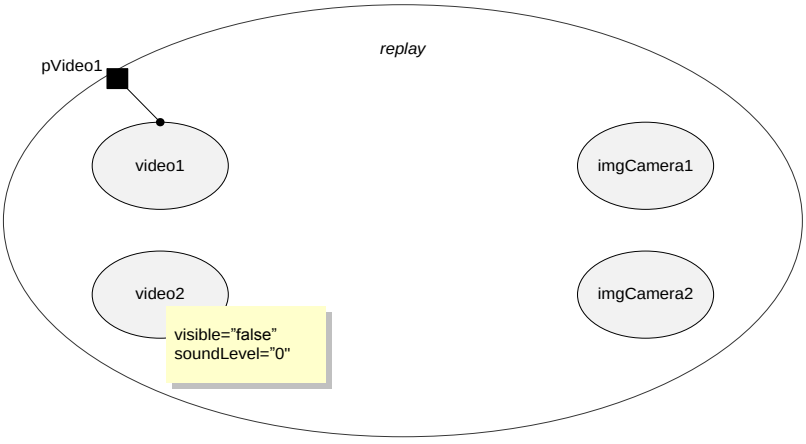


Figura 10.22 Exemplo 10.6, passo 1, contexto "replay" com as mídias, sendo que a porta "pVideo1" deve estar mapeada para "video1", que é o vídeo que deve tocar visível e com som inicialmente; "video2" deve referenciar um descritor com *visible*="false" e *soundLevel*="0".

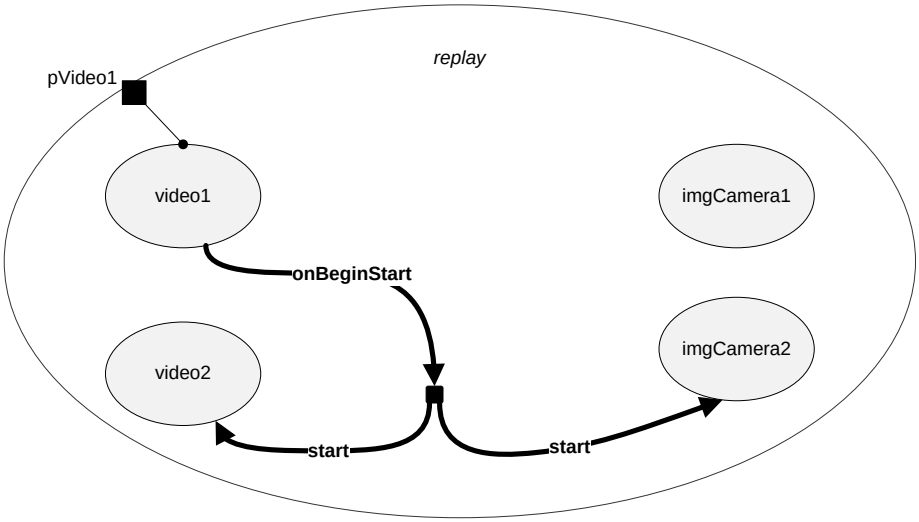


Figura 10.23 Exemplo 10.6, passo 2, elo para iniciar "video2" e "imgCamera2" assim que "video1" inicia, fazendo uso do conector "*onBeginStart*".

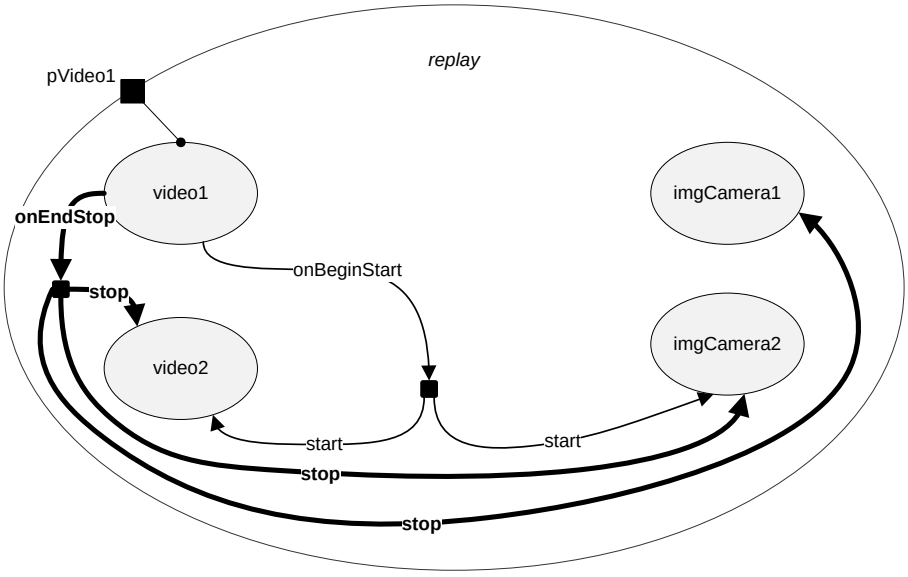


Figura 10.24 Exemplo 10.6, passo 3, elo para encerrar a apresentação de todas as mídias quando "video1" termina, fazendo uso do conector "*onEndStop*".

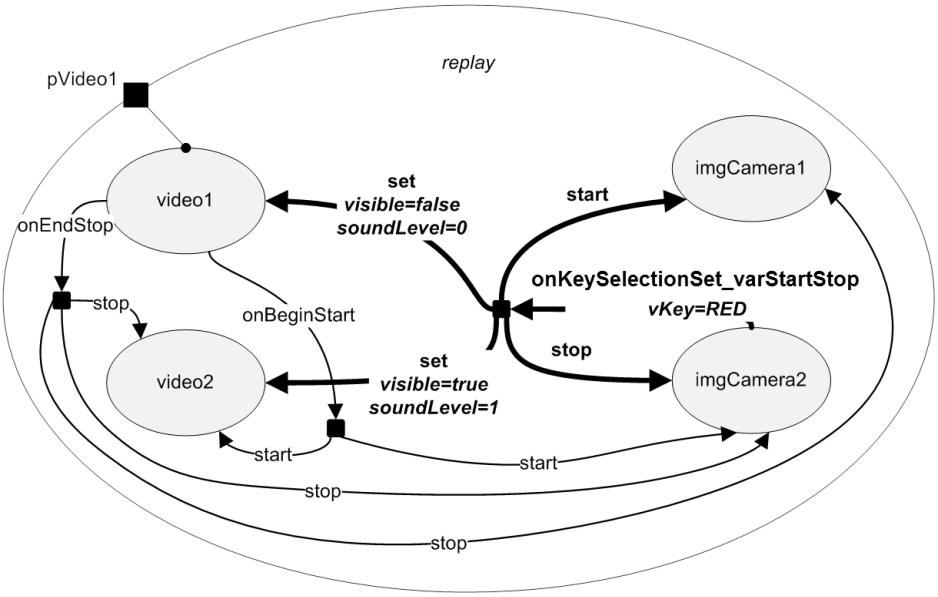


Figura 10.25 Exemplo 10.6, passo 4, lo para trocar a visibilidade do "video1" pela do "video2". Além disso, troca o botão de câmera, de "imgCamera2" para "imgCamera1".

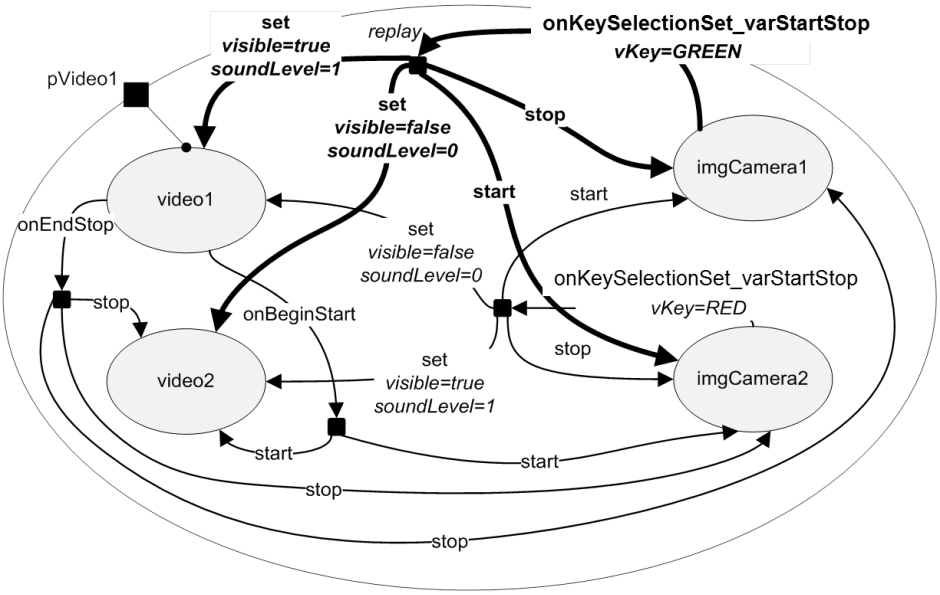


Figura 10.26 Exemplo 10.6, passo 5, elo análogo ao anterior, desta vez para trocar a visibilidade do "video2" pela do "video1". Além disso, troca o botão de câmera, de "imgCamera1" para "imgCamera2".

Podemos observar que foi utilizado o mesmo conector do exemplo anterior, “onKeySelectionSet_varStartStop”.

A Figura 10.27 apresenta as visões temporal e espacial do Exemplo 10.6.

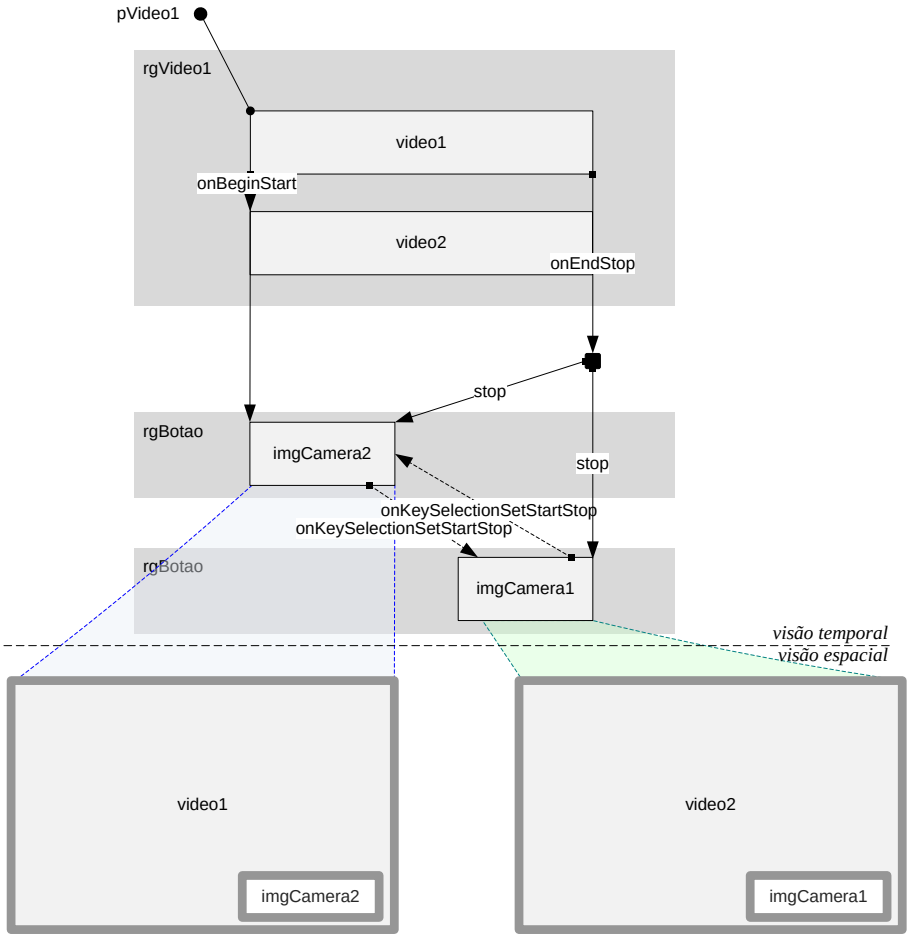


Figura 10.27 Visões temporal e espacial do Eemplo 10.6

Passo a passo

Passo 1: Criando novo descritor para o segundo vídeo

Como as duas mídias serão iniciadas ao mesmo tempo, mas com valores de propriedades de visibilidade e som distintos, é necessário criar um descritor diferente para o segundo vídeo, reusando a mesma região de “video1”, mas com os valores das propriedades correspondentes às definidas no seu descritor, para que não haja som nem exibição no início:

```
        <descriptor id="dVideo2" region="rgVideo1">
            <descriptorParam name="visible" value="false" />
            <descriptorParam name="soundLevel" value="0" />
        </descriptor>
```

Passo 2: Declarando as propriedades que serão manipuladas pelos elos

Para que seja possível alterar as propriedades *visible* e *soundLevel*, é necessário criar âncoras de propriedades nos nós de mídia correspondentes:

```
    <media id="video1" src="media/video1.mpg" descriptor="dVideo1">
        <property name="visible" />
        <property name="soundLevel" />
    </media>

    <media id="video2" src="media/video2.mpg" descriptor="dVideo2">
        <property name="visible" />
        <property name="soundLevel" />
    </media>
```

Passo 3: Criando os elos para manipular as propriedades, iniciar e parar mídias

É necessário criar dois elos para efetuar a seleção do vídeo, através do conector "onKeySelectionSet_varStartStop". Esse conector é utilizado para trocar as propriedades de visibilidade e volume do som, trocando o vídeo sendo exibido conforme o botão selecionado: o botão vermelho ativa o "video2" e o botão verde ativa o "video1":

```
<!-- toggle cameras -->
    <link xconnector="meusConectores#onKeySelectionSet_varStartStop">
        <bind component="imgCamera1" role="onSelection">
            <bindParam name="vKey" value="RED" />
        </bind>
        <bind component="video1" interface="visible" role="set">
            <bindParam name="var" value="true" />
        </bind>
        <bind component="video1" interface="soundLevel" role="set">
            <bindParam name="var" value="1" />
        </bind>
        <bind component="video2" interface="visible" role="set">
            <bindParam name="var" value="false" />
        </bind>
```

```
        <bind component="video2" interface="soundLevel" role="set">
           <bindParam name="var" value="0" />
        </bind>
        <bind component="imgCamera2" role="start" />
        <bind component="imgCamera1" role="stop" />
    </link>

    <link xconnector="meusConectores#onKeySelectionSet_varStartStop">
        <bind component="imgCamera2" role="onSelection">
           <bindParam name="vKey" value="GREEN" />
        </bind>
        <bind component="video2" interface="visible" role="set">
           <bindParam name="var" value="true" />
        </bind>
        <bind component="video2" interface="soundLevel" role="set">
           <bindParam name="var" value="1" />
        </bind>
        <bind component="video1" interface="visible" role="set">
           <bindParam name="var" value="false" />
        </bind>
        <bind component="video1" interface="soundLevel" role="set">
           <bindParam name="var" value="0" />
        </bind>
        <bind component="imgCamera1" role="start" />
        <bind component="imgCamera2" role="stop" />
    </link>
```

Observamos que, como o elo deve alterar algumas propriedades dos nós de vídeo, cada elemento de ligação (<bind>) deve especificar não apenas o componente de destino do elo (atributo *component*), mas também a sua propriedade (através do atributo *interface*), tal como definida no nó da mídia. Além disso, como o valor dessa propriedade deve ser modificado, o novo valor deve ser passado como parâmetro (através do elemento <bindParam>):

```
<bind component="video1" interface="visible" role="set">
    <bindParam name="var" value="false" />
</bind>
<bind component="video1" interface="soundLevel" role="set">
    <bindParam name="var " value="0" />
</bind>
```

Passo 4: Modificando o elo para iniciar as mídias

Para informar o usuário sobre as oportunidades de interação, o elo de exibição inicial do vídeo e dos botões deve exibir apenas a imagem do botão

vermelho, pois pressionar a tecla verde enquanto o "*video1*" está tocando não produz efeito:

```
<!-- início do video1 deve disparar a exibição do video2 e da imagem
do botão de mudança de câmera -->

 <link id="lVideo_Botoes_start" xconnector="connectors#onBeginStart">
   <bind component="video1" role="onBegin" />
   <bind component="video2" role="start" />
   <bind component="imgCamera2" role="start" />
 </link>
```

Exemplo 10.7 — Interrompendo uma Aplicação NCL no Caso de Vídeos Entrelaçados no Fluxo Elementar

Como apresentado nos Capítulos 8 e 9 e detalhado no Apêndice E, é possível que haja vídeos entrelaçados em um mesmo fluxo elementar. Suponha uma aplicação NCL "idApl" associada a um filme "pagu", que é interrompido por uma propaganda. É possível que a aplicação NCL esteja exibindo algum outro objeto de mídia sobre o vídeo "pagu" no momento dessa interrupção. A aplicação deve ser pausada e tornada invisível, até que a propaganda acabe. Ao término da propaganda, o filme "pagu" deve ser retomado e, juntamente com ele, a aplicação, que também deve ser tornada visível.

A igura 10.28 ilustra essa situação.

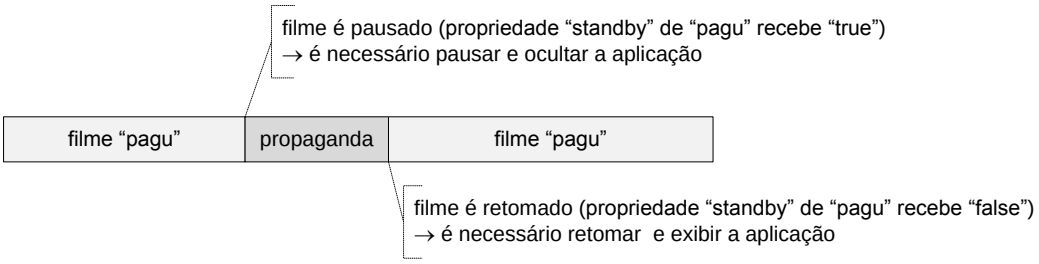


Figura 10.28 Entrelaçamento de vídeos no fluxo elementar.

A Listagem 10.20 apresenta o código que manipula a propriedade "visible" da aplicação NCL conforme o valor da propriedade embutida "standby" do objeto de mídia "filme" do fluxo elementar. Note que tanto a propriedade "visible" do elemento <body> da aplicação como a propriedade "standby" do objeto de mídia "pagu" devem ser explicitamente declaradas (a listagem não apresenta esse trecho de código).

```
<ncl id="idApl" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
  <head>
     <connectorBase>
        <causalConnector id="onEndAttribution_and_TestSetFalsePause">
           <compoundCondition operator="and">
              <simpleCondition role="onEndAttribution" />
              <assessmentStatement comparator="eq">
                <attributeAssessment role="testIfTrue"
                  attributeType="nodeProperty" eventType="attribution"/>
                <valueAssessment value="true"/>
              </assessmentStatement>
           </compoundCondition>
        <compoundAction operator="seq">
           <simpleAction role="setAsFalse"
             eventType=”attribution” actionType=”start” value="false"/>
           <simpleAction role="pause"/>
        </compoundAction>
        </causalConnector>
        <causalConnector id="onEndAttribution_and_TestSetTrueResume">
           <compoundCondition operator="and">
              <simpleCondition role="onEndAttribution"/>
              <assessmentStatement comparator="eq">
                <attributeAssessment role="testIfFalse"
                  attributeType="nodeProperty" eventType="attribution”/>
                <valueAssessment value="false” />
              </assessmentStatement>
           </compoundCondition>
        <compoundAction operator="seq">
           <simpleAction role="resume” />
           <simpleAction role="setAsTrue"
              eventType=”attribution” actionType=”start” value="true”/>
        </compoundAction>
        </causalConnector>
     </connectorBase>
  </head>
  <body>
     ...
     <link xconnector="onEndAttribution_and_TestSetFalsePause">
        <bind role="onEndAttribution" component="pagu"
                                              interface="standby”/>
        <bind role="testIfTrue" component="pagu" interface="standby”/>
        <bind role="setAsFalse" component="idApl" interface="visible”/>
        <bind role="pause" component="idApl” />
     </link>
     <link xconnector="onEndAttribution_and_TestSetTrueResume">
        <bind role="onEndAttribution" component="pagu"
                                              interface="standby”/>
        <bind role="testIfFalse" component="pagu" interface="standby”/>
        <bind role="resume" component="idApl” />
        <bind role="setAsTrue" component="idApl" interface="visible”/>
     </link>
  </body>
</ncl>
```

Listagem 10.20 Aplicação NCL preparada para uma interrupção quando há vídeos entrelaçados.

10.9 Elos do Tipo “get and set”

Os elos vistos até aqui definem como valores de parâmetro apenas valores literais fixos, ou seja, até o momento só podemos atribuir valores fixos a propriedades de um objeto. No entanto, às vezes é importante utilizar o valor atual de uma propriedade como o novo valor de uma outra propriedade.

Como vimos na Seção 10.3, se o valor do atributo *eventType* de elemento <simpleAction> for “attribution”, o elemento deve também definir o valor a ser atribuído, através de seu atributo *value*. Se esse valor for especificado como “$qualquerNome” (onde o “$” é símbolo reservado e “qualquerNome” é qualquer cadeia de caracteres, exceto um dos nomes reservados para papéis), o valor a ser atribuído deve ser obtido da propriedade ligada a *role*=“qualquerNome”, definida em um elemento <bind> do elemento <link> que utiliza o conector. Se esse valor não puder ser obtido, nenhuma atribuição deve ser realizada.

Devemos chamar a atenção para o fato de que, no conector usado pelo elo, o papel “qualquerNome” não precisa ser declarado, ele é declarado implicitamente. Em outras palavras, declarar o atributo *role*=“qualquerNome” em um elemento <bind> de um <link>, implica ter um papel implicitamente declarado como: <attributeAssessment role=“qualquerNome” eventType=“attribution” attributeType=“nodeProperty” />. Esse é o único caso possível de um elemento <bind> se referir a um papel não definido explicitamente em um conector. A única restrição é que o valor a ser atribuído deve obrigatoriamente ser o valor de uma propriedade (elemento <property>) de um componente da mesma composição onde o elo que referencia o evento é definido ou uma propriedade da própria composição onde o elo é definido ou, ainda, uma propriedade de um elemento acessível através de uma porta de um contexto ou switch (elementos <port> ou <switchPort>) aninhado na mesma composição onde o elo é definido.

Mas vamos a um exemplo para tornar claro o conceito. Suponha que exista um áudio que, ao ser continuado de um ponto onde foi pausado, deve pausar um outro áudio e tocar com o mesmo volume do áudio que pausou. Suponha ainda que essa mudança de áudios ocorra quando um vídeo de propaganda termina. A Listagem 10.21 apresenta um elo que define esse comportamento.

```
<causalConnector id="onEndSet_varPauseResume">
  <connectorParam name="var" />
  <simpleCondition role="onEnd" />
  <compoundAction operator="seq">
     <simpleAction role="set" value="$var" />
     <simpleAction role="pause" />
```

```
    <simpleAction role="resume" />
  </compoundAction>
</causalConnector>
                                                    ... trecho da seção <head>
```

```
                                                    ... trecho da seção <body>
<link xconnector="onEndSet_varPauseResume">
    <bind role="onEnd" component="propaganda"/>
    <bind role="getValue" component="rock" interface="soundLevel"/>
    <bind role="set" component="samba" interface="soundLevel">
       <bindParam name="var" value="$getValue"/>
    </bind>
    <bind role="pause" component="rock"/>
    <bind role="resume" component="samba"/>
</link>
```

Listagem 10.21 Utilização de um papel denominado *getValue* para alinhar o volume de dois áudios.

Nesse caso, a declaração de um papel "*getValue*" no elemento `<bind>` define implicitamente uma avaliação do tipo `<attributeAssessment role="getValue" eventType="attribution" attributeType="nodeProperty" />`.

10.10 Conectores e Elos com Atribuição ao Longo do Tempo para Efeito de Animação

Em todos os exemplos anteriores, as mudanças nos valores de propriedades disparadas por ações de elos foram instantâneas. No entanto, uma mudança brusca de algumas propriedades, como por exemplo a posição vertical de um objeto, pode desorientar o usuário. Para evitar esse tipo de problema, podemos criar um efeito de animação através da mudança gradual do valor da propriedade.

Como vimos na Seção 10.3, uma ação de atribuição (*role*="set") possui dois atributos opcionais, *duration* e *by*, cujos valores *defaults* são "0" e "indefinite", respectivamente. O atributo *duration* define o período de tempo decorrido entre o disparo do elo e a atribuição do valor final à propriedade ligada pelo elo. Já o atributo *by* define o incremento utilizado para cada atribuição. A atribuição de valores segue uma função linear, onde cada passo é definido pelo valor do atributo *by*.

A Listagem 10.22 apresenta o código de um conector e um elo que produzem um efeito de animação na propriedade "top" de um objeto. Note

que, no exemplo, quando o objeto de mídia "video1" atinge o trecho "aMenu", um outro objeto de mídia "menu" começa a ser exibido em uma posição dada pelo descritor que foi associado à exibição desse objeto. No entanto, o objeto "menu" se move verticalmente, desde o início de sua exibição, até que, passados dois segundos, os dois objetos "video1" e "menu" estejam alinhados pelo topo.

```
<causalConnector id="onBeginStartSet_vNewValue_vDuration_vBy">
      <connectorParam name="vNewValue" />
      <connectorParam name="vDuration" />
      <connectorParam name="vBy" />
      <simpleCondition role="onBegin" />
      <compoundAction operator="par">
        <simpleAction role="start" max="unbounded" />
        <simpleAction role="set" value="$vNewValue" max="unbounded"
                              duration="$vDuration" by="$vBy" />
      </compoundAction>
</causalConnector>
```

... trecho da seção <head>

... trecho da seção <body>

```
<link id="linkAlignTopPosition"
               xconnector="onBeginStartSet_vNewValue_vDuration_vBy">
        <bind role="onBegin" component="video1" interface="aMenu" />
        <bind role="start" component="menu" />
        <bind role="get" component="video1" interface="top" />
        <bind role="set" component="menu" interface="top">
               <bindParam name="vNewValue" value="$get" />
               <bindParam name="vDuration" value="2s" />
               <bindParam name="vBy" value="4" />
        </bind>
</link>
```

Listagem 10.22 Alinhamento gradual do topo de duas mídias.

10.11 Importação de Conectores

Conectores da base de conectores de um documento NCL podem ser importados através do elemento <importBase>, filho de <connectorBase>, de forma semelhante à importação das bases vistas nos outros capítulos. Basta definir os atributos *alias* ("apelido" do arquivo importado) e *documentURI* (a localização e o nome do arquivo que contém a base a ser importada).

A Listagem 10.23 ilustra a importação de uma base de conectores e o uso de um dos conectores importados. Esse exemplo assume que, no arquivo "conectores.ncl", existe um <causalConnector> com *id* "onKeySelectionStartStop".

```
<connectorBase>
    <importBase alias="meusConectores" documentURI="conectores.ncl"/>
</connectorBase>
                                                    ... trecho da seção <head>
```

```
                                                    ... trecho da seção <body>
<link xconnector="meusConectores#onKeySelectionStartStop">
    <bind component="menu" interface="pMenuItem1" role="onSelection"/>
    <bind component="menu" role="stop"/>
    <bind component="prog01" role="start"/>
</link>
```

Listagem 10.23 Importação de uma base de conectores.

10.12 Elos Referenciando Composições

Se um elemento <simpleAction> com valor de atributo *eventType* igual a "presentation" for associado por um elo a um elemento <context> ou <body> como um todo (ou seja, sem que uma de suas interfaces seja informada), a instrução deve obrigatoriamente ser refletida nas máquinas de estado de evento dos nós filhos da composição. As seguintes regras são seguidas pelo formatador NCL.

Se um elemento <context> ou <body> participar em um papel (role) *action* cujo tipo de ação é "start", quando essa ação for acionada, a instrução *start* também é aplicada a todos os eventos de apresentação mapeados pelas portas dos elementos <context> ou <body>. Se quisermos iniciar a apresentação usando uma porta específica, devemos indicar o *id* do elemento <port> como valor de interface do elemento <bind>.

Se um elemento <context> ou <body> participar em um papel (role) *action* cujo tipo de ação é "stop" ou "abort", quando essa ação for acionada, a instrução também é aplicada a todos os eventos de apresentação dos nós filhos da composição. Se a composição contiver elos sendo avaliados, as avaliações são suspensas e nenhuma ação correspondente é acionada.

Se um elemento <context> ou <body> participar em um papel (role) *action* cujo tipo de ação é "pause", quando essa ação for acionada, a instrução pause é aplicada a todos os eventos de apresentação dos nós filhos da composição que estejam no estado *occurring*. Se a composição contiver elos sendo avaliados, todas as avaliações devem ser suspensas até que uma ação *resume*, *stop* ou *abort* seja emitida. Se a composição contiver nós-filhos com eventos de apresentação no estado *paused* quando a ação *pause* for emitida, esses nós são identificados, porque, se a composição receber uma instrução *resume*, esses eventos não devem ser retomados.

Se um elemento <context> ou <body> participar em um papel (role) action cujo tipo de ação é "resume", quando essa ação for acionada, a instrução resume é aplicada a todos os eventos de apresentação dos nós filhos da composição que estejam no estado *paused*, exceto aqueles que já estavam pausados quando a composição foi pausada. Se a composição contiver elos com avaliações pausadas, elas são retomadas.

10.13 Referência Rápida — Conectores

A Tabela 10.8 sumariza os atributos e o conteúdo dos elementos que definem conectores. Como usual, os atributos obrigatórios estão sublinhados.

Tabela 10.8 Elementos, atributos e conteúdo que definem conectores no perfil NCL EDTV

Elementos	Atributos	Conteúdo
causalConnector	*id*	(connectorParam*, (simpleCondition | compoundCondition), (simpleAction | compoundAction))
connectorParam	*name*, *type*	—
simpleCondition	*role*, *delay*, *eventType*, *key*, *transition*, *min*, *max*, *qualifier*	—
compoundCondition	*operator*, *delay*	((simpleCondition | compoundCondition)+, (assessmentStatement | compoundStatement)*)
simpleAction	*role*, *delay*, *eventType*, *actionType*, *value*, *min*, *max*, *qualifier*, *repeat*, *repeatDelay*, *duration*, *by*	—
compoundAction	*operator*, *delay*	(simpleAction | compoundAction)+
assessmentStatement	*comparator*	(attributeAssessment, (attributeAssessment | valueAssessment))
attributeAssessment	*role*, *eventType*, *key*, *attributeType*, *offset*	—
valueAssessment	*value*	—
compoundStatement	*operator*, *isNegated*	(assessmentStatement | compoundStatement)+

ABNT NBR 15606-2 2011. Associação Brasileira de Normas Técnicas, "Televisão digital terrestre — Codificação de dados e especificações de transmissão para radiodifusão digital, Parte 2: Ginga-NCL para receptores fixos e móveis — Linguagem de aplicação XML para codificação de aplicações", Sistema Brasileiro de TV Digital Terrestre, *NBR 15606-2*.

ITU-T H.761 2011. Nested Context Language (NCL) and Ginga-NCL for IPTV. ITU-T Rec. H.761. Genebra.

Capítulo 11

Adaptação do Conteúdo e da Apresentação: Regras, *Switches* e *Switches* de Descritor

Este capítulo descreve como adaptar uma aplicação durante sua execução, através da avaliação de regras e a correspondente seleção de objetos ou de formas de apresentação (descritores) de objetos. Ao final deste capítulo, você será capaz de construir aplicações que fazem uso de regras, *switches* e *descriptorSwitches* para adaptar o conteúdo de uma aplicação e a forma de apresentação desse conteúdo.

11.1 Regras

As regras usadas em uma aplicação NCL são definidas no elemento <ruleBase>, filho do elemento <head>.[1] Cada regra possui a referência a uma propriedade do objeto *settings*, um operador de comparação e um valor, conforme a Listagem 11.1. O identificador de uma regra é opcional e quando definido deve seguir a mesma regra de formação para o atributo *id* definida no Capítulo 5.

```
<ruleBase>
   <rule id="rLegendaLigada" var="legenda"
                                    comparator="eq" value="ligada" />
   <rule id="rLegendaDesligada" var="legenda"
                               comparator="eq" value="desligada" />
</ruleBase>
```

Listagem 11.1 Definição de uma base de regras.

O valor do atributo *var* deve ser uma propriedade do elemento <media type="application/x-ncl-settings">. Já o atributo *comparator* pode assumir um dos valores apresentados na Tabela 11.1.[2]

Tabela 11.1 Operadores de Comparação que Podem ser Utilizados nas Regras

Valor	Significado
eq	igual a (*equal to*)
ne	diferente de (*not equal to*)
gt	maior que (*greater than*)
lt	menor que (*less than*)
gte	maior ou igual a (*greater than or equal to*)
lte	menor ou igual a (*less than or equal to*)

A NCL também permite definir regras compostas, através do elemento <compositeRule>, cujo atributo *operator* pode assumir os valores "and" ou "or". Uma regra para avaliar se a legenda está ligada e o idioma é português poderia ser definida do seguinte modo:

[1] Regras e seus atributos são definidos no módulo **TestRule** [ABNT, NBR 15606-2, 2011; ITU-T, H.761, 2011].

[2] Esses operadores de comparação são idênticos aos apresentados na Tabela 10.5.

```
<compositeRule id="idiomaPtlegendaOn" operator="and">
  <rule id="rLegendaLigada" var="legenda" comparator="eq" value="on"/>
  <rule id="rIdioma" var="idioma" comparator="eq" value="pt"/>
</compositeRule>
```

As regras são utilizadas para adaptação de conteúdo, através do elemento <switch> e para adaptação de apresentação, através do elemento <descriptorSwitch>, vistos a seguir.

11.2 Switch

Um <switch> é uma composição contendo nós (objetos de mídia, contextos, ou outros switches) alternativos, ou seja, dentre os quais apenas um será selecionado.[3] Um switch pode conter qualquer tipo de nó — de mídia, de contexto e outros *switches*. A decisão sobre qual nó será selecionado é dada por regras de mapeamento, definidas através de elementos <bindRule>.[4] As regras são avaliadas na ordem em que foram definidas. A primeira regra avaliada como verdadeira terá seu nó correspondente selecionado. Além disso, podemos definir um nó que será selecionado por *default*, no caso de nenhuma regra ser satisfeita, através do elemento <defaultComponent>.

A Listagem 11.2 apresenta uma base de regras que avaliam o valor da propriedade "system.language" e um *switch* que, ao ser acionado, apresenta a mídia de áudio correspondente à regra em vigor.

```
<ruleBase>
   <rule id="rEn" var="system.language" comparator="eq" value="en" />
   <rule id="rPt" var="system.language" comparator="eq"  value="pt" />
</ruleBase>
```

... trecho da seção <head>

```
<media id="noSettings" type="application/x-ncl-settings">
        <property name="system.language" />
</media>
...
<switch id="switchAudioIdioma">
   <bindRule rule="rEn" constituent="audioEn" />
   <bindRule rule="rPt" constituent="audioPt" />
   <defaultComponent component="audioPt" />
```

[3] *Switches* e seus atributos são definidos no módulo **ContentControl** [ABNT, NBR 15606-2, 2011; ITU-T, H.761, 2011].

[4] O elemento <bindRule> é definido no módulo **TestRuleUse** [ABNT, NBR 15606-2, 2011; ITU-T, H.761, 2011].

```
   <media id="audioEn" src="media/audioEn.mp3" descriptor="dAudio1" />
   <media id="audioPt" src="media/audioPt.mp3" descriptor="dAudio1" />
</switch>
                                                    ... trecho da seção <body>
```

Listagem 11.2 Regras e *switch* que seleciona o áudio conforme o idioma em vigor.

Nesse exemplo, a mídia "audioEn" será reproduzida caso a regra "rEn" seja válida, ou seja, caso "system.language" possua o valor "en". Se essa regra não for válida, o próximo mapeamento é avaliado, e assim sucessivamente, até uma regra válida ser alcançada ou até terminar o conjunto de mapeamentos daquele <switch>. Caso o idioma possua um valor diferente de "en" e "pt", o nó "audioPt" é apresentado, conforme definido pelo elemento <defaultComponent>.

A Figura 11.1 ilustra a visão estrutural do trecho de código da Listagem 11.2.

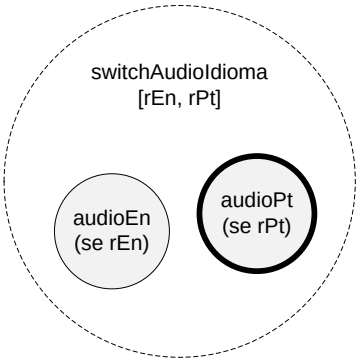


Figura 11.1 Visão estrutural ilustrando um <switch> com duas regras.

No exemplo anterior, o objeto *switch* selecionou um objeto por inteiro, ou seja, selecionou para exibição a "âncora de conteúdo total" do objeto, definida por *default*. Quando é necessário selecionar não apenas o objeto, mas também qual de suas interfaces, o elemento <switchPort> deve ser definido. O elemento <switchPort> permite a criação de interfaces em um elemento <switch>, que podem ser mapeadas a um conjunto de interfaces alternativas de nós internos, permitindo a um elo ancorar no componente e na interface escolhida quando o <switch> for processado. Um elemento <switchPort> conterá, assim, um conjunto de elementos de mapeamento. Um elemento de mapeamento (<mapping>) define um caminho a partir do <switchPort> para uma interface (atributo *interface*) de um dos componentes do <switch> (especificados por seu atributo *component*).

Como novo exemplo vamos assumir que, na Listagem 11.2, os objetos de mídia de áudio, quando escolhidos, tocarão apenas um trecho de seu

conteúdo, especificado pelas âncoras de conteúdo “trechoPt” e “trechoEn”, respectivamente. Para esse novo exemplo, teremos a Listagem 11.3.

```
                                                    ... trecho da seção <head>
<ruleBase>
   <rule id=“rEn” var=“system.language” comparator=“eq” value=“en” />
   <rule id=“rPt” var=“system.language” comparator=“eq”  value=“pt” />
</ruleBase>
```

```
                                                    ... trecho da seção <body>
<media id=“noSettings” type=“application/x-ncl-settings“>
       <property name=“system.language” />
</media>
...
<switch id=“switchAudioIdioma”>
   <switchPort id="spaudio">
        <mapping component="audioEn" interface=“trechoEn” />
        <mapping component="audioPt" interface=“trechoPt” />
   </switchPort>
   <bindRule rule=“rEn” constituent=“audioEn” interface=“ />
   <bindRule rule=“rPt” constituent=“audioPt” />
   <defaultComponent component=“audioPt” />
   <media id=“audioEn” src=“media/audioEn.mp3” descriptor=“dAudio1”>
     <area id=“trechoEn” begin=20s” end=“50s” />
   </media>
   <media id=“audioPt” src=“media/audioPt.mp3” descriptor=“dAudio1” />
     <area id=“trechoPt” begin=20s” end=“50s” />
   </media>
</switch>
```

Listagem 11.3 Regras e *switch* que seleciona um trecho do áudio conforme o idioma em vigor.

A Figura 11.2 ilustra a visão estrutural do trecho de código da Listagem 11.3.

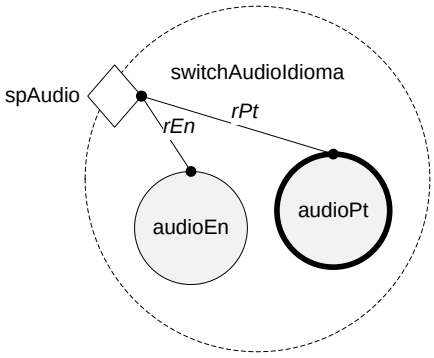


Figura 11.2 Visão estrutural ilustrando um <switch> com duas regras e um <switchPort> “spAudio”.

Embora não seja obrigatório, mesmo quando um *switch* seleciona objetos por inteiro, é boa prática de programação identificar esse tipo de seleção explicitamente utilizando o elemento <switchPort>. Seguindo essa regra, o exemplo da Listagem 11.2 ficaria como na Listagem 11.4.

```
                                                    ... trecho da seção <head
<ruleBase>
   <rule id="rEn" var="system.language" comparator="eq" value="en" />
   <rule id="rPt" var="system.language" comparator="eq"  value="pt" />
</ruleBase>
```

```
                                                    ... trecho da seção <body>
<media id="noSettings" type="application/x-ncl-settings">
       <property name="system.language" />
</media>
...
<switch id="switchAudioIdioma">
   <switchPort id="spaudio">
       <mapping component="audioEn" />
       <mapping component="audioPt" />
   </switchPort>
   <bindRule rule="rEn" constituent="audioEn" />
   <bindRule rule="rPt" constituent="audioPt" />
   <defaultComponent component="audioPt" />
   <media id="audioEn" src="media/audioEn.mp3" descriptor="dAudio1" />
   <media id="audioPt" src="media/audioPt.mp3" descriptor="dAudio1" />
</switch>
```

Listagem 11.4 Regras e *switch* que seleciona o áudio conforme o idioma em vigor, usando elementos <switchPort>.

A Figura 11.3 ilustra o uso de *switches* na escolha de objetos internos a contextos, cujo trecho de código NCL é dado pela Listagem 11.5.

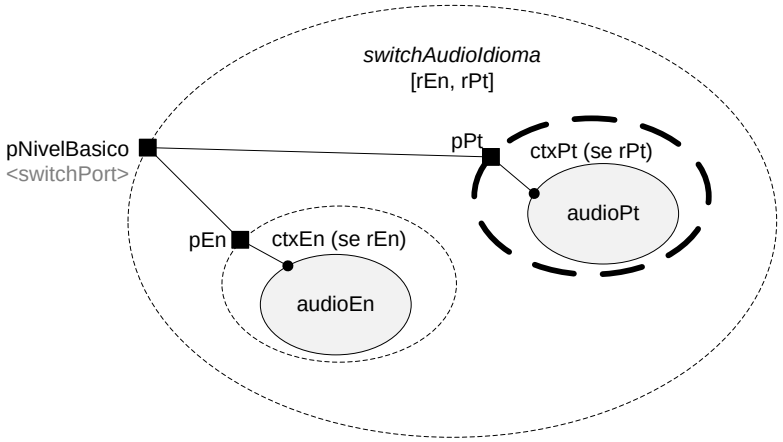


Figura 11.3 *Switch* contendo contextos.

```
<switch id="switchAudioIdioma">
        <bindRule rule="rEn" constituent="ctxEn" />
        <bindRule rule="rPt" constituent="ctxPt" />
        <defaultComponent component="ctxPt" />

        <switchPort id="pNivelBasico">
                <mapping component="ctxEn" interface="pEn" />
                <mapping component="ctxPt" interface="pPt" />
        </switchPort>

        <context id="ctxEn">
           <port id="pEn" component="audioEn" />
           <media id="audioEn" src="media/audioEn.mp3"
                                              descriptor="dAudio1" />
                <!-- outros nós e elos do contexto ctxEn -->
        </context>
        <context id="ctxPt">
           <port id="pPt" component="audioPt" />
           <media id="audioPt" src="media/audioPt.mp3"
                                              descriptor="dAudio1" />
                 <!-- outros nós e elos do contexto ctxPt -->
        </context>
</switch>
```

Listagem 11.5 *Switch* contendo contextos.

Sendo mais precisos, podemos dizer que uma referência (p. ex., por meio de um elo) a um componente interno a um switch deve ser feita através de um elemento <switchPort> ou, por *default* ao elemento <switch> sem especificar uma <switchPort>. Nesse último caso, é considerado como se a referência fosse feita a uma <switchPort> que contivesse elementos <mapping> para cada elemento filho do switch e se referindo às âncoras de conteúdo total desses elementos. Mais ainda, um elemento <switchPort> pode definir um mapeamento para apenas um subconjunto dos elementos filhos do switch. Quando um elo ancorando nessa <switchPort> é avaliado e nenhuma das regras que ligam aos elementos mapeados é satisfeita, o elemento definido pelo elemento <defaultComponent> deve ser o escolhido, caso o elemento <defaultComponent> tiver sido definido; em caso contrário nenhum elemento deve ser selecionado.

11.3 *Switch* de Descritor

Assim como utilizamos o elemento <switch> para selecionar um objeto conforme uma regra, podemos utilizar o elemento <descriptorSwitch> para selecionar um descritor conforme uma regra. Assim como no caso anterior,

são utilizados elementos <bindRule> para ligar uma regra ao descritor correspondente, como exemplificado na Listagem 11.6.

```
<descriptorBase>
      ...
      <descriptorSwitch id="dLegenda">
        <bindRule rule="rLegendaOn" constituent="dLegendaOn" />
        <bindRule rule="rLegendaOff" constituent="dLegendaOff" />
        <descriptor id="dLegendaOn" region="rgLegenda" />
        <descriptor id="dLegendaOff" region="rgLegenda">
                <descriptorParam name="visible" value="false" />
        </descriptor>
      </descriptorSwitch>
</descriptorBase>
```

Listagem 11.6 Definição de um <descriptorSwitch> que seleciona um descritor que exibe ou oculta as legendas, conforme as regras "*rLegendaOn*" e "*rLegendaOff*".

Na Listagem 11.6, o *switch* de descritor indica que, caso a legenda esteja ligada (regra "rLegendaOn" válida), o descritor utilizado é "dLegendaOn", que exibe a mídia correspondente na região "rgLegenda", com os parâmetros de exibição *default*. Caso a legenda esteja desligada, por outro lado (regra "rLegendaOff" válida), o descritor utilizado é o "dLegendaOff", através do qual a mídia é oculta (parâmetro "**visible**" com valor "false").

```
<media id="video1" src="media/melhoresMomentos.mpg"
                                        descriptor="dVideoFullscreen">
      <area id="a1" begin="2s" end="5s" />
      <area id="a2" begin="7s" end="9s" />
      <area id="a3" begin="11s" end="13s" />
</media>
<media id="legenda1" src="media/leg01.html" descriptor="dLegenda" />
<media id="legenda2" src="media/leg02.html" descriptor="dLegenda" />
<media id="legenda3" src="media/leg03.html" descriptor="dLegenda" />
...
<!-- legenda 1 -->
<link xconnector="onBeginStart">
      <bind component="video1" interface="a1" role="onBegin" />
      <bind component="legenda1" role="start" />
</link>
<link xconnector="onEndStop">
      <bind component="video1" interface="a1" role="onEnd" />
      <bind component="legenda1" role="stop" />
</link>
...
```

Listagem 11.7 Elos que fazem uso de um <descriptorSwitch> de maneira idêntica à que faziam de um <descriptor> homônimo.

11.4 Referência Rápida — Regras, *Switches* e *Switches* de Descritor

Nas tabelas a seguir, como é usual em nossas descrições, os atributos obrigatórios estão sublinhados.

A Tabela 11.2 apresenta os elementos e atributos utilizados para definir o elemento <rule>, conforme o perfil NCL EDTV.

Tabela 11.2 Elementos e Atributos que Definem Regras no Perfil EDTV

Elementos	Atributos	Conteúdo
ruleBase	*id*	(importBase|rule|compositeRule)+
rule	*id*, *var*, *comparator*, *value*	—
compositeRule	*id*, *operator*	(rule | compositeRule)+

A Tabela 11.3 apresenta os elementos e atributos utilizados para definir elementos <switch>, conforme o prefil NCL EDTV.

Tabela 11.3 Elementos e Atributos que Definem Elementos <switch> no Perfil NCL EDTV

Elementos	Atributos	Conteúdo
switch	*id*, *refer*	(defaultComponent?, (switchPort|bindRule|media|context|switch)*)
bindRule	*constituent*, *rule*	—
defaultComponent	*component*	—
switchPort	*id*	mapping+
mapping	*component*, *interface*	—

A Tabela 11.4 apresenta os elementos e atributos utilizados para definir elementos <descriptorSwitch>, conforme o perfil NCL EDTV.

Tabela 11.4 Elementos e Atributos que Definem Elementos <descriptorSwitch> no Perfil NCL EDTV

Elementos	Atributos	Conteúdo
descriptorSwitch	*id*	(defaultDescriptor?, (bindRule|descriptor)*)
defaultDescriptor	*descriptor*	—

Bibliografia

ABNT NBR 15606-2 2011. Associação Brasileira de Normas Técnicas, "Televisão digital terrestre — Codificação de dados e especificações de transmissão para radiodifusão digital, Parte 2: Ginga-NCL para receptores fixos e móveis — Linguagem de aplicação XML para codificação de aplicações", Sistema Brasileiro de TV Digital Terrestre, *NBR 15606-2*.

ITU-T H.761 2011. Nested Context Language (NCL) and Ginga-NCL for IPTV. ITU-T Rec. H.761. Genebra.

Capítulo 12

Metadados

Metadados são "dados sobre dados". Um metadado não contém informações *de* conteúdo utilizadas ou exibidas durante a apresentação de um documento. Em vez disso, contém informações *sobre* o conteúdo utilizado ou exibido, que podem ser processadas automaticamente. Por exemplo, podemos definir palavras-chave como metadados de uma aplicação NCL para facilitar buscas de aplicações com determinadas características.

12.1 Metadados em Aplicações NCL

A NCL permite dois tipos de metadados, representados pelos elementos <meta> e <metadata>. Esses elementos e seus atributos são definidos no módulo **Metainformation** e sumarizados na Tabela 12.1.

Tabela 12.1 Elementos, Atributos e Conteúdo (Elementos Filhos) Definidos pelo Módulo **Metainformation** para o Perfil EDTV

Elementos	Atributos	Conteúdo
meta	*name, content*	—
metadata	—	árvore RDF

O elemento <meta> é destinado a metadados simples, com uma propriedade (definida pelo atributo *name*) e um valor ou lista de valores (definidos pelo atributo *content*). Por exemplo, o autor, a data de criação e o tipo de licença de uma aplicação NCL podem ser representados conforme ilustrado pela Listagem 12.1.

```
<meta name="author" content="Telemidia" />
<meta name="data" content="2009-01-03" />
<meta name="license" content="cc 3.0 by-nc-nd" />
```

Listagem 12.1 Definição de metadados através de listas de propriedades.

O elemento <metadata> é destinado a metadados mais estruturados. Eles são representados como uma árvore RDF (1999), onde o elemento <metadata> atua como a raiz da árvore, como ilustrado na Listagem 12.2.

```
<metadata>
    <rdf:RDF
       xmlns:rdf = “http://www.w3.org/1999/02/22-rdf-syntax-ns#”
       xmlns:dc="http://purl.org/dc/elements/1.1/">
       <rdf:Description
          rdf:about=“http://www.telemidia.puc-rio.br/ex/ex01”>
          <dc:title>Meu primeiro exemplo NCL</dc:title>
          <dc:description>
             Exemplo simples, sem sincronismo de midia nem
interatividade, para familiarizar o desenvolvedor com a estrutura de
uma aplicacao NCL
          </dc:description>
          <dc:publisher>Telemidia</dc:publisher>
          <dc:date>2009</dc:date>
          <dc:rights>Copyright 2009 Telemidia</dc:rights>
          <dc:license> cc 3.0 by-nc-nd</dc:license>
          <dc:format> text/ncl”</dc:format>
          <dc:creator>
             <rdf:Seq ID=“CreatorsAlphabetical”>
               <rdf:li>Luiz Fernando Gomes Soares</rdf:li>
               <rdf:li>Simone Barbosa</rdf:li>
             </rdf:Seq>
          </dc:creator>
       </rdf:Description>
    </rdf:RDF>
</metadata>
```

Listagem 12.2 Definição de metadados através de árvore RDF.

Metadados podem ser associados a uma aplicação NCL como um todo, e, nesse caso, devem ser definidos como elementos filhos do elemento <head> de uma aplicação. Metadados podem também ser associados a contextos (definidos como elementos filhos deles), representados pelos elementos <body> e <context>. A Listagem 12.3 ilustra a definição de metadados em um contexto de propaganda.

```
<context id=“propaganda”>
       <meta name=“produto” content=“detergente LimpaBem” />
       <meta name=“agencia” content=“MarketingDoBom” />
       <meta name=“data” content=“2009-05-03” />
       ...
</context>
```

Listagem 12.3 Definição de metadados em um contexto.

12.2 Exemplo de Metadados na Aplicação *O Primeiro João*

Como exemplo, vamos retomar uma das listagens do Capítulo 3, sobre a animação *O Primeiro João* e vamos recheá-la de metadados, salientados em negrito na Listagem 12.4. Teremos metadados para:

1) informações sobre a aplicação em si, seus autores, *copyright* etc., inseridas no elemento <head>;
2) informações sobre o conteúdo dos objetos de mídia utilizados, inseridas no elemento <body>;
3) informações sobre a propaganda adicionada por interação, inseridas no elemento <context> que a define.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="switch" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
  <head>
    <ruleBase>
      <rule id="en" var="system.language" value="en" comparator="eq”/>
      <rule id="int" var="service.interactivity" value="true"
                                                    comparator="eq”/>
    </ruleBase>
    <regionBase>
      <region id="backgroundReg" width="100%" height="100%"
                                                    zIndex="1”/>
      <region id="screenReg" width="100%" height="100%" zIndex="2">
       <region id="frameReg" left="5%" top="6.7%"
                  width="18.5%"       height="18.5%" zIndex="3”/>
       <region id="iconReg" left="87.5%" top="11.7%"
                  width="8.45%"       height="6.7%" zIndex="3”/>
       <region id="shoesReg" left="15%" top="60%"
                       width="25%"    height="25%" zIndex="3”/>
       <region id="formReg" left="56.25%" top="8.33%"
                  width="38.75%" height="71.7%" zIndex="3”/>
      </region>
    </regionBase>
    <descriptorBase>
      <descriptor id="backgroundDesc" region="backgroundReg”/>
      <descriptor id="screenDesc" region="screenReg”/>
      <descriptor id="photoDesc" region="frameReg" explicitDur="5s”/>
      <descriptor id="audioDesc”/>
      <descriptor id="dribleDesc" region="frameReg” />
      <descriptor id="iconDesc" region="iconReg" explicitDur="6s”/>
      <descriptor id="shoesDesc" region="shoesReg” />
      <descriptor id="formDesc" region="formReg" focusIndex="1"
                                            explicitDur="15s”/>
    </descriptorBase>
```

```
    <connectorBase>
      <importBase documentURI="../causalConnBase.ncl" alias="conEx”/>
    </connectorBase>
    <meta name=“telemidia_ref” content=“TM-0132” />

    <metadata>
      <rdf:RDF
      xmlns:rdf = “http://www.w3.org/1999/02/22-rdf-syntax-ns#”
      xmlns:dc="http://purl.org/dc/elements/1.1/">
      <rdf:Description
        rdf:about=“http://www.telemidia.puc-rio.br/ex/Joao”>
        <dc:title>O Primeiro João</dc:title>
        <dc:description>
          Exemplo de uso de metadados em aplicações NCL
        </dc:description>
        <dc:publisher>Telemidia</dc:publisher>
        <dc:date>2009</dc:date>
        <dc:rights>Copyright 2009 Telemidia</dc:rights>
        <dc:license> cc 3.0 by-nc-nd</dc:license>
        <dc:format> text/ncl”</dc:format>
        <dc:creator>
          <rdf:Seq ID=“CreatorsAlphabetical”>
            <rdf:li>Luiz Fernando Gomes Soares</rdf:li>
            <rdf:li>Simone Barbosa</rdf:li>
          </rdf:Seq>
        </dc:creator>
      </rdf:Description>
      </rdf:RDF>
    </metadata>
  </head>

  <body>
    <metadata>
      <rdf:RDF
      xmlns:rdf = “http://www.w3.org/1999/02/22-rdf-syntax-ns#”
      xmlns:dc="http://purl.org/dc/elements/1.1/">
      <rdf:Description
        rdf:about=“http://www.telemidia.puc-rio.br/m/Joao”>
        <dc:title>O Primeiro João”</dc:title>
        <dc:description>
          Animação sobre o Garrincha
        </dc:description>
        <dc:format> avi</dc:format>
        <dc:creator>André Castelão</dc:creator>
      </rdf:Description>

      <rdf:Description
        rdf:about=“http://www.telemidia.puc-rio.br/m/vF01”>
```

```
      <dc:title>Cenas de futebol</dc:title>
      <dc:description>
        Vídeo com cenas de futebol no CIDS
      </dc:description>
      <dc:format> avi</dc:format>
      <dc:creator>PC-CIDS-VG</dc:creator>
    </rdf:Description>

    <rdf:Description
      rdf:about="http://www.telemidia.puc-rio.br/m/iJ01">
      <dc:title>Imagem de jogadores</dc:title>
      <dc:description>
        Imagem de jogadores de futebol no CIDS
      </dc:description>
      <dc:format> avi</dc:format>
      <dc:creator>PC-CIDS-VG</dc:creator>
    </rdf:Description>

  </rdf:RDF>
</metadata>

<port id="entry" component="animation"/>
<media id="background" src="../media/background.png"
                                  descriptor="backgroundDesc"/>
<media id="animation" src="../media/animGar.mp4"
                                        descriptor="screenDesc">
  <area id="segDrible" begin="12s"/>
  <area id="segPhoto" begin="41s"/>
  <area id="segIcon" begin="45s" end="51s"/>
</media>
<media id="choro" src="../media/choro.mp3"
                                        descriptor="audioDesc"/>
<media id="drible" src="../media/drible.mp4"
                                        descriptor="dribleDesc"/>
<media id="photo" src="../media/photo.png"
                                        descriptor="photoDesc"/>
<context id="advert">
  <meta name="genero" content="Propaganda" />
  <meta name="autor" content="PC-CIDS-VG" />
  <media id="reusedAnimation" refer="animation"
                                        instance="instSame">
    <property name="bounds"/>
  </media>
  <media id="icon" src="../media/icon.png" descriptor="iconDesc"/>
  <media id="shoes" src="../media/shoes.mp4"
                                        descriptor="shoesDesc"/>
  <switch id="form">
  <switchPort id="spForm">
    <mapping component="enForm"/>
```

```
        <mapping component="ptForm"/>
      </switchPort>
      <bindRule constituent="enForm" rule="en"/>
      <defaultComponent component="ptForm"/>
      <media id="ptForm" src="../media/ptForm.htm"
                                          descriptor="formDesc"/>
      <media id="enForm" src="../media/enForm.htm"
                                          descriptor="formDesc"/>
     </switch>
     <link id="lIcon" xconnector="conEx#onBeginStart">
      <bind role="onBegin" component="reusedAnimation"
                                          interface="segIcon"/>
      <bind role="start" component="icon"/>
     </link>
     <link id="lBegingShoes"
                connector="conEx#onKeySelectionStopSet_varStart">
      <bind role="onSelection" component="icon">
       <bindParam name="keyCode" value="RED"/>
      </bind>
      <bind role="start" component="shoes"/>
      <bind role="start" component="form" interface="spForm"/>
      <bind role="set" component="reusedAnimation" interface="bounds">
       <bindParam name="var" value="5%,6.67%,45%,45%"/>
      </bind>
      <bind role="stop" component="icon" />
     </link>
     <link id="lEndForm" xconnector="conEx#onEndSet_var">
      <bind role="onEnd" component="form" interface="spForm"/>
      <bind role="set" component="reusedAnimation"
                                          interface="bounds">
       <bindParam name="var" value="0,0,222.22%,222.22%"/>
      </bind>
     </link>
    </context>
    <link id="lMusic" xconnector="conEx#onBeginStart_delay">
     <bind role="onBegin" component="animation"/>
     <bind role="start" component="background">
      <bindParam name="delay" value="5s"/>
     </bind>
     <bind role="start" component="choro">
      <bindParam name="delay" value="5s"/>
     </bind>
    </link>
    <link id="lDrible" xconnector="conEx#onBeginStart">
     <bind role="onBegin" component="animation"
                                          interface="segDrible"/>
     <bind role="start" component="drible"/>
    </link>
```

```
    <link id="lPhoto" xconnector="conEx#onBeginStart">
      <bind role="onBegin" component="animation"
                                          interface="segPhoto"/>
      <bind role="start" component="photo" />
    </link>
    <link id="lEnd" xconnector="conEx#onEndStop">
      <bind role="onEnd" component="animation"/>
      <bind role="stop" component="background"/>
      <bind role="stop" component="choro"/>
    </link>
  </body>
</ncl>
```

Listagem 12.4 Definição de metadados em um contexto.

Bibliografia

ABNT NBR 15606-2 2011. Associação Brasileira de Normas Técnicas, "Televisão digital terrestre — Codificação de dados e especificações de transmissão para radiodifusão digital, Parte 2: Ginga-NCL para receptores fixos e móveis — Linguagem de aplicação XML para codificação de aplicações", Sistema Brasileiro de TV Digital Terrestre, *NBR 15606-2*.

ITU-T H.761 2011. Nested Context Language (NCL) and Ginga-NCL for IPTV. ITU-T Rec. H.761. Genebra.

SMIL 2.1 Specification, SMIL 2.1 — Synchronized Multimedia Integration Language — SMIL 2.1 Specification, W3C Recommendation. Dezembro de 2005.

Capítulo 13

Reúso

À medida que as aplicações se tornam mais complexas e que padrões de projeto emergem na definição de diversos elementos, torna-se necessário buscar mecanismos que aumentem a eficiência da autoria dessas aplicações. Como vimos nos capítulos anteriores, a NCL proporciona o reuso de diversos elementos definidos nas suas bases (p. ex., <regionBase>, <descriptorBase>, <connectorBase> etc.). Este capítulo apresenta mecanismos adicionais de reuso fornecidos pela NCL: reuso de conteúdo, importação de documentos, importação de bases de um documento e reuso de nós, intra- e interdocumentos.[1]

Ao final deste capítulo, você será capaz de reutilizar elementos definidos em diferentes aplicações NCL.

[1] Este capítulo se baseia em Soares, L.F.G. e Soares Neto, C.S. [2009]. O uso do material foi gentilmente cedido pelo Departamento de Informática da PUC-Rio.

13.1 Espelhamento de Conteúdo

Em geral, cada elemento <media> define um objeto de mídia distinto. Mesmo quando dois objetos têm o mesmo valor de atributo *src*, por *default* eles constituem objetos de mídia distintos, cada qual com sua máquina de estados independente. Isso significa que a apresentação de um objeto é totalmente independente da apresentação do outro. Há casos, no entanto, em que pode ser necessário exibir não apenas o mesmo conteúdo em paralelo, mas efetivamente que esse conteúdo exiba as mesmas unidades de informação em um dado momento. Isso é realizado definindo no atributo *src* de um elemento que ele é a cópia "espelhada" de outro, referenciado no atributo. Embora definido unidirecionalmente, a relação de espelhamento entre os objetos é reflexiva, simétrica e transitiva. Não existe nenhuma dependência de mestre-escravo, ou seja, o espelhamento é em duas vias.

A Listagem 13.1 define três elementos <media> exatamente com o mesmo conteúdo. Os objetos "video1" e "video2" são independentes e, portanto, podem ser apresentados em paralelo em diferentes momentos de seu conteúdo, ao passo que, ao iniciar o "video3", ele será exibido, também em paralelo, mas no mesmo ponto em que "video1" estiver sendo exibido.

```
<media id="video1" src="propaganda.mp4" descriptor= "d1"/>
<media id="video2" src="propaganda.mp4" descriptor= "d2"/>
<media id="video3" src="ncl-mirror://video1" descriptor= "d3"/>
```

Listagem 13.1 Objetos de mídia distintos, mas com o mesmo conteúdo (arquivo-fonte definido pelo atributo *src*).

A igura 13.1apresenta um *storyboard* envolvendo esses objetos, ilustrando o mecanismo de espelhamento.

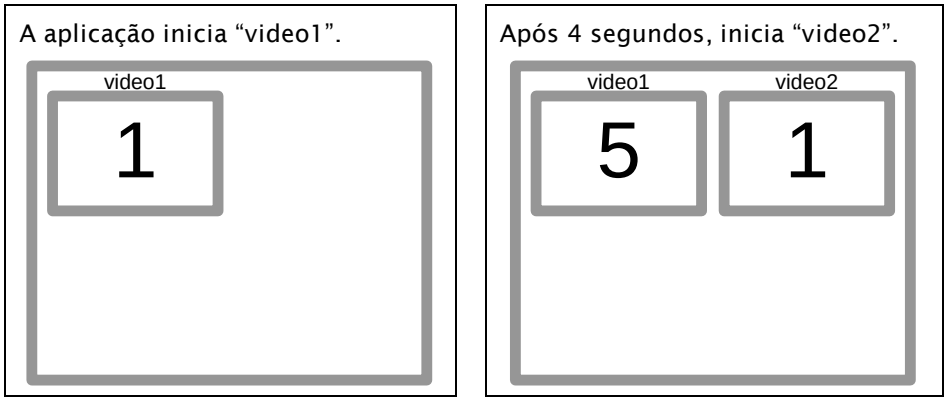


Figura 13.1 *Storyboard* ilustrando objetos de mídia independentes e espelhados.

13.2 Reúso de Objetos de Mídia

Como vimos brevemente no Capítulo 8, utilizamos os atributos *refer* e *instance* para reusar um objeto de mídia, incluindo seu conteúdo, seu descritor e quaisquer âncoras e propriedades definidos para ele.

O atributo *refer* indica o *id* do objeto que deve ser reutilizado. Existe uma única restrição do uso. Um atributo *refer* não pode referenciar um objeto definido na mesma composição que o contém diretamente.

Quando um elemento declara um atributo *refer*, todos os atributos e elementos filhos definidos pelo elemento referenciado são herdados. Todos os outros atributos e elementos filhos, se definidos pelo elemento que realiza a referência, são ignorados pelo formatador NCL, exceto o atributo *id* que deve ser definido e, em alguns, o atributo *descriptor*, que pode ser redefinido (nesse caso excepcional, o atributo *instance*, apresentado a seguir, deve ter o valor "new"). Ainda outra exceção é a possibilidade de se adicionar novos elementos filhos <area> e <property>. Se o novo elemento <property> adicionado tiver o mesmo atributo *name* de um elemento <property> já existente (definido no elemento <media> reutilizado), o novo elemento <property> adicionado é ignorado. Igualmente, se o novo elemento <area> adicionado tiver o mesmo atributo *id* de um elemento <area> já existente (definido no elemento <media> reutilizado), o novo elemento <area> adicionado é ignorado.

O atributo *instance* pode ter os seguintes valores:

- "new": objeto totalmente independente (reúso do código declarativo que especifica o objeto). Trata-se de um novo objeto que é uma cópia do objeto referenciado, incluindo suas âncoras de conteúdo e propriedades. Esse é o valor *default* para o atributo *instance*. Apenas nesse caso o atributo *descriptor* pode ser redefinido;

- "instSame": trata-se do mesmo objeto, iniciado para exibição junto com o objeto referido, isto é, o objeto em exibição incorporará, desde sua iniciação, *todas* as propriedades e âncoras de conteúdo definidas no elemento referenciado e nos elementos que o referenciam, e reportará todas as mudanças nas máquinas de estado de evento dessas interfaces a todos os elos associados a esses elementos <media>;

- “gradSame”: trata-se do mesmo objeto, mas que incorporará gradualmente as propriedades e âncoras de conteúdo definidas no elementos que referenciam e o referenciado, à medida que eles sofrerem ações de “start”. Apenas os elos associados aos elementos que já receberam a ação de start recebem a notificação de mudanças nas máquinas de estado de evento das interfaces já incorporadas.

Para facilitar o entendimento dessas diferentes formas de reúso, vamos considerar o código da Listagem 13.2.

```
<media id="video1" src="propaganda.mp4" descriptor="d1">
        <area id="a1_1" begin="3s" end="5s" />
</media>

<media id="video2_n" refer="video1" instance="new" descriptor="d2">
        <area id="a2_1" begin="1s" end="2s" />
        <area id="a2_2" begin="4s" end="6s" />
</media>

<media id="video3_i" refer="video1" instance="instSame">
        <area id="a3_1" begin="2s" end="5s" />
</media>

<media id="video4_g" refer="video1" instance="gradSame">
        <area id="a4_1" begin="1s" end="3s" />
        <area id="a4_2" begin="6s" end="9s" />
</media>
```

Listagem 13.2 Reúso de objetos de mídia através dos atributos *refer* e *instance*.

Vamos supor, ainda, que haja elos disparados no início e término de cada âncora, intitulados “link1_1_s”, “link1_1_e”, “link2_1_s”, “link2_1_e”, e assim por diante. A Figura 13.2 ilustra partes da visão estrutural correspondente. Podemos observar que a âncora “a1_1” é herdada por todos os objetos de mídia que se referem a “video1” e que, por isso, pode participar de outros elos nos diferentes contextos em que os objetos são definidos. Em particular, definimos um elo “link1_1_e_ctx4” no contexto “ctx4” para ilustrar essa característica.

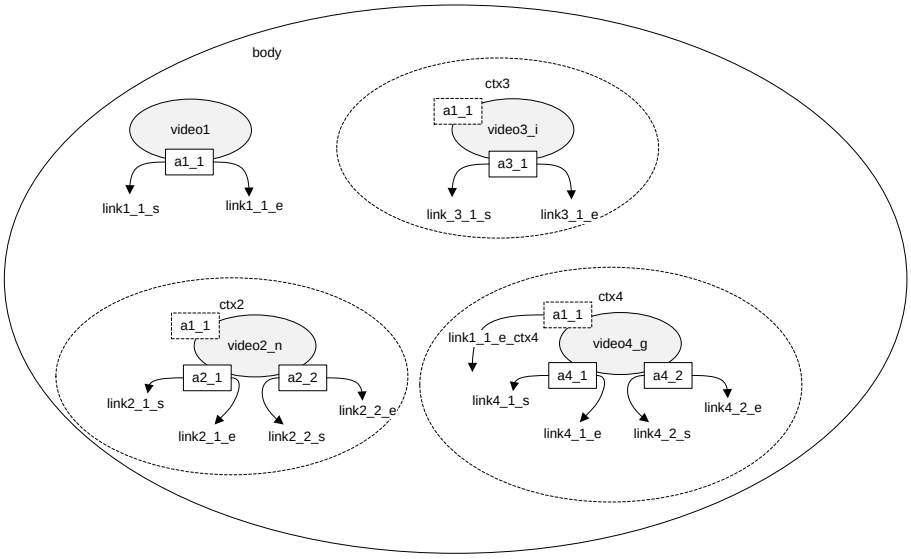


Figura 13.2 Visão estrutural parcial do exemplo de reuso de objetos de mídia.

A Tabela 13.1 ilustra o comportamento de uma aplicação NCL que inicia os objetos "video1", "video2_*n*" e "video4_g" em diferentes momentos (no início da aplicação, a três segundos do início e a quatro segundos do início, respectivamente) e termina "video2_n" e "video1" a oito e nove segundos do início, respectivamente.

Tabela 13.1 Comportamento da Aplicação de Exemplo de Reuso de Objetos de Mídia. Estado "o" significa "Occurring" e Estado "s" Significa "Sleeping"

Tempo do Início da Aplicação + Evento	Objeto	Estado da Âncora de Conteúdo Total	Tempo de Apresentação do Objeto	Âncoras em Occurring	Ativação de Elos
0 aplicação inicia **video1**	**video1**	**o**	**0**		
	video2_n	s	—		
	video3_i	**o**	**0**		
	video4_g	s	—		
1	video1	o	1		
	video2_n	s	—		
	video3_i	o	1		
	video4_g	s	—		
2	video1	o	2		
	video2_n	s	—		
	video3_i	o	2	**a3_1**	link3_1_s
	video4_g	s	—		

3 algum elo inicia **video2_n**	video1	o	3	**a1_1**	link1_1_s
	video2_n	**o**	**0**		
	video3_i	o	3	a3_1	
	video4_g	s	—		
4 algum elo inicia **video4_g**	video1	o	4	a1_1	
	video2_n	o	1	**a2_1**	link2_1_s
	video3_i	o	4	a3_1	
	video4_g	**o**	**4**		
5	video1	o	5		link1_1_e
	video2_n	o	2		link2_1_e
	video3_i	o	5		link3_1_e
	video4_g	o	5		link_1_1_e_ctx4
6	video1	o	6		
	video2_n	o	3		
	video3_i	o	6		
	video4_g	o	6	**a4_2**	link4_2_s
7	video1	o	7		
	video2_n	o	4	**a2_2**	link2_2_s
	video3_i	o	7		
	video4_g	o	7	a4_2	
8 algum elo encerra **video2_n**	video1	o	8		
	video2_n	**s**	**—**	a2_2	link2_2_e
	video3_i	o	8		
	video4_g	o	8	a4_2	
9 algum elo encerra **video1**	**video1**	**s**	**—**		
	video2_n	s	—		
	video3_i	**s**	**—**		
	video4_g	**s**	**—**		**link4_2_e**

Quando “video1” é iniciado, “video3_i” é automaticamente iniciado também. Quando “video4_g” é iniciado, ele assume o mesmo tempo de apresentação do objeto referido, “video1”. Como “video4_g” foi iniciado apenas a quatro segundos do início da aplicação, a âncora “a4_1” nunca ocorre e os elos “link4_1_s” e “link4_1_e” nunca são ativados.

Chamamos de *reúso de objeto de representação* as formas de reuso utilizadas pelos elementos <media> “video3_i” e “video4_g”: são o mesmo objeto de mídia em exibição, cujos elementos são iniciados juntos (no caso de *instance*=“instSame”) ou separadamente (no caso de *instance*=“gradSame”).

Por outro lado, “video2” é totalmente independente de “video1”. Trata-se de um reúso apenas do código declarativo que especifica o objeto, que chamamos de *reúso de objeto de dados*. Nesse tipo de reúso o objeto

"video2_n" aproveita todos os atributos, âncoras de conteúdo e propriedades do objeto referido ("video1"), mas se trata de um objeto de mídia diferente.

Uma observação importante é que esses elementos que representam o objeto de mídia não precisam nem devem estar num mesmo contexto. Pelo contrário, o reúso de objetos de mídia em diferentes contextos é a principal razão de ser desse mecanismo. Por exemplo, as âncoras de "video3_i" e "video4_g" podem indicar diferentes oportunidades de interação associadas a diferentes momentos de um mesmo vídeo, para iniciar um *quizz* e um *slide show*, respectivamente, cada qual em seu próprio contexto (contextos "ctx3" e "ctx4").

Reuso de objeto de mídia foi bastante explorado no Capítulo 3, nas várias versões da aplicação *O Primeiro João*. Sugerimos ao leitor revisitar aquele capítulo.

13.3 Reúso de Contextos

Diferentemente do reúso de objetos de mídia, o reúso de contextos é realizado apenas pelo atributo *refer* e consiste no reúso do código declarativo que o especifica. Em outras palavras, cria uma cópia do contexto referenciado. A Listagem 13.3 apresenta um exemplo de reúso de contexto.

```
<context id="menu">
        ...
</context>

<context id= "quizz">
        ...
        <context id="menu_no_quizz" refer="menu" />
</context>
```

Listagem 13.3 Reúso de contextos através do atributo *refer*.

13.4 Reúso entre Documentos NCL

O reúso apresentado nas seções anteriores envolveu elementos de um mesmo documento. A NCL oferece também diversas formas de reúso de elementos ou mesmo de documentos inteiros definidos em diferentes arquivos.

13.4.1 Importação de um Documento NCL

Como vimos nos capítulos anteriores, bases de regiões, de descritores, de transições e de regras podem ser importadas por um documento NCL por

meio do elemento <importBase> definido como filhos da base importadora e referenciando um URI da base a ser importada. A linguagem NCL permite, no entanto, que se importe também documentos, como um todo.

Para importar um ou mais documentos NCL, utilizamos o elemento <importedDocumentBase>, definido como filho do elemento <head> do documento importador, que agrupa um ou mais elementos <importNCL>, cada qual definindo um documento a ser importado.[2]

Quando se importa um documento por meio do elemento <importNCL>, todas as bases definidas dentro do documento importado, bem como o elemento <body> do documento, são importados, todos de uma vez. As bases são tratadas como se cada uma tivesse sido importada por um elemento <importBase>. O elemento <body> importado é tratado como um elemento <context>. É importante salientarmos que o elemento <importNCL> não "inclui" o documento NCL referido, mas apenas torna o documento referenciado visível para que os seus componentes possam ser reusados pelo documento que definiu o elemento <importNCL>, da mesma forma que reúsa objetos, como discutimos nas seções anteriores. Assim, o <body> importado, bem como quaisquer de seus nós, pode ser reusado dentro do documento NCL que realizou a importação.

O elemento <importNCL> tem dois atributos: *documentURI* e *alias*. O *documentURI* refere-se a um URI correspondente ao documento a ser importado. O atributo *alias* especifica um nome a ser utilizado quando for realizada uma referência a elementos desse documento importado. O nome deve ser único (type=ID) e seu escopo é restrito ao documento que definiu o atributo *alias*.

Para reusar um documento importado, criamos um contexto que se refere ao documento importado como "alias#id_do_documento_importado". Supondo que a aplicação prog1.ncl tenha *id*="prog1", o documento pode definir um contexto que se refere ao documento "prog1" inteiro. Como todo reúso de contexto, é criada uma cópia do contexto original.

Não apenas o documento como um todo, mas qualquer objeto definido pelo documento importado pode ser reusado. Para reusar um elemento, o atributo *refer* do elemento que fará o reúso deve conter o valor "alias#element_id". Como boa prática de programação, recomenda-se que o mesmo *alias* seja utilizado quando é necessário referir-se a elementos definidos nas bases do documento importado (<regionBase>, <connectorBase>, <descriptorBase> etc.).

[2] Os elementos de importação de documentos, bases e seus atributos são definidos no módulo **Import** [ABNT, NBR 15606-2, 2011; ITU-T, H.761, 2011].

Cabe ainda mencionar que, como em todas as importações de base, a operação do elemento <importNCL> também tem propriedade transitiva, ou seja, se o "documentoA" importar o "documentoB" que importa o "documentoC", então o "documentoA" importa o "documentoC". Entretanto, o *alias* definido para o "documentoC" dentro do "documentoB" é desconsiderado pelo "documentoA".

A Listagem 13.4 ilustra a importação de três documentos NCL e a definição dos contextos que se referem aos documentos importados. A listagem apresenta ainda um elo que inicia a apresentação de um dos contextos importados.

```
<ncl id="exemplo01"> ... </ncl>
                                        ... documento "prog1.ncl" a ser importado
```

```
<ncl id="exemplo02"> ... </ncl>
                                        ... documento "prog2.ncl" a ser importado
```

```
<ncl id="exemplo03"> ... </ncl>
                                        ... documento "prog3.ncl" a ser importado
```

```
<importedDocumentBase>
    <importNCL alias="docProg1" documentURI="prog1.ncl" />
    <importNCL alias="docProg2" documentURI="prog2.ncl" />
    <importNCL alias="docProg3" documentURI="prog3.ncl" />
</importedDocumentBase>
                    ... trecho da seção <head> do documento que importa os outros
```

```
                    ... trecho da seção <body> do documento que importa os outros
<context id="prog01" refer="docProg1#exemplo01"> ... </context>
<context id="prog02" refer="docProg2#exemplo02"> ... </context>
<context id="prog03" refer="docProg3#exemplo03"> ... </context>

<link xconnector="onKeySelectionStartStop">
    <bind role="onSelection" component="menu" interface="pMenuItem1"/>
    <bind role="stop" component="menu"/>
    <bind role="start" component="prog01"/>
</link>
```

Listagem 13.4 Importação de documentos NCL e uso de elementos do documento importado.

A Figura 13.3 apresenta a visão estrutural parcial do exemplo ilustrando a importação dos documentos tal como definida pela Listagem 13.4.

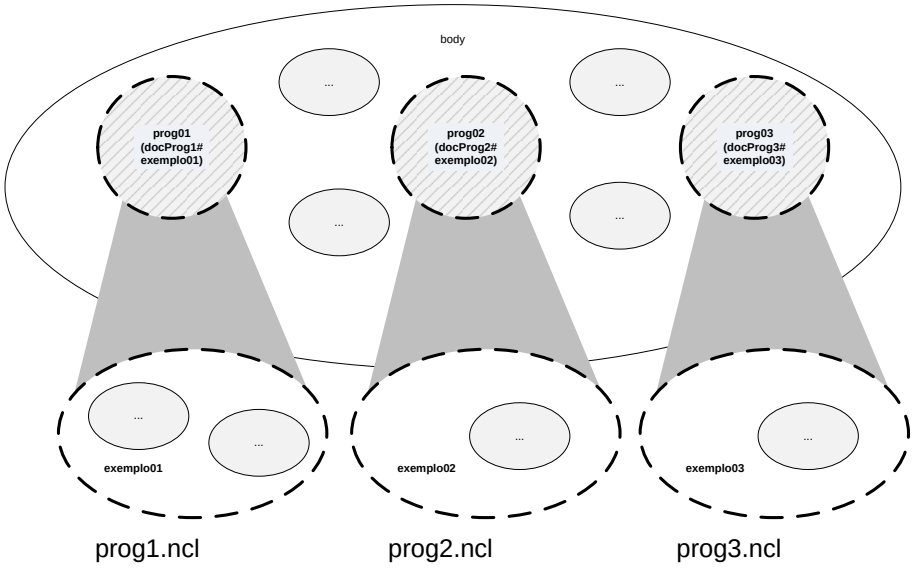


Figura 13.3 Visão estrutural parcial ilustrando a importação de documentos NCL.

13.4.2 Reúso de um Documento NCL como um Objeto de Mídia

Como mencionado no Capítulo 8, também é possível reusar um documento NCL inteiro como um objeto de mídia do tipo "application/x-ncl-ncl". A vantagem de se reusar um documento dessa maneira é a possibilidade de apresentá-lo em uma região específica, em um dispositivo específico, como será discutido nos Capítulos 14 e 15. A Listagem 13.5 ilustra o reúso de um documento NCL como um objeto de mídia a ser apresentado na região associada ao descritor "descPDA", ou seja, como se essa região fosse toda a tela do dispositivo disponível para apresentar o documento importado.

```
<media id="mirrorProg1" type="application/x-ncl-ncl"
            refer="docProg1#exemplo01"
            descriptor="descPDA" />
```

Listagem 13.5 Reúso de documento como um objeto de mídia.

13.5 Importação da Base de um Documento NCL

Além de objetos de mídia, contextos e documentos NCL inteiros, a NCL oferece um mecanismo de reúso de uma base definida em um arquivo externo,

como visto nos capítulos que descrevem cada base. Para isso, utilizamos o elemento <importBase> como um elemento aninhado ao elemento correspondente à base a ser importada. Para importar uma base de regiões, por exemplo, devemos definir um <importBase> dentro do elemento <regionBase>, conforme ilustrado pela Listagem 13.6.

```
<regionBase>
    <importBase alias="regioesDocumentario"
                documentURI="baseRegioesDocumentario.ncl" />
</regionBase>
```

Listagem 13.6 Importação de uma base de regiões.

O elemento <importBase> possui os seguintes atributos:

- *alias*: "apelido" do arquivo importado, ou seja, o nome que será utilizado como prefixo para se referir aos elementos importados, no formato "apelido#id_do_elemento_importado". No caso do exemplo, a base de conectores foi importada com o *alias* "*meusConectores*". Para se referir ao conector "*onEndStop*" dentro do arquivo importado, devemos utilizar a string "*meusConectores#onEndStop"*;
- *documentURI*: a localização e o nome do arquivo que contém a base a ser importada. Nesse exemplo, como o arquivo de conectores está no mesmo diretório que o arquivo da aplicação, bastou definir apenas o nome do arquivo;
- *region*: no caso de o arquivo importado conter uma base de regiões, define qual região da aplicação conterá as regiões importadas.
- *baseId:* no caso de o arquivo importado conter mais de uma base de regiões, define qual base se quer importar.

Vale observar que, quando uma base de descritores é importada, as regiões são importadas automaticamente. Em outras palavras, quando há um <importBase> na seção <descriptorBase>, não é necessário fazer o <importBase> correspondente às regiões na seção <regionBase>, que pode ficar vazia.

O uso mais comum de importação de bases de um documento é a importação de uma base de conectores definida em um documento NCL dedicado a esse fim. A Listagem 13.7 apresenta novamente o exemplo descrito no Capítulo 10, que ilustra a importação de uma base de conectores e o uso de um dos conectores importados. Esse exemplo assume que, no

arquivo “conectores.ncl”, existe um <causalConnector> com *id* “*onKeySelectionStartStop*”.

```
<connectorBase>
        <importBase alias="meusConectores"
documentURI="conectores.ncl" />
</connectorBase>
                                                    ... trecho da seção <head>
```

```
                                                    ... trecho da seção <body>
<link xconnector="meusConectores#onKeySelectionStartStop">
        <bind component="menu" interface="pMenuItem1"
role="onSelection"/>
        <bind component="menu" role="stop"/>
        <bind component="prog01" role="start" />
</link>
```

Listagem 13.7 Importação de uma base de conectores.

13.6 Referência Rápida — Importação de Documentos e Bases

A Tabela 13.2 apresenta os elementos e atributos relacionados ao reuso de documentos e bases de documentos NCL via importação. Como usual, os atributos obrigatórios estão sublinhados.

Tabela 13.2 Elementos e Atributos Relacionados ao Reúso de Documentos e Bases de Documentos NCL no Perfil NCL EDTV

Elementos	Atributos	Conteúdo
importedDocumentBase	*id*	(importNCL)+
importNCL	*alias*, *documentURI*	—
importBase	*alias*, *documentURI*, *region*, *baseId*	—

Bibliografia

ABNT, NBR 15606-2, 2011. Associação Brasileira de Normas Técnicas, “Televisão digital terrestre — Codificação de dados e

especificações de transmissão para radiodifusão digital, Parte 2: Ginga-NCL para receptores fixos e móveis — Linguagem de aplicação XML para codificação de aplicações", Sistema Brasileiro de TV Digital Terrestre, *NBR 15606-2*.

ITU-T, H.761, 2011. Nested Context Language (NCL) and Ginga-NCL for IPTV. ITU-T Rec. H.761. Genebra.

Soares, L.F.S., Soares Neto, C.S. "Reúso e importação em Nested Context Language." Monografias em Ciência da Computação do Departamento de Informática, PUC-Rio, N.º 13/09. Rio de Janeiro, março de 2009. ISSN 0103-9741.

PARTE III

Tópicos Avançados

Capítulo 14

Objetos Hipermídia Declarativos em NCL

Como vimos no Capítulo 8, a linguagem NCL aceita não apenas objetos de mídia convencionais (perceptuais), com conteúdo de vídeo, imagem, áudio e texto, mas também objetos de mídia cujo conteúdo é composto por código declarativo ou imperativo na definição de seus elementos <media>.

Objetos de mídia com código imperativo ou funcional são tratados no Capítulo 17. Neste capítulo, discutiremos como objetos com conteúdo hipermídia especificado por uma linguagem declarativa podem ser definidos, como eles podem se relacionar com outros objetos em um documento NCL e como exibidores (*players*) para esses objetos se comportam.[1]

Em particular, objetos de mídia com código NCL serão abordados. Em outras palavras, uma aplicação NCL pode conter objetos de mídia representando outras aplicações NCL, recursivamente. Aliado ao suporte a múltiplos dispositivos de exibição, discutidos no Capítulo 15, esse é um conceito ímpar de NCL.

[1] Este capítulo se baseia em Soares (2009). O uso do material foi gentilmente cedido pelo Departamento de Informática da PUC-Rio.

14.1 Integrando Objetos Hipermídia Declarativos à NCL

Um objeto de mídia com conteúdo hipermídia especificado por um código declarativo, também denominado objeto hipermídia declarativo, é definido em NCL pelo elemento <media> com o atributo *type* recebendo o valor "application/x-???", onde ??? depende da linguagem declarativa usada. Por exemplo, "application/x-ncl-NCL" é usado no middleware Ginga para objetos com código NCL embutidos no documento NCL pai. A única exceção atual a essa regra é o objeto declarativo do tipo "text/html", que contém código HTML/XHTML.

Em um objeto hipermídia declarativo, o atributo *src* deve referenciar a localização do código declarativo que compõe o conteúdo do objeto. A Listagem 14.1 ilustra um exemplo de especificação de objeto hipermídia declarativo com código NCL. Note que o fato de usarmos a extensão ".ncl" nos desobriga da definição do atributo *type*, como usual.

```
<media id="objetoNCLembutido" src="exe.ncl" descriptor="nclDesc"/>
```

Listagem 14.1 Objeto de mídia com código NCL.

Igual para qualquer outro objeto de mídia, o elemento <media> representando um objeto hipermídia com código declarativo pode definir âncoras de conteúdo (através do elemento <area>) e propriedades (através do elemento <property>). Também igual para qualquer outro objeto de mídia, o atributo *descriptor*, opcional, deve referir-se a um elemento <descriptor> que é responsável pela iniciação de propriedades necessárias à apresentação do objeto, como, por exemplo, sua posição na tela em que será exibido e em que dispositivo de exibição.

Cabe ao exibidor do objeto hipermídia declarativo a responsabilidade de interpretar a semântica associada a suas âncoras de conteúdo, propriedades e descritor associado.

O descritor pode definir, além do exibidor que deverá ser utilizado, uma série de propriedades que serve para iniciar esse exibidor. Por exemplo, no caso de um objeto de mídia de tipo "application/x-ncl-NCL", seu exibidor, um formatador NCL, é capaz de obedecer à semântica NCL usual dessas propriedades e iniciar valores do objeto *settings*, contido no objeto hipermídia declarativo, com os valores passados pelo descritor. Por exemplo, a região

especificada pelo descritor é utilizada para iniciar a variável system.screenSize do novo documento NCL sendo iniciado.[2]

14.2 Interfaces de Objetos de Hipermídia Declarativos

Objetos hipermídia declarativos podem definir âncoras de conteúdo (através do elemento <area>) e propriedades (através do elemento <property>), como é usual para todo objeto de mídia de uma aplicação NCL.

14.2.1 Âncoras de Conteúdo

Um objeto hipermídia com código declarativo é visto pelo formatador NCL como composto de uma série de cadeias temporais (o Apêndice F apresenta uma discussão detalhada do conceito). Uma cadeia temporal corresponde a uma sequência de eventos (ocorrências no tempo), iniciada pelo evento que corresponde ao início da apresentação do objeto hipermídia declarativo. Como existem eventos imprevisíveis, isto é, eventos cuja ocorrência no tempo só pode ser determinada durante a apresentação do objeto de mídia declarativo (como, por exemplo, as interações do telespectador), a cadeia temporal por inteiro só pode ser determinada quando o último evento imprevisível ocorrer. Assim, pela nossa definição de conteúdo de um objeto hipermídia declarativo, pode não ser possível determiná-lo *a priori* em todos os casos.

Como exemplo, vamos retomar a nossa velha conhecida animação de *O Primeiro João*, do Capítulo 3, agora em nova versão. Durante o vídeo da animação, uma propaganda de chuteira, representada por um objeto hipermídia declarativo, será apresentada, como no Capítulo 3, mas agora em um dispositivo secundário (o leitor não deve se preocupar neste capítulo com detalhes da exibição em múltiplos dispositivos; eles serão aprofundados no Capítulo 15). No período da possível exibição da propaganda, um ícone é exibido no dispositivo primário, a tela de uma TV (canto superior direito da Figura 14.1). Ao mesmo tempo, um ícone da chuteira é exibido no dispositivo secundário e, se selecionado, dispara a exibição de um vídeo, propaganda da chuteira, e de um formulário HTML, para compra; tudo sobre uma imagem de fundo no dispositivo secundário, como ilustra a Figura 14.1. Ao término

[2] Outras variáveis system.screenSize(i) podem ser iniciadas por parâmetros do descritor passados em seus elementos <descriptorParam>. Como todo objeto de mídia, todas essas propriedades podem também ser definidas por elementos <property> do objeto hipermídia declarativo.

do vídeo de propaganda, toda a exibição no dispositivo secundário é finalizada.

Figura 14.1 Exibição do objeto hipermídia declarativo NCL em dispositivo secundário.

A Figura 14.2 apresenta a visão estrutural dessa nova versão da aplicação NCL.

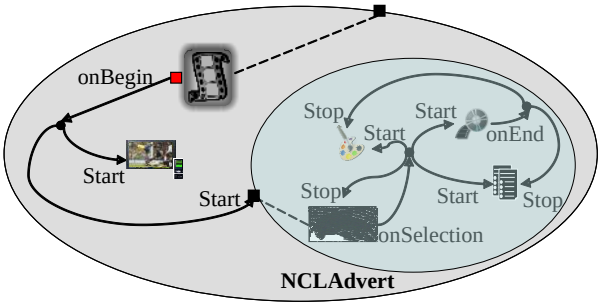


Figura 14.2 Visão estrutural de *O Primeiro João* com objeto hipermídia declarativo NCL.

O objeto hipermídia declarativo NCLAdvert da Figura 14.2 define a cadeia temporal, única, da Figura 14.3, que começa com o início da apresentação (evento de apresentação) do ícone da chuteira. Após seu início, o ícone da chuteira pode ser selecionado (evento imprevisível de seleção) em qualquer instante de tempo "X", fazendo com que a cadeia continue conforme a coluna da direita da Figura 14.3 ou, então, caso não haja seleção, a cadeia termina, com o fim da apresentação do ícone da chuteira, que tem um tempo máximo de exibição de seis segundos. Devemos salientar que, embora nesse exemplo só tenhamos uma cadeia temporal, existem casos em que o conteúdo do objeto hipermídia declarativo define mais de uma cadeia. Isso acontecerá,

por exemplo, se o objeto hipermídia declarativo NCL tiver mais de uma porta de entrada.

Início, Apresentação, ícone da chuteira	0s
Fim, Apresentação, ícone da chuteira	6s

Início, Seleção, ícone da chuteira	(0+X) s
Fim, Apresentação, ícone da chuteira	(0+X) s
Início, Apresentação, imagem de fundo	(0+X) s
Início, Apresentação, vídeo de propaganda	(0+X) s
Início, Apresentação, formulário HTML	(0+X) s
Fim, Apresentação, vídeo de propaganda	(X+13) s
Fim, Apresentação, formulário HTML	(X+13) s

Figura 14.3 Cadeia temporal de NCLAdvert.

Embora o conteúdo de um objeto hipermídia declarativo nem sempre possa ser definido *a priori*, âncoras de conteúdo podem ser definidas, como sempre, por meio de elementos <area>.

Âncoras de conteúdo podem definir trechos nas cadeias temporais, por meio do atributo *clip* do elemento <area>. O valor do atributo *clip* é dado por "(chainId, beginOffset, endOffset)".

O parâmetro *chainId* identifica uma das cadeias temporais definidas pelo objeto hipermídia com código declarativo. Os parâmetros *beginOffset* e *endOffset* determinam o tempo de início e de fim da âncora de conteúdo, respectivamente, relativo ao início da cadeia. Na grande maioria das vezes, um objeto hipermídia com código declarativo define apenas uma única cadeia; nesse caso, o parâmetro *chainId* pode ser omitido. Da mesma forma, os parâmetros *beginOffset* e *endOffset* podem ser omitidos, quando então assumem seus valores *default*: "0s" e o valor correspondente ao tempo final da cadeia, respectivamente.

No caso específico de um objeto hipermídia declarativo com código NCL, uma cadeia é identificada pela *porta* de entrada do corpo da aplicação NCL, identificada pelo atributo *src* do objeto hipermídia declarativo: elemento <port>, filho do elemento <body> do documento NCL que define o conteúdo do objeto hipermídia declarativo. Na Listagem 14.2, vemos uma âncora que inicia quando a cadeia "entry" (identificador de uma porta do elemento <body> de "exemplo.ncl") atingir cinco segundos e termina junto com o fim da cadeia.

```
<media id="objetoNCLembutido" src="exemplo.ncl"
       descriptor="nclDesc">
     <area id="ancora1" clip="(entry,5s,)"/>
</media>
```

Listagem 14.2 Âncoras de conteúdo em objetos de mídia declarativos.

Uma âncora de conteúdo de um objeto hipermídia declarativo pode também fazer referência a qualquer âncora de conteúdo definida internamente no próprio código declarativo identificado pelo atributo *scr* do objeto hipermídia declarativo. Nesse último caso, o atributo *label* do elemento <area> deve ser usado. É ele que define a âncora de conteúdo e deve ter um valor tal que o exibidor do objeto hipermídia declarativo seja capaz de identificar uma de suas âncoras de conteúdo definidas internamente. A máquina de estado de eventos da âncora definida é, nesse caso, idêntica à da âncora de conteúdo que ela referencia.

Como exemplo, no caso de um objeto de mídia do tipo "application/x-ncl-NCL", uma de suas âncoras de conteúdo (elemento <area> do objeto hipermídia declarativo) deve obrigatoriamente fazer referência a uma porta, filha direta do elemento <body> do objeto, por meio do atributo *label*, que deve ter como valor o identificador da porta. Essa porta, por sua vez, pode ser mapeada em uma âncora de conteúdo (elemento <area>) de algum objeto interno ao objeto hipermídia declarativo. Note, assim, que um objeto hipermídia declarativo pode externar âncoras de conteúdo definidas internamente para manipulação por elos definidos no documento NCL pai que contém o objeto hipermídia declarativo.

Como usual na NCL, um objeto hipermídia declarativo sempre define uma âncora representando o conteúdo total do nó. Essa âncora, que nós anteriormente conhecemos com o nome "âncora de conteúdo total", é declarada por omissão (*default*). Essa âncora tem, entretanto, um sentido especial em um objeto hipermídia declarativo. Ela representa a apresentação de *todas* as cadeias definidas pelo objeto. Quando o objeto declarativo sofre uma instrução *start* sem especificar nenhuma âncora, a "âncora de conteúdo total" é assumida, como usual, e todas as cadeias têm sua exibição disparada em paralelo.

14.2.2 Propriedades

Um elemento <property>, filho de um elemento <media> com código declarativo, pode ser usado em NCL para parametrizar o comportamento do

exibidor do objeto. Semelhante às âncoras de conteúdo, cabe ao exibidor de mídia a responsabilidade de interpretar a semântica de cada propriedade.

As propriedades são atributos do nó de mídia como um todo. No caso de um objeto hipermídia com conteúdo declarativo, elas podem ser as propriedades usuais de um objeto de mídia, como *left*, *height*, *explicitDur*, *soundLevel* etc., como também fazer referência a qualquer variável (ou propriedade) interna do objeto hipermídia declarativo. Nesse último caso, o atributo *name* do elemento <property>, que define a propriedade, deve ter um valor tal que o exibidor do objeto hipermídia com conteúdo declarativo seja capaz de identificar uma de suas variáveis (ou propriedades) definida internamente.

Como exemplo, no caso de um objeto de mídia do tipo "application/x-ncl-NCL", uma de suas propriedades (elemento <property> do objeto hipermídia com código declarativo) pode fazer referência a uma porta, filha direta do elemento <body> do objeto, por meio de seu atributo *name*, que deve ter como valor o identificador da porta. Essa porta, por sua vez, pode ser mapeada em uma propriedade (elemento <property>) de algum objeto interno ao objeto hipermídia com conteúdo declarativo, incluindo o objeto *settings*. Note, assim, que um objeto hipermídia declarativo pode externar propriedades definidas internamente para manipulação por elos definidos no documento NCL pai que contém o objeto hipermídia declarativo.

Na Listagem 14.3 temos duas propriedades definidas. A primeira se refere ao atributo "bounds" do nó de mídia com conteúdo declarativo, especificando seu posicionamento na tela de exibição. A segunda se refere a uma porta do elemento <body> do objeto de mídia, cujo identificador é dado pelo valor "globalName". Essa porta, por sua vez, é mapeada em uma propriedade definida internamente em um dos elementos filhos do objeto de mídia.

```
<media id="objetoNCLembutido" src="exemplo.ncl"

       descriptor="nclDesc">
     <property name="bounds"/>
     <property name="globalName"/>
     <area id="ancora1" label="(entry, 5s,)"/>
</media>
```

Listagem 14.3 Propriedades em objetos de mídia declarativo.

Cada mudança no valor de uma variável (ou propriedade) interna realizada pela execução do objeto hipermídia com conteúdo declarativo é refletida no valor da propriedade correspondente desse objeto. Essa mudança pode servir de condição para o disparo de elos internos ao documento NCL pai do objeto hipermídia com conteúdo declarativo.

Da mesma forma, cada mudança em uma propriedade de um objeto hipermídia com conteúdo declarativo sofrida por ações externas se reflete no valor da variável (ou propriedade) correspondente, interna ao objeto. Essas mudanças também podem servir para disparo de procedimentos internos aos objetos hipermídia declarativos. Por exemplo, em um objeto de mídia do tipo "application/x-ncl-NCL", essas mudanças podem servir de condições para disparo de elos internos ao objeto de mídia.

14.3 Exemplo O Primeiro João

Nesta seção vamos completar o exemplo iniciado na Seção 14.2.1 sobre a aplicação *O Primeiro João*, ilustrada na visão estrutural da Figura 14.2.

Para começar, vamos codificar o objeto hipermídia declarativo NCL, como ilustrado na Listagem 14.4. A essa altura do livro, a codificação dispensa explicações, mas vamos salientar dois pontos. Primeiro, note que o pano de fundo da aplicação foi especificado para ocupar toda a região que lhe for passada pelo descritor do objeto hipermídia declarativo. Segundo, observe que a aplicação começa pela porta de entrada "interaction" do elemento <body>. Ele representa a cadeia temporal definida pelo objeto.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Exemplo de objeto de mídia do tipo "application/x-ncl-
NCL" -->
<ncl id="NCLadvert"
         xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
 <head>
   <regionBase>
     <region id="backgroundReg" width="100%" height="100%"
                                                  zIndex="1">
       <region id="iconReg" width="100%" height="100%"
                                                  zIndex="2"/>
       <region id="shoesReg" left="5%" top="30%" width="40%"
                                  height="40%" zIndex="2"/>
       <region id="formReg" left="50%" top="5%" width="45%"
                                  height="90%" zIndex="2"/>
     </region>
   </regionBase>
   <descriptorBase>
     <descriptor id="backgroundDesc" region="backgroundReg"/>
     <descriptor id="shoesDesc" region="shoesReg"/>
     <descriptor id="formDesc" region="formReg"
                                        focusIndex="1"/>
     <descriptor id="iconDesc" region="iconReg"
                                        explicitDur="6s"/>
```

```
    </descriptorBase>
    <connectorBase>
      <importBase documentURI="../causalConnBase.ncl"
                                                  alias="conEx"/>
    </connectorBase>
  </head>
  <body>
    <port id="interaction" component="icon"/>
    <media id="icon" src="../media/icon.png"
                                  descriptor="iconDesc"/>
    <media id="background" src="../media/background.png"
                                  descriptor="backgroundDesc"/>
    <media id="shoes" src="../media/shoes.mp4"
                                  descriptor="shoesDesc"/>
    <media id="ptForm" src="../media/ptForm.htm"
                                  descriptor="formDesc"/>
    <link id="lBeginAdvert"
  xconnector="conEx#onKeySelectionStopStart">
      <bind role="onSelection" component="icon"/>
      <bind role="stop" component="icon"/>
      <bind role="start" component="background"/>
      <bind role="start" component="ptForm"/>
      <bind role="start" component="shoes"/>
    </link>
    <link id="lEndShoes" xconnector="conEx#onEndStop">
      <bind role="onEnd" component="shoes"/>
      <bind role="stop" component="background"/>
      <bind role="stop" component="ptForm"/>
    </link>
  </body>
</ncl>
```

Listagem 14.4 Codificação do objeto hipermídia NCLAdvert.

Voltando à visão estrutural da Figura 14.2, podemos agora definir a aplicação NCL pai, como ilustrado pela Listagem 14.5. Vários pontos da especificação merecem ser destacados:

1. Uma âncora “anchor2” é definida para o objeto hipermídia declarativo “NCLAdvert” fazendo referência à cadeia temporal iniciada pela porta “interaction”. Na Listagem 14.4, essa porta do elemento <body> indica que a cadeia deve ser apresentada, iniciando-se pela exibição do ícone da chuteira.

2. Pelo descritor “nclDesc” passado para a exibição do objeto de mídia “NCLAdvert”, a exibição deve ocupar toda a tela (100% nas duas dimensões,

como especificado) do dispositivo secundário. Pela Listagem 14.4, o ícone da chuteira ocupa toda essa região delimitada para o dispositivo secundário.

3. Pela Listagem 14.5, o objeto de mídia "NCLAdvert" define uma aplicação NCL filha da aplicação NCL identificada por "glue". Contudo, a aplicação NCLAdvert, da Listagem 14.4, também embute um outro objeto hipermídia declarativo: o objeto "ptForm", contendo código declarativo HTML. Vemos, assim, um aninhamento de objetos de mídia declarativos que é bastante comum em aplicações para múltiplos dispositivos, como veremos no Capítulo 15. Mais ainda, o objeto declarativo HTML pode embutir outra aplicação NCL, recursivamente. E mais ainda, o objeto declarativo HTML pode embutir código imperativo ECMAScript. Um objeto hipermídia com código declarativo sempre pode embutir outros objetos quaisquer.

4. Devemos observar que o link "l01", ao iniciar o objeto hipermídia declarativo, declarado na Listagem 14.5 com a propriedade "*focusIndex*=1", passa-lhe o controle das teclas de navegação, ao atribuir o valor "1" à propriedade *service.currentKeyMaster* do objeto *settings*.

5. Por ter o foco, o ícone da chuteira é selecionado pela tecla "ENTER" do controle remoto. Na verdade, se a propriedade *service.currentKeyMaster* do objeto *settings* não tivesse sido colocada em 1, não haveria problema, pois o foco recairia no ícone da chuteira de qualquer jeito, pois ele é o único que tem o *focusIndex* entre os objetos sendo exibidos. No entanto, isso exigiria que a tecla "ENTER" fosse pressionada duas vezes: uma para que o objeto "NCLAdvert" ganhasse o controle das teclas de navegação e, mais uma vez, para que a seleção da chuteira fosse efetivada.

6. Quando o ícone da chuteira é selecionado, o vídeo propaganda começa a ser exibido, bem como o formulário HTML que, por ter a propriedade "*focusIndex*=1", ganha o foco para navegação. O acionamento da tecla "ENTER" faz com que esse objeto ganhe o controle das teclas de navegação.

7. Finalmente, devemos notar que o acionamento da tecla "BACK" do controle remoto tira o controle da navegação do documento HTML, passando para o objeto de mídia "NCLAdvert", embora o foco continue sobre o documento HTML. Um novo acionamento da tecla "BACK" retira o controle da navegação do objeto "NCLAdvert", passando para a aplicação NCL pai.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Generated by NCL Eclipse -->
<ncl id="glue"
        xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
 <head>
   <regionBase>
     <region id="videoReg" width="100%" height="100%"
                                                zIndex="1">
```

```
      <region id="imageReg" left="87.5%" top="11.7%"
                  width="8.45%" height="6.7%" zIndex="2"/>
    </region>
  </regionBase>
  <regionBase device="systemScreen(2)">
    <region id="nclReg" width="100%" height="100%"
                                          zIndex="1"/>
  </regionBase>
  <descriptorBase>
    <descriptor id="videoDesc" region="videoReg"/>
    <descriptor id="imageDesc" region="imageReg"
                                      explicitDur="6s"/>
    <descriptor id="nclDesc" region="nclReg"
                                      focusIndex="1"/>
  </descriptorBase>
  <connectorBase>
    <causalConnector id="onBeginStartSet_var">
      <connectorParam name="var"/>
      <simpleCondition role="onBegin"/>
      <compoundAction operator="seq">
        <simpleAction role="start" max="unbounded"
                                      qualifier="par"/>
        <simpleAction role="set" value="$var"/>
      </compoundAction>
    </causalConnector>
    <causalConnector id="onEndStop">
      <simpleCondition role="onEnd"/>
      <simpleAction role="stop" max="unbounded"
                                      qualifier="par"/>
    </causalConnector>
  </connectorBase>
</head>
<body>
    <port id="entry" component="animation"/>
    <media id="animation" type="video/mpeg"
          src="../media/animGar.mp4" descriptor="videoDesc">
      <area id="anchor1" begin="10s" end="30s"/>
    </media>
    <media id="secIcon" type="image/png"
            src="../media/icon.png" descriptor="imageDesc"/>
    <media id="NCLAdvert" type="application/x-ncl-ncl"
            src="../media/advert.ncl" descriptor="nclDesc">
      <area id="anchor2" `clip="(interaction,,)"/>
    </media>
    <media id="globalVar"
```

```
                          type="application/x-ginga-settings">
      <property name="service.currentKeyMaster"/>
    </media>
    <link id="l01" xconnector="onBeginStartSet_var">
      <bind role="onBegin" component="animation"
                                        interface="anchor1"/>
      <bind role="start" component="secIcon"/>
      <bind role="start" component="NCLAdvert"
                                        interface="anchor2"/>
      <bind role="set" component="globalVar"
                  interface="service.currentKeyMaster">
        <bindParam name="var" value="NCLAdvert"/>
      </bind>
    </link>
    <link id="l02" xconnector="onEndStop">
      <bind role="onEnd" component="animation"
                                        interface="anchor1"/>
      <bind role="stop" component="secIcon"/>
      <bind role="stop" component="NCLAdvert"/>
    </link>
  </body>
</ncl>
```

Listagem 14.5 Aplicação *O Primeiro João* com objeto hipermídia declarativo.

Bibliografia

Soares, L.F.S. "Nested Context Language 3.0 Part 11 — Declarative Objects in NCL: Nesting Objects with NCL Code in NCL Documents. Monografias em Ciência da Computação do Departamento de Informática, PUC-Rio, N.º 12/09. Rio de Janeiro, março de 2009. ISSN 0103-9741.

[Soares et al. 2009] Relating Declarative and Imperative Objects through the NCL Glue Language. Soares, L. F. G.; Moreno, M.F; Sant'Anna, F. *Proceedings of the ACM Symposium on Document Engineering*. Munich, Alemanha. Setembro de 2009.

Capítulo 15

Programando para Múltiplos Dispositivos

Um documento NCL pode ser exibido em múltiplos dispositivos. Esses dispositivos podem se registrar em classes de dois tipos: aquelas em que seus dispositivos membros devem ser aptos para executar exibidores de mídia e aquelas, ditas passivas, em que não é exigido de seus dispositivos membros a capacidade de processar as funções de um exibidor de mídia.

Este capítulo apresenta as várias formas de um documento NCL especificar em que dispositivos serão exibidos seus objetos, e o comportamento do formatador NCL e dos vários exibidores de mídia nesses dispositivos, ilustrando com alguns exemplos.[1]

[1] Este capítulo foi baseia-se em Soares [2009]. O uso do material foi gentilmente cedido pelo Departamento de Informática da PUC-Rio.

15.1 Especificando o Dispositivo de Exibição

Vamos começar este capítulo com uma série de definições. Chamamos de "dispositivos de execução" aos dispositivos capazes de processar atividades de exibição. Para nós, entretanto, o mais importante é definir os chamados "dispositivos de exibição", que é o nome dado à junção de um dispositivo de execução com seus dispositivos de saída (somente os de saída). Vamos chamar de "aparelho", ao conjunto dispositivo de execução e seus dispositivos de saída e entrada associados. Vamos também definir um "domínio" como um conjunto de aparelhos. Finalmente, vamos chamar de "dispositivo-base" o dispositivo de execução que contém o formatador NCL raiz de todo o documento a ser apresentado em um domínio, e de "aparelho-base" o aparelho que contém esse dispositivo.

Como vimos no Capítulo 6, cada elemento <regionBase> de um documento NCL pode estar associado a uma classe de dispositivos onde ocorrerá a apresentação. Para identificar a associação, o elemento <regionBase> define o atributo *device*, que pode ter os valores "systemScreen(i)" ou "systemAudio(i)", sendo *i* um número inteiro maior ou igual a zero. Quando o atributo não for especificado, nenhuma classe é associada, e a apresentação deve ocorrer no próprio dispositivo de exibição que está executando a aplicação.

Dispositivos de execução podem se cadastrar em classes ("systemScreen(i)" ou "systemAudio(i)") de um domínio. O número de classes de um domínio pode ser obtido da variável de ambiente "system.classNumber" do nó Settings. Um dispositivo pode pertencer a mais de uma classe.

Relembrando ainda o Capítulo 6, uma classe (i) escolhida define as variáveis globais de ambiente: system.screenSize(i), system.screenGraphicSize(i) e system.audioType(i), especificadas no nó *Settings*.

15.1.1 Classes de Dispositivos

Existem dois tipos de classe de dispositivos de exibição: "passiva" e "ativa".

Um dispositivo de exibição que se cadastra em uma classe do tipo "ativa" deve ser capaz de executar as funções de exibidores de objetos de mídia, exigidos por essa classe, conforme é discutido no Apêndice H.

De um dispositivo de exibição que se cadastra em uma classe do tipo "passiva" não é exigida a capacidade de executar as funções de exibidores de

mídia. Ele deve ser capaz apenas de apresentar o mapa de memória de vídeo que lhe é passado e exibir as amostras de áudio que lhe são passadas. O exibidor de cada objeto de mídia, nesse caso, será executado por um dispositivo de execução (chamado dispositivo pai), que é responsável pela criação do mapa e do conjunto de amostras de áudio que embute a apresentação dos objetos.

O mapa criado (ou a sequência de amostras de áudio criada) pelo dispositivo pai, para os dispositivos em classe passiva que comanda, pode também ser exibido no próprio dispositivo pai. Para tanto, uma região no dispositivo pai deve ser referenciada pelo elemento <regionBase>, através de seu atributo *region*, que define a classe passiva. Devemos ressaltar que o atributo *region* de um elemento <regionBase> só terá algum sentido quando esse elemento estiver associado a uma classe do tipo "passiva".

Uma classe ativa deve especificar quais exibidores devem estar disponíveis em todos os seus dispositivos cadastrados.

Algumas simplificações foram definidas para esse procedimento. Entre elas é assumido que, quando existir apenas um dispositivo de exibição, ele não precisa se cadastrar. Na verdade, o dispositivo-base tem uma classe (ativa) só para ele, em que nenhum outro dispositivo pode se cadastrar, e essa classe não precisa ser declarada. As classes "systemScreen (1)" ou "systemAudio(1)" são reservadas como classes passivas e as classes "systemScreen (2)" ou "systemAudio(2)" como classes ativas, onde todos os seus dispositivos de exibição cadastrados são capazes de executar todos os exibidores de objetos mídia especificados pelo sistema que adotou a NCL como linguagem, incluindo os exibidores de objetos imperativos, funcionais e hipermídia declarativos.

Qualquer que seja o tipo de classe, passiva ou ativa, seus dispositivos de exibição só podem exibir objetos de mídia vindos de um mesmo dispositivo, chamado "dispositivo pai". Uma classe não pode ter mais de um dispositivo pai em um dado momento. Mais ainda, o "dispositivo-base" não pode receber objetos (diretamente ou embutidos em mapas de memória/amostras de áudio), para exibir, de outro dispositivo exibidor do domínio. Em outras palavras, não há possibilidade de um dispositivo ser ascendente ou descendente de si mesmo, formando ciclos.

15.1.2 Comportamento dos Dispositivos de Entrada

No início de uma apresentação, todos os dispositivos de entrada associados ao domínio dos dispositivos do documento NCL que especifica a aplicação são controlados pelo dispositivo-base.

Quando um elemento <media> em exibição por um dispositivo registrado em uma classe ativa receber o foco e for selecionado, o exibidor de seu conteúdo ganha o controle de todos os dispositivos de entrada do mesmo aparelho do dispositivo de exibição e dos dispositivos de entrada de todos os dispositivos que estão em classes que serão suas descendentes. O exibidor pode, então, seguir suas próprias regras para navegação. O controle dos dispositivos de entrada é devolvido ao dispositivo pai quando a tecla de mnemônico "BACK" for pressionada. Nesse caso, o foco vai para o elemento identificado pelo atributo *service.currentFocus* do nó *settings* (elemento <media> do tipo application/x-ncl-settings) em exibição controlada pelo dispositivo pai.

Devemos notar que pode haver mais de um dispositivo com controle de navegação por teclas, mas cada um com dispositivos de entrada diferentes dos outros.

De posse de todas as informações desta Seção 15.1, podemos agora analisar as várias alternativas para exibição em múltiplos dispositivos.

15.2 Comportamento de Dispositivos na Classe Passiva

Como comentamos anteriormente, no caso de uma classe passiva, todo o processamento de um exibidor de mídia estará a cargo de um dispositivo de exibição capaz de executar exibidores de objetos de mídia, que chamamos de dispositivo pai.

Nesse caso, o dispositivo pai se comunica com os dispositivos de exibição da classe (dispositivos filhos) passando ou o mapa de memória do plano de exibição (*frame buffer*), no caso de dispositivos visuais, ou amostras de áudio, no caso de dispositivos com saída sonora. Um dispositivo de exibição visual deve apenas ser capaz de varrer a matriz e apresentar os pixels correspondentes em sua tela. Um dispositivo de exibição sonora deve apenas ser capaz de varrer e apresentar as amostras de áudio na caixa de som.

Ainda que mais de um dispositivo de exibição seja associado a uma mesma classe, quando um elemento <media> for exibido nessa classe, apenas uma única instância de exibição deve ser criada (no dispositivo pai) e compartilhada por todos os dispositivos. No jargão do modelo NCM, um objeto de representação único é criado e exibido em todos os dispositivos filhos.

É importante salientarmos que não existe zIndex para os mapas de memória/amostras de áudio nos dispositivos filhos. Todos são exibidos com

zIndex=0. Se vários mapas/amostras são recebidos, apenas o último recebido será exibido.

Se um elemento <media>, em exibição na classe passiva, receber o foco e for selecionado, o exibidor de seu conteúdo ganhará o controle do foco (ou seja, de todos os dispositivos de entrada controlados pelo dispositivo pai). O exibidor pode então seguir suas próprias regras para navegação. O controle de foco é devolvido ao dispositivo pai quando a tecla "BACK" for pressionada.

A Figura 15.1 retoma o exemplo da Seção 3.7 (Figura 3.9), *O Primeiro João*, com algumas pequenas modificações para que o contexto da propaganda não seja mais exibido no dispositivo-base (tela principal), mas em dispositivos secundários registrados em uma dada classe passiva. Como o ícone da chuteira não mais será exibido na tela principal, vamos introduzir uma nova imagem (um novo ícone), a ser apresentada na tela principal apenas como aviso de que há informações no dispositivo secundário. Uma vez que o vídeo principal (vídeo da animação) não será mais redimensionado, a imagem de fundo foi colocada dentro do contexto da propaganda (para servir de fundo no dispositivo que apresentará a propaganda), e os elos para o redimensionamento do vídeo principal foram retirados; como consequência, uma vez que não mais existirão relacionamentos com o vídeo principal dentro do contexto de propaganda, o reuso do vídeo foi retirado do contexto. A porta para o contexto de propaganda, mapeada para o ícone da chuteira, foi reintroduzida. Finalmente, o final da exibição do vídeo de propaganda terminará com todo o contexto da propaganda, e não mais um tempo fixo para a manipulação do formulário para compra. Recordando o Capítulo 3, agora já com as modificações, nossa aplicação, bem simples, ilustra como é fácil, em NCL, introduzir vários objetos de mídia sincronizados no tempo em múltiplos dispositivos:

1. uma música de fundo (um chorinho) (<media id="choro" .../>), que começa assim que termina a apresentação inicial do vídeo e inicia a animação propriamente dita;
2. um objeto de vídeo (<media id="drible" ...>), que é exibido em paralelo e sincronizado com o famoso drible do vaivém do Mané, retratado na animação;
3. uma imagem (<media id="photo" .../>), uma foto, que é exibida junto com a cena do marcador caído no chão, após mais um drible desconcertante;
4. uma imagem, um ícone, sinalizando a existência de conteúdo em dispositivos secundários, que é exibida quando o marcador do Mané cai no chão de pernas para o alto;

5. uma imagem, um ícone, de uma chuteira (<media id="icon" .../>), que é exibida para seleção por interatividade, quando o marcador do Mané cai no chão de pernas para o alto, nos dispositivos secundários;
6. uma imagem de fundo (<media id="background" .../>), exibida nos dispositivos secundários quando o ícone do item anterior for selecionado através da tecla vermelha do controle remoto;
7. um formulário (<media id="ptForm" ...) exibido nos dispositivos secundários quando for iniciada a apresentação da imagem de fundo;
8. um vídeo de propaganda da chuteira (<media id="shoes" .../>), também exibido nos dispositivos secundários quando é iniciada a apresentação da imagem de fundo e que, ao final de sua apresentação, termina a imagem de fundo e o formulário.

Toda a apresentação do contexto da propaganda (ícone da chuteira, vídeo da propaganda, imagem de fundo e formulário) é definida dentro do contexto (<context id="advert" ...>), como anteriormente. São os objetos desse contexto que queremos exibir em um dispositivo diferente do da exibição dos outros objetos de mídia do exemplo, cuja visão estrutural da aplicação é apresentada na Figura 15.1.

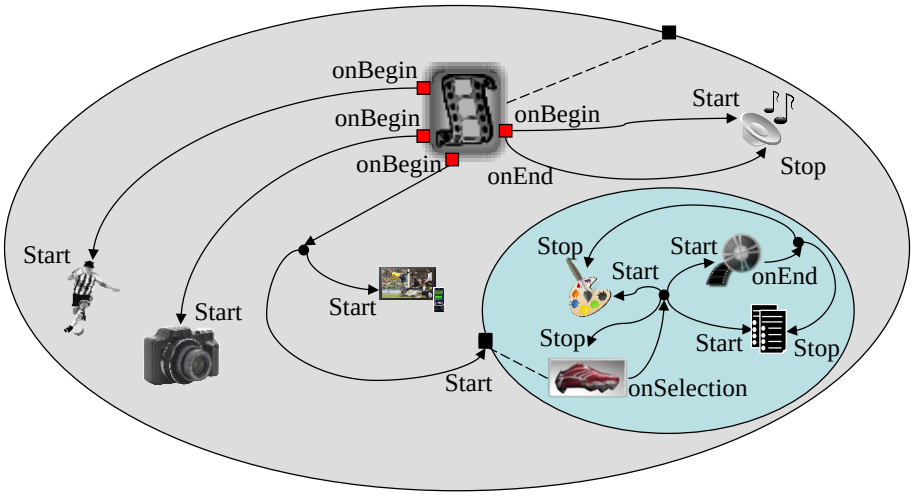


Figura 15.1 Visão estrutural da nova versão do exemplo *O Primeiro João*.

Na Listagem 15.1, o documento NCL do exemplo é apresentado. Note a definição de duas bases de regiões. A primeira define onde serão exibidos todos os objetos, exceto os relacionados à propaganda dentro do contexto da Figura 15.1. Essa base de regiões não especifica o atributo *device*, indicando por omissão (*default*) que os objetos devem ser exibidos no dispositivo-base

(que executa o documento NCL). A segunda, identificando o dispositivo "systemScreen(1)", define onde serão exibidos todos os objetos da propaganda (vídeo da propaganda, imagem de fundo e formulário).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Exemplo de uso de múltiplos dispositivos -->
<ncl id="devices" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
 <head>
   <regionBase>
     <region id="screenReg" width="100%" height="100%" zIndex="1">
       <region id="frameReg" left="5%" top="6.7%" width="18.5%"
                                    height="18.5%" zIndex="2"/>
       <region id="secIconReg" left="87.5%" top="11.7%" width="8.45%"
                                    height="6.7%" zIndex="2"/>
     </region>
   </regionBase>
   <regionBase device="systemScreen(1)">
     <region id="backgroundReg" width="100%" height="100%"
                                                  zIndex="1">
       <region id="iconReg" width="100%" height="100%" zIndex="2"/>
       <region id="shoesReg" left="5%" top="30%" width="40%"
                                          height="40%" zIndex="2"/>
       <region id="formReg" left="50%" top="5%" width="45%"
                                          height="90%" zIndex="2"/>
     </region>
   </regionBase>
   <descriptorBase>
     <descriptor id="backgroundDesc" region="backgroundReg"/>
     <descriptor id="screenDesc" region="screenReg"/>
     <descriptor id="photoDesc" region="frameReg" explicitDur="5s"/>
     <descriptor id="audioDesc"/>
     <descriptor id="dribleDesc" region="frameReg"/>
     <descriptor id="iconDesc" region="iconReg" explicitDur="6s"/>
     <descriptor id="secIconDesc" region="secIconReg"
                                                 explicitDur="6s"/>
     <descriptor id="shoesDesc" region="shoesReg"/>
     <descriptor id="formDesc" region="formReg" focusIndex="1"/>
   </descriptorBase>
   <connectorBase>
     <importBase documentURI="../causalConnBase.ncl" alias="conEx"/>
   </connectorBase>
 </head>
 <body>
   <port id="entry" component="animation"/>
   <media id="animation" src="../media/animGar.mp4"
                                    descriptor="screenDesc">
     <area id="segDrible" begin="12s"/>
     <area id="segPhoto" begin="41s"/>
     <area id="segIcon" begin="45s" end="51s"/>
   </media>
   <media id="choro" src="../media/choro.mp3" descriptor="audioDesc"/>
   <media id="drible" src="../media/drible.mp4"
                                          descriptor="dribleDesc"/>
   <media id="photo" src="../media/photo.png" descriptor="photoDesc"/>
   <media id="secIcon" src="../media/secIcon.png"
```

```
                                        descriptor="secIconDesc"/>
    <context id="advert">
      <port id="pIcon" component="icon"/>
      <media id="icon" src="../media/icon.png" descriptor="iconDesc"/>
      <media id="background" src="../media/background.png"
                                        descriptor="backgroundDesc"/>
      <media id="shoes" src="../media/shoes.mp4"
                                              descriptor="shoesDesc"/>
      <media id="ptForm" src="../media/ptForm.htm"
                                              descriptor="formDesc"/>
      <link id="lBegingAdvert"
                      xconnector="conEx#onKeySelectionStopStart">
        <bind role="onSelection" component="icon">
               <bindParam name="keyCode" value="RED"/>
        </bind>
        <bind role="stop" component="icon"/>
        <bind role="start" component="background"/>
        <bind role="start" component="ptForm"/>
        <bind role="start" component="shoes"/>
      </link>
      <link id="lEndShoes" xconnector="conEx#onEndStop">
        <bind role="onEnd" component="shoes"/>
        <bind role="stop" component="ptForm"/>
        <bind role="stop" component="background"/>
      </link>
    </context>
    <link id="lMusic" xconnector="conEx#onBeginStart_delay">
      <bind role="onBegin" component="animation"/>
      <bind role="start" component="choro">
        <bindParam name="delay" value="5s"/>
      </bind>
    </link>
    <link id="lDrible" xconnector="conEx#onBeginStart">
      <bind role="onBegin" component="animation"
                                              interface="segDrible"/>
      <bind role="start" component="drible"/>
    </link>
    <link id="lPhoto" xconnector="conEx#onBeginStart">
      <bind role="onBegin" component="animation"
                                              interface="segPhoto"/>
      <bind role="start" component="photo"/>
    </link>
    <link id="lEnd" xconnector="conEx#onEndStop">
      <bind role="onEnd" component="animation"/>
      <bind role="stop" component="choro"/>
    </link>
    <link id="lIcon" xconnector="conEx#onBeginStart">
      <bind role="onBegin" component="animation" interface="segIcon"/>
      <bind role="start" component="secIcon"/>
      <bind role="start" component="advert" interface="pIcon"/>
    </link>
  </body>
</ncl>
```

Listagem 15.1 *O Primeiro João* com múltiplos dispositivos de exibição.

Devemos notar que, se no exemplo o ícone da chuteira for selecionado, o vídeo da propaganda e o formulário aparecerão em todos os dispositivos cadastrados como “systemScreen(1)”, como apresenta a Figura 15.2. Caso o formulário seja selecionado em um desses dispositivos, qualquer navegação feita por qualquer dispositivo de entrada controlado pelo dispositivo pai será exibida em todos os dispositivos de saída cadastrados em “systemScreen(1)”. Caso queiramos que a navegação no formulário seja individual em cada dispositivo, a solução da próxima seção deve ser adotada.

Figura 15.2 Múltiplos dispositivos em classe passiva com mapas de memórias idênticos.

A Listagem 15.1 mostra como é fácil especificar a apresentação em múltiplas classes de dispositivos. Apenas um elemento (<regionBase device="systemScreen(1)">) teve de ser adicionado para distinguir a apresentação em múltiplos dispositivos daquela onde todos os objetos seriam apresentados apenas no aparelho de TV.

Em NCL, o mapa de vídeo apresentado nos dispositivos secundários pode também ser apresentado em uma região do dispositivo pai. Para tanto, é suficiente que adicionemos um atributo ao elemento <regionBase>, que define a classe passiva “systemScreen(i)”, referindo-se a uma região filha do elemento <regionBase> associado ao dispositivo pai. A Listagem 15.2 ilustra o procedimento para o exemplo da Listagem 15.1.

```
<head>
    ...
  <regionBase>
    <region id="screenReg" width="100%" height="100%" zIndex="1">
      <region id="memoryMap" left="87.5%" top="87.5%" width="10%"
                                                  height="10%" zIndex="2"/>
     ...
    </region>
  </regionBase>
  <regionBase device="systemScreen(1)" region="memoryMap">
    ...
  </regionBase>
    ...
</head>
```

Listagem 15.2 Mapa de memória de vídeo apresentada no dispositivo pai.

15.3 Comportamento de Dispositivos na Classe Ativa

No caso de uma classe ativa, todo o processamento para exibição de um objeto de mídia a ela direcionado será delegado aos dispositivos registrados na classe, pelo que chamamos anteriormente de dispositivo pai.

Quando mais de um dispositivo de exibição for associado a uma mesma classe ativa, e um elemento <media> for exibido nessa classe, uma instância de exibição deve ser criada pelo exibidor de cada dispositivo. No jargão do modelo NCM, um objeto de representação é criado em cada dispositivo e controlado por ele. Na definição de um elo, elementos <bind> que se referem a esse elemento <media> devem ter seu atributo de cardinalidade *max* com o valor "unbounded".

Toda ação de apresentação ou atribuição (*stop*, *abort*, *pause* e *resume*) realizada sobre o elemento <media> cujas várias instâncias de exibição foram criadas deve refletir em todas as instâncias. As ações devem ser aplicadas em qualquer ordem. Qualquer condição derivada desse elemento <media> é considerada satisfeita somente quando a condição for satisfeita em todas as instâncias (simultaneamente ou não), se o atributo *qualifier* do elemento <bind> for igual a "and", ou se a condição for satisfeita em pelo menos uma instância, se o atributo *qualifier* do elemento <bind> for igual a "or".

Enquanto nenhum dos elementos <media> em exibição nessa classe for selecionado (após receber o foco), toda a interação do usuário nos diversos dispositivos de entrada é passada ao dispositivo pai.

Se um elemento <media> em exibição nessa classe receber o foco e for selecionado, o exibidor de seu conteúdo ganha o controle do foco (ou seja, de todos os dispositivos de entrada do mesmo aparelho do dispositivo de exibição e de todos os aparelhos cujo dispositivo de exibição estiver em classes que serão suas descendentes, como vimos na Seção 15.1.2) para navegação por teclas (Key Navigation). O exibidor pode então seguir suas próprias regras para navegação. O controle de foco é devolvido ao dispositivo pai quando a tecla de mnemônico "BACK" for pressionada.

A Figura 15.3 retoma o exemplo da seção anterior. Nela o contexto de propaganda foi substituído por um objeto de mídia NCL, elemento <media type="application/x-ncl-NCL" .../>, por nós discutido no Capítulo 14. De resto, tudo é muito parecido com o exemplo anterior, como mostra a visão estrutural na figura.

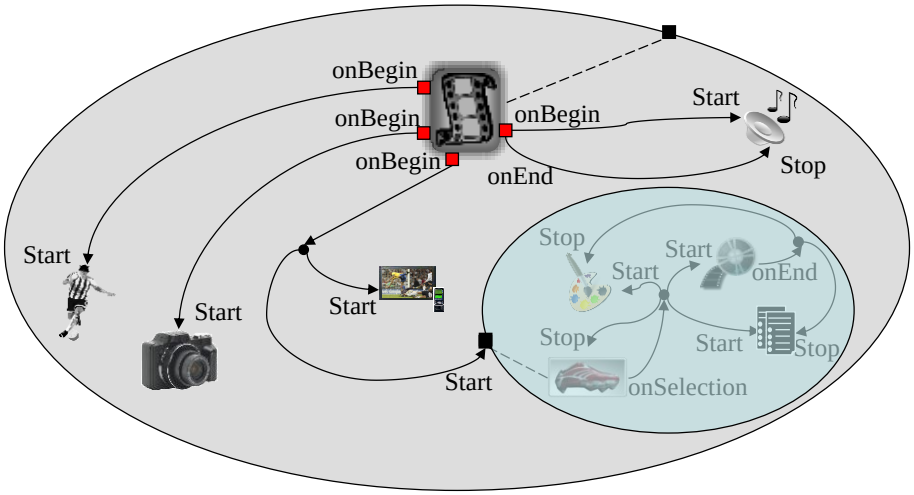


Figura 15.3 Visão estrutural com objeto de mídia do tipo "application/x-ncl-NCL".

Na Figura 15.3, toda a parte interna do objeto de mídia NCLAdvert (elemento <media type="application/x-ncl-NCL") é definida em um novo nó (documento) NCL, apresentado na Listagem 15.3.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Exemplo de objeto de mídia do tipo "application/x-ncl-NCL" -->
<ncl id="advert" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
  <head>
    <regionBase>
  <region id="backgroundReg" width="100%" height="100%" zIndex="1">
    <region id="iconReg" width="100%" height="100%" zIndex="2"/>
    <region id="shoesReg" left="5%" top="30%" width="40%"
                                              height="40%" zIndex="2"/>
    <region id="formReg" left="50%" top="5%" width="45%"
                                              height="90%" zIndex="2"/>
  </region>
    </regionBase>
    <descriptorBase>
      <descriptor id="backgroundDesc" region="backgroundReg"/>
      <descriptor id="shoesDesc" region="shoesReg"/>
      <descriptor id="formDesc" region="formReg" focusIndex="1"/>
      <descriptor id="iconDesc" region="iconReg" explicitDur="6s"/>
    </descriptorBase>
    <connectorBase>
      <importBase documentURI="../causalConnBase.ncl"
                                                    alias="conEx"/>
    </connectorBase>
  </head>
  <body>
    <port id="pIcon" component="icon"/>
    <media id="icon" src="../media/icon.png" descriptor="iconDesc"/>
    <media id="background" src="../media/background.png"
                                     descriptor="backgroundDesc"/>
    <media id="shoes" src="../media/shoes.mp4" descriptor="shoesDesc"/>
    <media id="ptForm" src="../media/ptForm.htm"
                                              descriptor="formDesc"/>
    <link id="lBegingAdvert"
                        xconnector="conEx#onKeySelectionStopStart">
      <bind role="onSelection" component="icon">
        <bindParam name="keyCode" value="RED"/>
      </bind>
      <bind role="stop" component="icon"/>
      <bind role="start" component="background"/>
      <bind role="start" component="ptForm"/>
      <bind role="start" component="shoes"/>
    </link>
    <link id="lEndShoes" xconnector="conEx#onEndStop">
      <bind role="onEnd" component="shoes"/>
      <bind role="stop" component="ptForm"/>
      <bind role="stop" component="background"/>
    </link>
  </body>
</ncl>
```

Listagem 15.3 Documento NCL da propaganda da chuteira.

Queremos que, na nova versão do documento da aplicação *O Primeiro João* com o novo objeto de mídia NCLAdvert, uma vez que o ícone de existência de informação em dispositivos secundários seja exibido, o novo objeto de mídia seja apresentado em outra classe de dispositivos de exibição, e também queremos que, ao iniciar a apresentação de tal objeto, ele ganhe o controle dos dispositivos de entrada do aparelho correspondente ao dispositivo de exibição. Como vimos anteriormente, uma vez que são criadas instâncias para cada dispositivo de exibição da classe, cada dispositivo de exibição terá uma navegação independente.

Para passar o controle dos dispositivos de entrada para o objeto de mídia NCLAdvert, a variável global "service.currentKeyMaster" do elemento <media type="application/x-ncl-settings" ...> será usada.

Primeiramente, tal variável precisa ser definida em um elemento <property>, para permitir seu uso no documento, como mostra a Listagem 15.4.

```
<media id="globalVar" type="application/x-ncl-settings">
   <property name="service.currentKeyMaster"/>
 </media>
```

Listagem 15.4 Externalização da propriedade "service.currentKeyMaster".

Ao ser iniciada a apresentação do objeto de mídia NCLAdvert, o controle do foco deve ser passado a ele pelo mesmo elo usado para sua iniciação, como mostra a Listagem 15.5. Uma vez que a apresentação da propaganda é iniciada, como a propriedade "service.currentKeyMaster" recebe o identificador do objeto NCLAdvert, o objeto recebe o controle da navegação por teclas (Key Navigation) do aparelho de exibição correspondente.

```
<link id="lBeginAdvert" xconnector="conEx#onBeginSet_varStart">
   <bind role="onBegin" component="animation" interface="segIcon"/>
   <bind role="start" component="secIcon"/>
   <bind role="start" component="NCLAdvert"/>
   <bind role="set" component="globalVar"
                            interface="service.currentKeyMaster">
       <bindParam name="var" value="NCLAdvert"/>
   </bind>
</link>
```

Listagem 15.5 Iniciação e passagem de controle para o objeto de mídia NCLAdvert.

O documento completo dessa nova versão da aplicação *O Primeiro João* é apresentado na Listagem 15.6.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Exemplo de uso de múltiplos dispositivos com navegação
independente -->
<ncl id="indDevices" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
  <head>
    <regionBase>
      <region id="screenReg" width="100%" height="100%" zIndex="1">
        <region id="frameReg" left="5%" top="6.7%"
                        width="18.5%" height="18.5%" zIndex="2"/>
        <region id="secIconReg" left="87.5%" top="11.7%"
                        width="8.45%" height="6.7%" zIndex="2"/>
      </region>
    </regionBase>

    <regionBase device="systemScreen(2)">
      <region id="NCLAdvertReg" width="100%" height="100%" zIndex="1"/>
    </regionBase>

    <descriptorBase>
      <descriptor id="screenDesc" region="screenReg"/>
      <descriptor id="photoDesc" region="frameReg" explicitDur="5s"/>
      <descriptor id="audioDesc"/>
      <descriptor id="dribleDesc" region="frameReg"/>
      <descriptor id="secIconDesc" region="secIconReg"
                                                  explicitDur="6s"/>
      <descriptor id="NCLAdvertDesc" region="NCLAdvertReg"/>
    </descriptorBase>
    <connectorBase>
      <importBase documentURI="../causalConnBase.ncl" alias="conEx"/>
    </connectorBase>
  </head>
  <body>
    <port id="entry" component="animation"/>
    <media id="animation" src="../media/animGar.mp4"
                                            descriptor="screenDesc">
      <area id="segDrible" begin="12s"/>
      <area id="segPhoto" begin="41s"/>
      <area id="segIcon" begin="45s" end="51s"/>
    </media>
    <media id="globalVar" type="application/x-ncl-settings">
      <property name="service.currentKeyMaster"/>
    </media>
    <media id="choro" src="../media/choro.mp3" descriptor="audioDesc"/>
    <media id="drible" src="../media/drible.mp4"
                                            descriptor="dribleDesc"/>
    <media id="photo" src="../media/photo.png" descriptor="photoDesc"/>
```

```
    <media id="secIcon" src="../media/icon.png"
                                    descriptor="secIconDesc"/>
    <media id="NCLAdvert" src="advert.ncl" descriptor="NCLAdvertDesc"/>
    <link id="lMusic" xconnector="conEx#onBeginStart_delay">
      <bind role="onBegin" component="animation"/>
      <bind role="start" component="choro">
        <bindParam name="delay" value="5s"/>
      </bind>
    </link>
    <link id="lDrible" xconnector="conEx#onBeginStart">
      <bind role="onBegin" component="animation"
                                          interface="segDrible"/>
      <bind role="start" component="drible"/>
    </link>
    <link id="lPhoto" xconnector="conEx#onBeginStart">
      <bind role="onBegin" component="animation" interface="segPhoto"/>
      <bind role="start" component="photo"/>
    </link>
    <link id="lEnd" xconnector="conEx#onEndStop">
      <bind role="onEnd" component="animation"/>
      <bind role="stop" component="choro"/>
    </link>
    <link id="lBeginAdvert" xconnector="conEx#onBeginSet_varStart">
      <bind role="onBegin" component="animation" interface="segIcon"/>
      <bind role="start" component="secIcon"/>
      <bind role="start" component="NCLAdvert"/>
      <bind role="set" component="globalVar"
                            interface="service.currentKeyMaster">
        <bindParam name="var" value="NCLAdvert"/>
      </bind>
    </link>
  </body>
</ncl>
```

Listagem 15.6 *O Primeiro João* com múltiplos dispositivos de exibição independentes.

A Figura 15.4 ilustra o mesmo momento definido na Figura 15.2, só que para múltiplos dispositivos em classe ativa. Note que, por ser classe ativa, a exibição pode ser diferente em cada dispositivo da classe (focalize a atenção no formulário, que na Figura 15.4 mostra a compra já efetuada).

Figura 15.4 Múltiplos dispositivos em classe ativa com exibições diferentes.

15.4 Comportamento de Dispositivos Cadastrados nas Classes Ativa e Passiva

Esta seção apenas ressalta que um mesmo dispositivo de exibição pode estar cadastrado simultaneamente em classes passivas e ativas, desde que todas sejam controladas por um mesmo dispositivo pai em um dado instante, como usual. Nesse caso, ele herda o comportamento dos dois tipos de classe.

Recordando:

- um dispositivo de exibição só pode exibir objetos recebidos (diretamente ou embutidos em mapas de memória/amostras de áudio) pelas classes em que se cadastrou, ou exibir objetos confinados (contidos) nos objetos recebidos (por exemplo, um objeto de mídia dentro de um elemento <media> recebido com conteúdo possuindo código declarativo);
- o dispositivo-base não pode receber objetos (diretamente ou embutidos em mapas de memória/amostras de áudio) para exibir de outro dispositivo exibidor do domínio;
- quando um objeto for recebido para exibição em uma classe ativa, uma instância independente será criada para cada dispositivo de exibição da classe e o comportamento deve ser igual ao descrito para aquela classe;
- quando objetos forem passados pela classe passiva através de mapa de memória/amostras de áudio, será criada apenas uma instância para cada objeto, no aparelho do dispositivo exibidor que enviou o objeto.

No entanto, devemos ressaltar:

- Não existe zIndex para os mapas de memória/amostras de áudio. Todos são exibidos com zIndex=0 nos dispositivos filhos. Se vários mapas de memória/amostras de áudio forem recebidos, apenas o último recebido será exibido. Toda exibição de objetos recebidos em uma classe ativa se superpõe aos mapas de memória/amostras de áudio recebidos.

15.5 Formatador Distribuído

Um formatador NCL pode, por meio da análise do seu grafo temporal (veja Apêndice G), descobrir que parte de uma cadeia temporal será exibida em uma classe ativa de dispositivos de exibição e que os dispositivos dessa classe são capazes de exibir documentos NCL. Nesse caso, em vez de o formatador instanciar a exibição de cada objeto de mídia no dispositivo de destino, ele pode tirar proveito da situação e passar toda a cadeia para o formatador NCL desse dispositivo, que então se encarregará de instanciar cada exibidor de mídia da cadeia.

Para o leitor interessado nessa opção recomendamos uma leitura atenta do Apêndice G.

15.6 Adaptando Múltiplos Dispositivos para um Ambiente com um Único Dispositivo

Trabalhar com classes de dispositivos libera o autor de uma aplicação NCL da preocupação de quantos e quais dispositivos estão registrados nas classes, número que pode variar com o tempo de exibição de uma aplicação. Entretanto, quando o número de dispositivos registrados em uma classe é "zero", nenhum objeto de mídia endereçado a essa classe será apresentado. Nesse caso, seria bem conveniente permitir ao autor especificar uma apresentação alternativa, e isso é possível.

Conforme vimos no Capítulo 7, o posicionamento inicial da apresentação de um objeto de mídia é determinado em NCL pelo elemento <descriptor> referido pelo objeto. Esse elemento associa o objeto de mídia a uma região de apresentação que, por sua vez, está associada a uma classe de dispositivos. Uma variável global `system.devNumber(i)` do nó settings (<media type="application/x-ncl-settings" ...>) mantém o número de dispositivos registrados em uma classe `systemScreen(i)`. Assim, pelo teste do valor dessa variável, um elemento <descriptorSwitch> é capaz de selecionar uma região dessa classe, caso haja algum dispositivo nela registrado ou uma apresentação alternativa.

O exemplo da Listagem 15.7 ilustra a definição de uma regra que é satisfeita quando o número de dispositivos registrados na classe `systemScreen(2)` é zero. Se substituirmos o elemento `<descriptor id="NCLAdvertDesc" region="NCLAdvertReg"/>` do exemplo da Listagem 15.6 pelo elemento `<descriptorSwitch id="NCLAdvertDesc">` da Listagem 15.7 e, se não houver nenhum dispositivo registrado na classe ativa `systemScreen(2)`, a propaganda definida pelo elemento `<media id="NCLAdvert" src="advert.ncl">` será exibida no aparelho de TV da classe-base, como ilustra a Figura 15.5.

```
<head>
   <ruleBase>
     <rule id="single" var="system.devNumber(2)" value="0"
                                                    comparator="eq"/>
   </ruleBase>
   <regionBase>
     <region id="screenReg" width="100%" height="100%" zIndex="1">
        ...
        <region id="NCLAdvertSingleReg" left="5%" top="5%"
                              width="30%" height="30%" zIndex="2"/>
     </region>
   </regionBase>
   <regionBase device="systemScreen(2)">
     <region id="NCLAdvertMultiReg" width="100%" height="100%"
                                                    zIndex="1"/>
   </regionBase>

     <descriptorBase>
        ...
        <descriptorSwitch id="NCLAdvertDesc">
          <bindRule constituent="NCLAdvertSingleDesc" rule="single"/>
          <defaultDescriptor descriptor="NCLAdvertMultiDesc"/>
          <descriptor id="NCLAdvertSingleDesc"
                                      region="NCLAdvertSingleReg"/>
          <descriptor id="NCLAdvertMultiDesc"
                                      region="NCLAdvertMultiReg"/>
        </descriptorSwitch>
     </descriptorBase>
     ...
</head>
```

Listagem 15.7 Apresentação alternativa à exibição em uma classe sem dispositivos registrados.

Figura 15.5 Apresentação em um único dispositivo por ausência de registros em classes.

Bibliografia

ABNT, NBR 15606-2 (2011). Associação Brasileira de Normas Técnicas, "Televisão digital terrestre — Codificação de dados e especificações de transmissão para radiodifusão digital, Parte 2: Ginga-NCL para receptores fixos e móveis — Linguagem de aplicação XML para codificação de aplicações", Sistema Brasileiro de TV Digital Terrestre, *NBR 15606-2*.

Soares, L.F.S. e Rodrigues, R.F. (2006). "Nested Context Model 3.0 Part 8 — NCL (Nested Context Language) Digital TV Profiles." Monografias em Ciência da Computação do Departamento de Informática, PUC-Rio, N.º 35/06. Rio de Janeiro, outubro de 2006. ISSN 0103-9741.

Soares, L.F.S. (2009). "Nested Context Model 3.0 Part 12 — Support to Multiple Exhibition Devices." Monografias em Ciência da Computação do Departamento de Informática, PUC-Rio, N.º 03/09. Rio de Janeiro, janeiro de 2009. ISSN 0103-9741.

Capítulo 16

Comandos de Edição NCL

Comandos de edição NCL podem vir por diferentes modos: pela rede (canal de difusão, canal de interatividade ou outra rede qualquer), através dos objetos imperativos embutidos no próprio documento NCL, ou mesmo através de alguma aplicação externa comandada por um usuário da aplicação.

Através de comandos de edição, documentos NCL podem ser criados e modificados em tempo de exibição.

Este capítulo descreve os comandos de edição NCL, deixando para o Apêndice F a discussão sobre a forma como esses comandos podem ser transportados, tanto através de estruturas de dados recebidas sem solicitação, como através de estruturas de dados recebidas sob demanda.[1]

[1] Este capítulo foi baseado em Soares *et al*. [2006]. O uso do material foi gentilmente cedido pelo Departamento de Informática da PUC-Rio.

16.1 Introdução

O núcleo da máquina de apresentação de documentos NCL é composto pelo formatador NCL e pelo módulo Gerenciador de Base Privada.

O formatador NCL é responsável por receber um documento NCL e controlar sua apresentação, tentando garantir que as relações especificadas entre os objetos de mídia sejam respeitadas. O formatador lida com aplicações NCL que são coletadas dentro de uma estrutura de dados conhecida como base privada. Como exemplo, no Sistema Brasileiro de TV Digital Terrestre, o *middleware* Ginga associa pelo menos uma base privada a um canal de televisão.

Os documentos NCL em uma base privada podem ser iniciados, pausados, retomados, parados e referir uns aos outros.

O Gerenciador de Base Privada é responsável por receber comandos de edição de documentos NCL e pela execução desses comandos, incluindo a edição de documentos NCL ativos (documentos sendo apresentados), ou seja, edições ao vivo.

Comandos de edição podem ser recebidos por diferentes vias. Por exemplo, em um ambiente de TV digital terrestre é usual adotar o protocolo DSM-CC (ver Apêndice B) para o transporte de comandos de edição gerados pelos provedores das emissoras de TV. É também possível receber comandos de edição pelo canal de interatividade ou mesmo diretamente do telespectador, fazendo uso de uma aplicação residente no receptor. As diferentes vias de recepção de comandos de edição são assunto do Apêndice F.

16.2 Comandos de Edição NCL

Os comandos de edição NCL [Soares *et al.*, 2006] são envelopados em uma estrutura chamada *descritor de evento*. Cada descritor de evento (de edição) tem uma estrutura composta basicamente por um *id*, uma referência de tempo e um campo de dados privados. A identificação define univocamente cada evento de edição (e não cada tipo de comando). A referência de tempo indica o exato momento de disparar o evento. Tempo de referência igual a zero informa que o evento de edição deve ser disparado imediatamente após ser recebido (eventos carregando esse tipo de referência de tempo são comumente conhecidos como eventos "*do it now*"). O campo de dados privados oferece suporte para identificação do comando e definição de parâmetros do evento de edição, como apresentado na Tabela 16.1.

Tabela 16.1 Descritor de Evento para Comandos de Edição

Sintaxe	Número de Bits
EventDescriptor ( ) {	
eventid	16
eventNPT	33
privateDataLength	8
commandTag	8
sequenceNumber	7
finalFlag	1
privateDataPayload	8 a 1928
FCS	8
}	

O campo *commandTag* identifica univocamente os tipos de comandos de edição, como especificado na Tabela 16.2. Para permitir o envio de um comando completo com mais de 255 bytes em mais de um descritor de evento, todos os descritores de um mesmo comando devem ser numerados e enviados em sequência (isto é, não podem ser multiplexados com outros comandos de edição com o mesmo *commandTag*), com o *finalFlag* igual a 1, exceto para o último descritor, que deve ter o campo *finalFlag* igual a 0. O campo *privateDataPayload* contém os parâmetros do comando de edição. Finalmente, o campo FCS contém um *checksum* de todo o campo *privateData*, inclusive o *privateDataLength.*

Os comandos de edição NCL são divididos em três grupos: o primeiro para operação da base privada (para abrir, ativar, desativar, fechar e salvar bases privadas); o segundo para manipulação de documentos (para adicionar, remover e salvar um documento em uma base privada aberta e para iniciar, pausar, retomar e parar apresentações de documentos em uma base privada ativa); e o último para manipular entidades NCL de um documento em uma base privada aberta. Para cada entidade NCL, foram definidos os comandos *add* e *remove*. Se uma entidade já existir, o comando *add* tem a semântica de atualização (alteração).

Os comandos *add* têm entidades NCL como seus argumentos (parâmetros de comando especificados em XML). As entidades são definidas utilizando uma notação sintática idêntica àquela usada pelos esquemas NCL [Soares *et al.*, 2006], com exceção do comando *addInterface*: o atributo *begin* ou *first* de um elemento <area> pode receber o valor "now", especificando o NPT atual do objeto determinado pelo argumento *nodeId* (ver Tabela 16.2). A consistência do documento é mantida pelo formatador NCL, quer a entidade especificada já exista ou não, no sentido de que todos os atributos de identidade obrigatórios têm de ser definidos. Há apenas uma exceção a essa regra - o atributo *interface* de um elemento <bind> filho de um

elemento <link> pode ser deixado inconsistente, referindo-se a um elemento <area> a ser preenchido por um comando *addInterface* cujo atributo *begin* possui o valor “now". Nesse caso, o elemento <link> deve obrigatoriamente ser avaliado assim que o comando *addInterface* for recebido.

Se o parâmetro de comando baseado em XML for curto o suficiente, ele pode ser transportado diretamente no *payload* dos descritores de evento. Se não, o *privateDatePayload* transporta um conjunto de pares de referência {uri, id}, com a interpretação dada a seguir.

No caso de arquivos recebidos pelo canal de difusão (documentos NCL ou objetos de documentos NCL enviados sem solicitação), cada par relaciona um caminho de arquivo ou diretório de arquivos e sua respectiva localização no sistema de transporte. Não é necessária a inclusão de um par {uri, id} no comando para cada arquivo enviado por difusão. Mas é necessário que a partir dos pares {uri, id} incluídos no comando todo arquivo recebido possa ter o seu caminho absoluto (uri) no sistema transmissor deduzido.

No caso de arquivos recebidos sob demanda pelo canal de interatividade ou localizados no próprio receptor, nenhum par de referências necessita ser enviado, exceto se o arquivo for o da especificação do documento NCL ou da especificação XML do objeto NCL que deverá ser adicionado, segundo o comando de adição correspondente. Nesse caso, o par {uri, “null”} deve ser enviado especificando o caminho do arquivo a ser buscado.

A Tabela 16.2 mostra as *strings* de comando e, entre parênteses, os parâmetros transportados como conteúdo (*payload*) do descritor de evento de edição (*nclEditingCommand*) [Soares *et al.*, 2006].

Tabela 16.2 Comandos de Edição para o Gerenciador da Base Privada Ginga

***String* de Comando**	***Tag* de Comando**	**Descrição**
openBase (baseId, location)	0x00	Abre uma base privada existente, localizada pelo parâmetro *location*. Se a base privada não existir ou se o parâmetro *location* não for informado, uma nova base é criada com o identificador *baseId*. O parâmetro *location* deve especificar o dispositivo e o caminho onde está a base a ser aberta
activateBase (baseId)	0x01	Ativa uma base privada aberta. Todas as aplicações ficam então aptas a serem iniciadas.
deactivateBase (baseId)	0x02	Desativa uma base privada aberta. Todas as suas aplicações devem ser terminadas.
saveBase (baseId, location)	0x03	Salva todo o conteúdo da base privada em um dispositivo de armazenamento persistente (se disponível). O parâmetro *location* deve especificar o dispositivo e o caminho para salvar a base
closeBase (baseId)	0x04	Fecha a base privada aberta e descarta todo o seu conteúdo

addDocument (baseId, {uri, id}+)	0x05	Adiciona um documento NCL a uma base privada aberta. Os arquivos do documento NCL podem ser: 1) Enviados pela rede de difusão de dados como um conjunto de arquivos enviados sem solicitação; nesse caso, o par {uri, id} é usado para relacionar um conjunto de caminhos de arquivos especificados no documento NCL com suas respectivas localizações no sistema de transporte (veja exemplos no Apêndice F) NOTA O conjunto de pares de referência deve ser suficiente para que se possa mapear qualquer referência a arquivos presentes na especificação do documento NCL na sua localização concreta na memória do dispositivo receptor. 2) Recebidos sob demanda pelo canal de interatividade ou já serem residentes no receptor; para esses arquivos, nenhum par {uri, id} necessita ser enviado, exceto o par {uri, "null"} associado ao documento NCL que deverá ser adicionado na base *baseId*, se o documento NCL não for recebido sem solicitação (pushed file)
removeDocument (baseId, documentId)	0x06	Remove um documento NCL de uma base privada aberta
startDocument (baseId, documentId, interfaceId, offset, nptBaseId, nptTrigger) NOTA O parâmetro *offset* especifica um valor de tempo NOTA O nptTrigger é um valor de NPT, e o nptBaseId é um identificador de uma base de tempo NPT	0x07	Inicia a reprodução de um documento NCL em uma base privada ativa, iniciando a apresentação a partir de uma interface específica do documento. A referência do tempo transportada no campo *nptTrigger* estabelece o ponto de início do documento, com respeito à base de tempo NPT identificada pelo campo *nptBaseId*. Três casos podem ocorrer: 1) Se *nptTrigger* for diferente de zero e for maior ou igual ao valor de NPT corrente da base temporal NPT identificada por *nptBaseId*, espera-se até que NPT atinja o valor dado em *nptTrigger* e começa a exibição do documento do seu ponto inícial no tempo+*offset* 2) Se *nptTrigger* for diferente de zero e for menor que o valor de NPT corrente da base temporal identificada por *nptBaseId*, o início da exibição do documento é imediata e deslocada no tempo de seu ponto incial do valor "*offset*+(NPT – *nptTrigger*)$_{seconds}$" NOTA Somente nesse caso o parâmetro *offset* pode receber um valor negativo, mas *offset*+(NPT – *nptTrigger*)$_{seconds}$ deve ser um valor positivo 3) Se *nptTrigger* for igual a 0, a exibição do documento é imediata e a partir de seu ponto inícial no tempo + *offset*
stopDocument (baseId, documentId)	0x08	Cessa a apresentação de um documento NCL em uma base privada ativa. Todos os eventos do documento que estão em andamento devem ser parados
pauseDocument (baseId, documentId)	0x09	Pausa a apresentação de um documento NCL em uma base privada ativa. Todos os eventos do documento que estão em andamento devem ser pausados
resumeDocument (baseId, documentId)	0x0A	Retoma a apresentação de um documento NCL em uma base privada ativa. Todos os eventos do documento que foram previamente pausados pelo o comando de edição *pauseDocument* devem ser retomados
saveDocument (baseId, documented, location)	0x2E	Salva um documento NCL de uma base privada aberta em um dispositivo de armazenamento persistente (se disponível). O parâmetro *location* deve especificar o dispositivo e o caminho no dispositivo onde o documento será salvo. Se o documento NCL estiver sendo exibido, ele deve primeiro ser parado (todos os eventos no estado *occurring* devem ser parados)

addRegion (baseId, documentId, regionBaseId, regionId, xmlRegion)	0x0B	Adiciona um elemento <region> como filho de outro <region>, no <regionBase>, ou como filho do <regionBase> (*regionId*=”null”) de um documento NCL em uma base privada aberta
removeRegion (baseId, documentId, regionId)	0x0C	Remove um elemento <region> de um <regionBase> de um documento NCL em uma base privada aberta
addRegionBase (baseId, documentId, xmlRegionBase)	0x0D	Adiciona um elemento <regionBase> ao elemento <head> de um documento NCL em uma base privada aberta. Se a especificação XML do regionBase for enviada em um sistema de transporte como um sistema de arquivo, o parâmetro *xmlRegionBase* é apenas uma referência para esse conteúdo
removeRegionBase (baseId, documentId, regionBaseId)	0x0E	Remove um elemento <regionBase> do elemento <head> de um documento NCL em uma base privada aberta
addRule (baseId, documentId, xmlRule)	0x0F	Adiciona um elemento <rule> ao <ruleBase> de um documento NCL em uma base privada aberta
removeRule (baseId, documentId, ruleId)	0x10	Remove um elemento <rule> do <ruleBase> de um documento NCL em uma base privada aberta
addRuleBase (baseId, documentId, xmlRuleBase)	0x11	Adiciona um elemento <ruleBase> ao elemento <head> de um documento NCL em uma base privada aberta. Se a especificação XML do ruleBase for enviada em um sistema de transporte como um sistema de arquivo, o parâmetro *xmlRuleBase* é apenas uma referência para esse conteúdo
removeRuleBase (baseId, documentId, ruleBaseId)	0x12	Remove um elemento <ruleBase> do elemento <head> de um documento NCL em uma base privada aberta
addConnector (baseId, documentId, xmlConnector)	0x13	Adiciona um elemento <connector> ao <connectorBase> de um documento NCL em uma base privada aberta
removeConnector (baseId, documentId, connectorId)	0x14	Remove um elemento <connector> do <connectorBase> de um documento NCL em uma base privada aberta
addConnectorBase (baseId, documentId, xmlConnectorBase)	0x15	Adiciona um elemento <connectorBase> ao elemento <head> de um documento NCL em uma base privada aberta. Se a especificação XML do connectorBase for enviada em um sistema de transporte como um sistema de arquivo, o parâmetro *xmlConnectorBase* é apenas uma referência para esse conteúdo
removeConnectorBase (baseId, documentId, connectorBaseId)	0x16	Remove um elemento <connectorBase> do elemento <head> de um documento NCL em uma base privada aberta
addDescriptor (baseId, documentId, xmlDescriptor)	0x17	Adiciona um elemento <descriptor> ao <descriptorBase> de um documento NCL em uma base privada aberta
removeDescriptor (baseId, documentId, descriptorId)	0x18	Remove um elemento <descriptor> do <descriptorBase> de um documento NCL em uma base privada aberta
addDescriptorSwitch (baseId, documentId, xmlDescriptorSwitch)	0x19	Adiciona um elemento <descriptorSwitch> ao <descriptorBase> de um documento NCL em uma base privada aberta. Se a especificação XML do descriptorSwitch for enviada em um sistema de transporte como um sistema de arquivo, o parâmetro *xmlDescriptorSwitch* é apenas uma referência para esse conteúdo
removeDescriptorSwitch (baseId, documentId, descriptorSwitchId)	0x1A	Remove um elemento <descriptorSwitch> do <descriptorBase> de um documento NCL em uma base privada aberta
addDescriptorBase (baseId, documentId, xmlDescriptorBase)	0x1B	Adiciona um elemento <descriptorBase> ao elemento <head> de um documento NCL em uma base privada aberta. Se a especificação XML do descriptorBase for enviada em um sistema de transporte como um sistema de arquivo, o parâmetro *xmlDescriptorBase* é apenas uma referência para esse conteúdo
removeDescriptorBase (baseId, documentId, descriptorBaseId)	0x1C	Remove um elemento <descriptorBase> do elemento <head> de um documento NCL em uma base privada aberta
addTransition (baseId, documentId, xmlTransition)	0x1D	Adiciona um elemento <transition> ao <transitionBase> de um documento NCL em uma base privada aberta

removeTransition (baseId, documentId, transitionId)	0x1E	Remove um elemento <transition> do <transitionBase> de um documento NCL em uma base privada aberta
addTransitionBase (baseId, documentId, xmlTransitionBase)	0x1F	Adiciona um elemento <transitionBase> ao elemento <head> de um documento NCL em uma base privada aberta. Se a especificação XML do transitionBase for enviada em um sistema de transporte como um sistema de arquivo, o parâmetro *xmlTransitionBase* é apenas uma referência para esse conteúdo no carrossel
removeTransitionBase (baseId, documentId, transitionBaseId)	0x20	Remove um elemento <transitionBase> do elemento <head> de um documento NCL em uma base privada aberta
addImportBase (baseId, documentId, docBaseId, xmlImportBase)	0x21	Adiciona um elemento <importBase> à base (elemento <regionBase>, <descriptorBase>, <ruleBase>, <transitionBase> ou <connectorBase>) de um documento NCL em uma base privada aberta
removeImportBase (baseId, documentId, docBaseId, documentURI)	0x22	Remove um elemento <importBase>, cujo atributo documentURI é identificado pelo parâmetro *documentURI*, a partir da base (elemento <regionBase>, <descriptorBase>, <ruleBase>, <transitionBase> ou <connectorBase>) de um documento NCL em uma base privada aberta
addImportedDocumentBase (baseId, documentId, xmlImportedDocumentBase)	0x23	Adiciona um elemento <importedDocumentBase> ao elemento <head> de um documento NCL em uma base privada aberta
removeImportedDocumentBase (baseId, documentId, importedDocumentBaseId)	0x24	Remove um elemento <importedDocumentBase> do elemento <head> de um documento NCL em uma base privada aberta
addImportNCL (baseId, documentId, xmlImportNCL)	0x25	Adiciona um elemento <importNCL> ao elemento <importedDocumentBase> de um documento NCL em uma base privada aberta
removeImportNCL (baseId, documentId, documentURI)	0x26	Remove um elemento <importNCL>, cujo atributo documentURI é identificado pelo parâmetro *documentURI*, a partir do <importedDocumentBase> de um documento NCL em uma base privada aberta
addNode (baseId, documentId, compositeId, {uri, id}+)	0x27	Adiciona um nó (elemento <media>, <context> ou <switch>) a um nó *de composição* (elemento <body>, <context> ou <switch>) de um documento NCL em uma base privada aberta. A especificação XML do nó e seu conteúdo de mídia podem: 1)Ser enviados pela rede de difusão de dados como um conjunto de arquivos enviados sem solicitação; nesse caso, o par {uri, id} é usado para relacionar um conjunto de caminhos de arquivos definidos no documento XML da especificação do nó, com suas respectivas localizações no sistema de transporte (veja exemplos no Apêndice F) NOTA Os conjuntos de pares de referência devem ser suficientes para que o *middleware* possa mapear qualquer referência a arquivos, presentes na especificação do nó, na sua localização concreta na memória do dispositivo receptor 2) Recebidos sob demanda pelo canal de interatividade como um conjunto de arquivos ou já serem residentes no receptor; para esses arquivos, nenhum par {uri, id} necessita ser enviado, exceto o par {uri, "null"} associado à especificação XML do nó que deverá ser adicionado em *compositeId*, caso o documento XML não seja recebido sem solicitação (*pushed file*)
removeNode(baseId, documentId, compositeId, nodeId)	0x28	Remove um nó (elemento <media>, <context> ou <switch>) de um nó de composição (elemento <body>, <context> ou <switch>) de um documento NCL em uma base privada aberta

addInterface (baseId, documentId, nodeId, xmlInterface)	0x29	Adiciona uma interface (<port>, <area>, <property> ou <switchPort>) a um nó (elemento <media>, <body>, <context> ou <switch>) de um documento NCL em uma base privada aberta
removeInterface (baseId, documentId, nodeId, interfaceId)	0x2A	Remove uma interface (<port>, <area>, <property> ou <switchPort>) de um nó (elemento <media>, <body>, <context> ou <switch>) de um documento NCL em uma base privada aberta. O parâmetro *interfaceID* deve identificar um atributo *name* de um elemento <property> ou um atributo *id* de um elemento <port>, <area> ou <switchPort>
addLink (baseId, documentId, compositeId, xmlLink)	0x2B	Adiciona um elemento <link> a um nó de composição (elemento <body>, <context> ou <switch>) de um documento NCL em uma base privada aberta
removeLink (baseId, documentId, compositeId, linkId)	0x2C	Remove um elemento <link> de um nó de composição (elemento <body>, <context> ou <switch>) de um documento NCL em uma base privada aberta
setPropertyValue(baseId, documentId, nodeId, propertyId, value)	0x2D	Atribui o valor a uma propriedade. O parâmetro *propertyId* deve identificar um atributo *name* de um elemento <property> ou um atributo *id* de elemento <switchPort>. O <property> ou <switchPort> deve pertencer a um nó (elemento <body>, <context>, <switch> ou <media>) de um documento NCL em uma base privada aberta identificada pelos parâmetros

A Tabela 16.3 apresenta os identificadores utilizados nos comandos de edição e suas definições.

Tabela 16.3 Identificadores Usados nos Comandos de edição

Identificadores	Definição
baseId	O identificador de uma base privada. Usualmente, em ambientes de TV digital, uma base privada é associada a um canal de TV. Assim, um identificador do canal de TV pode ser usado como valor de *baseId*
documentId	Atributo *id* de um elemento <ncl> de um documento NCL
nptTrigger	Um valor de NPT
nptBaseId	Identificador *contentId* de uma base temporal NPT
regionId	Atributo *id* de um elemento <region> de um documento NCL
ruleId	Atributo *id* de um elemento <rule> de um documento NCL
connectorId	Atributo *id* de um elemento <connector> de um documento NCL
descriptorId	Atributo *id* de um elemento <descriptor> de um documento NCL
descriptorSwitchId	Atributo *id* de um elemento <descriptorSwitch> de um documento NCL
transitionId	Atributo *id* de um elemento <transition> de um documento NCL
regionBaseId	Atributo *id* de um elemento <regionBase> de um documento NCL

ruleBaseId	Atributo *id* de um elemento <ruleBase> de um documento NCL
connectorBaseId	Atributo *id* de um elemento <connectorBase> de um documento NCL
descriptorBaseId	Atributo *id* de um elemento <descriptorBase> de um documento NCL
transitionBaseId	Atributo *id* de um elemento <transitionBase> de um documento NCL
docBaseId	Atributo *id* de um elemento <regionBase>, <ruleBase>, <connectorBase>, <descriptorBase> ou <transitionBase> de um documento NCL
documentURI	Atributo documentURI de um elemento <importBase> ou um elemento <importNCL> de um documento NCL
importedDocumentBaseId	Atributo *id* de um elemento <importedDocumentBase> de um documento NCL
compositeID	Atributo *id* de um elemento <body>, <context> ou <switch> de um documento NCL
nodeId	Atributo *id* de um elemento <body>, <context>, <switch> ou <media> de um documento NCL
interfaceId	Atributo *id* de um elemento <port>, <area>, <property> ou <switchPort> de um documento NCL
linkId	Atributo *id* de um elemento <link> de um documento NCL
propertyId	Atributo *id* de um elemento <property> ou <switchPort> de um documento NCL

A Seção 16.3 apresenta alguns exemplos simples de comandos de edição NCL. Os comandos de edição *addDocument* e *addNode*, bem mais complexos, são exemplificados no Apêndice F, onde são apresentadas várias opções de transporte para esses comandos.

16.3 Exemplos de Comandos de Edição NCL

Vamos agora aplicar os vários conceitos apresentados neste capítulo em uma série de exemplos.

A série de exemplos é composta dos seguintes passos, realizados através de comandos de edição NCL:

1. Abrir uma base privada.
2. Ativar a base privada aberta.
3. Adicionar um documento na base privada aberta.
4. Iniciar a exibição do documento inserido.

5. Adicionar uma região à base de regiões do documento e em seguida removê-la.
6. Adicionar uma interface (âncora de conteúdo) a um objeto do documento.
7. Adicionar um novo objeto ao documento.
8. Adicionar um elo ligando a nova interface adicionada ao novo objeto adicionado.
9. Parar a exibição do documento.
10. Salvar o documento.
11. Fechar a base privada.

Note que todo comando a partir do passo 4 se dará com o documento em exibição, ou seja, todos os comandos de 5 em diante são de edição ao vivo.

Passemos. então. à realização de cada passo.

16.3.1 Abrir uma Base Privada

Para abrir uma nova base privada, anteriormente inexistente, devemos enviar o comando:

```
openBase (baseId="TV GINGA")
```

A não-especificação do parâmetro *location* indica a abertura de uma nova base privada.

O descritor de evento terá a sintaxe vista na Tabela 16.4.

Tabela 16.4 Descritor de evento para abrir uma base privada

Campo	Valor
eventid	Qualquer valor de 16 bits
eventNPT	0
privateDataLength	Comprimento do restante do comando
commandTag	0x00
sequenceNumber	0x00
finalFlag	0
privateDataPayload	TV GINGA
FCS	8 bits de *checksum*

Note que o comando determina uma abertura imediata da base (evento do tipo “do it now”), que terá o identificador “TV GINGA”.

16.3.2 Ativar a Base Privada aberta

Para ativar a nova base privada, devemos enviar o comando:

```
activateBase (baseId="TV GINGA").
```

O descritor de evento terá a sintaxe vista na Tabela 16.5.

Tabela 16.5 Descritor de evento para ativar uma base privada aberta

Campo	Valor
eventid	Qualquer valor de 16 bits
eventNPT	0
privateDataLength	Comprimento do restante do comando
commandTag	0x01
sequenceNumber	0x00
finalFlag	0
privateDataPayload	TV GINGA
FCS	8 bits de *checksum*

Note que o comando, também nesse caso, determina a ativação imediata (evento do tipo "do it now") da base "TV GINGA".

16.3.3 Adicionar um Documento na Base Privada Aberta

Vamos adicionar o documento weatherConditions.ncl à base privada TV GINGA. Suponha que o documento e seus arquivos de conteúdo estão no sistema de arquivos mostrado na Figura 16.1.

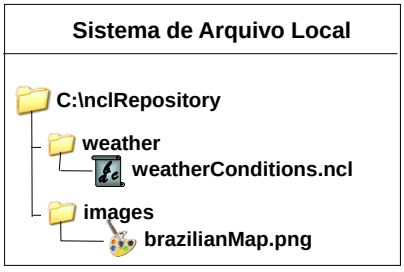


Figura 16.1 Sistema de arquivos do documento a ser adicionado.

Suponha também que todos os arquivos do documento serão enviados em um único carrossel de objetos (veja Apêndice F) e que todos os objetos de mídia do documento têm seu atributo *src* especificado de forma relativa à localização da especificação NCL do documento. Nesse caso, basta que o comando de edição faça referência ao caminho do diretório onde está o documento NCL e onde ele será transportado no carrossel (no caso, por exemplo: Service Domain "0x1", module "0x1" e object "0x2"), pois todos os arquivos poderão ter seus caminhos absolutos resolvidos a partir desse relacionamento, como discutido no Apêndice F.

O comando de edição a ser enviado é dado por:

```
addDocument (baseId="TV GINGA",
{uri,id}={"C:\nclRepository\weather","0x1, 0x1, 0x2"})
```

O descritor de evento terá a sintaxe vista na Tabela 16.6.

Tabela 16.6 Descritor de evento para adicionar um documento a uma base privada aberta

Campo	Valor
eventid	Qualquer valor de 16 bits
eventNPT	0
privateDataLength	Comprimento do restante do comando
commandTag	0x05
sequenceNumber	0x00
finalFlag	0
privateDataPayload	TV GINGA, {"C:\ nclRepository\weather","0x1, 0x1, 0x2"}
FCS	8 bits de *checksum*

Note que, mais uma vez, o comando determina a adição imediata do documento na base.

16.3.4 Iniciar a Exibição do Documento

Vamos agora supor que o documento weatherConditions.ncl tenha um objeto de vídeo (*id*= "noticias") sobre um noticiário em exibição e que tal objeto tem como fonte o fluxo de vídeo principal de um programa. Vamos também supor que, quando o fluxo de vídeo atingir o valor de NPT= "49", estará no momento em que o noticiário apresentará a previsão do tempo e que, por isso, desejamos iniciar a exibição do documento weatherConditions.ncl. Note que esse é um caso comum em que a base

temporal para o início do documento é proveniente do fluxo do vídeo principal sintonizado.[2]

Para iniciar a apresentação do documento, devemos enviar o comando:

```
startDocument (baseId="TV GINGA",
documentId="Jornal Ginga", interfaceId="porta",
offset="0", nptTrigger="49",nptBaseId="null"),
```

em que a interface de entrada do documento tem *id* = "porta".

O descritor de evento terá a sintaxe vista na Tabela 16.7.

Tabela 16.7 Descritor de evento para iniciar a exibição de um documento

Campo	Valor
eventid	Qualquer valor de 16 bits
eventNPT	0
privateDataLength	Comprimento do restante do comando
commandTag	0x07
sequenceNumber	0x00
finalFlag	0
privateDataPayload	TV GINGA, Jornal Ginga, porta, 0, 49, null
FCS	8 bits de *checksum*

Note que o comando determina o início da exibição em NPT = "49".

16.3.5 Adicionar uma Região à Base de Regiões e em Seguida Removê-la

Apenas como exemplo de remoção, vamos adicionar e remover uma região da base de regiões (*id* = "regBase").

Para adicionar uma nova região, o seguinte comando deve ser enviado:

```
addRegion (baseId="TV GINGA", documentId="Jornal
Ginga", regionBaseId="regBase", regionId="null",
xmlRegion="<region id="regiaoX" width="100%"
height="100%" zIndex="1"/>")
```

[2] No caso do Sistema Brasileiro de TV Digital Terrestre, quando em um comando startDocument, o parâmetro nptBaseId for "null", o valor do campo eventId do descritor de eventos de fluxo deve estar listado em um objeto de eventos de fluxo DSM-CC cujo campo *id* de seu STR_NPT_USE tap (campo use do tap) identifica a base de tempo NPT (o campo *id* do tap deve obrigatoriamente conter o contentId da base de tempo).

O descritor de evento terá a sintaxe vista na Tabela 16.8.

Tabela 16.8 Descritor de evento para acrescentar uma região a uma base de regiões

Campo	Valor
eventid	Qualquer valor de 16 bits
eventNPT	0
privateDataLength	Comprimento do restante do comando
commandTag	0x0B
sequenceNumber	0x00
finalFlag	0
privateDataPayload	TV GINGA, Jornal Ginga, regBase, null, <region id="regiaoX" width="100%" height="100%" zIndex="1"/>
FCS	8 bits de *checksum*

Note que, mais uma vez, o comando determina a adição imediata (evento do tipo "do it now") da região.

Para remover a nova região, o seguinte comando deve ser enviado:

```
removeRegion (baseId="TV GINGA",
documentId="Jornal Ginga", regionId="regiaoX")
```

O descritor de evento terá a e sintaxe vista na Tabela 16.9.

Tabela 16.9 Descritor de evento para remover uma região

Campo	Valor
eventid	Qualquer valor de 16 bits
eventNPT	0
privateDataLength	Comprimento do restante do comando
commandTag	0x0B
sequenceNumber	0x00
finalFlag	0
privateDataPayload	TV GINGA, Jornal Ginga, regiaoX
FCS	8 bits de *checksum*

16.3.6 Adicionar uma Interface a um Objeto do Documento

Para adicionar uma nova âncora de conteúdo ao objeto de vídeo (*id*="noticias"), o seguinte comando deve ser enviado:

```
addInterface (baseId="TV GINGA", documentId="Jornal
Ginga", nodeId="noticias", xmlInterface="<area
id="tempoRio" first="72npt" last="75npt"/>")
```

Note que a âncora começa quando o vídeo chega ao tempo 72 npt e termina em 75 npt.

O descritor de evento terá a sintaxe vista na Tabela 16.10.

Tabela 16.10 Descritor de evento para adicionar uma interface a um objeto de um documento

Campo	Valor
eventid	Qualquer valor de 16 bits
eventNPT	0
privateDataLength	Comprimento do restante do comando
commandTag	0x29
sequenceNumber	0x00
finalFlag	0
privateDataPayload	TV GINGA, Jornal Ginga, noticias, <area id="tempoRio" first="12npt" last="15npt"/>
FCS	8 bits de *checksum*

16.3.7 Adicionar um Novo Objeto ao Documento

Vamos agora adicionar ao documento weatherConditions.ncl um objeto com uma imagem que, posteriormente, desejaremos exibir quando o vídeo (notícias) atingir a âncora que introduzimos na seção anterior. O objeto será adicionado como filho do elemento <body id= "idBody">.

Suponha que o arquivo XML que especifica o objeto deva ser buscado em ftp://salgueiro.telemidia.puc-rio.br/tmp e que o conteúdo da imagem deva ser buscado no mesmo site. Nesse caso, apenas um par {uri, id} deve ser enviado, com id= "null".

O comando de edição a ser enviado é dado por:

```
addNode (baseId="TV GINGA", documentId="Jornal Ginga",
compositeId="idBody",
{uri,id}={"ftp://salgueiro.telemidia.puc-rio.br/tmp",
"null"})
```

O descritor de evento terá a sintaxe vista na Tabela 16.11.

Tabela 16.11 Descritor de evento para acrescentar um objeto a um documento

Campo	Valor
eventid	Qualquer valor de 16 bits
eventNPT	0
privateDataLength	Comprimento do restante do comando
commandTag	0x27
sequenceNumber	0x00
finalFlag	0
privateDataPayload	TV GINGA, Jornal Ginga, idBody, {"ftp://salgueiro.telemidia.puc-rio.br/tmp", "null"}
FCS	8 bits de *checksum*

Note que, mais uma vez, o comando determina a adição imediata do elemento de mídia.

16.3.8 Adicionar um Elo Ligando a Nova Interface Adicionada ao Novo Objeto Adicionado

Suponha que o elo que permitirá a exibição da imagem adicionada ao documento tão logo o vídeo (notícias) chegue a 72 npt, tenha sido definido em um arquivo que está sendo transportado no carrossel de objetos com a localização dada por Service Domain="0x1", module="0x1" e object="0x3".

O seguinte comando de adição deve ser então enviado:

```
addLink (baseId="TV GINGA", documentId="Jornal Ginga",
compositeId="idBody", xmlLink="(0x1, 0x1, 0x3)")
```

O descritor de evento terá a e sintaxe vista na Tabela 16.12.

Tabela 16.12 Descritor de evento para acrescentar um elo a um documento

Campo	Valor
eventid	Qualquer valor de 16 bits
eventNPT	0
privateDataLength	Comprimento do restante do comando
commandTag	0x2B
sequenceNumber	0x00
finalFlag	0
privateDataPayload	TV GINGA, Jornal Ginga, idBody, (0x1, 0x1, 0x3)
FCS	8 bits de *checksum*

16.3.9 Parar a Exibição do Documento

Para parar a exibição do documento, devemos enviar o comando:

```
stopDocument (baseId="TV GINGA",
   documentId="Jornal Ginga")
```

O descritor de evento terá a sintaxe vista na Tabela 16.13.

Tabela 16.13 Descritor de evento para parar a exbição de um documento

Campo	Valor
eventid	Qualquer valor de 16 bits
eventNPT	0
privateDataLength	Comprimento do restante do comando
commandTag	0x08
sequenceNumber	0x00
finalFlag	0
privateDataPayload	TV GINGA, Jornal Ginga
FCS	8 bits de *checksum*

16.3.10 Salvar o Documento

Para salvar o documento, devemos enviar o comando:[3]

```
saveDocument (baseId="TV GINGA", documentId="Jornal
      Ginga", location="C:\baseDeDocumentos")
```

O descritor de evento terá a sintaxe vista na Tabela 16.14.

Tabela 16.14 Descritor de evento para savar um documento

Campo	Valor
eventid	Qualquer valor de 16 bits
eventNPT	0
privateDataLength	Comprimento do restante do comando
commandTag	0x2E
sequenceNumber	0x00
finalFlag	0
privateDataPayload	TV GINGA, Jornal Ginga, C:\baseDeDocumentos
FCS	8 bits de *checksum*

[3] Cabe ressaltar que não seria necessário o envio do comando para parar a execução do documento, uma vez que a execução do comando para salvar um documento tem como primeira ação parar sua apresentação.

16.3.11 Fechar a Base Privada Aberta

Finalmente, para fechar a base privada, devemos enviar o comando:

```
closeBase (baseId="TV GINGA").
```

O descritor de evento terá a sintaxe vista na Tabela 16.15.

Tabela 16.15 Descritor de evento para fechar uma base privada

Campo	Valor
eventid	Qualquer valor de 16 bits
eventNPT	0
privateDataLength	Comprimento do restante do comando
commandTag	0x04
sequenceNumber	0x00
finalFlag	0
privateDataPayload	TV GINGA
FCS	8 bits de *checksum*

Bibliografia

ABNT, NBR 15606-2 (2011). Associação Brasileira de Normas Técnicas, "Televisão digital terrestre — Codificação de dados e especificações de transmissão para radiodifusão digital, Parte 2: Ginga-NCL para receptores fixos e móveis — Linguagem de aplicação XML para codificação de aplicações", Sistema Brasileiro de TV Digital Terrestre, *NBR 15606-2*.

ISO/IEC 13818-1 (2000). International Organization for Standardization/International Eletrotecnical Committee, "Information Technology — Generic coding of moving pictures and associated audio information, Part 1: Systems", *ISO/IEC 13818-1*.

ISO/IEC 13818-6 (1998). International Organization for Standardization/International Eletrotecnical Committee, "Information Technology — Generic coding of moving pictures and associated audio information, Part 6: Extensions for DSM-CC", *ISO/IEC 13818-6*.

Soares, L.F.S. e Rodrigues, R.F.; Costa, R.R; Moreno, M. (2006). "Nested Context Model 3.0 Part 9 — NCL Live Editing Commands. Monografias em Ciência da Computação do Departamento de Informática, PUC-Rio, N.º 35/06. Rio de Janeiro, dezembro de 2006. ISSN 0103-9741.

Capítulo 17

Objetos Imperativos em NCL

A NCL aceita não apenas objetos cujo conteúdo é composto por código declarativo na definição de seus elementos <media>, como vimos no Capítulo 14, mas também objetos cujo conteúdo é composto por código imperativo.

Neste capítulo, discutiremos como objetos com código imperativo podem ser definidos, como eles podem se relacionar com outros objetos em um documento NCL e como os exibidores (*engines*) para esses objetos se comportam.[1]

Objetos e exibidores NCLua (objetos imperativos com código Lua[2]) são por definição parte dos perfis da linguagem NCL para TV digital, e a eles dedicaremos os capítulos seguintes deste livro. Lua é a principal linguagem de script de NCL, e é linguagem-padrão do Sistema Nipo-Brasileiro de TV Digital terrestre, e da Recomendação ITU-T H.761 para serviço IPTV, na especificação-padrão do *middleware* Ginga.

[1] Este capítulo foi baseado em Soares *et al.* (2008). O uso do material foi gentilmente cedido pelo Departamento de Informática da PUC-Rio.

[2] Na verdade Lua é uma linguagem multiparadigma: imperativa e funcional.

17.1 Integrando Objetos Imperativos à NCL

Como vimos no Capítulo 1,o universo das aplicações de TVD (TV Digital) pode ser particionado em um conjunto de aplicações declarativas e um conjunto de aplicações imperativas. A entidade inicial de uma aplicação, isto é, aquela que dispara a aplicação, é que define a que conjunto a aplicação pertence, dependendo de essa entidade ser codificada segundo uma linguagem declarativa ou imperativa. Note que aplicações declarativas podem conter entidades imperativas e vice-versa; o que as caracteriza é apenas a entidade inicial.

Linguagens declarativas enfatizam a descrição declarativa de uma tarefa, em vez de sua decomposição passo a passo, em uma definição algorítmica do fluxo de execução de uma máquina, como fazem as descrições imperativas. Por ser de mais alto nível de abstração, tarefas descritas de forma declarativa são mais fáceis de ser concebidas e entendidas, sem exigir um programador especialista, como é usualmente necessário nas tarefas descritas de forma imperativa. Contudo, uma linguagem declarativa muitas vezes visa a um determinado domínio de aplicações e define um modelo específico para esse domínio.

Linguagens imperativas de propósito geral são bem expressivas, porém a um elevado custo. Como mencionado, elas usualmente exigem um programador especialista, geralmente colocam em risco a portabilidade de uma aplicação, e o controle da aplicação é muito mais sujeito a erros cometidos pelo programador.

A autoria de aplicações puramente declarativas usando linguagens de domínio específico é vantajosa quando ela depende apenas de recursos previstos no projeto da linguagem. Quando uma aplicação necessita de funcionalidades não-previstas pela linguagem declarativa, a solução pode se tornar complicada ou até mesmo impossível.

Uma solução para esse impasse consiste em adicionar algum suporte imperativo à linguagem declarativa de domínio específico; assim, o autor de aplicações poderá usar a forma declarativa sempre que possível e tirar proveito da forma imperativa quando necessário. Quando uma tarefa casar com o modelo da linguagem declarativa, o paradigma declarativo será, em geral, a melhor escolha. No entanto, nos casos em que o foco de realização de uma tarefa não casar com o foco da linguagem declarativa, o paradigma imperativo será, em geral, a melhor escolha. De fato, a solução pode advir pela adição, à linguagem declarativa de domínio específico, de suporte proveniente do uso de linguagens de propósito geral, incluindo, além das linguagens imperativas, as linguagens funcionais, linguagens lógicas etc.

Em NCL, a realização de algumas tarefas é complicada sem auxílio imperativo, tal como processamento matemático, manipulação sobre textos, animações complexas e colisões para objetos gráficos, enfim, de modo geral, tarefas que necessitem da especificação de algoritmos e estruturas de dados que não aquelas providas de forma nativa pela linguagem.

Objetos imperativos podem ser incluídos em documentos NCL definindo novos tipos de elemento <media>, cujo conteúdo (localizado através do atributo *src*) seria composto por códigos em alguma linguagem imperativa[3]. Por exemplo, os perfis BDTV e EDTV da NCL para o Sistema Brasileiro de TV Digital terrestre incluem os tipos application/x-ncl-NCLua, para objetos de mídia com código Lua (extensão de arquivo .lua), e application/x-ginga-NCLet, para objetos de mídia com código Java (Xlet) (extensão de arquivo .class ou .jar), esse último só para dispositivos fixos. A Recomendação H.761 para serviços IPTV inclui apenas o tipo application/x-ncl-NCLua.

Alguns requisitos devem ser cumpridos na integração de linguagens declarativas e imperativas.

No caso de linguagens declarativas e imperativas já especificadas e implementadas, deve-se mexer o mínimo nas linguagens, para evitar o surgimento de dependências mútuas que comprometam a evolução independente de cada uma delas.

Linguagens declarativas, como a NCL, são facilmente compreendidas por autores de conteúdo audiovisual que não possuem base de programação, ao contrário de linguagens imperativas. Sendo assim, na integração de linguagens declarativas e imperativas, também é desejado que haja o mínimo de intercalação entre seus códigos de modo a simplificar a divisão de tarefas entre equipes de profissionais técnicos e não-técnicos em linguagens de programação.

Por fim, em apresentações multimídia, o documento declarativo deve ser o componente-mestre no qual todos os relacionamentos entre entidades da aplicação, sejam elas declarativas ou não, devem ser definidos explicitamente (por meio de elementos da linguagem declarativa), impedindo que entidades imperativas sobrepujem essa hierarquia por meio de acessos diretos à estrutura do documento.

---

[3] Devemos mais uma vez enfatizar que NCL, como uma linguagem cola, não restringe nem prescreve qualquer tipo de objeto de mídia. Assim poderíamos também ter objetos de mídia com código puramente funcional, ou em uma linguagem lógica etc. Esses novos objetos poderiam até seguir as mesmas definições dadas neste capítulo. No entanto, vamos, no momento, nos restringir a objetos imperativos, ou seja, cujo conteúdo é definido por trechos de código especificados em uma linguagem seguindo o paradigma imperativo, incluindo linguagens, como Lua, que não são puramente imperativas.

Todos os requisitos explicitados nos três parágrafos anteriores guiaram a integração de objetos imperativos à NCL. Um objeto com código imperativo deve ser escrito em um arquivo separado do documento NCL, que apenas o referencia. Os códigos imperativos usam a mesma abstração para objetos de mídia usada para imagens, vídeos e outras mídias (ver Parte II). Em outras palavras, um documento NCL apenas relaciona objetos de mídia, não importando qual o tipo de seu conteúdo. Mais ainda, como para todo objeto de mídia, a comunicação com o mundo externo ao objeto somente pode ser feita através de elos (elementos <link>, ver Capítulo 10) ou da leitura de variáveis do nó NCL settings (elemento <media type="application/x-ncl-settings">, ver Capítulo 8), que também só podem ser escritas através do uso de elos.

Resumindo, a motivação de mínima intrusão destaca três requisitos, que foram obedecidos pela NCL na sua integração com entidades especificadas em uma linguagem imperativa:

1. Tanto a NCL quanto a linguagem imperativa são alteradas o mínimo possível na viabilização da integração.

2. Do ponto de vista do desenvolvedor, é mantida uma fronteira bem delineada entre os dois paradigmas de programação.

3. Em termos de projeto, a relação entre as duas máquinas[4] de execução (declarativa NCL e imperativa) é ortogonal; em outras palavras, operações no ambiente de uma máquina não produzem efeitos imprevistos no ambiente da outra máquina.

17.2 Elemento <media type="application/x-???">

Um objeto de mídia com código imperativo é definido em NCL pelo elemento <media> com o atributo *type* recebendo o valor "application/x-???", onde ??? depende da linguagem imperativa usada. Por exemplo, "application/x-ncl-NCLua" é usado no *middleware* Ginga para objetos com código Lua. O atributo *src* deve, nesse caso, referenciar a localização do código imperativo que compõe o conteúdo do objeto. A Listagem 17.1 ilustra um exemplo de especificação de objeto imperativo com código Lua. Note que o fato de usarmos a extensão ".lua" nos desobriga da definição do atributo *type*.

[4] Uma máquina de execução declarativa (também chamada de *player*, *formatador*, *agente de usuário* etc.) é aquela que inicia e gerencia todo o ciclo de vida de uma aplicação declarativa. Analogamente, uma máquina de execução imperativa (chamada de *engine*, *máquina virtual* etc.) é aquela que inicia e gerencia todo o ciclo de vida de uma aplicação imperativa.

```
<media id="objetoLua" src="exemplo.lua">
      <property name="propriedade1"/>
      <property name="propriedade2" value="0"/>
      <area id="ancora1" label="final"/>
      <area id="ancora2"/>
</media>
```

Listagem 17.1 Objeto de mídia com código Lua.

Igualmente para qualquer outro objeto de mídia, o elemento <media> representando um objeto de mídia com código imperativo pode definir âncoras de conteúdo (através do elemento <area>) e propriedades (através do elemento <property>). Também igual para qualquer outro objeto de mídia, o atributo *descriptor*, opcional, deve se referir a um elemento <descriptor> que é responsável pela iniciação de propriedades necessárias à apresentação (execução) do objeto.

Em uma implementação de um formatador NCL, cabe ao exibidor de um objeto de mídia a responsabilidade de interpretar a semântica associada a suas âncoras. Por exemplo, em caso de elementos <media> do tipo "imagem", uma âncora pode definir uma região em pixels da imagem completa; em caso de elementos <media> do tipo "video", uma âncora pode definir um trecho temporal do vídeo. No caso de objetos imperativos, cabe ao programador de cada objeto especificar (programar) o comportamento de suas âncoras. Um possível e comum comportamento é o de executar um trecho do código (uma função, um método etc.) de mesmo nome da âncora, especificado através do atributo *label* do elemento <area>, como ilustrado na Listagem 17.1 pela "ancora1". É comum também o uso de âncoras apenas para definição de máquinas de estados, cuja execução é comandada pelo código imperativo, como ilustrado na Listagem 17.1 pela "ancora2". Essas últimas âncoras poderiam ser usadas em papéis de condição de elementos <link>, disparando ações em outros objetos de um documento NCL, como discutiremos na Seção 17.3.1.

Como é usual na NCL, um objeto imperativo sempre define uma âncora representando o conteúdo total do nó. Essa âncora, que anteriormente conhecemos com o nome "âncora de conteúdo total", é declarada por omissão (*default*). Essa âncora tem, entretanto, um sentido especial em um objeto de mídia imperativo. Ela representa a execução de "*qualquer*" trecho de código do objeto.

Todo objeto imperativo define também por omissão (*default*) uma outra âncora chamada "âncora de conteúdo principal". Cada vez que um objeto imperativo é iniciado (ação "start" no objeto) sem especificar uma de suas âncoras de conteúdo ou propriedade, a "âncora de conteúdo principal" é assumida e, como consequência, o trecho de código a ela associado é executado. Em todos os outros casos de referência a um objeto imperativo

sem especificar uma de suas âncoras de conteúdo ou propriedade, a "âncora de conteúdo total" é assumida.

Um elemento <property>, filho de um elemento <media>, é usado para parametrizar o comportamento do exibidor do objeto. Como para as âncoras de conteúdo, cabe ao exibidor de mídia a responsabilidade de interpretar a semântica de cada propriedade. Por exemplo, um exibidor de texto deve ser capaz de interpretar as propriedades fontStyle e fontSize; já um exibidor de som, as propriedades soundLevel e bassLevel. No caso de objetos imperativos, cabe ao programador de cada objeto especificar (programar) o comportamento de suas propriedades. Uma abordagem possível e comum é mapear cada propriedade em variáveis de mesmo nome do código imperativo a executar, como ilustrado na Listagem 17.1 pela "propriedade1" ou, então, a trechos de código (funções, métodos etc.) que tratem de forma especial o valor atribuído à propriedade, como ilustrado na Listagem 17.1 pela "propriedade2".

De forma análoga a qualquer objeto de mídia, toda âncora de conteúdo ou propriedade possui uma máquina de estado de evento (veja Figura 17.1) associada.

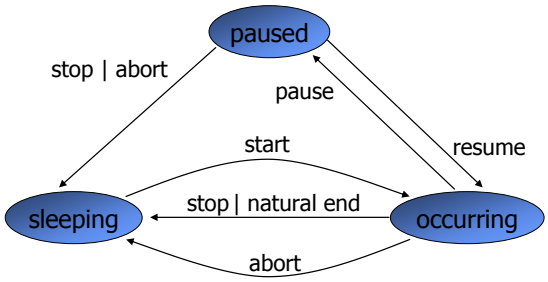


Figura 17.1 Máquina de estado associada a âncoras de conteúdo ou propriedade.

17.3 Comportamento Esperado de Exibidores de Objetos Imperativos em Aplicações NCL

Um exibidor de objeto imperativo (máquina de execução da linguagem) deve obrigatoriamente prover a interface do ambiente de execução imperativo com o formatador NCL, como se segue.

17.3.1 Ponte entre os Ambientes Declarativo e Imperativo

Os autores podem definir elos NCL para iniciar, parar, pausar, retomar ou abortar uma âncora de conteúdo de um objeto imperativo. Nesses casos,

callbacks no código imperativo são disparados. A forma como esses *callbacks* são definidos é responsabilidade de cada código associado com o objeto imperativo.

Por outro lado, um código imperativo pode também comandar o início, a parada, a pausa, a retomada ou o aborto de suas âncoras de conteúdo, através de alguma API oferecida pela linguagem. Essas transições podem ser utilizadas como condições de elos NCL para disparar ações em outros objetos do mesmo documento NCL.

Pelos dois parágrafos anteriores, temos a primeira forma de estabelecer uma sincronização de duas vias entre o código imperativo e o restante do documento NCL.

A outra forma que um código imperativo pode ser sincronizado com outros objetos de um documento NCL é através de elementos <property>, já definidos na seção anterior. Naquela seção, vimos que um elemento <property> de um objeto de mídia imperativo pode ser mapeado para um trecho de código (função, método etc.) ou para um atributo do código. Quando é mapeado para um trecho de código, uma ação de elo "start" aplicada à propriedade causa a execução do código com os valores atribuídos interpretados como parâmetros de entrada. O atributo *name* do elemento <property> deve obrigatoriamente ser utilizado para identificar o trecho de código imperativo. Quando o elemento <property> é mapeado para um atributo do código imperativo, a ação "start" deve atribuir um valor ao atributo.

Um elemento <property> também pode estar associado a um *assessment role* de um elo NCL. Nesse caso, o formatador NCL consulta o valor da propriedade para avaliar a expressão do elo. Se o elemento <property> for mapeado para um atributo de código, seu valor é retornado pelo exibidor do objeto de mídia imperativo ao formatador NCL. Se o elemento <property> for mapeado para um trecho de código, ele é chamado e o valor do resultado de sua execução é retornado pelo exibidor do objeto de mídia imperativo ao formatador NCL.

17.3.2 Modelo de Execução de um Objeto Imperativo

O ciclo de vida de um objeto de mídia imperativo é controlado pelo formatador NCL. O formatador é responsável por disparar a execução do objeto e por mediar a comunicação desse objeto com outros objetos do documento NCL, como vimos na seção anterior.

Como todo exibidor de objeto de mídia, uma vez instanciado, o exibidor do objeto imperativo executa um procedimento de iniciação. Diferentemente do que acontece para os outros exibidores de mídia, esse procedimento deve ser especificado (programado) pelo autor do objeto. O procedimento de iniciação é executado apenas uma vez, para cada instância, e cria todos os trechos de códigos e estrutura de dados que podem ser usados durante a execução do objeto. Esse procedimento é também o responsável por registrar um ou mais "tratadores de evento" para a comunicação com o formatador NCL. O leitor deve ter em mente que pelo menos o código associado à "âncora de conteúdo principal" deve ser criado no procedimento de iniciação.

Depois do procedimento de iniciação, a execução do objeto imperativo se torna orientada a eventos, nas duas direções. Isto é, qualquer ação comandada pelo formatador NCL é entregue aos tratadores de evento registrados, e qualquer modificação na máquina de estado de eventos, comandadas por instruções do código imperativo, também programadas pelo autor do objeto, é enviada como um evento ao formatador NCL (como caso particular, o final natural da execução de um trecho de código).

A Listagem 17.2 ilustra o uso de âncoras de conteúdo em um objeto imperativo com código Lua. Uma ação de *start* (disparada por um elo no sentido do ambiente imperativo) no objeto Lua aciona o procedimento de iniciação do objeto (se ele já não foi acionado e o exibidor já não está instanciado). Na Listagem 17.2, os trechos de código que podem ser usados na execução do objeto são então criados (no caso, apenas o código associado à âncora de conteúdo principal) e o tratador de eventos é registrado. O evento de "start" da execução (apresentação) é então passado ao tratador de eventos, que aciona a execução do código associado à âncora de conteúdo principal, mostrado na figura.

Ainda na Listagem 17.2, a execução do código, em certo ponto, coloca a máquina de estado da âncora de atributo *label*="final" no estado "ocorrendo", devendo essa mudança de estado ser notificada pelo exibidor do objeto de mídia imperativo ao formatador NCL (`event.post ('out', evt)`, na figura). Essa transição de estado pode, por exemplo, ser a condição de disparo de um elo. Esse exemplo simples ilustra como cabe ao programador controlar a execução das máquinas de estado associadas às suas âncoras de conteúdo (e às suas propriedades, embora estas não sejam mostradas no exemplo). Ao final, podemos observar mais uma vez como o programa controla a sinalização ao formatador NCL, agora para notificar o fim da execução do objeto imperativo (`event.post ({class='ncl', type='presentation', action='stop'})`.

```
function handler (evt)
   if (evt.class=='ncl') and
      (evt.type=='presentation') and
      (evt.action=='start') then
   ...
      evt.area='final'
       event.post ('out', evt)
   ...
       event.post ('out', {
      class='ncl', type='presentation', action='stop'
       })
   end
end
Event.register (handler)
```

Listagem 17.2 Objeto de mídia com código Lua.

Diferentemente dos procedimentos realizados para outros tipos de elementos <media>, se um exibidor de objeto de mídia com código imperativo receber uma instrução *start* para um evento associado a um elemento <area> e esse evento estiver no estado *sleeping*, ele inicia a execução do código imperativo associado ao elemento, mesmo se outra parte do código imperativo do objeto de mídia estiver em execução (pausado ou não). Contudo, se o evento associado ao elemento-alvo <area> estiver no estado *occurring* ou *paused*, a instrução *start* é ignorada pelo exibidor imperativo, que continuará controlando a execução anteriormente iniciada. Como consequência, o leitor deve estar atento ao fato de que, diferentemente do que ocorre para os outros elementos <media>, uma ação de "stop", "pause", "resume" ou "abort" de um elo deve ser ligada a uma interface do nó imperativo, que não é ignorada quando a ação é aplicada, ao contrário dos outros tipos de objeto de mídia de um documento NCL.

A instrução *start* emitida pelo formatador para um evento associado a um elemento <property> é aplicada a um objeto de mídia imperativo independentemente do fato de ele estar em execução ou não (neste último caso, seu exibidor deve já ter sido instanciado para que a ação se realize). No primeiro caso, a instrução *start* precisa identificar o objeto de mídia imperativo, um evento de atribuição monitorado e um valor a ser passado ao código imperativo associado ao evento. No segundo caso, pode também identificar o elemento <descriptor> que será usado quando da execução do objeto (análogo ao que é feito para a instrução *start* em um evento de apresentação).

Uma vez terminada a execução do objeto imperativo, sua instância não precisa ser eliminada. O leitor deve lembrar do Capítulo 7 que o atributo

playerLife pode permitir o reúso de uma instância de exibidor para a apresentações futuras.

Neste ponto, uma releitura do exemplo *O Primeiro João*, da Seção 3.12 do Capítulo 3, pode ser útil para firmar os conceitos discutidos neste capítulo.

O Apêndice H traz uma discussão das várias ações sobre eventos de apresentação e atribuição definidos por um objeto de mídia, incluindo os objetos de mídia imperativos.

Bibliografia

Soares, L.F.S.; Sant'Anna, F.F; Cerqueira, R. (2008). "Nested Context Language 3.0 Part 10 — Imperative Objects in NCL: The NCLua Scripting Language." Monografias em Ciência da Computação do Departamento de Informática, PUC-Rio, N.º 02/08. Rio de Janeiro, janeiro de 2008. ISSN 0103-9741.

Soares, L.F.S.; Moreno, M.F.; Sant'Anna (2009). "Relating Declarative and Imperative Objects through the NCL Glue Language." *Proceedings of the ACM Symposium on Document Engineering*. Munique, setembro de 2009.

Capítulo 18

Programando com Objetos NCLua

No capítulo anterior, vimos como objetos com código imperativo podem ser definidos, como eles podem se relacionar com outros objetos em um documento NCL e como exibidores (*engines*) para esses objetos se comportam. Neste capítulo, nos dedicamos aos objetos NCL com código Lua, os chamados *scripts* NCLua.[1]

Ao final deste capítulo, você será capaz de construir objetos com código imperativo em NCLua e utilizá-los em sua aplicação NCL.

[1] Este capítulo foi baseado em [Sant'Anna et al. 2009]. O uso do material foi gentilmente cedido pelo Departamento de Informática da PUC-Rio e por seus autores.

18.1 A Linguagem Lua

Desde o início de seu desenvolvimento, no início dos anos 1990, Lua foi projetada para ser usada em conjunto com outras linguagens, não sendo comum encontrar programas escritos puramente em Lua. Nesse sentido, Lua é normalmente usada para permitir que uma aplicação principal seja estendida ou adaptada através do uso de *scripts*. A aplicação principal pode ser um *videogame*, em que um script Lua é usado para definir o comportamento de um personagem; um editor de textos, que permite que os textos sejam acessados e modificados por scripts Lua; ou, de maneira mais geral, aplicações que usam Lua em scripts de configuração. Esse tipo de uso caracteriza Lua como uma linguagem de scripts no seu sentido mais puro. O próprio nome da linguagem, Lua, remete à ideia de uma *linguagem satélite*.

Lua é uma linguagem de fácil aprendizado, que combina sintaxe procedural com declarativa, com poucos comandos primitivos. Dessa característica resulta uma implementação leve e muito eficiente quando comparada com linguagens de propósitos similares. Lua também apresenta alto grau de portabilidade, podendo ser executada com todas as suas funcionalidades em diversas plataformas, tais como computadores pessoais, celulares, sistemas embarcados e consoles de *videogames*.

As características de Lua mencionadas — simplicidade, eficiência e portabilidade — além de sua licença livre, casam perfeitamente com o cenário de TV digital. Uma linguagem simples é bem-vinda onde é comum equipes formadas não só por programadores, mas também por *designers* e produtores de conteúdo. A portabilidade é importante quando o *middleware* deve ser desenvolvido para dispositivos com características conflitantes, como celulares e *set-top boxes*. A eficiência e o tamanho da linguagem requerem menos custos com *hardware*. Já a licença livre de *royalties* reduz a custo zero a adoção do interpretador por unidade produzida. Um indicador de como a linguagem Lua se adapta bem a esse tipo de cenário é a liderança de Lua como linguagem de script em *videogames*, nicho que compartilha as mesmas características descritas.

Uma apresentação mais detalhada da sintaxe e funcionalidades de Lua fica fora do escopo deste livro. Explicaremos os conceitos da linguagem necessários, conforme forem utilizados nos exemplos deste capítulo.

18.1.1 Extensões de NCLua

Para se adequar ao ambiente de TV Digital e se integrar à NCL, a linguagem Lua foi estendida com novas funcionalidades. Por exemplo, um NCLua precisa se comunicar com o documento NCL para saber quando o seu

objeto <media> correspondente é iniciado por um elo. Um NCLua também pode responder a teclas do controle remoto ou desenhar livremente dentro da região NCL a ele destinada. Essas funcionalidades são específicas da linguagem NCL e, obviamente, não fazem parte da biblioteca-padrão de Lua. O que diferencia um NCLua de um programa Lua puro é o fato de ser controlado pelo documento NCL no qual está inserido e utilizar as extensões descritas a seguir.

Além da biblioteca-padrão de Lua, os seguintes módulos estão disponíveis para scripts NCLua:

- módulo `event`: permite que objetos NCLua se comuniquem com o documento NCL e outras entidades externas (tais como controle remoto e canal de interatividade);
- módulo `canvas`: oferece funcionalidades para desenhar objetos gráficos na região do NCLua;
- módulo `settings`: oferece acesso às variáveis definidas no objeto *settings* do documento NCL (objeto do tipo "application/x-ncl-settings");
- módulo `persistent`: exporta uma tabela com variáveis persistentes entre execuções de objetos imperativos.

O Capítulo 10 da ABNT, NBR 15606-2 [2011] lista detalhadamente todas as funções suportadas por cada módulo mencionado.

As seguintes funções da biblioteca-padrão de Lua são dependentes de plataforma e por isso não estão disponíveis para scripts NCLua:

- no módulo `package`: a função `loadlib;`
- no módulo `io`[2]: todas as funções;
- no módulo `os`: as funções `clock, execute, exit, getenv, remove, rename, tmpname` e `setlocale;`
- no módulo `debug`: todas as funções.

18.2 Programação Orientada a Eventos

No Capítulo 17 discutimos o modelo de execução e comunicação de objetos imperativos embutidos em documentos NCL. No caso de objetos NCLua, os

[2] O modulo `io assíncrono` está disponível apenas para aplicações residentes.

mecanismos de integração com o documento NCL se fazem através do paradigma de programação orientada a eventos.

Não somente a comunicação com o documento NCL, mas também qualquer interação com entidades externas à aplicação, como o canal de interatividade, controle remoto e temporizadores, se faz pela difusão e recepção de eventos. O módulo `event` de NCLua, usado para esse fim, é a mais importante extensão à linguagem Lua, e seu entendimento é essencial para desenvolver qualquer aplicação que utilize objetos NCLua.

A Figura 18.1 exibe ao centro um NCLua, envolto por diversas entidades com as quais ele pode interagir. Para se comunicar com um NCLua, uma entidade externa deve inserir um evento na fila indicada na figura, que é então redirecionado às funções tratadoras de eventos, definidas pelo programador do script NCLua. Enquanto cada tratador, um de cada vez, processa um evento, nenhum outro evento da fila é tratado. Sendo assim, fica a cargo do programador escrever tratadores que executem o mais rápido possível, de maneira a evitar o congestionamento da fila. Um NCLua também pode se comunicar com entidades externas postando eventos dentro de seus tratadores, como mostram as setas saindo do NCLua.[3]

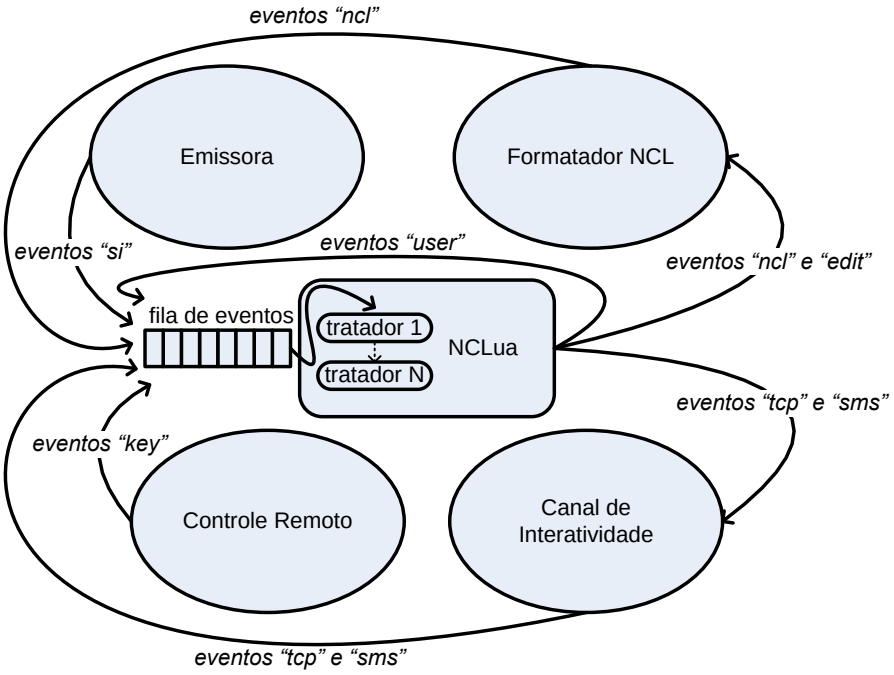


Figura 18.1 Paradigma de programação orientado a eventos.

[3] A fila de eventos é controlada pelo sistema e não é visível a um NCLua.

A principal vantagem do uso do paradigma de eventos é a sua característica de *acoplamento fraco* entre as entidades do sistema. Como se pode observar pela figura, a remoção ou adição de uma entidade não acarreta mudanças internas em nenhuma outra entidade. Essa característica vai ao encontro dos requisitos de mínima intrusão levantados no Capítulo 17.

Para ser informado quando eventos externos são recebidos, um NCLua deve registrar pelo menos uma função de tratamento em seu corpo através de uma chamada à função `event.register` (o nome da função a ser registrada pode ser qualquer um). O código de um NCLua segue uma estrutura comum a todos os scripts, como na Listagem 18.1.

```
...                                  -- código de iniciação
function tratador (evt)
      ...                      -- código de um tratador
end
event.register(tratador) -- registro de pelo menos um tratador
```

Listagem 18.1 Paradigma de programação orientado a eventos

O código de iniciação, a definição do tratador e seu registro são executados antes que qualquer evento gerado pelo documento NCL (ou qualquer entidade externa ao script) seja sinalizado aos tratadores NCLua, inclusive o de início de apresentação do objeto. Após esse processo de carga do script, efetuado pelo sistema, apenas o código do tratador é chamado, toda vez que ocorre um evento externo. O código de iniciação pode ser utilizado para criar objetos e funções auxiliares que serão usadas pelos tratadores.

Eventos são representados por tabelas Lua com chaves e valores descrevendo seus atributos. Como exemplo, a função tratadora pode receber um evento (parâmetro `evt` do tratador, na Listagem 18.1), indicando que a tecla vermelha do controle remoto foi pressionada pelo telespectador.

```
evt = {
  class  = 'key',
  type   = 'press',
  key    = 'RED',
}
```

Listagem 18.2 Representação de evento em NCLua.

A função `event.post` é utilizada para que um NCLua poste eventos e possa, por exemplo, enviar dados pelo canal de interatividade ou sinalizar seu estado ao documento NCL. No exemplo a seguir (Listagem 18.3), o NCLua sinaliza ao documento o seu fim natural.

```
event.post {
    class  = 'ncl',
    type   = 'presentation',
    action = 'stop',
}
```

Listagem 18.3 Exemplo de evento postado por um NCLua para sinalizar ao documento NCL o seu fim natural.

Como um NCLua (por meio de seus tratadores) deve executar rapidamente, a função de envio de eventos nunca aguarda o retorno de um valor. Caso o destino necessite retornar uma informação ao NCLua, o fará através do envio de um novo evento.

18.2.1 Classes de Eventos

O campo `class` de uma tabela representando um evento é obrigatório e tem a finalidade de separar os eventos em categorias. A classe identifica não somente a origem de eventos passados aos tratadores, mas também o seu destino, caso o evento seja gerado e postado por um script NCLua.

As seguintes classes de eventos estão definidas:

- Classe `ncl`: usada na comunicação entre um NCLua e o documento NCL que contém o objeto de mídia.
- Classe `key`: representa o pressionamento de teclas do controle remoto pelo usuário.
- Classe `tcp`: permite acesso ao canal de interatividade por meio do protocolo *tcp*.
- Classe `sms`: usada para envio e recebimento de mensagens SMS.
- Classe `edit`: permite que os comandos de edição ao vivo apresentados no Capítulo 16 sejam disparados a partir de scripts NCLua.
- Classe `si`: provê acesso a um conjunto de informações multiplexadas em um fluxo de transporte e transmitidas periodicamente por difusão.
- Classe `user`: através dessa classe, as aplicações podem estender sua funcionalidade criando seus próprios eventos.

Observe, pela Figura 18.1, que há eventos apenas de entrada, apenas de saída ou que são usados em ambos os sentidos.

O modelo orientado a eventos de NCLua foi projetado para suportar outras entidades externas estendendo o modelo básico, bastando para isso definir novas classes de eventos.

18.3 Interagindo com o Documento NCL

Assim como qualquer objeto de mídia, um NCLua interage com o documento NCL através de elos.

Em elos que acionam um NCLua, a condição satisfeita faz com que o NCLua receba um evento da classe `ncl` descrevendo a ação a ser tomada. No trecho da Listagem 18.4, quando o elo for disparado com o início de "videoId", o tratador de eventos de NCLua receberá o evento no código do objeto NCLua.

```
<link xconnector="onBeginStart">
    <bind role="onBegin" component="videoId"/>
    <bind role="start"   component="luaId"/>
</link>
```

arquivo NCL que contém o objeto NCLua

arquivo NCLua

```
-- Evento recebido pelo tratador do NCLua no
-- disparo do elo:
evt = {
    class  = 'ncl',
    type   = 'presentation',
    action = 'start',
}
```

Listagem 18.4 Exemplo de códigos NCL e NCLua que tratam um evento de apresentação de um objeto NCL.

Já em elos cuja condição depende de um NCLua, a ação do elo será disparada quando o NCLua sinalizar o evento que casa com a condição esperada. No trecho da Listagem 18.5, quando o NCLua sinalizar o evento indicado, o elo será disparado e a imagem exibida.

```
<link xconnector="onBeginStart">
    <bind role="onEnd" component="luaId"/>
    <bind role="start" component="imagemId"/>
</link>
```
arquivo NCL que contém o objeto NCLua

arquivo NCLua

```
-- O elo acima será disparado quando o evento a seguir
-- for postado pelo NCLua 'luaId':
event.post {
    class  = 'ncl',
    type   = 'presentation',
    action = 'stop',
}
```

Listagem 18.5 Elo disparado pelo código do objeto NCLua.

O tipo "presentation" indica que os eventos se referem à apresentação do NCLua. Como nenhuma âncora foi especificada, é assumida a âncora de conteúdo total, conforme descrito no Capítulo 17.

Como os dois exemplos indicam, a interação de um NCLua com o documento NCL se dá sempre através da classe de eventos `ncl`, seja para receber instruções do formatador, seja para notificar o estado de suas âncoras.

Exemplo 18.11 — Ciclo de Vida de Objetos NCLua

Este exemplo apresenta uma aplicação com o fim de ilustrar o modelo de execução de objetos imperativos NCLua em documentos NCL.

Ao iniciar a aplicação, três objetos NCLua são iniciados, cada um com um comportamento interno diferente:

- O primeiro NCLua executa indefinidamente.
- O segundo NCLua termina a si próprio no mesmo momento em que é iniciado.
- O terceiro NCLua termina a si próprio três segundos após ser iniciado.

Associados a cada NCLua há um botão verde e um vermelho, indicando se o NCLua está ocorrendo ou terminado, respectivamente. As visões temporal e

espacial da aplicação, mostradas na Figura 18.2 tem o comportamento descrito.

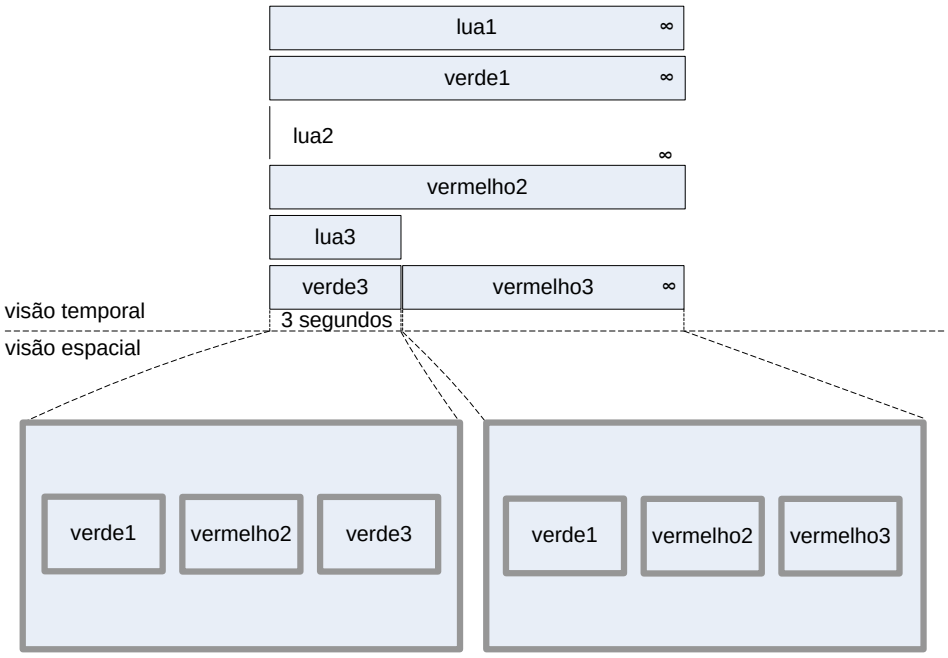


Figura 18.2 Visões temporal e espacial do Exemplo 18.1.

A Figura 18.3 exibe a visão estrutural do documento NCL. O primeiro NCLua é ligado aos outros por meio de um elo "onBeginStart". Como o primeiro NCLua está ligado à porta de entrada do documento, os três objetos iniciam juntamente com a aplicação. Cada NCLua se liga por meio de um elo "onBeginStart" com seu respectivo botão verde, para que eles sejam exibidos com o início de cada NCLua. Para esconder seu respectivo botão verde e exibir o vermelho, cada NCLua também utiliza um elo "onEndStopStart" com seus botões, conforme mostra a figura.

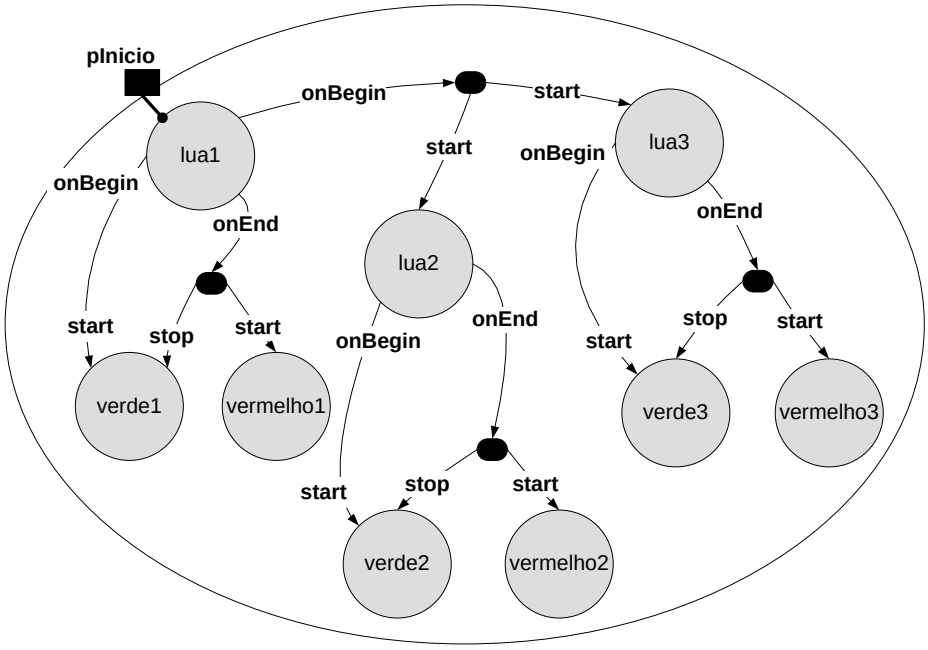


Figura 18.3 Visão estrutural do Exemplo 18..

O documento NCL é responsável por definir e iniciar os três objetos NCLua, assim como pela *cola lógica* entre cada NCLua e seus botões correspondentes, conforme definido na Listagem 18.6.

```
<body>
    <!-- INICIA primeiro o NCLua -->
    <port id="pInicio" component="lua1"/>

    <!-- MIDIAS -->
    <media id="lua1"  src="1.lua"/>
    <media id="lua2"  src="2.lua"/>
    <media id="lua3"  src="3.lua"/>
    <media id="verde1"    src="verde1.png" descriptor="ds1"/>
    <media id="vermelho1" src="verm1.png"  descriptor="ds1"/>
    <media id="verde2"    src="verde2.png" descriptor="ds2"/>
    <media id="vermelho2" src="verm2.png"  descriptor="ds2"/>
    <media id="verde3"    src="verde3.png" descriptor="ds3"/>
    <media id="vermelho3" src="verm3.png"  descriptor="ds3"/>

    <!-- BEGIN lua1 -> START lua2/lua3 -->
    <link xconnector="onBeginStart">
       <bind role="onBegin" component="lua1"/>
```

```
      <bind role="start"   component="lua2"/>
      <bind role="start"   component="lua3"/>
   </link>

   <!-- BEGIN luaN -> START greenN -->
   <link xconnector="onBeginStart">
      <bind role="onBegin" component="lua1"/>
      <bind role="start" component="verde1"/>
   </link>
   ...  <!-- Código idêntico para os outros NCLua -->

   <!-- END luaN -> STOP greenN | START redN -->
   <link xconnector="onEndStopStart">
      <bind role="onEnd" component="lua1"/>
      <bind role="stop"  component="verde1"/>
      <bind role="start" component="vermelho1"/>
   </link>
   ...  <!-- Código idêntico para os outros NCLua -->
</body>
```

Listagem 18.6 Código NCL do Exemplo 18.1.

Note como neste exemplo o NCL não aciona o término de nenhum objeto NCLua. Esse papel ficou a cargo de cada script NCLua, como mostrado a seguir.

- O primeiro NCLua é um script vazio (sem nenhuma linha de código). Em particular, como não possui um tratador de eventos, nunca sinaliza o seu término para o documento NCL. O efeito visual é a exibição permanente do primeiro botão verde.

```
-- 1.lua
-- vazio
```

Listagem 18.7 Código do arquivo 1.lua do Exemplo 18.1.

- O segundo NCLua registra um tratador de eventos que gera seu fim natural ao receber um "start" do documento NCL. Visualmente, assim que o segundo botão verde é exibido, é exibido instantaneamente o botão vermelho correspondente (o botão verde pode nem ser visto).

```
-- 2.lua:
function tratador (evt)
  if (evt.class == 'ncl') and (evt.type ==
'presentation')
     and (evt.action == 'start') then
    event.post {
        class  = 'ncl',
        type   = 'presentation',
        action = 'stop'
    }
  end
end
event.register(tratador)
```

Listagem 18.8 Código do arquivo 2.lua do Exemplo 18.1.

Tão logo o evento indicando seu início é recebido, o NCLua posta um evento para sinalizar o seu fim natural.

- O terceiro NCLua registra um tratador de eventos que cria um temporizador de três segundos que, por sua vez, gera seu fim natural ao expirar. Como efeito visual, temos a exibição do terceiro botão verde e, após três segundos, a mudança para vermelho.

```
-- 3.lua:
function tratador (evt)
  if (evt.class == 'ncl') and
     (evt.type == 'presentation') and
     (evt.action == 'start') then
    event.timer(3000,
        function()
            event.post {
                class  = 'ncl',
                type   = 'presentation',
                action = 'stop'
            }
        end)
  end
end
event.register(tratador)
```

Listagem 18.9 Código do arquivo 3.lua do Exemplo 18.1.

O temporizador de três segundos (3.000 milissegundos) é criado assim que o evento de início é recebido, passando a função que deve ser executada quando o temporizador expira. Essa função posta um evento idêntico ao do segundo NCLua para sinalizar seu fim natural.

Enquanto executa indefinidamente, o primeiro NCLua não consome recursos e pode responder a eventos (apesar de não o fazer por não possuir um tratador para tal fim). O mesmo ocorre com o terceiro NCLua enquanto aguarda os três segundos para terminar.

18.3.1 Eventos em Âncoras de Conteúdo e Propriedades

No exemplo anterior, apenas a âncora de conteúdo principal de cada NCLua foi acionada. No entanto, âncoras de conteúdo específicas e propriedades também podem ser relacionadas entre o documento NCL e o objeto NCLua.

Dados os tipos de eventos gerados pelo formatador NCL, o campo `type` de um evento da classe `ncl` pode assumir os valores "presentation" ou "attribution", conforme o atributo *eventType* dos conectores NCL (ver Seção 10.5). Eventos do tipo "selection" são tratados pela classe `key`.

18.3.1.1 Eventos do Tipo "presentation"

Eventos de apresentação estão associados à apresentação de âncoras de conteúdo, sendo identificadas pelo campo `label` do evento. O campo `action` indica a ação a ser realizada ou sinalizada pelo NCLua, dependendo de ele estar recebendo ou gerando o evento.

Um evento de apresentação possui a seguinte estrutura:

- `class: 'ncl'`
- `type: 'presentation'`
- `label: [string]` — rótulo da âncora associada ao evento
- `action: [string]` — pode assumir os seguintes valores: `'start'`, `'stop'`, `'abort'`, `'pause'` e `'resume'`

18.3.1.2 Eventos do Tipo "attribution"

Eventos de atribuição estão associados às propriedades do objeto NCLua, sendo identificados pelo campo `name`.

O campo `value` é preenchido com o valor a ser atribuído à propriedade e é sempre uma string, uma vez que vem de um atributo NCL. A ação de "start" em um evento de atribuição corresponde ao papel "set" (ou "start") de um elo NCL.

Um evento de atribuição possui a seguinte estrutura:

- `class: 'ncl'`
- `type: 'attribution'`
- `name: [string]` — nome da propriedade associada ao evento
- `action: [string]` — pode assumir os seguintes valores: `'start'`, `'stop'`, `'abort'`, `'pause'` e `'resume'`
- `value: [string]` — novo valor a ser atribuído à propriedade

O campo `action` de um evento `ncl` (seja ele de apresentação ou atribuição) pode assumir os valores correspondentes aos seus tipos, como mostrado nas Tabelas 10.6 e 10.7. No entanto, o nome das transições na Tabela 10.6 é usado sem o "s" final ("starts" vira "start"), de maneira a unificar a sintaxe para eventos recebidos ou sinalizados pelo NCLua.

Exemplo 18.2 — Contador de Cliques

Vamos supor uma aplicação NCL que exiba um botão *Clique aqui* em quatro momentos diferentes. Se o usuário selecioná-lo com o controle remoto por pelo menos três vezes, ao final da apresentação será exibido o botão *Você ganhou*, caso contrário será exibido o botão *Você Perdeu*. A Figura 18.4 mostra as visões temporal e espacial da aplicação.

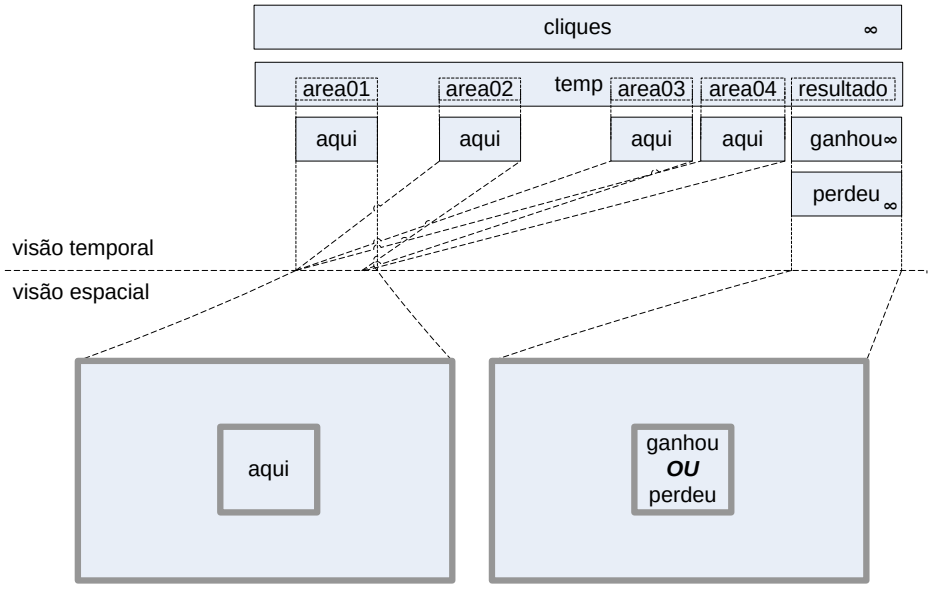


Figura 18.4 Visões temporal e espacial do Exemplo 18.2.

Não é possível fazer a contagem de cliques puramente em NCL de forma simples, uma vez que não há suporte a expressões aritméticas na linguagem. Neste exemplo, usaremos um NCLua para contar e armazenar o número de

cliques em uma propriedade, que será consultada ao final para determinar o resultado.

O documento NCL mostrado a seguir define um temporizador (“temp”) com quatro âncoras temporais (“area01” até “area04”) durante as quais o botão *Clique aqui* é exibido. O temporizador também define uma âncora temporal para exibir o resultado após o botão ser exibido por quatro vezes. Além do temporizador e os botões, o documento define um NCLua responsável pela contagem dos cliques e exporta a propriedade “contador” para manter esse valor. A Figura 18.5 mostra a visão estrutural da aplicação.

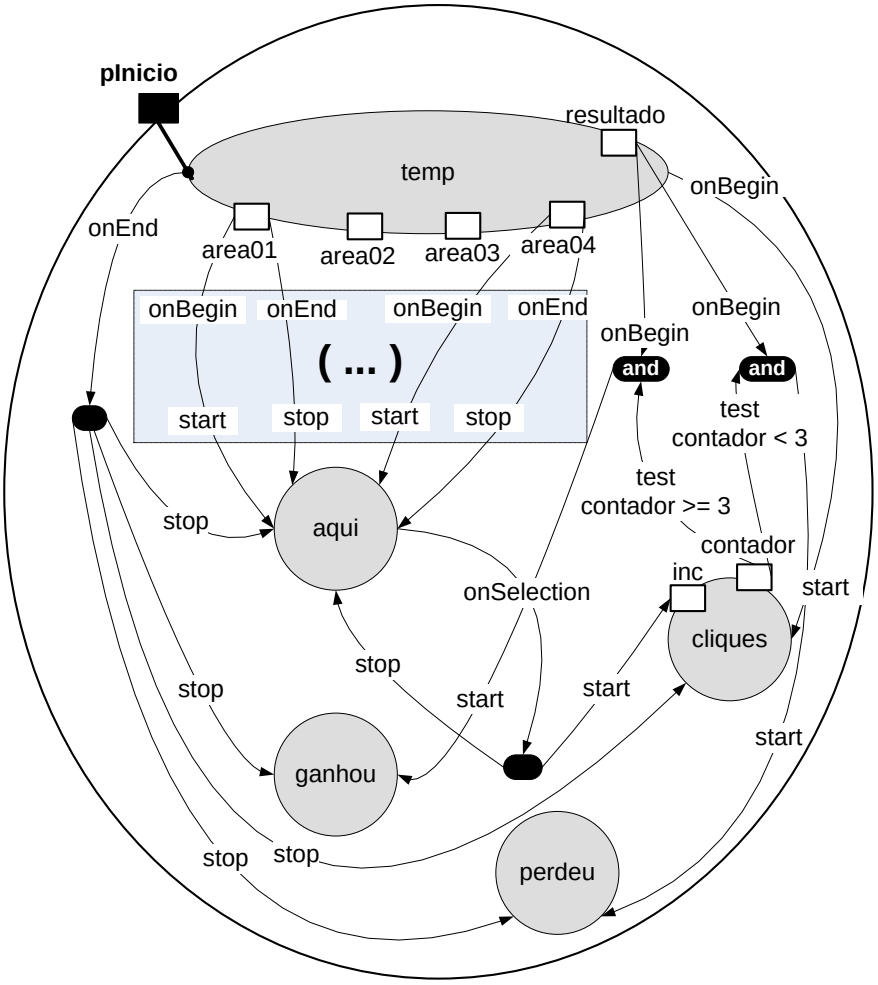


Figura 18.5 Visão estrutural do Exemplo 18.2.

A porta de entrada da aplicação é o temporizador, que também dispara o NCLua por meio de um elo “onBeginStart”. Cada âncora de exibição do botão *Clique aqui* possui um elo “onBeginStart” e “onEndStop” para o

mesmo. Toda vez que o botão é selecionado pelo usuário (o que só pode acontecer enquanto está sendo exibido), a âncora "inc" do NCLua é iniciada e o botão é escondido, conforme o elo "onSelectionStopStart" saindo do próprio botão. O tratamento dado à âncora "inc" e à propriedade "contador" é especificado no código do NCLua. Ao final da âncora "resultado" do temporizador, dois elos testam se o valor da propriedade "contador" é maior ou menor que três cliques. Dependendo do resultado, a imagem *Você ganhou* ou *Você perdeu* é exibida.

O documento NCL para a aplicação é exibido na Listagem 18.1.

```
<body>
  <!-- INÍCIO PELO TEMPORIZADOR -->
  <port id="pInicio" component="temp"/>

  <!-- TEMPORIZADOR -->
  <media id="temp">
    <!-- ÂNCORAS PARA EXIBIR O BOTÃO "CLIQUE AQUI" -->
    <area id="area01" begin="3s"  end="6s"/>
    <area id="area02" begin="10s" end="13s"/>
    <area id="area03" begin="17s" end="20s"/>
    <area id="area04" begin="24s" end="27s"/>
    <!-- ÂNCORA PARA EXIBIR O RESULTADO -->
    <area id="resultado" begin="35s"/>
  </media>

  <!-- NCLua -->
  <media id="cliques" src="cliques.lua">
    <area id=”incrementa” label="inc"/>
    <property name="contador"/>
  </media>

  <!-- "CLIQUE AQUI"/"VOCÊ GANHOU"/"VOCÊ PERDEU" -->
  <media id="aqui"   descriptor="dsBotao" src="aqui.jpg"/>
  <media id="ganhou" descriptor="dsBotao" src="ganhou.jpg"/>
  <media id="perdeu" descriptor="dsBotao" src="perdeu.jpg"/>

  <!-- TEMPORIZADOR -> NCLua -->
  <link xconnector="onBeginStart">
    <bind role="onBegin" component="temp"/>
    <bind role="start"   component="cliques"/>
  </link>

  <!-- AREA[N] -> BOTÃO "CLIQUE AQUI" -->
  <link xconnector="onBeginStart">
    <bind role="onBegin" component="temp"
```

```
                                          interface="area01"/>
    <bind role="start" component="aqui"/>
  </link>
  ... <!-- Código idêntico para as outras âncoras -->

  <link xconnector="onEndStop">
    <bind role="onEnd" component="temp" interface="area01"/>
    <bind role="stop" component="aqui"/>
  </link>
  ... <!-- Código idêntico para as outras âncoras -->

  <!-- SELECT "CLIQUE AQUI" -> CLIQUES.INCREMENTA -->
  <link xconnector="onSelectionStopStart">
    <bind role="onSelection" component="aqui"/>
    <bind role="stop" component="aqui"/>
    <bind role="start" component="cliques"
                                    interface="incrementa"/>
  </link>

  <!-- TESTE DO RESULTADO -->
  <link xconnector="onCondGteBeginStart">
  <linkParam name="var" value="3"/>
    <bind role="onBegin  component="temp"
                                    interface="resultado"/>
    <bind role="attNodeTest" component="cliques"
                                     interface="contador"/>
    <bind role="start" component="ganhou"/>
  </link>
  <link xconnector="onCondLtBeginStart">
    <linkParam name="var" value="3"/>
    <bind role="onBegin" component="temp"
                                   interface="resultado"/>
    <bind role="attNodeTest" component="cliques"
                                    interface="contador"/>
    <bind role="start" component="perdeu"/>
  </link>
</body>
```

Listagem 18.10 Código NCL parcial do Exemplo 18.2.

O código do NCLua deve lidar em sua função tratadora de eventos com as duas interfaces com o documento NCL — a âncora "inc" e a propriedade "contador". A Listagem 18.11 apresenta o código do script.

```
local contador = 0
function tratador (evt)
   contador = contador + 1
   local evtContador = {
      class = 'ncl',
      type  = 'attribution',
      name  = 'contador',
      value = contador,
   }
   evtContador.action = 'start'
   event.post(evtContador)
   evtContador.action = 'stop'
   event.post(evtContador)
   event.post {
      class  = 'ncl',
      type   = 'presentation',
      label  = 'inc',
      action = 'stop',
   }
end
event.register(tratador,'ncl','presentation','inc','start')
```

Listagem 18.11 Código NCLua do Exemplo 18.2.

O script começa declarando uma variável para guardar a contagem de cliques. A variável não possui relação direta com a propriedade do NCLua, de mesmo nome, definida no documento NCL. O manuseio da propriedade é feito através da postagem de eventos, conforme apresentado a seguir.

A função "*tratador*" é então definida e registrada (na última linha do código). Os parâmetros extras na chamada à função *register* servem para filtrar apenas os eventos desejados; no caso, eventos *ncl* de início de apresentação da âncora "inc". Quando o botão é selecionado, iniciando a âncora "inc", a função *tratador* é executada. A função incrementa o contador interno e posta um evento de início de atribuição (*action*='start'*)*, para indicar a mudança na propriedade "contador".

IMPORTANTE: Note que é necessário sinalizar também o fim da atribuição, fazendo com que a máquina de estados da propriedade "contador" volte ao estado "sleeping" e futuras atribuições surtam efeito. Pelo mesmo motivo, é necessário sinalizar o término da âncora "inc", postando o evento de "stop" da apresentação, no fim da função.

O NCLua não tem como saber o momento exato em que a propriedade "contador" será consultada pelo documento NCL — por isso, sempre que incrementa o valor, também atualiza a propriedade.

Neste ponto, uma releitura do exemplo *O Primeiro João*, da Seção 3.12, pode ser útil para firmar os conceitos discutidos até aqui neste capítulo.

18.4 Desenhando na Região do Objeto

Quando um NCLua é carregado, o formatador NCL cria um objeto gráfico para representar a região associada à mídia NCLua no documento. Esse objeto gráfico é pré-carregado na variável global "*canvas*" do script, e é através dela que todas as operações gráficas são efetuadas. Caso o objeto de mídia NCLua não esteja associado a nenhuma região, o valor do "*canvas*" será igual a "*nil*".

Como exemplo (Listagem 18.12), caso a região a seguir esteja associada ao objeto NCLua, a variável "*canvas*" do script irá representá-la, com tamanho 300×100 e posicionada em (20,200).

```
<region id="luaRegion" width="300" height="100"
                       top="200" left="20"/>
```

Listagem 18.12 Código NCL de uma região a ser associada com um objeto NCLua

Existem diversas operações gráficas suportadas pelo módulo de canvas, tais como desenho de linhas, textos e imagens. A lista completa de operações pode ser consultada no Capítulo 10 da NBR 15606-2 [ABNT, 2011]. O exemplo a seguir cria um novo canvas representando a imagem passada, desenhando-a centralizada na região e acompanhada de uma legenda. A Figura 18.6 exibe o resultado visual da execução do script da Listagem 18.3.

Figura 18.6 Resultado da execução do script da Listagem 18.13.

```
local regLarg, regAlt = canvas:attrSize()

local img = canvas:new('ginga.png')
local imgLarg, imgAlt = img:attrSize()
local imgX = (regLarg - imgLarg) / 2
local imgY = (regAlt - imgAlt) / 2
canvas:compose(imgX, imgY, img)

local txt = 'TV Digital se faz com Ginga'
local txtLarg, txtAlt = canvas:measureText(txt)
local txtX = (regLarg - txtLarg) / 2
local txtY = imgY + imgAlt + 2
canvas:attrColor('white')
canvas:drawText(txtX, txtY, txt)


canvas:flush()
```

Listagem 18.13 Script ilustrando o uso do canvas.

O script da Listagem 18.13 começa guardando as dimensões da região inteira do NCLua nas variáveis `regLarg` e `regAlt` com a chamada à função `canvas:attrSize()`, que retorna a largura e a altura do canvas. Em seguida, o método `canvas:new()` recebe uma imagem e retorna um novo canvas, "*img*", que a representa. As dimensões da imagem são então recuperadas e utilizadas para calcular a posição centralizada na qual a imagem é desenhada na região (`imgX` e `imgY`). A chamada `canvas:compose(imgX,imgY,img)` sobrepõe a imagem do *Ginga* à região do NCLua na posição informada.[4]

Para desenhar a legenda do texto, primeiramente é medido o tamanho que o texto ocupa no canvas, com a chamada à `canvas:measureText()`. A posição horizontal centralizada do texto é calculada de maneira similar à da imagem. Já sua posição vertical é calculada para um pouco abaixo da imagem. Por fim, a chamada a `canvas:attrColor('white')` altera o atributo de cor para branco, em futuras operações sobre o canvas, para então desenhar o texto na posição calculada. Apenas com a chamada à função `canvas:flush()` as operações gráficas descritas são de fato efetuadas.

Como se pode deduzir ao observar o exemplo, um objeto `canvas` guarda em seu estado diversos atributos sob os quais as primitivas gráficas devem operar. Os atributos são acessados através de métodos de prefixo `attr`

[4] As coordenadas passadas para todos os métodos gráficos são sempre relativas ao ponto mais à esquerda e no topo do canvas (0,0), como é comum entre sistemas gráficos.

acompanhado do nome do atributo (por exemplo, `attrColor`), e servem tanto para leitura, quando chamados sem parâmetros (como é o caso da chamada a `attrSize()`), como para escrita, quando chamados com os novos parâmetros (como é o caso da chamada a `attrColor('white')`).

Como mencionamos, a referência completa do módulo *canvas*, com todos os atributos e operações gráficas suportados, pode ser encontrada no Capítulo 10 da NBR 15606-2 [ABNT, 2011].

Exemplo 18.3 — Gráficos e Controle Remoto

Este exemplo apresenta um jogo bastante simples, no qual são exibidos os logotipos das linguagens Lua e NCL. O usuário deve mover com o controle remoto o logotipo de Lua até o de NCL, quando é exibida uma imagem indicando o fim do jogo. O exemplo explora os eventos da classe `key` e o pacote gráfico `canvas`. A Figura 18.7 mostra as visões temporal e espacial da aplicação.

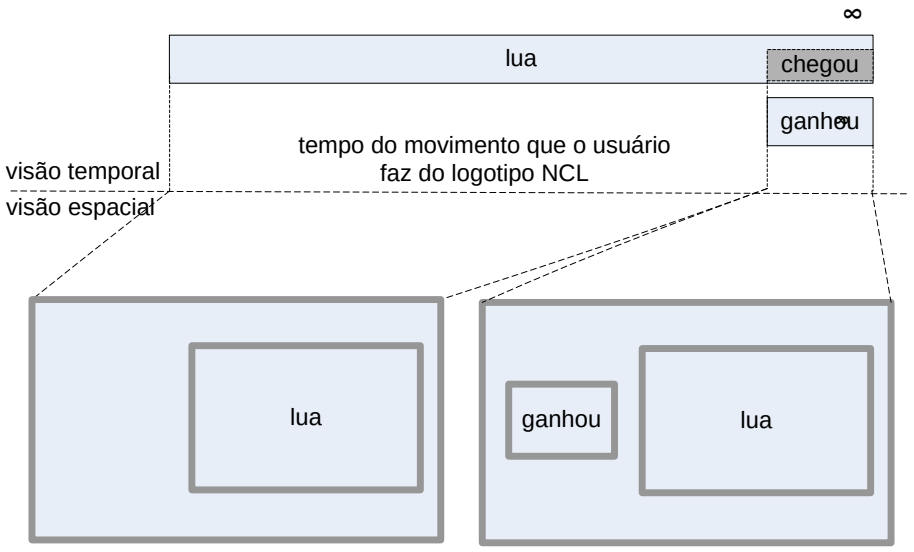


Figura 18.7 Visões temporal e espacial do Exemplo 18.3.

O documento NCL possui como mídias apenas o objeto NCLua ("*lua*") com o jogo e a imagem "Você ganhou" ("*ganhou*"), que é exibida quando o logotipo de Lua encontra o de NCL (acontecimento representado pela âncora "chegou" do NCLua). O jogo inicia assim que o documento é carregado. A Figura 18.18 mostra a visão estrutural da aplicação.

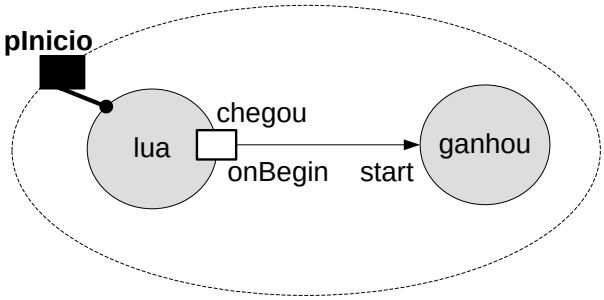


Figura 18.8 Visão estrutural do Exemplo 18.3.

O código NCL é bem simples, como definido na Listagem 18.14. Note que a aplicação NCL propositalmente não tem fim, para manter sua simplicidade.

```
<body>
  <port id="pInicio" component="lua"/>
  <media id="lua" src="jogo.lua" descriptor="dsLua">
    <area id="chegou" label=”chegou”/>
  </media>
     <media id="ganhou" src="ganhou.jpg"
                                      descriptor="dsGanhou"/>

  <link xconnector="onBeginStart">
    <bind role="onBegin" component="lua" interface="ganhou"/>
    <bind role="start" component="ganhou"/>
  </link>
</body>
```

Listagem 18.14 Código NCL do Exemplo 18.3.

Nos exemplos anteriores, o código da aplicação estava concentrado no documento NCL; agora praticamente todo o código está dentro do NCLua. As primeiras linhas do código NCLua criam duas tabelas para representar os logotipos de Lua e NCL (Listagem 18.15).

```
local img = canvas:new('lua.png')
local larg, alt = img:attrSize()
local logoLua = { canvas=img, x=10,y=10, larg=larg,alt=alt }

local img = canvas:new('ncl.png')
local larg, alt = img:attrSize()
local logoNcl = { canvas=img, x=10,y=10, larg=larg,alt=alt }
```

Listagem 18.15 Primeira parte do código NCLua do Exemplo 18.3.

Conforme a Listagem 18.15, o logotipo de Lua é representado pela tabela *logoLua*, guardando o *canvas* com a imagem *lua.png*, as coordenadas *x* e *y* onde ela deve ser desenhada, e sua largura e altura. A tabela *logoNcl* guarda os mesmos atributos (mas para a imagem *ncl.png*) para representar o logotipo de NCL.

A Listagem 18.16 define a função de redesenho da tela, chamada durante a execução do NCLua toda vez que o usuário movimenta o logotipo de Lua. Sempre que chamada, a função `redraw` desenha um retângulo preto ocupando a região inteira (para limpá-la) e em seguida compõe os canvas das duas tabelas `logoNcl` e `logoLua`, sobre o canvas principal, em suas posições correntes.

```
function redraw ()
    canvas:attrColor('black')
    canvas:drawRect('fill', 0,0, canvas:attrSize())
    canvas:compose(logoNcl.x, logoNcl.y, logoNcl.canvas)
    canvas:compose(logoLua.x, logoLua.y, logoLua.canvas)
    canvas:flush()
end
```

Listagem 18.16 Segunda parte do código NCLua do Exemplo 18.3.

Por fim, a Listagem 18.17 define a função tratadora de eventos, responsável por tratar as teclas do controle remoto e por sinalizar, ao documento NCL, o momento em que os logotipos se sobrepõem.

```
function tratador (evt)
  -- apenas eventos de tecla interessam
      if evt.class == 'key' and evt.type == 'press'
      then
        -- apenas as setas que movem o logotipo Lua
interessam
        if evt.key == 'CURSOR_UP' then
          logoLua.y = logoLua.y - 10
        elseif evt.key == 'CURSOR_DOWN' then
          logoLua.y = logoLua.y + 10
        elseif evt.key == 'CURSOR_LEFT' then
          logoLua.x = logoLua.x - 10
        elseif evt.key == 'CURSOR_RIGHT' then
          logoLua.x = logoLua.x + 10
        end
    -- testa se os logotipos estão sobrepostos
    if sobrepondo(logoLua, logoNcl) then
          -- sinaliza que a ancora "chegou" esta ocorrendo
```

```
          event.post {
        class  = 'ncl',
        type   = 'presentation',
        label  = 'chegou',
        action = 'start',
      }
    end
    redraw()  -- redesenha a tela inteira
      end
end
event.register(tratador)
```

Listagem 18.17 Terceira parte do código NCLua do Exemplo 18.3.

Na Listagem 18.17, o script inicia testando se alguma das teclas direcionais foi pressionada (campo `key` do evento de tecla), alterando a posição do logotipo de Lua de acordo com a tecla. Caso os logotipos estejam se sobrepondo, é postado o evento para sinalizar que a âncora "chegou" iniciou. A função "*sobrepondo*" foi omitida sem prejuízo do entendimento. Por fim, a tela é redesenhada, qualquer que seja a tecla que tenha sido pressionada.

Em um evento da classe *key*, o valor da tecla é uma *string*, guardada no campo *key*. O campo *type* pode ser *'press'* ou *'release'*, dependendo de a tecla ter sido pressionada ou solta.

18.4.1 Programando com Animações

Imaginemos agora que o logotipo de Lua fosse movimentado com o passar do tempo, como em uma animação, em vez de ser guiado pelo controle remoto. Ao receber a instrução de "start", o trecho hipotético da Listagem 18.18 atualiza a posição do logotipo a cada 30 milissegundos, até que sua posição chegue a 100.

```
function tratador (evt)
   if evt.action == 'start' then
      while logoLua.x < 100 do
         logoLua.x = logoLua.x + 5
         redraw()
         sleep(30)
      end
   end
end
event.register(tratador, 'ncl', 'presentation')
```

Listagem 18.18 Exemplo (ruim) de animação em NCLua.

O problema com esse código é que a chamada à função `sleep` bloqueia o tratador, não permitindo que outros eventos sejam processados, inviabilizando essa proposta.

Uma possível solução seria criar uma função de `update` a ser chamada a cada 30 milissegundos por um temporizador (Listagem 18.19).

```
function update ()
   logoLua.x = logoLua.x + 5
   redraw()
   if logoLua.x < 100 then
      event.timer(update, 30)
   end
end
function tratador (evt)
   if evt.action == 'start' then
      update()
   end
end
event.register(tratador, 'ncl', 'presentation')
```

Listagem 18.19 Exemplo de animação em NCLua que utiliza um temporizador.

Pela listagem anterior, ao ser chamado, o tratador ativa a função `update`, que atualiza a posição do logotipo de Lua, chama a função de redesenho e, caso o logotipo ainda não tenha alcançado a posição 100, se programa para ser chamado novamente após 30 milissegundos de espera.

Essa solução não é tão legível quanto um *loop* localizado, mas mantém o NCLua reativo a possíveis eventos que possam chegar durante os 30 milissegundos.

18.4.2 Corrotinas de Lua

O mecanismo de corrotinas de Lua simplifica muito a programação de aplicações em que muitos objetos interagem e devem estar permanentemente sincronizados.

Apesar de serem comumente comparadas a *threads*, as corrotinas são, na verdade, muito mais próximas do conceito comum de função (ou rotina). Da mesma forma que as funções, chamadas a corrotinas são síncronas, isto é, o código que chama uma corrotina sempre aguarda o seu retorno. No entanto, uma corrotina pode explicitamente suspender sua própria execução, *preservando o seu estado corrente* (isto é, variáveis locais, contador de instruções), antes do seu término completo. Ao se suspender, uma corrotina retorna imediatamente o controle a quem a chamou. Nesse caso, a corrotina

pode, a partir de outro trecho do código, ter sua execução continuada do ponto onde parou, executando até o seu término ou nova suspensão. Note que uma corrotina que não se suspende antes de terminar é exatamente uma rotina comum.

O uso de corrotinas em Lua é feito através das seguintes primitivas:

- coroutine.create (f): retorna uma nova corrotina a partir da função passada como parâmetro. Cria os meios pelos quais o estado de uma corrotina é preservado entre suspensões
- coroutine.resume (co): recomeça a execução da corrotina do ponto onde parou. Também serve para iniciar a corrotina
- coroutine.yield (): suspende a execução da corrotina em execução
- coroutine.status (co): retorna o estado da corrotina passada, que pode ser *running*, *suspended*, *normal* ou *dead*

Voltando ao exemplo da animação do logotipo de Lua, agora podemos usar o *loop* problemático da Listagem 18.18, bastando colocá-lo dentro de uma corrotina e trocando a chamada *sleep()* por *coroutine.yield()* (Listagem 18.20).

```
function animaLogoLua ()
   while lua.x < 100 do
      lua.x = lua.x + 5
      redraw()
          coroutine.yield() -- sleep(30)
       end
    end
    coAnimaLogoLua = coroutine.create(animaLogoLua)

function update ()
   coroutine.resume(coAnimaLogoLua)
   if coroutine.status(coAnimaLogoLua) ~= 'dead' then
      event.timer(30, update)
   end
end

function tratador (evt)
   if evt.action == 'start' then
          update()
       end
end
event.register(tratador, 'ncl', 'presentation')
```

Listagem 18.20 Exemplo de uso de corrotinas para realizar uma animação.

Na listagem, a função *update* acorda a corrotina a cada 30 milissegundos até que ela termine, o que acontece quando o logotipo chega à posição 100 quando o *loop* é encerrado.

O uso de corrotinas simula a execução sequencial de código em situações em que, na verdade, uma entidade externa à aplicação controla sua execução (o temporizador, no exemplo apresentado).

É possível, ainda, trocar valores entre chamadas e suspensões de corrotinas, analogamente à passagem de parâmetros de funções. Para uma descrição mais detalhada do suporte a corrotinas de Lua, o manual da linguagem deve ser consultado [Ierusalimschy *et al*., 2006].

Exemplo 18.4 — Corrida de Cavalos (Parte I)

Imaginemos a simulação de uma corrida de cavalos. Neste exemplo, a primeira parte da simulação, o script para a animação de apenas um cavalo, é criado. A segunda parte, apresentada no Exemplo 18.5, reúsa esse script na definição de todos os cavalos.

A Figura 18.9 apresenta uma tira de imagens, com o cavalo em diversas posições. A cada intervalo de tempo, uma imagem diferente do cavalo será mostrada na tela, dando uma sensação de realidade à animação.

Figura 18.9 Imagem dos cavalos do Exemplo 18.4.

A tabela representando o cavalo é um pouco diferente da usada para representar os logotipos do exemplo anterior (Listagem 18.21).

```
local img = canvas:new('cavalo.png')
local larg, alt = img:attrSize()
local cavalo = { canvas=img,  quadro=0,
                 larg=larg/5, alt=alt }
```

Listagem 18.21 Tabela representando os cavalos

A largura do cavalo é igual à largura da imagem dividida pelo número de quadros da tira. Além disso, a tabela também guarda o quadro a ser exibido, que varia durante a animação.

A função de redesenho deve exibir apenas a parte da imagem dos cavalos que representa o quadro corrente. A função *compose* aceita parâmetros extras para indicar que região do canvas de origem (cavalo.img) deve ser composta.

Os parâmetros são as posições x,y da origem e a largura e altura a partir desse ponto (Listagem 18.22).

```
function redraw ()
   canvas:attrColor('black')
   canvas:drawRect('fill', 0,0, canvas:attrSize())
   canvas:compose( cavalo.x, cavalo.y, cavalo.canvas,
                   cavalo.quadro*cavalo.larg,0,
                   cavalo.larg,cavalo.alt) )
   canvas:flush()
end
```

Listagem 18.22 Função de redesenho

Para animar o cavalo, usaremos uma corrotina, como no exemplo anterior (Listagem 18.23).

```
function animaCavalo ()
   local partida = 10     -- posição da linha de partida
   local chegada = 500    -- posição da linha de chegada
   local mudaQuadro = 20  -- muda de quadro a cada 20px
   cavalo.x = partida
   while cavalo.x < chegada do
      coroutine.yield()
      local deslocamento = 3 + math.random(1,3)
      cavalo.x = cavalo.x + deslocamento
      mudaQuadro = mudaQuadro - deslocamento
      if mudaQuadro < 0 then
         quadro = (quadro + 1) % 6
         mudaQuadro = 20
      end
      redraw()
   end
end
coAnimaCavalo = coroutine.create(animaCavalo)
```

Listagem 18.23 Corrotina para animação dos cavalos

O deslocamento do cavalo, a cada 30 milissegundos, varia de 3 a 6 pixels, de acordo com a chamada à função *math.random*, para que a aplicação tenha alguma imprevisibilidade. O código para a função *update* e o tratador de eventos são iguais aos do exemplo anterior, bastando trocar o nome da corrotina de animação.

Novamente, o uso de uma corrotina permite que o código do cavalo seja escrito sequencialmente, facilitando o desenvolvimento e o entendimento do mesmo.

18.5 Reúso de Código Lua

Conforme mencionado por diversas vezes, um documento NCL não contém código Lua diretamente, mas referencia um objeto contendo o código Lua. Isso cria dois ambientes isolados de programação e também permite que um mesmo código Lua possa ser reusado em diversos objetos de mídia NCL. Para tornar ainda mais versátil essa abordagem, elos de atribuição em NCL podem ser usados para informar parâmetros a scripts Lua. Âncoras em objetos de mídia NCLua permitem tanto que o código Lua sirva como condição para o disparo de elos quanto permitem que o código Lua trate ações provenientes de elos, como já vimos.

O reúso de código Lua permite, por exemplo, que se definam componentes gráficos (ou *widgets*), tais como caixas de texto ou painéis de opção, uma única vez. Cada componente gráfico poderia ser reusado e parametrizado em documentos NCL como um objeto de mídia orquestrado, tal como qualquer outro objeto de mídia.

Exemplo 18.5 — Corrida de Cavalos (Parte II)

Este exemplo estende o Exemplo 18.4 para ilustrar uma corrida entre cavalos. O mesmo script NCLua, responsável pela animação de um único cavalo, é reusado na especificação de quatro objetos de mídia, demonstrando o uso de NCL como um orquestrador entre objetos de código imperativo de igual conteúdo.

A Figura 18.10 apresenta a visão estrutural do exemplo. O objeto de mídia "cavalo1" é a porta de início da exibição do documento. Quando "cavalo1" é iniciado, os outros objetos de mídia "cavalo2", "cavalo3" e "cavalo4" também são iniciados. Os quatro objetos de mídia referenciam o mesmo arquivo de extensão .lua.

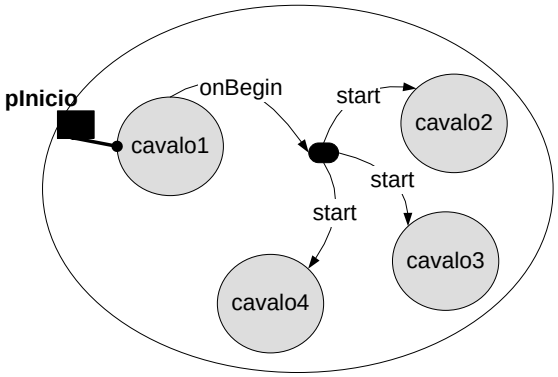


Figura 18.10 Visão Estrutural do Exemplo 18.5.

A Figura 18.11 ilustra as visões temporal e espacial do exemplo. Na visão temporal, as quatro animações em NCLua aparecem com o mesmo tempo total de exibição, por simplificação. Na prática, conforme é explicado no Exemplo 18.4, o tempo total de cada animação é imprevisível, já que cada cavalo possui um fator de deslocamento que varia de 4 a 6 pixels a cada 30 milissegundos.

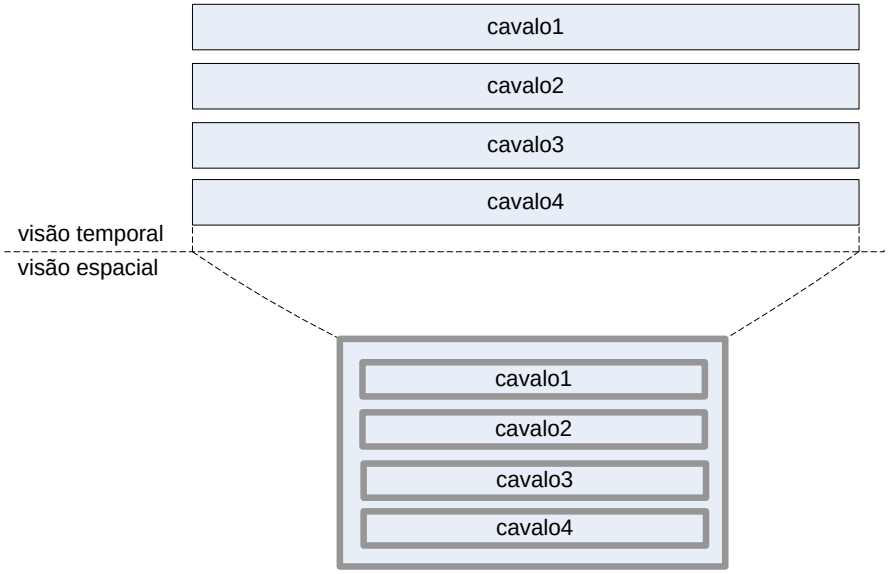


Figura 18.11 Visões temporal e espacial do Exemplo 18.5.

Exemplo 18.6 — Passagem de Valores

Esta seção apresenta uma aplicação que ilustra o reúso de código Lua em documentos NCL e a passagem de valores de propriedades entre objetos NCLua.

Quando inicia a aplicação, um campo de entrada e dois campos de saída são exibidos na tela. Enquanto o usuário preenche o campo de entrada, o primeiro campo de saída é automaticamente atualizado com o texto que vai sendo digitado. Apenas quando o usuário encerra a digitação por meio do botão OK do controle remoto ou através do ENTER no teclado alfanumérico, o texto digitado até o momento é copiado para o segundo campo de saída.

A Figura 18.12 ilustra a visão estrutural deste exemplo. O objeto NCLua "entrada" é disparado no início do documento, o que faz iniciar também os outros dois objetos NCLua "saida1" e "saida2". O foco para a digitação inicia no campo de entrada onde a propriedade "texto" armazena o valor atual digitado pelo usuário. A cada mudança no valor da propriedade "texto",

seu valor é copiado para a propriedade “texto” do objeto “saida1”, o que atualiza o que é exibido pelo primeiro campo de saída. Por outro lado, apenas quando a âncora “selecao” do objeto “entrada” é iniciada, seu valor é copiado para a propriedade “texto” do objeto “saida2”.

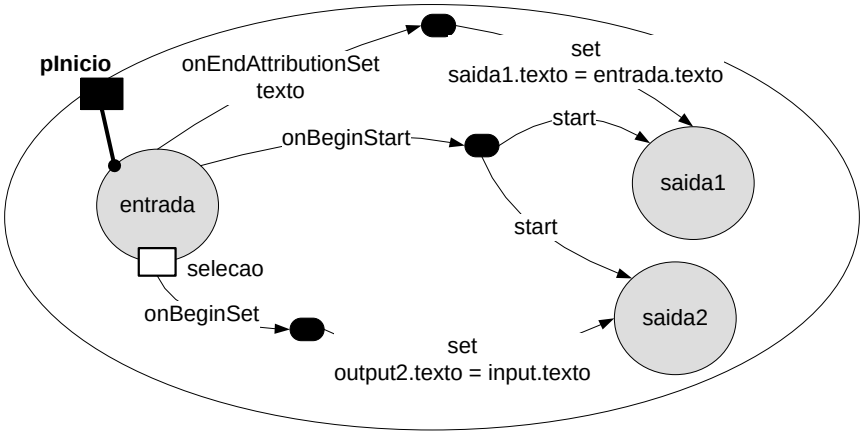


Figura 18.12 Visão estrutural do Exemplo 18.5.

Os campos de entrada e saída são implementados em dois scripts Lua diferentes. Ambos os códigos tratam a propriedade “texto” mantendo-a com o texto visualizado. No caso do campo de entrada, a propriedade “texto” é controlada pelo próprio NCLua e é alterada toda vez que uma nova tecla é pressionada. No caso do campo de saída, a propriedade “texto” deve ser controlada pelo documento NCL, através de elos de atribuição.

Neste exemplo, os objetos NCLua podem ser vistos como caixas-pretas, não importando o conteúdo dos arquivos .lua. Para o autor do documento NCL basta saber a interface que cada um dos NCLua oferece com suas propriedades “texto” e a âncora “selecao”, especificamente no campo de entrada. Dessa forma, os arquivos .lua podem ser reusados em outras aplicações.

O campo de entrada é representado em NCL pelo código na Listagem 18.24.

```
<media id="entrada" src="input.lua"  descriptor="dsInput">
    <area id="selecao" label=”sel”/>
    <property name="texto"/>
</media>
```

Listagem 18.24 Elemento <media> para a entrada

O objeto possui a âncora “selecao”, que é iniciada sempre que o usuário pressiona OK no controle remoto ou ENTER no teclado alfanumérico.

Os campos de saída são representados com o código da Listagem 18. 25.

```
<media id="saida1" src="output.lua" descriptor="dsOutput1">
    <property name="texto"/>
</media>
<media id="saida2" src="output.lua" descriptor="dsOutput2">
    <property name="texto"/>
</media>
```

Listagem 18.25 Elementos <media> para as saídas

A parte mais importante do documento é a que contém os elos responsáveis por copiar o conteúdo do campo de entrada para os campos de saída.

O primeiro campo de saída é atualizado sempre que a propriedade "texto" do campo de entrada é alterada (Listagem 18.26).

```
<link xconnector="onEndAttributionSet">
    <bind role="onEndAttribution" component="entrada"
          interface="texto"/>
    <bind role="set" component="saida1" interface="texto">
        <bindParam name="var" value="$get"/>
    </bind>
    <bind role="get" component="entrada" interface="texto"/>
</link>
```

Listagem 18.26 Relacionando o campo de entrada e o primeiro campo de saída

A associação com o papel "get", definido no próprio elo, permite consultar o valor de uma propriedade de qualquer objeto, guardando-o na variável "$get". No exemplo, a propriedade consultada é "texto" do objeto "entrada". A variável $get, por sua vez, é utilizada na associação com o papel de atribuição "set", fazendo com que o valor da entrada seja copiado para a propriedade "texto" do objeto "saida1".

O segundo campo de saída é atualizado somente quando a âncora "selecao" do objeto de "entrada" é iniciada (Listagem 18.27).

```
<link xconnector="onBeginSet">
    <bind role="onBegin" component="entrada"
          interface="selecao"/>
    <bind role="set" component="saida2" interface="texto">
        <bindParam name="var" value="$get"/>
    </bind>
    <bind role="get" component="entrada" interface="texto"/>
</link>
```

Listagem 18.27 Relacionando o campo de entrada e o segundo campo de saída

O mesmo mecanismo de cópia de valores de uma propriedade para outra também é usado para a cópia do valor digitado no campo de entrada para o segundo campo de saída.

Bibliografia

ABNT, NBR 15606-2, 2011. Associação Brasileira de Normas Técnicas, "Televisão digital terrestre — Codificação de dados e especificações de transmissão para radiodifusão digital, Parte 2: Ginga-NCL para receptores fixos e móveis — Linguagem de aplicação XML para codificação de aplicações", Sistema Brasileiro de TV Digital Terrestre, *NBR 15606-2*.

Ierusalimschy, R.; Figueiredo, L.H.; Celes, Waldemar. "Lua 5.1 Reference Manual." *Lua.org*, Agosto de 2006. ISBN 85-903798-3-3.

ITU-T, H.761, 2011. Nested Context Language (NCL) and Ginga-NCL for IPTV. ITU-T Rec. H.761.

Sant'Anna F.; Soares Neto, C.S.; Barbosa, S.D.J. "Aplicações Declarativas NCL com Objetos NCLua Imperativos Embutidos". Monografias em Ciência da Computação do Departamento de Informática, PUC-Rio, N.º 17/09. Rio de Janeiro, junho de 2009. ISSN 0103-9741.

Soares, L.F.G.; Sant'Anna, F.F; Cerqueira, R. (2008). "Nested Context Language 3.0 Part 10 — Imperative Objects in NCL: The NCLua Scripting Language." Monografias em Ciência da Computação do Departamento de Informática, PUC-Rio, N.º 02/08. Rio de Janeiro, janeiro de 2008. ISSN 0103-9741.

Soares, L.F.G.; Moreno, M.F.; Sant'Anna (2009). "Relating Declarative and Imperative Objects through the NCL Glue Language." *Proceedings of the ACM Symposium on Document Engineering*. Munique, setembro de 2009

Apêndices

Apêndice A

Codificação Digital na Comunicação de Dados Multimídia

A digitalização presenciada nos sistemas de telecomunicação, em paralelo aos avanços que a tecnologia digital já estava proporcionando aos sistemas computacionais, motivou, nos anos 1990, a idéia de que a convergência dessas duas áreas traria benefícios incomparáveis. A evolução tecnológica tornou possível o desenvolvimento de sistemas para processamento e comunicação de informações representadas pela integração de várias mídias: texto, áudio, vídeo etc. O efeito da convergência sobre os usuários é, entretanto, menos tecnológico e mais prático. O que se percebe, ao longo desse processo, é a crescente disponibilidade de um conjunto de serviços totalmente integrados, oferecidos de forma padronizada, de fácil acesso e a preços baixos, além da introdução de serviços, chamados de valor adicionado, que se juntam a um serviço principal complementando-o de forma a agregar valor e gerar mais interesse por parte do consumidor.

A mistura de diferentes tipos de mídia em sistemas integrados de transmissão de informação decorrente da convergência digital exige novas formas para a codificação eficiente dos dados. Neste capítulo, apresentaremos os principais conceitos envolvidos na codificação digital dessas informações e na sua comunicação.

A.1 Informação e Sinal

Os seres humanos adquirem informação através de seus sentidos: visão, audição, tato, olfato e paladar. Esses sentidos são denominados *mídias*[1] *de percepção*. As informações adquiridas são então codificadas em estruturas de dados que denominamos *mídias de representação*, ou simplificadamente *mídias*. São exemplos de mídias de representação as mídias texto, gráfico, áudio e vídeo. Note que não existe uma correspondência biunívoca entre as mídias de percepção e de representação e que ainda é, no mínimo, pouco usual a utilização de mídias representando informações adquiridas pelo olfato, tato e paladar, embora já existam estudos nesse sentido.

Podemos definir [Soares *et al.*, 1995] *sinais* como ondas que se propagam através de algum meio físico, possuindo, por exemplo, uma amplitude que varia ao longo do tempo, correspondendo à codificação da informação transmitida. Um sinal pode ser distorcido durante sua transmissão, por ele ter em suas componentes de frequência atenuações diferentes devido a limitações do meio de transmissão. É provável também termos perda ou deformação de parte do sinal por ruídos, como também discutido em Soares *et al*. (1995). Ao transmitir informações esperamos, no entanto, preservar seu significado e recuperar seu entendimento.

Podemos, então, introduzir informalmente o conceito de *qualidade de sinal* e *qualidade da informação* transmitida, através de um exemplo. Um vídeo transmitido a 30 quadros por segundo (padrão de TV), certamente tem uma qualidade de sinal melhor do que se fosse transmitido a 10 quadros por segundo (imagine, por exemplo, que a cada três quadros dois são perdidos). Se estivéssemos filmando uma paisagem sem qualquer movimento, a qualidade da informação transmitida seria, entretanto, a mesma (nesse caso extremo bastaria até mesmo a transmissão de um único quadro), independentemente do sinal recebido. Se, no entanto, o vídeo tivesse muito movimento, a qualidade da informação transmitida não seria boa a 10 quadros por segundo e a sensação obtida seria a de várias imagens com movimentos bruscos, sem naturalidade.

Do exemplo anterior, podemos notar que um sinal pode carregar muita informação redundante. Técnicas para redução dessa redundância, denominadas técnicas de compressão e de compactação, são usualmente empregadas. Sobre elas teremos muito o que discutir ao longo deste apêndice. Vamos, no entanto, primeiramente, aprofundar um pouco mais a discussão sobre informações e sinais, analógicos e digitais.

---

[1] Em português, mídia vem da palavra latina *medius* (de onde derivou a palavra anglo-saxônica *medium*), e seu plural é mídias (correspondendo à palavra anglo-saxônica *media*).

Inicialmente, os computadores estavam restritos ao processamento e comunicação de dois tipos de dados — letras e números. Códigos para números (binários, BCD, ponto flutuante IEEE etc.) estão hoje padronizados e estabilizados. Códigos para caracteres alfanuméricos (ASCII, EBCDIC etc.) são também amplamente aceitos. Enfim, a mídia textual é hoje razoavelmente bem entendida como codificação digital.

A mídia gráfica foi a segunda mídia a ganhar representação nos computadores digitais. Ela possui dois formatos: o vetorial e o matricial. O formato vetorial é bastante utilizado em modelagem geométrica e nele as figuras são representadas por um conjunto de segmentos de reta ou curvas, dados pelas coordenadas de seus pontos e pelos atributos das linhas que os unem. Imagens no formato matricial são usualmente chamadas de imagens estáticas. Nesse formato, as imagens são divididas em pequenas regiões, chamadas de elementos de fotografia, ou pixels (*picture elements*, algumas vezes também chamados de *pels*). Para cada uma dessas regiões guarda-se sua informação codificada de cor. Quanto maior o número de bits para codificar a cor, mais cores pode-se codificar e mais próximo pode-se chegar da cor original. Temos, assim, uma matriz de *M* linhas e *N* colunas, onde cada elemento representa um dos MxN pixels que compõem a imagem. Na reprodução da imagem, os pixels são reconstruídos utilizando-se a informação de cor armazenada na matriz. Quanto menor for o tamanho do pixel, mais fiel será sua coloração com relação à original, mas maior será a matriz da imagem. Ao tamanho da matriz dá-se o nome de *resolução geométrica* da imagem. A quantidade de bits utilizados para armazenar a cor de um pixel chama-se *resolução de cor* da imagem.

Informações capturadas nas mídias textual e gráfica são originalmente digitais. Por isso, muitas vezes essas mídias são referidas como *mídias discretas*. Já informações geradas por fontes sonoras e de vídeo apresentam variações contínuas de amplitude, constituindo o tipo de informação que comumente é percebida pelos sentidos humanos através de sinais que denominamos analógicos. Devido a isso, as mídias de vídeo e áudio são usualmente referidas como *mídias contínuas*.

É importante que se entenda que qualquer tipo de informação (seja analógica ou digital) pode ser codificada em uma estrutura de dados (mídia de representação) digital, e essa codificação digital pode ser transmitida em um sinal analógico ou digital, como veremos.

A codificação digital de informações tem, em geral, vantagens em uma comunicação. Isso se deve, principalmente, à possibilidade de podermos restaurar, no receptor, a informação codificada original, mesmo na presença de distorções, falhas ou ruídos no sistema de transmissão.

A.2 Conversão de Sinais

Para utilizarmos as vantagens da codificação digital, devemos transformar as mídias contínuas de áudio e vídeo, normalmente adquiridas através de sinais analógicos. A essa transformação chamamos de conversão analógica digital, ou conversão A/D.

Uma vez processados e transmitidos, os sinais digitais[2] podem ter de ser transformados em analógicos para percepção pelos sentidos humanos. A essa transformação chamamos de conversão digital analógica, ou simplesmente conversão D/A.

A.2.1 Conversão A/D

O teorema de Nyquist [Soares *et al*., 1995] assegura que uma taxa de amostragem de 2*W* vezes por segundo é o suficiente para recuperar um sinal com largura de banda de *W* Hz. Isso quer dizer que, de um sinal analógico, basta guardar os valores das amplitudes de suas amostras tomadas a intervalos regulares de 1/2*W* segundos para que se possa ter toda a informação necessária para reconstruí-lo integralmente. O processo de amostrar e guardar os valores dessas amostras, ilustrado na Figura A.1, é conhecido como *Pulse Amplitude Modulation (PAM)*.

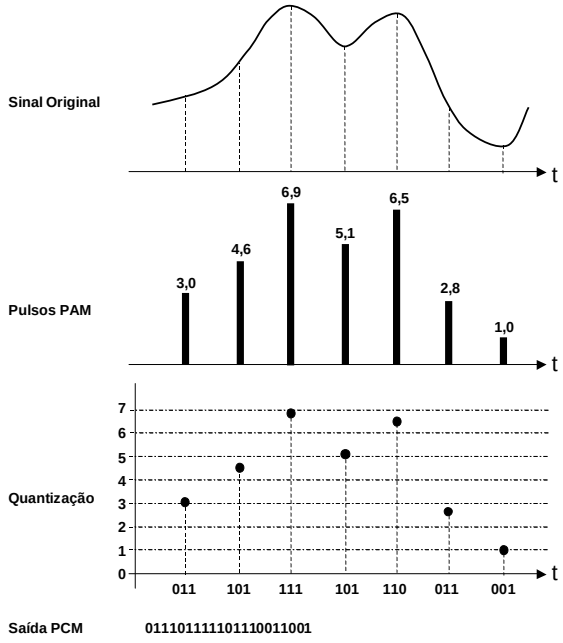


Figura A.1 Pulsos PAM e PCM.

[2] Um sinal digital pode ser transformado em sinal analógico, para transmissão em um dado meio, também pelo processo de modulação, comentado no Capítulo1.

A partir dos pulsos PAM, podemos produzir os pulsos PCM (*Pulse Code Modulation*) através de um processo conhecido como *quantização*, em que cada amostra PAM é aproximada a um inteiro de *n* bits. No exemplo da Figura A.1, escolhemos $n = 3$, dando origem a oito níveis (2^3) de quantização. A saída PCM corresponde ao resultado dessa quantização.

Podemos calcular, a partir desse processo, denominado *conversão A/D*, a taxa gerada pela transmissão de informação analógica através de sinais digitais.[3] Considere o caso de sinais de voz, por exemplo. Se assumirmos que a banda passante necessária desses sinais tem largura igual a 3.100 Hz, a taxa de amostragem de Nyquist é, nesse caso, igual a 6.200 amostras por segundo. Normalmente, amostra-se a uma taxa maior, para facilitar a construção dos codecs (codificadores/decodificadores). Se escolhermos uma taxa de 8.000 amostras por segundo e codificarmos cada amostra com oito bits, a taxa gerada será $8.000 \times 8 = 64$ Kbps, que é a taxa definida pelo padrão ITU-T G.711 [ITU-T G.711, 1988] para telefonia digital.

A Figura A.1 ilustra o caso em que os níveis de quantização são igualmente espaçados, ou seja, o *quantum* ΔQ (diferença entre níveis) é constante. Como consequência, o erro máximo de quantização é dado por $\Delta Q/2$. Chamamos esse caso de *quantização linear*. Nele, as amostras pequenas são, em termos proporcionais, mais penalizadas pelo erro de quantização do que as grandes.

Para melhorar a qualidade do sinal amostrado, podemos usar uma quantização logarítmica, onde o sinal é primeiro transformado, logaritmicamente, de forma a manter o erro máximo de quantização aproximadamente constante, a despeito da amplitude da amostra. Várias funções logarítmicas foram propostas e estudadas com esse intento. Duas dessas funções são largamente utilizadas e padronizadas, sendo denominadas *lei A* e *lei* μ (Figura A.2). A primeira é mais utilizada na Europa, enquanto a segunda predomina nos Estados Unidos.

[3] Neste apêndice, consideraremos sempre o sinal digital gerado a partir de uma informação codificada digitalmente com um bit por intervalo de sinalização, ou seja, um sinal onde sua taxa em bauds é a mesma que sua taxa em bits por segundo [Soares *et al*., 1995].

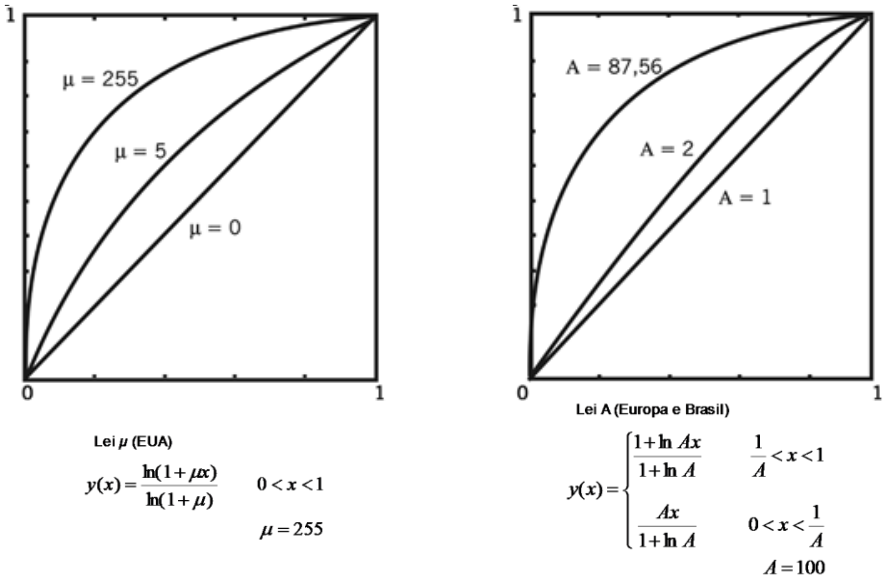


Figura A.2: Lei A e lei μ.

A Tabela A.1 apresenta o resultado da conversão A/D de alguns sinais de áudio e vídeo.

Tabela A.1 Conversão A/D para alguns sinais de áudio e vídeo

Sinais	Faixa Passante	Frequência de Amostragem	Codificação Bits/Amostra (b/a)	Taxa de Bits
Voz (ITU-T G711)	300–3.400 Hz	8.000 Hz	Log PCM (8b/a)	64 Kbps
Qualidade CD (estéreo)	0– 21 KHz	44,1 KHz	Log PCM (16 b/a por canal)	1,41 Mbps (2 × 720,6 Kbps)
Vídeo (NTSC - luminância)	0–4,2 MHz	10 MHz	8 b/a	80 Mbps

A.2.2 Conversão D/A

Pode-se demonstrar que um trem de pulsos PCM, obtido pela amostragem de um sinal em uma frequência maior ou igual à dada pelo teorema de Nyquist, tem o mesmo espectro de frequência que o sinal amostrado, no intervalo de frequências dado pela banda passante desse sinal. A conversão D/A se faz, então, pela simples passagem do trem de pulsos

PCM por um filtro na faixa passante (e, assim, com a largura de banda) do sinal originalmente amostrado.

Não fosse pelo erro de quantização, o sinal obtido da saída do filtro seria idêntico ao sinal analógico original.

Note que o sinal de saída é tão mais próximo do sinal original quanto menor for o erro de quantização. O erro de quantização, por sua vez, é tão menor quanto maior o número de níveis de quantização, ou seja, quanto maior o número de bits utilizados na codificação.

A.2.3 Outros Codificadores de Onda

A codificação PCM representa cada amostra pelo seu valor absoluto, mas essa não é a única representação possível. Alternativamente, é possível representar apenas a diferença entre o valor de uma amostra e o valor de sua antecessora. Esse esquema de codificação é denominado *DPCM* (*Differencial Pulse Code Modulation*). Quando os valores das amostras são muito próximos uns dos outros, necessitamos de um menor número de níveis para representar as diferenças do que o necessário para representar os valores absolutos das amostras, para um mesmo erro de quantização. Como um menor número de níveis pode representar um menor número de bits para codificar todos os níveis, utilizando o DPCM poderemos ter um menor número de bits gerados pela codificação. O DPCM é um caso particular de *codificação preditiva*, em que o valor predito da amostra corrente é o valor da amostra anterior, guardando-se (codificando-se) então o erro (diferença) de predição.

A ideia do DPCM pode ser ainda refinada um pouco mais, variando-se dinamicamente os níveis de quantização, dependendo de o sinal variar muito ou pouco. Dessa forma, prevê-se não apenas o valor da amostra corrente baseado na amostra anterior, mas também o valor do quantum, baseado em uma função, bem conhecida, dos valores de amostras anteriores. Esse esquema é denominado *ADPCM*, de *Adaptative Differencial Pulse Code Modulation*.

Existem ainda outras formas de codificação que independem do tipo do sinal analógico. Vamos citar apenas mais uma, a *SBC (SubBand Coding)*. Na codificação por sub-bandas, o espectro de frequência do sinal é dividido em várias bandas de frequência. Cada banda é então tratada como se representasse um sinal distinto, e nela é aplicada qualquer uma das técnicas anteriores. A vantagem da SBC é que, através da análise de um sinal, podemos identificar suas bandas mais importantes no transporte da informação. Para essas bandas, utilizamos um erro de quantização menor do que aquele usado nas bandas menos importantes, ou seja, podemos codificar

as bandas menos importantes utilizando um número menor de bits por amostras.

A.3 Compressão e Compactação

Um sinal digital, em geral, carrega muita informação redundante. Se eliminarmos essa redundância conseguiremos reduzir em muito a quantidade de bits gerados, que, em alguns casos, pode ser muito grande; pela Tabela A.1, por exemplo, observa-se que apenas 1 minuto de vídeo preto-e-branco gera 600 Mbytes.

Quando eliminamos apenas a redundância de um sinal, não há perda de informação e dizemos que fizemos uma *compactação*, ou *compressão sem perdas*. No entanto, podemos também diminuir a quantidade de bits com alguma perda de informação. Dependendo de quem for o usuário da informação, parte dela pode ser considerada pouco útil. Raramente é necessário manter o sinal original intacto no caso das mídias vídeo, áudio e imagens estáticas, uma vez que o usuário final perderia de qualquer forma parte da informação por limitações físicas; tal é o caso do ouvido e do olho humanos. Vemos, assim, que a quantidade de informação que podemos perder é dependente do usuário, mas ela também pode depender da tarefa em desenvolvimento: por exemplo, perder um pouco da nitidez de um vídeo em uma videotelefonia é perfeitamente aceitável, enquanto a perda da qualidade do vídeo pode ser inadmissível em uma aplicação médica. Quando na redução dos dados gerados há perda de informação, dizemos que fizemos uma *compressão com perdas*, ou simplesmente *compressão*.

Existem várias técnicas de compressão sem perdas (compactação) que podem ser aplicadas a qualquer tipo de dados, independentemente da mídia representada. As Seções A.3.1 a A.3.5 são dedicadas a algumas dessas técnicas mais usuais. As técnicas de compressão com perdas serão estudadas para cada mídia em particular nas Seções A.3.6 a A.3.8.

A.3.1 Codificação por Carreira

O desempenho da codificação por carreira (*run length coding*) depende muito da estatística dos dados de entrada. Ela consiste simplesmente em representar os dados pelo seu valor e o número de vezes que ele se repete. A unidade para codificação pode ser um bit, um byte, um caractere, um pixel, uma amostra etc. A Figura A.3 ilustra o caso da unidade ser um pixel de 8 bits e o caso da unidade ser o bit.

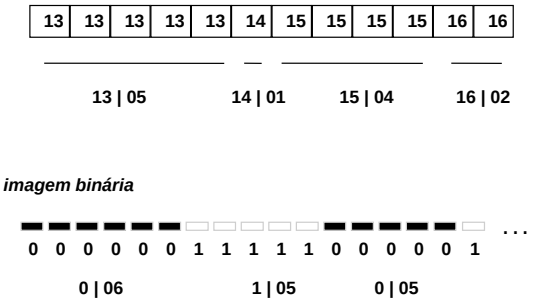


Figura A.3 Codificação por carreira.

A.3.2 Codificação de Shannon-Fano

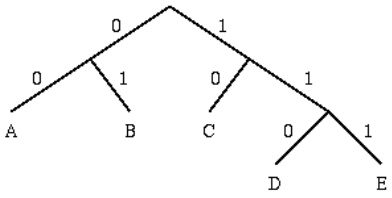
Para as duas próximas técnicas de codificação, imagine que nossos dados sejam apenas os símbolos A, B, C, D e E (que podem representar um pixel, uma amostra de áudio ou vídeo, um caractere etc.) e que eles ocorram na frequência dada pela tabela a seguir.

Símbolo	A	B	C	D	E
Número de Ocorrências	15	7	6	6	5

A codificação de Shannon-Fano constrói a árvore de codificação seguindo o seguinte algoritmo:

1. Alinhe os símbolos de acordo com suas frequências/probabilidades, por exemplo, ABCDE.
2. Divida recursivamente em duas partes, cada uma com aproximadamente o mesmo número de contagem (soma das frequências).

A árvore gerada fica então:



E temos a seguinte codificação, gerando uma compactação de 117:89.

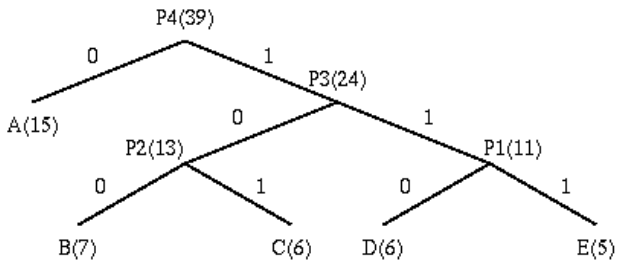
Símbolo	Frequência	Código	Subtotal (n.º de bits)
A	15	00	30
B	7	01	14
C	6	10	12
D	6	110	18
E	5	111	15

A.3.3 Codificação de Huffman

Tomando o mesmo exemplo da seção anterior, a codificação de Huffman constrói a árvore de codificação seguindo o seguinte algoritmo:

1. Iniciação: ponha todos os nós em uma lista ABERTA. Mantenha a lista alinhada todo o tempo de aplicação do algoritmo (por exemplo, ABCDE).
2. Repita até que a lista ABERTA contenha apenas um nó:
    a. Pegue os dois nós de mais baixas frequências/probabilidades e crie um nó pai para ambos.
    b. Atribua ao nó pai a soma das frequências/probabilidades dos filhos e o insira na lista ABERTA.
    c. Atribua códigos 0 e 1 aos dois ramos da árvore e retire os filhos da lista ABERTA.

A árvore gerada fica então:



E temos a seguinte codificação, gerando uma compactação de 117:87.

Símbolo	Frequência	Código	Subtotal (n.º de bits)
A	15	0	15
B	7	100	21
C	6	101	18
D	6	110	18
E	5	111	15

Note que, quer usando a codificação de Huffman ou de Shannon-Fano, o decodificador deve usar o mesmo dicionário de códigos gerado pelo codificador para recuperar os símbolos originais.

A.3.4 Codificação de Lempel-Ziv-Welch

Um procedimento diferente dos dois anteriores é processar símbolo a símbolo e ir construindo o dicionário de códigos passo a passo. À medida que o dicionário vai sendo construído, ele pode ser usado na codificação do próximo símbolo, dinamicamente.

No esquema de Lempel e Ziv, posteriormente estendido por Welch, o dicionário é construído como uma estrutura de dados, uma tabela que mantém sequências de símbolos, em conjunto com um identificador único (código) para toda a sequência. A tabela contém até, digamos, 2^j posições (sequências). Ela é iniciada simplesmente com o conjunto dos 2^k possíveis símbolos, isto é, todas as sequências de tamanho 1. Os melhores desempenhos são conseguidos quando k << j, dependendo, obviamente, do grau de redundância dos dados.

A codificação se inicia definindo a sequência de símbolos corrente *S* como o primeiro símbolo a codificar. Note que *S* é membro do dicionário. A codificação então continua como a seguir:

1. Se não existem mais símbolos para codificar, dê como saída o código da sequência (j bits) para S. Em caso contrário,
2. Pegue o próximo símbolo P e concatene a S, obtendo a nova sequência SP.
3. Se SP já estiver no dicionário, faça S = SP e volte para o passo 1. Em caso contrário,
4. Dê como saída o código da sequência (j bits) para S.
5. Adicione SP ao dicionário, se ainda houver espaço.
6. Faça S = P e volte para o passo 1.

Note que, assim procedendo, o decodificador não tem a necessidade de conhecer *a priori* todo o dicionário, pois pode reconstruí-lo passo a passo, dinamicamente, a partir dos dados codificados. A decodificação se inicia definindo a sequência de símbolos corrente *S* como a entrada no dicionário correspondente ao primeiro código a decodificar e dando como saída o símbolo *S*. Se houver mais códigos a decodificar:

1. Leia próximo código X.
2. Se houver a entrada no dicionário (P) correspondente a X:
   a) Dê como saída P.
   b) Adicione S concatenado ao primeiro símbolo de P no dicionário, caso a entrada não exista.

   Se não houver a entrada no dicionário (P) correspondente a X:
   a) Faça P igual a S concatenado ao primeiro símbolo de S e adicione ao dicionário.
   b) Dê como saída P.
3. Faça S igual a P e volte para o passo 1, se houver mais símbolos a decodificar.

A.3.5 Outras Técnicas de Compactação

Além das técnicas anteriormente mencionadas, outras são encontradas, bem como variantes das primeiras. Uma, no entanto, merece destaque por ser comumente utilizada: a codificação aritmética.

A codificação aritmética também parte do conhecimento da frequência de ocorrências dos símbolos, tal qual as codificações de Shannon-Fano e de Huffman. Baseado na frequência de ocorrências, intervalos são associados aos símbolos e, a partir desses intervalos, novos intervalos são construídos para sequências de símbolos. Uma sequência pode então ser codificada por qualquer número dentro do intervalo calculado para a sequência, garantindo sua decodificação posterior sem ambiguidade. Em Cormen *et al*. (2002) pode-se encontrar a especificação detalhada do algoritmo.

A.3.6 Compressão em Imagem Estática

As imagens geram, normalmente, uma quantidade de informação muito grande. Se levarmos em consideração a correlação do valor (cor) de cada pixel, poderemos reduzir a quantidade de informação gerada.

Existe grande número de formatos para imagens estáticas, com base em esquemas diferentes de compressão. Dentre os mais usuais atualmente

podemos citar os formatos *TIFF* e *GIF*, que são baseados no algoritmo de Lempel-Ziv, apresentado na Seção A.3.4, e o padrão ISO para imagens estáticas, chamado JPEG [ISO/IEC 10918-1, 1994], baseado em transformadas, como veremos mais adiante.

A forma mais simples de compressão de uma imagem é a redução da sua resolução geométrica. Isso implica aumentarmos o tamanho da região de um pixel. Tal procedimento pode ser feito até um certo limite, dependendo do usuário final e do dispositivo de exibição, para evitar o efeito apresentado na Figura A.4, onde vemos primeiro a imagem original dividida em pixels e, em seguida, a imagem reproduzida; a diferença se deve ao fato de o pixel representar uma região grande.

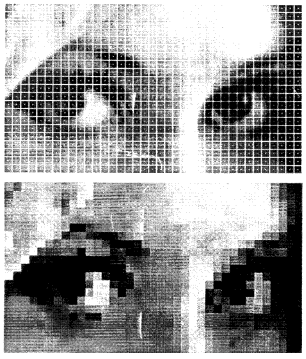


Figura A.4 Efeito do tamanho da região de um pixel na codificação/decodificação.

Outra forma de compressão é a redução do espaço de cor pela simples utilização de um número menor de bits para representação de cada pixel. Cabe antes, no entanto, uma discussão sobre como as cores são representadas.

Através das adições das cores vermelha, verde e azul, podemos obter *quase todas* as cores visíveis pelo olho humano. Assim, uma representação bem comum é a RGB (de *Red*, *Green*, *Blue*), na qual um pixel é representado pelos valores dessas componentes. É comum encontrarmos o padrão RGB 8-8-8, em que se utilizam 8 bits para codificação de cada componente, e o padrão 5-6-5, em que é reservado um número maior de bits (6) para a componente verde, por ser o olho humano mais sensível a essa componente. Outra representação bastante utilizada é o sistema YC_rC_b. A componente Y é denominada *luminância*, e é uma medida da sensibilidade do olho humano às várias componentes de frequência de uma cor. Para as fontes usuais de luz provenientes de dispositivos de vídeo, Y é dada por:

$$Y = 0{,}299R + 0{,}587G + 0{,}114B$$

As componentes C_r e C_b são chamadas de *crominância* e sua definição varia de padrão para padrão, bem como o seu nome (componentes I e Q, no padrão NTSC de TV, e componentes U e V, no padrão PAL de TV). C_r e C_b são os nomes utilizados pelos padrões MPEG e JPEG, e têm seus valores dados por:

$$C_r = ((R - Y)/1{,}6) + 0{,}5$$
$$C_b = ((B - Y)/2) + 0{,}5$$

O leitor deve observar que o conjunto de equações para Y, C_r e C_b é linearmente independente (isso também acontece para as outras definições de C_r e C_b), ou seja, dados Y, C_r e C_b, obtêm-se facilmente R, G e B.

O olho humano é mais sensível à luminância do que à crominância; assim, na compressão pela redução da resolução de cor, podemos utilizar um número menor de bits para as componentes da crominância. Mais comum, no entanto, é utilizarmos uma menor resolução geométrica para as componentes de crominância do que aquela utilizada para a luminância.

Uma outra forma de compressão de imagem estática, geralmente sem perdas, é a codificação preditiva. De forma análoga à explicação da Seção A.2.3, na codificação preditiva de uma imagem realiza-se uma predição do valor do pixel baseada em valores de outros pixels da imagem. A diferença do valor real para o valor predito é então codificada. A Figura A.5 ilustra o caso.

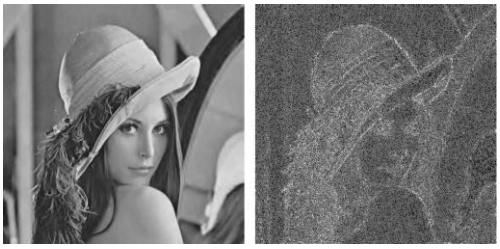


Figura A.5 Imagem original e imagem com apenas os erros de predição.

Se, na imagem, os pixels tiverem valores muito próximos, pode-se usar um número menor de bits para armazenar o erro da predição do que aquele usado para codificar o valor absoluto do pixel. Além disso, uma imagem com poucos contornos vai gerar muitos valores pequenos ou mesmo o valor zero, tornando o emprego de um outro método de compressão posterior bem eficiente.

Antes de continuarmos nossa discussão sobre técnicas de compressão, devemos ressaltar que, normalmente, as técnicas de compressão são seguidas pela aplicação de algum esquema de compactação. Muitas vezes, o esquema de compressão simplesmente prepara os dados para que possam sofrer maior compactação. Nada impede também que apliquemos várias técnicas de compressão em sequência.

Um exemplo de codificação preditiva pode ser encontrado no padrão JPEG [ISO/IEC 10918-1, 1994] no modo sem perdas, onde a codificação de Huffman é aplicada após a codificação preditiva. No esquema apresentado na Figura A.6, o codificador por entropia utiliza a codificação de Huffman. Note também, pela figura, que existem sete possíveis predições para um pixel *X*.

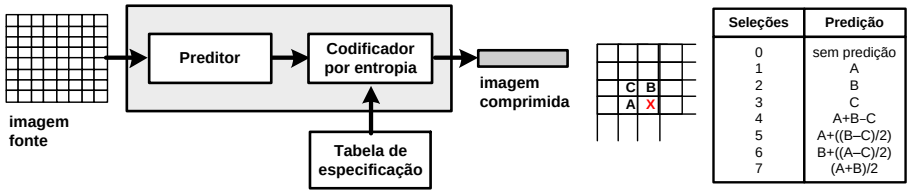


Figura A.6 JPEG sem perdas.

Existem ainda outras técnicas para compressão de imagem, tais como a codificação por sub-bandas (similar à apresentada na Seção A.2.3) e a quantização vetorial. Entretanto, nos deteremos apenas em mais uma, por ser utilizada nos padrões JPEG e MPEG: a codificação por transformadas.

O leitor já deve ter percebido, pelas várias referências à Seção A.2.3, que existem várias similaridades entre amostras no tempo e pixels. Na verdade, podemos considerar os pixels como se fossem amostras do "sinal imagem", só que amostras obtidas não no tempo, mas no espaço. É exatamente por isso que podemos aplicar todas as técnicas da Seção A.2.3 nas imagens estáticas. Note também que, em um sinal de vídeo, o grupo de várias amostras temporais forma um quadro (por exemplo, no sistema de TV brasileiro, existem 30 quadros por segundo). Esse quadro é uma imagem estática onde as amostras temporais do vídeo são os pixels (amostras espaciais). Esse fato é que nos permite não somente tratar o vídeo como um sinal contínuo e nele aplicarmos todas as técnicas de compressão conhecidas para sinal contínuo, como também tratá-lo como uma sequência de imagens estáticas no tempo, aplicando as mesmas técnicas de compressão utilizadas para imagens.

Uma vez que uma imagem estática pode ser considerada uma sequência de amostras espaciais, podemos agora pensar, como fizemos com os sinais analógicos, em aplicar uma transformada (por exemplo, Fourier) para descrever o mesmo sinal no domínio da frequência. Só que, agora, no domínio das frequências espaciais. Como estamos com um sinal discreto, precisaremos de uma transformada discreta. Poderíamos usar a transformada discreta de Fourier, como mencionado, mas outra transformada leva a melhores resultados na compressão: a *transformada discreta de cossenos*.

No espaço bidimensional de uma imagem de 8×8 pixels, a transformada discreta de cossenos (*FDCT — Forward Discrete Cosine Transform*) é dada por:

$$F(u,v) = \frac{1}{4} C(u)C(v) \sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y) \cos\left[\frac{(2x+1)u\pi}{16}\right] \cos\left[\frac{(2y+1)v\pi}{16}\right]$$

$$C(w) = \frac{1}{\sqrt{2}} \quad para \;\; w = 0$$

$$C(w) = 1 \quad para \;\; w = 1, 2, \ldots, 7$$

E a transformada inversa (*IDCT — Inverse Discrete Cosine Transform*) por:

$$f(x,y) = \frac{1}{4} \sum_{u=0}^{7} \sum_{v=0}^{7} C(u)C(v)F(u,v) \cos\left[\frac{(2x+1)u\pi}{16}\right] \cos\left[\frac{(2y+1)v\pi}{16}\right]$$

No domínio da frequência, as mudanças abruptas que acontecem nos contornos de uma figura estão concentradas nas frequências mais altas. Assim, uma imagem com poucos contornos deve concentrar seus coeficientes nas frequências baixas. Mais ainda, os coeficientes das frequências altas são menos importantes, e perdas nesses coeficientes podem diminuir um pouco a nitidez da imagem, mas para muitas aplicações isso pode ser aceitável.

Pelos motivos mencionados no parágrafo anterior, após uma imagem ser transformada os coeficientes gerados são quantizados de forma diferenciada, usando maior precisão (quantum menor) para as frequências mais baixas. Assim procede o padrão JPEG [ISO/IEC 10918-1, 1994] no modo sequencial, dividindo uma imagem em blocos de 8×8 pixels e aplicando uma compressão em cada bloco, conforme o diagrama mostrado na Figura A.7. A imagem é varrida uma única vez, da esquerda para a direita, de cima para baixo.

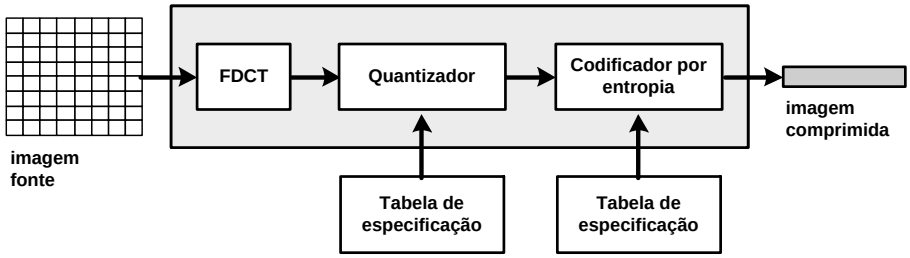


Figura A.7 Codificação JPEG modo sequencial.

No JPEG, após a aplicação da transformada discreta de cossenos e a quantização dos coeficientes, estes são organizados da mais baixa frequência para a mais alta,[4] quando então é aplicada a codificação por carreira, seguida da codificação de Huffman (ou codificação aritmética), ilustradas na Figura A.7 pelo bloco "codificador por entropia".

A decodificação JPEG modo sequencial é ilustrada na Figura A.8.

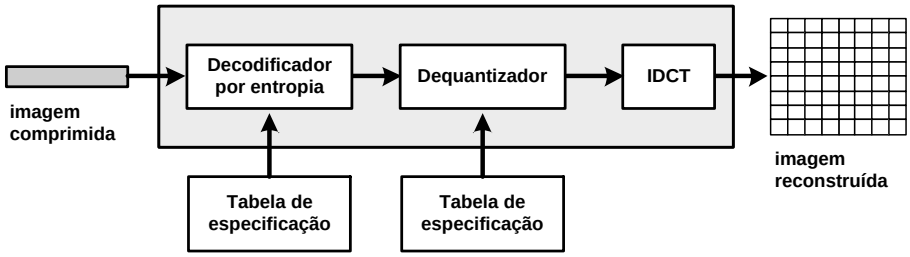


Figura A.8 Decodificação JPEG modo sequencial.

O padrão JPEG ainda possui mais dois modos de compressão: o *progressivo* e o *hierárquico*.

O modo progressivo também utiliza a transformada de cossenos, mas a imagem é codificada em várias varreduras. Na variante denominada *seleção de espectro*, a cada varredura são codificados coeficientes das mesmas frequências de todos os blocos, da mais baixa frequência para a maior frequência. Já na variante denominada *aproximação sucessiva*, primeiro é codificado o coeficiente de mais baixa frequência de todos os blocos e depois são codificados, paulatinamente, os bits dos demais coeficientes de todos os blocos, do mais significativo para o menos significativo.

[4] Na verdade, um passo adicional existe no JPEG, quando os coeficientes DC (frequência zero) de um bloco são codificados pela diferença entre seu valor e o valor do coeficiente DC do bloco anterior.

Note que, no modo progressivo, os primeiros dados da imagem que são decodificados já permitem ter uma visão completa da cena, embora ainda sem muita nitidez. Com o restante dos dados, a nitidez vai se tornando cada vez melhor. Isso pode ser útil em vários casos. Muitas vezes, navegando na Web, queremos apenas ter uma ideia da informação que vem na página a seguir. Certas vezes essa informação nem é aquela que desejamos. Ter uma visão rápida dessa informação, ainda que não com todos os detalhes, pode acelerar em muito a tarefa sendo executada. Outras vezes podemos estar trafegando com a imagem em um meio de pequena banda passante ou mesmo em um trecho congestionado de uma rede. Nesses casos, o descarte seletivo dos dados que não trazem muita informação pode ser a única forma viável de manter a aplicação em funcionamento. Como veremos, um bom sistema de comunicação deve poder identificar a parte da informação que ele deve manter íntegra e quais partes ele pode perder, em caso de necessidade ou conveniência.

O modo JPEG hierárquico também permite separar da imagem os dados mais relevantes dos menos relevantes, mas através do aumento progressivo da resolução geométrica. Nesse modo, a imagem é codificada com múltiplas resoluções, de forma que a menor resolução pode ser decodificada sem a necessidade de se ter a resolução maior.

A.3.7 Compressão em Áudio

A compressão de um sinal de áudio depende muito do tipo de sinal. Vamos começar pela voz humana.

Um ser humano falando emite surtos de voz apenas durante 35% a 40% do tempo de fala. O restante do tempo é preenchido com silêncio que existe entre palavras e entre uma sentença e outra. Se pudermos detectar esse silêncio e eliminá-lo da codificação, de forma que ele possa ser recuperado na decodificação, reduziremos muito a quantidade de dados gerados. Essa técnica é aplicada à telefonia digital com o nome TASI (*Time Assignment Digital Interpolation*). Ainda como outra característica da voz e do ouvido humanos, a perda tolerada de surto de voz e de silêncio é muito diferente. Perdas da ordem de 1% da informação do surto de voz são toleráveis,[5] ao passo que podemos tolerar a perda de até 50% do silêncio. Note que, com a detecção de silêncio, transformamos um tráfego de voz contínuo em um tráfego em rajadas.

[5] Na realidade, a porcentagem de perda depende do tamanho do surto de voz e se a perda ocorre no início ou no meio do surto. Em Gruber (1985] é possível encontrar uma discussão sobre o assunto.

Uma outra forma de comprimir a voz humana é codificar, em vez de suas amostras, os parâmetros de um modelo analítico do trato vocal capaz de gerar aquelas amostras. No método conhecido como LPC (*Linear Predictive Coding*), apenas os parâmetros que descrevem o melhor modelo que se adapta às amostras é codificado. Um decodificador LPC usa esses parâmetros para a geração sintética da voz, que é usualmente parecida com a original. O resultado é inteligível, mas a tonalidade é aquela de um "robô falando" ("voz metálica').

O CELP (*Code Excited Linear Predictor*) é bastante similar ao LPC. O codificador CELP gera os mesmos parâmetros LPC, mas computa os erros entre a fala original e a fala gerada pelo modelo sintético. Tanto os parâmetros do modelo analítico do trato vocal quanto uma representação comprimida dos erros são codificados. A representação comprimida é um índice em um vetor de excitação (que pode ser pensado como um livro de códigos compartilhado pelo codificador e o decodificador). O resultado do CELP tem uma qualidade de fala muito boa a uma taxa bem baixa. A Tabela A.2 apresenta alguns padrões para voz recomendados pelo ITU-T [ITU-T G.711, 1988; ITU-G.722, 1988; ITU-T G.723, 1996; ITU-T G.726, 1990; ITU-T G.728, 1992; ITU-T G.729, 1996].

Tabela A.2 Padrões ITU-T para voz

Padrão	Algoritmo	Taxa de Compressão (Kbps)	Recursos de Processamento Necessários	Qualidade da Voz Resultante	Atraso Adicionado
G.711	PCM	48, 56, 64 (sem compressão)	Nenhum	Excelente	Nenhum
G.722	SBC/ADPCM	64 (faixa passante de 50 Hz a 7 KHz)	Moderado	Excelente	Alto
G.723	MP-MLQ	5.3, 6.3	Moderado	Boa (6.3) Moderada (5.3)	Alto
G.726	ADPCM	16, 24, 32, 40	Baixo	Boa (40) Moderada (24)	Muito baixo
G.728	LD-CELP	16	Muito alto	Boa	Baixo
G.729	CS-ACELP	8	Alto	Boa	Baixo

Mais especificamente para áudio, de forma geral, um padrão muito importante é o MPEG áudio [ISO/IEC 11172-,3 1993; ISO/IEC 13818-3, 1998; ISO/IEC 13818-7, 1997; ISO/IEC 14496-3, 2004].

O MPEG áudio leva em conta o modelo psicoacústico humano para realizar uma compressão "perceptualmente sem perdas". O modelo divide o domínio de frequência audível (entre 20 Hz e 20 KHz) em 32 bandas, chamadas bandas críticas. O sistema de audição tem uma resolução limitada e

dependente da frequência. A medida perceptualmente uniforme de frequências pode ser expressa em termos das larguras das *bandas críticas*. A Figura A.9 mostra a sensibilidade do ouvido humano nas diversas frequências.

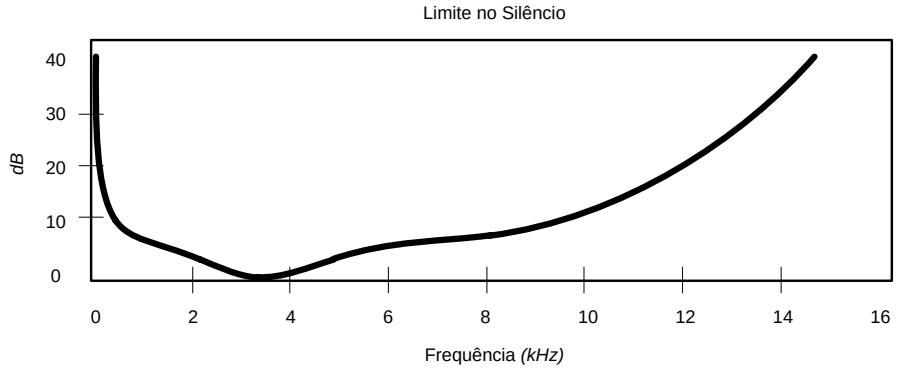


Figura A.9 Sensibilidade do ouvido humano.

Note que a sensibilidade do ouvido ilustrada na Figura A.9 é medida em *decibéis* (*dB SPL — dB Sound Preassure Level* — ou, simplificadamente, *dB*). O decibel é uma unidade conveniente para expressar o que se chama de *nível sonoro*. O som, de forma geral, tem uma medida de *intensidade* que é a potência transferida por uma onda sonora por área de uma superfície que intercepta essa onda. O decibel nada mais é do que uma forma comparativa de analisar valores. Nesse caso, em vez de fornecer o valor absoluto da intensidade sonora para uma frequência, podemos fornecer o seu valor dividido pela menor intensidade perceptível ao ouvido humano[6] e utilizar uma escala logarítmica para representar essa razão, já que o intervalo de intensidades produzido pela voz humana é muito grande. Assim, o nível sonoro em decibéis β é definido como:

$$\beta = 10 \log_{10} \left(\frac{I}{I_0} \right)$$

onde I_0 é o menor valor de intensidade sonora perceptível ao ouvido e I é a intensidade do som sendo medido. Note que, para $I = I_0$, temos $\beta = 0$ *dB*, ou seja, a nossa referência de menor intensidade perceptível corresponde a 0 dB.

Voltando ao MPEG áudio, seu modelo leva em conta o mascaramento de frequências, característica do ouvido humano que, quando submetido a um sinal de certa amplitude em dada frequência, mascara as outras frequências

[6] O valor de referência é definido como $I_0 = 10^{-12}$ watts/m^2. Ele corresponde a aproximadamente o limiar da audição humana. No entanto, o limiar de audição varia de frequência para frequência e com a intensidade do som, como descoberto por Fletcher e Munson em 1933.

ao redor que possuam uma amplitude abaixo de um certo limite. A Figura A.10 ilustra o caso.

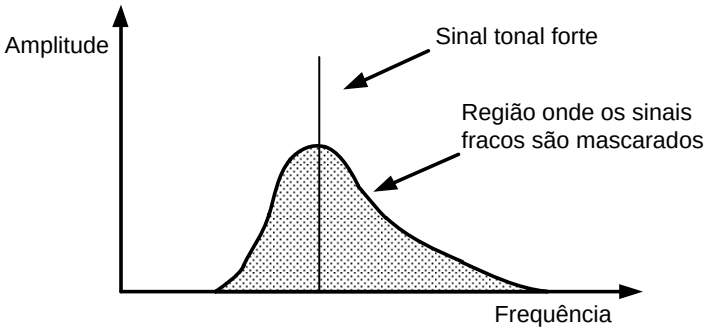


Figura A.10 Mascaramento de frequências.

O MPEG áudio transforma o sinal para o domínio da frequência e aplica o mascaramento de frequências, codificando apenas aquelas componentes de frequência que não são mascaradas. A Figura A.11 ilustra o procedimento.

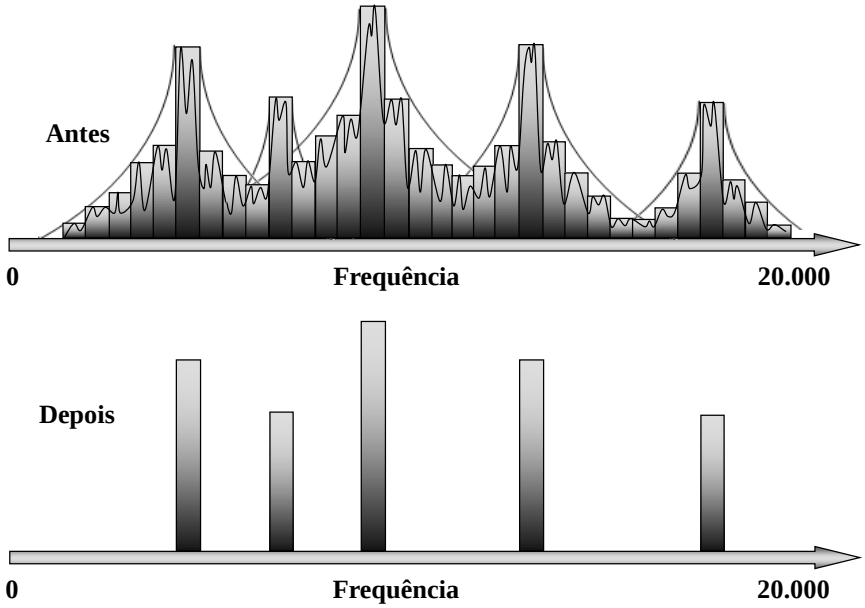


Figura A.11 Mascaramento de frequências nas bandas críticas MPEG.

O MPEG áudio também leva em consideração resultados psicoacústicos que mostram que, para frequências maiores que 2 KHz, o ouvido percebe a

imagem estereofônica baseado mais no envelope temporal do áudio do que em sua estrutura mais refinada. Assim, no modo "intensity stereo", o codificador soma as frequências mais altas do sinal estereofônico em um único sinal. Os canais de esquerda e direita são reconstruídos com a mesma forma, mas com magnitudes diferentes, baseadas em fatores de escala.

Na codificação MPEG, para cada intervalo de tempo de áudio codificado (isto é, para cada conjunto de amostras), existe um número fixo de bits total para todas as 32 sub-bandas. Escolhe-se o número de bits de uma banda de forma a minimizar a percepção auditiva do ruído de quantização levando em conta, como já mencionamos, o mascaramento de frequências.

O MPEG 1 áudio [ISO/IEC 11172-3, 1993] define três métodos de compressão, denominados camadas 1, 2 e 3 (MP1, MP2, MP3).

O MP1 agrupa 12 amostras para cada uma das 32 sub-bandas. Cada grupo de 12 recebe, então, os bits para codificação e, se o número de bits não for zero, um fator de escala.

O MP2 e o MP3 ainda levam em conta uma outra característica psicoacústica, o mascaramento temporal. Quando é emitido um tom em uma dada frequência e com uma certa amplitude, esse tom mascara os tons, na mesma frequência, abaixo de uma certa amplitude, que varia no tempo. A Figura A.12 ilustra o fato com um tom emitido a 60 dB.

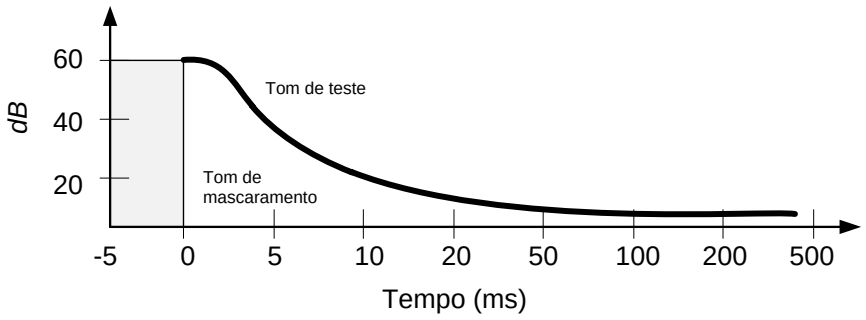


Figura A.12 Mascaramento temporal.

O MP2 codifica os dados em grupos maiores: para cada sub-banda, agrupa três grupos de 12 amostras. Isso modela um pouco da máscara temporal, pois faz-se uma alocação de bits e usam-se três fatores de escala para cada trio de 12 amostras.

Tanto o MP1 quanto o MP2 usam bandas uniformes, isto é, do mesmo tamanho, que não modelam bem o ouvido humano, como pode ser visto pela

Figura A.9. O MP3 usa bandas não-uniformes e faz uma alocação de bits melhor. Faz também melhor cálculo do quantum de cada banda, isto é, melhor distribuição de bits, usando inclusive o conceito de reservatório de bits (como mencionado, o número de bits total para as 32 bandas é fixo para cada grupo de amostras, mas no MP3 pode haver empréstimos de bits de um grupo de amostras para outro).

O MP2 e o MP3 também permitem o *MS stereo mode*, além do *stereo intensity mode*. O modo MS stereo codifica os sinais de frequências mais altas dos canais direito e esquerdo como a soma (*middle-channel*) e a diferença (*side-channel*). Técnicas de sintonização são então utilizadas para comprimir o sinal side-channel.

A Tabela A.3 apresenta uma comparação das várias camadas MPEG 1 áudio.

Tabela A.3 Camadas MPEG 1 áudio (a codificação pode ser realizada com taxas de amostragem de 32, 41.1 e 48 KHz)

Camadas	Taxa de Bits Alvo (Kbps)	Taxa de Compressão	Qualidade	Retardo Máximo (ms)
MP1	32 a 448	4:1		50
MP2	32 a 384	6:1		100
MP3	32 a 320	12:1	Telefonia: 8 Kbps Rádio AM: 32 Kbps Rádio FM: 64 Kbps CD: 128 Kbps	150

As camadas do padrão MPEG 1 áudio, denominadas fase 1 (MP1, MP2 e MP3) preveem apenas o uso de dois canais em um fluxo de áudio, ou seja, apenas o estéreo tradicional. O padrão MPEG 2 [ISO/IEC 13818-3, 1998] introduz algumas extensões. O padrão MPEG 2 áudio (fase 2) comum, chamado de "BC" (*Backward Compatible*), tem as mesmas camadas e usa os mesmos algoritmos com os mesmos parâmetros do áudio MPEG 1. A diferença é que o MPEG 2 prevê a formação de fluxos de áudio com até seis canais (5.1), em vez de implementar apenas o estéreo com dois canais.

A codificação AAC (*Advanced Audio Coding*) é novidade do áudio trazida pelo MPEG-2, conhecida como NBC (*Non Backward Compatible*), de não-compatível com MPEG 1 (também conhecido como MPEG-2 Parte 7 [ISO/IEC 13818-7, 1997] ou MPEG-4 Parte 3 [ISO/IEC 14496-3, 2004]).

Essa codificação é mais eficiente que o MPEG 1 (MP3 etc.), tolera até 48 canais de áudio e 15 canais de frequências baixas, com taxas de amostragem de 8 a 96 KHz. A AAC necessita de menos processamento e, consequentemente, tem retardo menor na (de)codificação.

A AAC é considerada o estado da arte para áudio de alta qualidade em uma taxa de bits típica de 128 Kbps. Abaixo dessa taxa, a qualidade do áudio começa a degradar, o que pode ser compensado por técnicas de melhoramento, como SBR (*Spectral Band Replication*) e PS (*Parametric Stereo*).

A SBR é uma técnica de extensão que permite a mesma qualidade de som a aproximadamente metade da taxa de bits. A combinação de AAC e SBR é chamada de HE-AAC (*High efficiency-AAC*) versão 1 [ISO/IEC 14496-3, 2004], também conhecida como "aacPlus v1".

A PS aumenta a eficiência de codificação ainda mais, através de uma representação paramétrica da imagem estéreo de um sinal de entrada. A combinação de AAC, SBR e PS é chamada de HE-AAC (*High efficiency-AAC*) versão 2 [ISO/IEC 14496-3, 2004].

Note assim que HE-AAC, definido no padrão MPEG-4, é um superconjunto do núcleo AAC, que estende a alta qualidade de áudio AAC para taxas de bits mais baixas. Decodificadores HE-AAC v2 são capazes de decodificar fluxos HE-AAC v1 e fluxos AAC. Por sua vez, decodificadores HE-AAC v1 são também capazes de decodificar fluxos AAC. Como vimos no Capítulo 1, o sistema brasileiro de TV digital terrestre adotou o padrão MPEG-4 para a codificação do áudio principal de um programa [ABNT NBR 15602-2 ,2007], com as características apresentadas na Tabela 1.1.

Além das codificações apresentadas, ainda existem várias outras em uso atualmente. Entre elas podemos citar a DD e a DTS.

AC-3 era o antigo nome da codificação chamada hoje de Dolby Digital (DD). Essa codificação é propriedade da empresa Dolby, mas foi adotada pelos Estados Unidos como codificação de áudio a ser utilizada nos DVDs e em HDTV (*High Definition TV*). Ela utiliza seis canais de áudio, sendo cinco com qualidade de CD (20 Hz a 20 KHz) e um apenas para as baixas frequências (20 a 120 Hz). A taxa dessa codificação é de cerca de 384 Kbps. A Figura A.13 apresenta a distribuição sugerida de autofalantes para os seis canais de áudio.

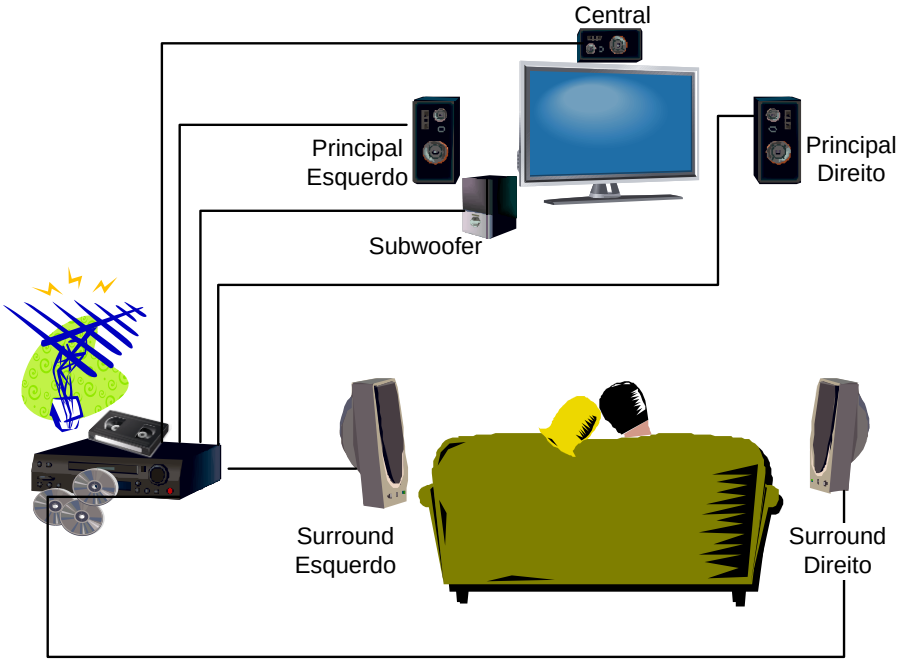


Figura A.13 Áudio multicanal.

A Europa usa ainda uma outra codificação proprietária, a DTS. Essa codificação também é multicanal, mas a taxa gerada é de cerca de 1,536 Mbps. Tanto DD quanto DTS trabalham com codificação por sub-bandas (até 576 na DD). Testes exaustivos com especialistas em áudio não conseguem chegar a uma definição sobre qual codificação é a melhor. No entanto, trilhas DD obviamente ocupam menos espaço de armazenamento (e de banda, quando transmitidas) e, por isso, a codificação DD tem sido preferida pelos fabricantes de DVD.

A.3.8 Compressão em Vídeo

Como vimos na Seção A.2, o sinal de vídeo pode ser codificado usando qualquer uma das técnicas lá mencionadas: PCM, DPCM, ADPCM, SBC etc. Também conforme vimos na Seção A.3.6, um vídeo pode ser considerado como uma sequência de imagens estáticas ou quadros. Cada um desses quadros pode ser codificado usando as mesmas técnicas empregadas para as imagens estáticas. Em particular, poderíamos empregar a codificação JPEG em cada quadro. Essa técnica constitui a base da codificação chamada MJPEG (*Motion JPEG*). Entretanto, ao empregarmos essa codificação, estaremos levando em conta apenas a redundância de informação contida em

um quadro (redundância *intraquadro*), quando a maior redundância pode estar nas informações contidas em quadros consecutivos (redundância *interquadros*).

Nesta seção nos deteremos na análise de dois padrões de codificação de vídeo que levam em conta não apenas a redundância intraquadro, mas também a existente interquadros: os padrões H.261 e MPEG vídeo. Antes, porém, vamos analisar alguns padrões de sinal de vídeo.

Sinais de TV são, em geral, adquiridos no formato RGB, mas transmitidos no formato YC_rC_b, no qual a resolução geométrica dos canais de crominância é menor que a de luminância, levando em conta a maior sensibilidade do olho humano à luminância, como discutimos na Seção A.3.6. Os sinais são então multiplexados e modulados, gerando um sinal chamado vídeo composto modulado, conforme ilustrado na Figura A.14.

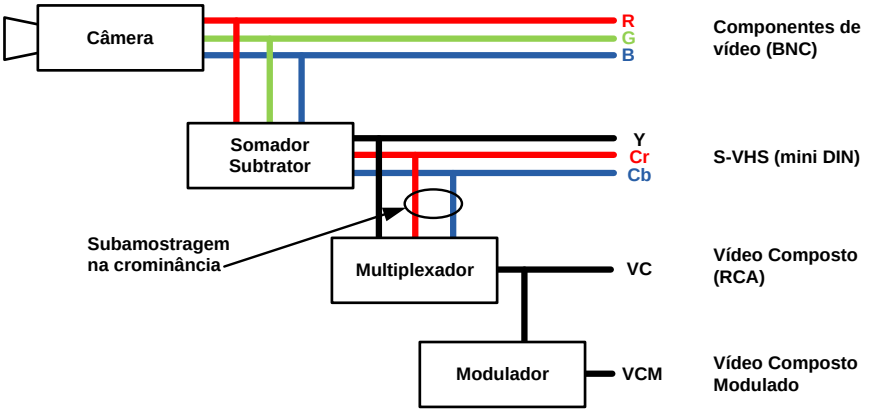


Figura A.14 Geração de sinal de vídeo de TV.

Sistemas de vídeo apresentam informações como uma sequência de quadros, sendo cada quadro composto de linhas. Um dos sistemas de distribuição de televisão mais utilizados usa 525 linhas por quadro, a uma taxa de 30 quadros por segundo (é o padrão M de TV, utilizado tanto pelo padrão americano NTSC quanto pelo padrão brasileiro PAL-M).[7] As televisões têm uma relação de aspecto de 4:3, dando ao padrão M uma resolução para a luminância de 700 × 525 pixels por quadro.

Nem todas as linhas da televisão são visíveis. A maioria dos formatos de vídeo digital está relacionada com a área visível para cada padrão de

[7] Na verdade, a taxa de quadros é de 29,97 quadros por segundo para TV em cores. Os padrões europeus, em geral, usam 25 quadrospor segundo e 625 linhas por quadro.

televisão. A recomendação BT 601-4 [ITU-R BT.601-4, 1994] especifica 483 linhas ativas, com 720 pixels por linha. Apenas 480 das 483 linhas e 704 dos 720 pixels (os primeiros e últimos 8 pixels são descartados) são usados para codificação. O padrão especifica a subamostragem de crominância 4:2:2, conforme dado pela Figura A.15, indicando que, para cada dois valores de luminância na horizontal, apenas um de cada crominância deve ser gerado. Isso implica uma taxa gerada de:

$$704 \times 480 \times 29{,}97 \times 16 = 162 \text{ Mbps}$$

A Figura A.15 apresenta também outras subamostragens de crominância utilizadas em outros padrões de codificação, como veremos.

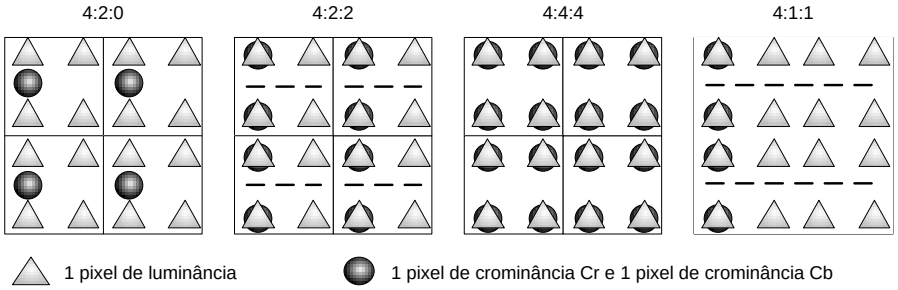


Figura A.15 Subamostragem de crominância MPEG 2.

O padrão H.261 do ITU-T, discutido mais adiante, usa os formatos CIF (*Common Interchange Format*), com 288 linhas e 352 pixels por linha, e QCIF (*Quarter CIF*), com 144 linhas e 176 pixels por linha, para a luminância. Sua extensão, H.263, acrescenta três novos formatos: o sub-QCIF (128 × 96), o 4CIF (704 × 576), também conhecido como SCIF, e o 16CIF (1408 × 1152). Todos os formatos citados possuem subamostragem de crominância (relação de aspecto) 4:2:0, conforme ilustrado na Figura A.15.

O padrão MPEG 1 vídeo pode codificar imagens de até 4.096 linhas × 4.096 pixels × 60 quadros por segundo. No entanto, a maioria das aplicações usa o formato SIF, com 240 linhas, 352 pixels por linha e subamostragem de crominância 4:2:0.

O padrão MPEG 2 pode codificar imagens de até 16.383 linhas × 16.383 pixels. O padrão é organizado tal qual o padrão MPEG-4, como veremos, em diversos perfis e níveis, que especificam o formato utilizado. Exemplos de formato são: nível baixo (240 linhas × 352 pixels por linha × 30 quadros por segundo — idêntico ao SIF MPEG 1), nível principal, visando à codificação com qualidade de TV (720 × 480 × 30), e os níveis altos, visando à TV de alta resolução — HDTV e à produção de filmes (em geral, 1280 ×

720 × 30; 1920 × 1080 × 30 ou 1440 × 1152 × 30). O padrão permite subamostragem de crominância 4:2:0, 4:2:2 e 4:4:4.

Note que vários formatos são menores que os tamanhos de TV atuais. Note também que as imagens de TV são significativamente menores do que as telas atuais das estações de trabalho. Quase todos os formatos de vídeo digital apresentam a imagem em uma área menor do que a tela da estação. Alguns padrões, no entanto, chegam a resoluções suficientes para atender à qualidade das TVs de alta resolução, a HDTV.

H.261

H.261 [ITU-T H.261, 1993] é o padrão de compressão mais usado em sistemas de videoconferência. Seu objetivo inicial foram as aplicações para redes comutadas por circuito, mais especificamente RDSI-FE (Redes Digitais de Serviços Integrados de Faixa Estreita). Daí decorre sua geração de tráfego CBR (*Constant Bit Rate*, isto é, taxa constante) nas taxas de $p \times 64$ Kbps, p variando de 1 a 30.

H.261 divide cada quadro (QCIF ou CIF) em macroblocos de 16 × 16 pixels, gerando seis blocos de 8 × 8 pixels (4 de luminância e 2 de crominância), conforme ilustra a Figura A.16.

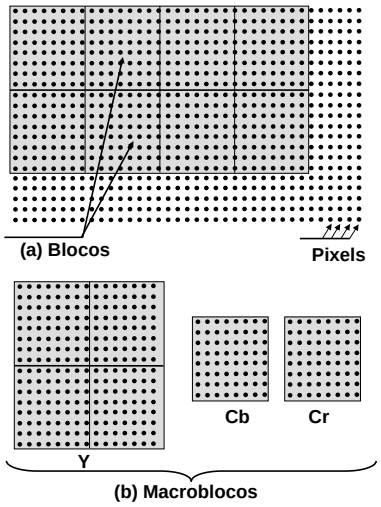


Figura A.16 Blocos H.261.

O padrão define dois tipos de codificação. Na codificação intraquadro, o macrobloco é codificado levando em conta apenas a redundância espacial do bloco. Cada bloco passa por um processo muito parecido com o descrito no

JPEG modo sequencial, gerando os blocos codificados. Na codificação preditiva, é realizada uma pesquisa no quadro anterior à procura de um macrobloco, o mais parecido possível com o macrobloco que se quer codificar (a pesquisa é realizada apenas na componente de luminância e apenas em uma região que circunda o macrobloco). O erro de predição (diferença entre os macroblocos) é então enviado para codificação, sofrendo o mesmo processo descrito para a codificação intraquadro. No caso da codificação preditiva, deve também ser codificado o vetor (chamado de vetor de deslocamento), que especifica o deslocamento entre o macrobloco corrente e o macrobloco do quadro anterior usado para a predição.

Todo macrobloco sofre uma codificação intraquadro e preditiva. A que gera a maior compressão é escolhida. Uma vez codificados, os quadros são enviados a um buffer que vai regular o fluxo de informação para uma taxa de bits constante. Lembre-se de que o H.261 foi pensado para uma rede comutada por circuitos. Como a taxa de entrada no buffer é VBR (*Variable Bit Rate*, isto é, taxa variável), se não fosse tomada nenhuma providência poderia ocorrer estouro de buffer ou falta de bits codificados. Para que isso não aconteça, o tamanho do passo do quantizador (o quantum), dos coeficientes gerados a partir da transformada de cossenos, é ajustado, quando necessário, conforme a quantidade de dados no buffer, para se chegar à taxa CBR desejada de saída.

O ajuste no passo do quantizador afeta a qualidade do vídeo gerado, privilegiando os vídeos com poucas mudanças de cena. Pouca mudança de cena implica maior compressão, isto é, menos bits gerados na codificação que entra no buffer, o que faz o processo de realimentação diminuir o quantum para aumentar a quantidade de bits gerados. Como consequência da diminuição do quantum, tem-se uma imagem de melhor qualidade. Como em aplicações de videoconferência e videotelefonia não existem, em geral, muitas mudanças de cenas, o padrão é bem apropriado para elas.

O padrão H.263 [ITU-T H.263, 2005] estende o padrão H.261, introduzindo novos formatos de quadros, como discutimos anteriormente, otimizando o H.261 para codificação de vídeo a taxas inferiores a 64 Kbps e adicionando facilidades para maior qualidade e melhores serviços. Contudo, as idéias básicas de compressão são as mesmas.

MPEG-1 e MPEG-2 Vídeo

Idênticos ao H.261, os padrões MPEG-1 e MPEG-2 vídeo [ISO/IEC 11172-2, 1993 e ISO/IEC 13818-2, 2000] dividem um quadro em macroblocos, como apresentado na Figura A.16, válido também para o MPEG com amostragem de crominância 4:2:0.

Cada bloco pode ser codificado usando apenas a informação intraquadro, de forma idêntica ao que foi apresentado para o JPEG. Quadros em que todos os blocos são codificados dessa forma são denominados *quadros I*.

Um macrobloco pode também ser codificado de forma preditiva, semelhante ao H.261. No entanto, a predição MPEG-1 é mais rica, uma vez que pode ser feita baseada em quadros passados e em quadros futuros da sequência de um vídeo. Quando a predição é feita baseada em um quadro passado, tal qual o H.261, é codificado o erro de predição (diferença do macrobloco que se quer codificar para o macrobloco de referência do quadro passado), usando os mesmos procedimentos usados para os macroblocos intraquadros e o vetor de movimento (que dá a posição relativa do macrobloco que se quer codificar para o macrobloco de referência do quadro passado). Quadros codificados usando esse tipo de predição são chamados *quadros P*. A predição é sempre baseada no primeiro quadro do tipo I ou P anterior.

A estimação do movimento (estimação do macrobloco mais próximo daquele que se quer codificar) se dá dentro de uma região do quadro de referência, conforme ilustra a Figura A.17. Para TV, por exemplo, obtém-se bom desempenho com p = 15 pixels em cenas comuns de noticiário e p = 63 em cenas de muito movimento, como por exemplo, cenas de esporte.

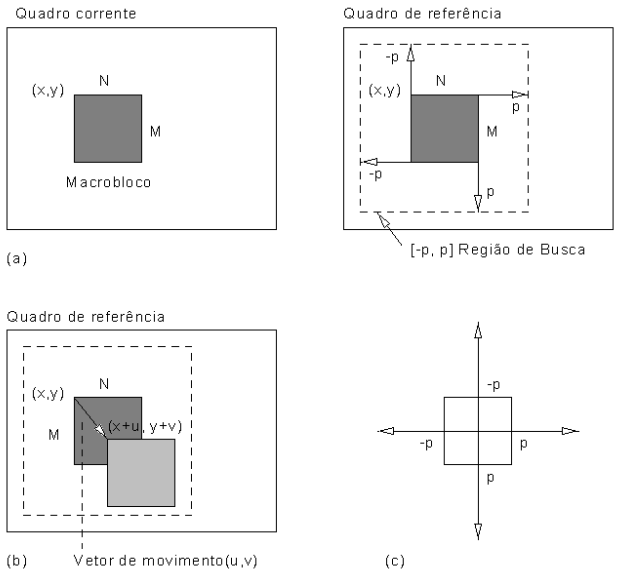


Figura A.17 Estimação de movimento.

Macroblocos também podem ser codificados com base no primeiro quadro I ou P, posterior ou anterior. Nesse caso, teremos dois quadros de referência para a procura do melhor casamento. A codificação pode ser realizada com base no quadro anterior no quadro posterior, ou na interpolação do quadro anterior e posterior. Quadros codificados usando esse tipo de predição são chamados *quadros B*.

Quadros B apresentam como vantagem o fato de, normalmente, apresentarem uma compressão maior que os quadros I e P (Tabela A.4). São também quadros que, se perdidos, não afetam tanto a qualidade da imagem, pois o erro cometido não se propagará, uma vez que esses quadros não são usados como referência de predição (note que a perda de um quadro I ou P implica a perda de todos os quadros até o próximo quadro I). No entanto, quadros B introduzem um retardo extra no processo de codificação porque devem ser codificados fora da sequência, além de exigirem mais processamento para codificação.

Tabela A.4 Tamanhos típicos de quadros MPEG 1

Tipo de Quadro	Tamanho (Kbytes)	Compressão
I	18	2:1
P	6	7:1
B	2,5	50:1
Média	4,8	27:1

Ao contrário do H.261, que escolhe sempre a melhor codificação para o macrobloco, no MPEG o padrão de codificação de quadros é um parâmetro da escolha do usuário, isto é, o usuário escolhe qual a sequência de quadros I, P ou B a ser utilizada. Note que essa escolha vai depender das aplicações. Por exemplo, como veremos na Seção A.4.3, o retardo de transferência pode ser crítico em aplicações em que há interatividade em tempo real entre os participantes, como em um sistema de videoconferência. Nesse caso, o número de quadros B em sequência não pode ser muito grande, devido ao retardo que isso introduz.

O padrão MPEG-1 trabalha, em geral, com o formato SIF, embora maior resolução também seja permitida por sua sintaxe. O padrão MPEG-2, como já mencionado, admite vários formatos de quadros e diferentes resoluções para as componentes de crominância. De fato, o MPEG-2 especifica vários conjuntos de parâmetros de restrição, que são definidos nos

seus *perfis* e *níveis*. Um perfil especifica as facilidades de codificação que serão utilizadas. Um nível especifica a resolução dos quadros, as taxas de bits etc. Várias combinações de perfis e níveis foram definidas, como veremos mais adiante. O MPEG-2 usa os mesmos tipos de quadro I, P e B, como o MPEG-1, mas introduz outros métodos de predição [Netravali *et al*., 1995] para lidar com vídeo entrelaçado.[8]

O MPEG-2 também apresenta várias extensões de escalabilidade [ISO/IEC 13818-2, 2000]. As extensões proveem, basicamente, duas ou mais sequências de bits, ou *camadas*, que podem ser combinadas para prover um único sinal de vídeo de alta qualidade. A *camada-base* pode, por definição, ser totalmente decodificada por si mesma, de forma a prover um vídeo de baixa qualidade. Como veremos a seguir, muitas das técnicas empregadas são semelhantes às codificações progressivas e hierárquica do JPEG.

Com a *escalabilidade SNR* (*Signal to Noise Ratio*), outra camada pode ser adicionada à camada-base, oferecendo melhoria na precisão dos coeficientes da DCT (*Discrete Cosine Transform*), adicionando valores de correção para serem utilizados antes da decodificação (aplicação da DCT inversa). Essa extensão também provê a codificação do vídeo na resolução 4:2:2, tendo por camada-base o vídeo na resolução 4:2:0.

Escalabilidade por partição de dados é uma versão simplificada da escalabilidade SNR. Com essa extensão, até um certo número de coeficientes de frequências DCT é enviado pela camada-base. Os coeficientes restantes são enviados por outra camada a ser adicionada à básica.

Na *escalabilidade espacial*, a camada-base tem uma resolução espacial (resolução geométrica de cada imagem) menor do que a do vídeo original. A camada de melhoramento é então acrescida à camada-base para se obter a resolução original.

Na *escalabilidade temporal*, a camada-base tem uma resolução temporal (número de quadros por segundo) menor do que a do vídeo original. Novamente, a camada de melhoramento é adicionada à camada-base para a obtenção da resolução original.

Escalabilidade é especialmente útil em redes que permitem distinguir os tipos de fluxos e privilegiar a entrega do mais importante. Assim, em caso de necessidade ou conveniência de perda, um sinal de vídeo com um mínimo de qualidade ainda poderá ser recebido.

[8] Vídeo entrelaçado é tipicamente usado em TV, onde são primeiro varridas as linhas ímpares e depois as pares. Aos conjuntos de linhas ímpares e pares chamamos campos. Assim, um quadro é composto de dois campos.

No MPEG-2, o perfil principal (*main profile*) utiliza os quadros I, P e B e uma amostragem de cor 4:2:0. O perfil simples (*simple profile*) é basicamente o perfil principal sem os quadros B. O perfil escalável SNR (*SNR Profile*) adiciona a escalabilidade SNR ao perfil principal. O perfil escalável espacialmente (*spatially scalable profile*) adiciona a escalabilidade espacial ao perfil escalável SNR. O perfil alto adiciona cor 4:2:2 ao perfil escalável espacialmente. Todos os perfis são limitados ao máximo de três camadas. O nível principal (*main level*), como mencionado no início desta seção, está definido basicamente para o vídeo ITU-R Rec. 601. O nível simples (*simple level*) é definido para vídeo SIF. Os dois níveis altos para HDTV são o nível alto 1440 (*high-1440 level*), com um máximo de 1.440 pixels por linha, e o nível alto (*high level*) com no máximo 1.920 pixels por linha. Nem todas as combinações de perfis e níveis foram padronizadas. No futuro, quando necessário, perfis e níveis poderão ser adicionados ao padrão.

MPEG-4 Vídeo

O MPEG 4 (ISO/IEC 14496) foi finalizado em outubro de 1998 e tornou-se um padrão internacional nos primeiros meses de 1999. No final de 1999 foram acrescentadas novas extensões (MPEG-4 versão 2), tornando-se um padrão internacional formal no começo de 2000.

O padrão em sua forma atual é dividido em várias partes, entre elas:

- Parte 1— Sistema
- Parte 2 — Vídeo
- Parte 3 — Áudio
- Parte 10 — Codificação Avançada de Vídeo

Das diversas partes, em termos de codificação de vídeo, interessa-nos especialmente a Parte 10, por incorporar todas as melhorias trazidas pelo MPEG-4 com relação ao padrão MPEG-2.

Em 2001, o grupo de trabalho da especificação MPEG da ISO reconheceu o potencial dos trabalhos desenvolvidos pelo grupo H.26L (VCEG — *Video Coding Expert Group*) do ITU-T, como evolução dos trabalhos do H.263, e a partir de então os trabalhos dos dois grupos se fundiram (JVT — *Joint Video Team*) para a definição de um novo padrão de codificação de vídeo. O resultado foram dois padrões idênticos: ISO MPEG-4 Parte 10 [ISO/IEC 14496-10, 2005] e ITU-T H.264, oficialmente conhecidos como AVC (*Advanced Video Coding*), publicado pela primeira vez em 2003.

Como o MPEG-2, o padrão é dividido em perfis. O perfil *baseline* visa a fluxos de vídeo simples (de baixa resolução); por exemplo, para videotelefonia; o perfil principal visa à qualidade de TV de definição padrão; o perfil alto (perfil estendido) visa a fluxos de vídeo de alta definição. Esse último perfil é subdividido em quatro perfis: o perfil alto (para o suporte a vídeo de 8 bits por amostra com subamostragem de crominância 4:2:0); o perfil alto 10 (para o suporte a vídeo de até 10 bits por amostra com subamostragem de crominância 4:2:0); o perfil alto 4:2:2 (para o suporte a vídeo de até 10 bits por amostra com subamostragem de crominância 4:2:2) e o perfil alto 4:4:4 (para o suporte a vídeo de até 12 bits por amostra com subamostragem de crominância 4:4:4 e transformada de cor residual inteira para codificação de sinal RGB).

Os elementos funcionais básicos (predição, transformada, quantização, codificação por entropia) do AVC têm poucas diferenças com relação aos padrões anteriores (MPEG-1, MPEG-2, MPEG-4 Parte 2, H.261, H.263). As mudanças importantes ocorrem em detalhes de cada um dos elementos.

O processo de codificação processa quadros de vídeo em unidades de macroblocos (16×16 pixels). Ele forma a predição do macrobloco baseada em dados já previamente codificados, ou do quadro corrente (predição intra) ou de quadros já codificados e transmitidos (predição inter).

O método de predição AVC é mais flexível do que os dos padrões por nós anteriormente discutidos, permitindo uma predição mais precisa e, assim, uma melhor compressão. Os blocos de predição são de tamanhos variáveis. A predição intra pode usar blocos de tamanho 16×16 ou 4×4. A predição inter pode usar blocos 16×16, 16×8, 8×16, 8×8, 8×4, 4×8 ou 4×4.

Diferentemente do MPEG-1, do MPEG-2 e do MPEG-4 Parte 2, a predição de cada macrobloco pode ser baseada em um ou dois quadros quaisquers, passados ou futuros, que já foram codificados. Essa possibilidade é extremamente importante quando o movimento na cena é periódico.

Um bloco, obtido após a predição, é transformado usando uma transformada inteira 4×4 ou 8×8 (a transformada inteira é uma forma aproximada da transformada discreta de cosseno).

Os valores e parâmetros resultantes (elementos sintáticos) são convertidos em código binário usando codificação por carreia e/ou codificação aritmética, também com algumas melhoras incorporadas.

A Figura A.18 ilustra o processo de codificação, e a Figura A.19, o processo de decodificação .

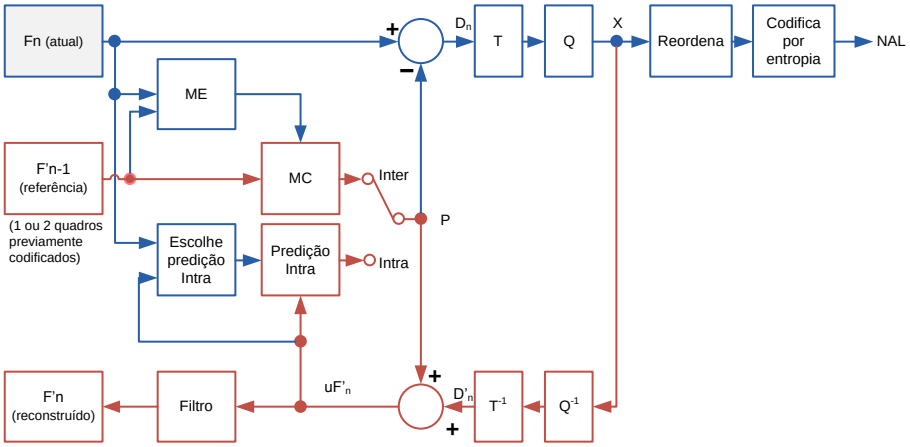


Figura A.18 Processo de codificação AVC.

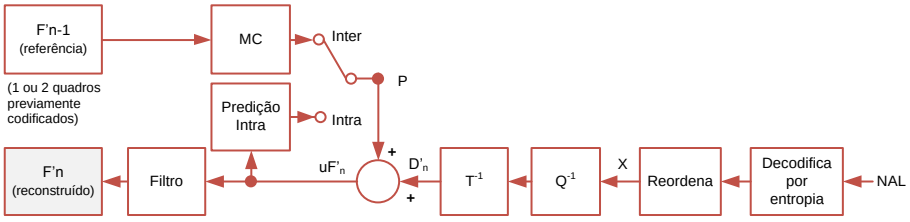


Figura A.19 Processo de decodificação AVC.

A.3.9 Multiplexação e Sincronização

Tanto o H.261 quanto o MPEG padronizam a forma como informações audiovisuais devem ser multiplexadas (unidas em um único fluxo). No caso do MPEG, a padronização *MPEG System* [ISO/IEC 11172-1, 1993 e ISO/IEC 13818-1, 2000] é responsável por essa especificação, como mencionamos no Capítulo 1. Ambos os padrões adicionam aos fluxos elementares de áudio e vídeo informações para suas exibições sincronizadas. A sincronização é realizada pela adição de selos de tempo (*timestamps*) a conjuntos de amostras codificadas de vídeo e áudio, baseadas em um relógio compartilhado. A Figura A.20 ilustra o procedimento.

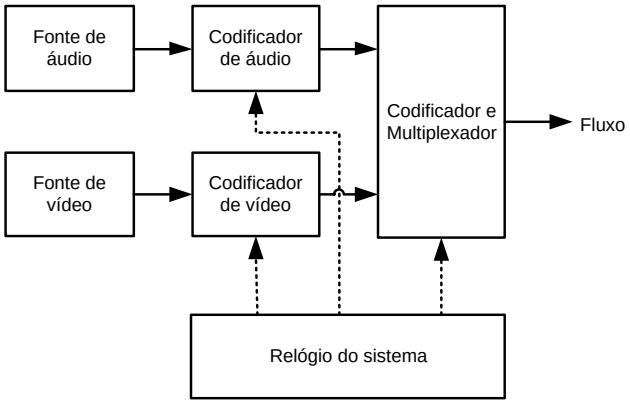


Figura A.20 Multiplexação e sincronização dos fluxos de áudio e vídeo.

Um fluxo MPEG-1 é organizado em duas camadas: a camada *pack* e a camada *packet*. A camada *pack* contém informações utilizadas por todos os fluxos elementares, e a camada *packet*, as informações particulares a cada fluxo. Um fluxo MPEG consiste em um ou mais *packs*. O cabeçalho *pack* contém informações de temporização do sistema e sobre as taxas de dados. O cabeçalho *packet* contém a identificação do fluxo elementar, os requisitos de armazenamento e informações de temporização. Os dados *packet* contêm um número de bytes variável de um mesmo fluxo elementar. Assim, depois de remover o cabeçalho *packet*, os dados *packet* de todos os *packets* com o mesmo identificador são concatenados para a recuperação de um fluxo elementar. Até 32 fluxos de áudio e 16 fluxos de vídeo podem ser multiplexados em um fluxo MPEG-1. A Figura A.21 apresenta a estrutura de camadas MPEG-1 System [ISO/IEC 11172-1, 1993].

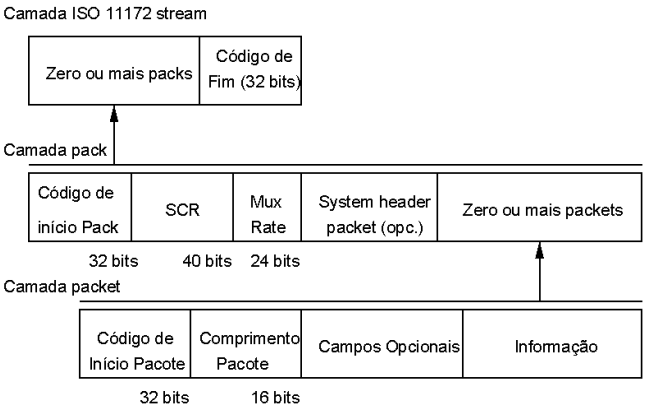


Figura A.21 Camadas MPEG-1 System.

A função do MPEG-2 System [ISO/IEC 13818-1, 2000] é idêntica à do MPEG-1. Contudo, o MPEG-2 System especifica dois formatos de dados: o fluxo de programa (*program stream*) e o fluxo de transporte (*transport stream*) (Figura A.22), conforme já mencionamos no Capítulo 1. O fluxo de programa é similar e compatível com o fluxo MPEG-1 System. Ele foi otimizado para aplicações multimídia e para ser processado por software.

O fluxo de transporte pode transportar múltiplos programas simultaneamente e é otimizado para aplicações nas quais a perda de dados é comum. O fluxo de transporte consiste em pacotes de tamanho fixo (188 bytes, incluindo 4 bytes de cabeçalho).

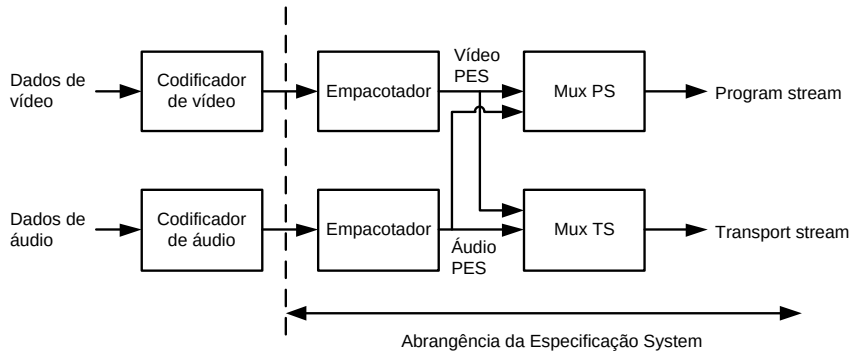


Figura A.22 MPEG-2 System.

Similar ao MPEG-1 e MPEG-2, o MPEG-4 System [ISO/IEC 14496-1, 2001] é desenvolvido para fornecer multiplexação de fluxos de dados elementares, sincronização e empacotamento. Adicionalmente, o MPEG 4 System fornece parâmetros de representação/manipulação básicos (translação, rotação e zoom) no cabeçalho da camada de fluxo de dados de cada objeto.

Diferentemente da codificação linear de áudio e vídeo do MPEG-1 e MPEG-2, a codificação MPEG 4 é, no entanto, baseada em objetos, isto é, as cenas audiovisuais são codificadas em termos de objetos. Um objeto pode ser uma imagem, um vídeo ou um áudio.

Objetos codificados separadamente fornecem três benefícios: reusabilidade — a abordagem orientada a objeto permite aos autores reusarem material audiovisual mais rapidamente; escalabilidade — objetos podem ser codificados usando diferentes resoluções espaciais e temporais (a resolução do objeto pode ser ajustada para casar com a capacidade do meio de transporte); interatividade — porque os objetos audiovisuais são compostos em quadros no decodificador, o usuário pode controlar a saída.

O MPEG-4 considera uma cena como sendo composta de *objetos de vídeo (Video Objects) — VOs*. Os VOs têm propriedades como forma, movimento, textura etc. Eles vão constituir as entidades no fluxo de bits que o usuário pode manipular e ter acesso. Um *plano de objetos de vídeo* (*Video Object Plane* — VOP) é uma ocorrência de um VO em dado instante de tempo. Cada quadro consiste em vários VOPs. Uma cena que contém somente um VOP pode corresponder aos padrões correntes, tais como MPEG-1, MPEG-2 e MPEG-4. Cada VOP tem sua própria resolução espacial e temporal.

Uma cena dividida em objetos, como mencionado, possui uma organização hierárquica. Uma informação adicional é enviada com os VOPs a fim de informar ao receptor como compor a cena. A descrição de cada cena baseia-se em conceitos da *Virtual Reality Modeling Language (*VRML — linguagem de modelagem de realidade virtual). Contudo, o padrão MPEG 4 introduziu novos conceitos de modelagem e otimizou os já existentes, dando origem a uma linguagem diferente e mais poderosa: *Binary Format for Scenes* (BIFS).

Note que, ao dividir uma cena em objetos e relacioná-los utilizando uma linguagem de descrição de cenas, o padrão MPEG-4 está de fato tentando padronizar uma linguagem que pode ser usada na descrição de relacionamentos para a transmissão de dados assíncronos, conforme discutimos na Seção 1.2.3.1 do Capítulo 1. Naquele capítulo comentamos que o padrão MPEG-2 System, utilizado em todos os principais sistemas de televisão digital terrestre, especificava os vários tipos de serviço, mas não especificava a linguagem usada para sincronização no serviço assíncrono. O padrão MPEG-4 vai um passo adiante na especificação do BIFS. Mais recentemente discute-se dentro do MPEG-4 a adoção de uma linguagem mais eficiente e de mais alto nível (MPEG-4 Parte 20) dentro da representação denominada LASeR (*Lightweight Application Scene Representation*) [ISO/IEC 14496-20, 2006].

A.4 Requisitos de Comunicação das Diversas Mídias

As características e requisitos de comunicação exigidos pelos diversos tipos de mídia são muito diferentes. Várias características devem ser consideradas ao classificarmos fontes de tráfego. A *natureza* do tráfego gerado é uma de suas características mais importantes, dando origem a três classes básicas: a classe de tráfego contínuo com taxa constante (*Constant Bit Rate* — CBR), a classe de tráfego em rajadas (*bursty*) e a classe de tráfego contínuo com taxa variável (*Variable Bit Rate* — VBR).

Na *classe de tráfego contínuo com taxa constante*,[9] o tráfego, como o próprio nome diz, é constante e, por conseguinte, sua taxa média é igual à sua taxa de pico. Essa taxa é o único parâmetro necessário para caracterizar tal fonte.

As fontes cujo tráfego gerado tem característica de *rajadas* apresentam períodos ativos (durante os quais há geração de informação pela fonte, que opera na sua taxa de pico) intercalados por períodos de inatividade (durante os quais a fonte não produz tráfego algum). Para caracterizar uma fonte com tráfego em rajadas não é suficiente utilizarmos a taxa média de geração de informação, já que essa taxa não representa corretamente o seu comportamento. A taxa média nem sequer representa uma taxa na qual a fonte opera em algum momento. Muito mais significativas são informações sobre a distribuição das rajadas ao longo do tempo, a duração das rajadas e a taxa de pico atingida durante as rajadas. Alguns parâmetros comumente utilizados para caracterização desse tipo de tráfego incluem a *duração média dos períodos de atividade* e a *explosividade* (*burstiness*) da fonte — a razão entre a taxa de pico e a taxa média de utilização do canal.[10]

Por fim, as *fontes de tráfego contínuo com taxa variável* apresentam variações na taxa de transmissão ao longo do tempo. Parâmetros como a média e a variância da taxa de transmissão podem ser utilizados para caracterizar o comportamento de fontes com essa característica. O parâmetro de explosividade (*burstiness*) também é bastante utilizado na caracterização dessas fontes.

Requisitos sobre a qualidade do serviço de comunicação desejado (QoS), tais como retardo máximo de transferência, variação estatística do retardo (*jitter*), vazão média, taxas aceitáveis de erro de bit e de pacote de dados, variam muito de uma mídia para outra e são dependentes da aplicação. De forma geral, podemos caracterizar as diversas mídias, quanto aos requisitos de comunicação exigidos, como se segue.

A.4.1 Texto

O tráfego gerado por informações em texto é, em sua grande maioria, de rajada. Para compreender essa natureza do tráfego, pense na comunicação de um terminal com um computador durante a execução interativa de um programa. A vazão média dos dados vai depender muito da aplicação,

---

[9] Em geral, os padrões de comunicação utilizam a palavra *contínuo* para caracterizar *sem interrupção* e a *taxas constantes*. Note, no entanto, que temos, além do tráfego em rajadas, o tráfego sem interrupção mas com taxa variável. Em geral, os padrões chamam apenas de *tráfego com taxa variável* (VBR) a ambos os tráfegos (em rajadas e contínuo com taxa variável), independentemente de serem sem interrupção ou não.

[10] Existem outras definições para a medida da explosividade da fonte: a razão entre o desvio padrão e a taxa média gerada, por exemplo.

variando desde alguns poucos bits por segundo para aplicações de correio eletrônico, até alguns megabits por segundo em transferência de arquivos. Para texto, excetuando-se algumas aplicações em tempo real, como por exemplo para controle de processos críticos, o retardo máximo de transferência e a variação estatística do retardo não constituem problemas, sendo seus requisitos, em geral, facilmente satisfeitos pelo sistema de comunicação. Quanto à tolerância a erros, na grande maioria das aplicações não se pode tolerar erro nem em um único bit: suponha, por exemplo, o caso da perda de um bit numa transferência eletrônica de fundos.

A.4.2 Imagem

O tráfego gerado em aplicações gráficas animadas, onde vários quadros são gerados em intervalos regulares de tempo, tem características bem semelhantes às da mídia de vídeo, comentadas mais à frente. Excetuando o caso de imagens animadas, a natureza do tráfego gerado pela mídia gráfica também é de rajadas, com vazões médias chegando a algumas dezenas de megabits por segundo. Como em textos, o retardo máximo e a variação estatística do retardo, em geral, não são relevantes.

Como discutido na Seção A.1, as imagens gráficas podem estar no formato vetorial ou matricial. Para imagens no formato matricial e sem compressão, a taxa de erro de bit pode ser bem maior que a taxa de erro de pacote, uma vez que, em geral, não haverá nenhum problema se, por exemplo, um único pixel de uma tela ficar azul em vez de verde. O mesmo não se pode dizer da perda de um pacote, que poderá, por exemplo, apagar um bloco da imagem na tela. Para imagens no formato vetorial e imagens (vetoriais ou matriciais) em que foram utilizadas técnicas de compressão ou compactação, a tolerância à perda depende muito da aplicação e de seus usuários. Como discutimos na Seção A.3.6, existem métodos de compressão que identificam a porção mais importante dos dados de uma imagem. Para esses dados, deve-se evitar ao máximo as perdas. As porções menos importantes podem ser descartadas, se necessário (seja por erro na transmissão, por congestionamento no sistema de comunicação ou mesmo porque o usuário final não necessita delas para obter a informação que deseja). Um sistema de comunicação deve poder identificar as porções que ele deve manter íntegras. Outro caso importante, com relação às perdas, são as imagens que não são processadas somente pelo olho humano, mas também pelo computador como, por exemplo, imagens médicas ou cartográficas. Nesse caso, a perda de um único bit (seja devido à comunicação ou ao método de compressão) pode ser intolerável (imagine uma doença que se quer diagnosticar através de uma imagem médica).

A.4.3 Áudio

A mídia de áudio tem características bem distintas das mencionadas nos dois parágrafos anteriores, principalmente em aplicações de tempo real com interatividade, como os serviços conversacionais do ITU-T. Começando pela natureza do tráfego gerado, a mídia de áudio se caracteriza por gerar um tráfego contínuo com taxa constante. Mesmo quando, no sinal de voz, é realizada a compactação por detecção de silêncio, por exemplo, passando a se caracterizar como um tráfego de rajada [Gruber, 1982], ele deve ser reproduzido no destino a uma taxa constante. O tráfego gerado para comunicação dessa mídia é do tipo CBR, caso não seja empregada nenhuma técnica de compactação ou compressão. Em caso contrário, o tráfego se caracteriza como VBR e, às vezes, como no caso da voz com detecção de silêncio, como um tráfego em rajadas.

A vazão média gerada pela mídia de áudio depende da qualidade do sinal, da codificação e compactação ou compressão utilizadas. Para sinais de voz, por exemplo, já apresentamos a técnica PCM, que gera 64 Kbps se utilizarmos 8 bits para codificar cada amostra (tomada a cada 125 μseg, isto é, 8.000 amostras por segundo). Com qualidade aproximadamente igual, a codificação ADPCM gera 32 Kbps. Sinais de áudio de alta qualidade (qualidade de CD estéreo, por exemplo) geram taxas bem superiores, como, por exemplo, os CDs de áudio, onde a taxa é de 1,411 Mbps, como vimos na Seção A.2.1.

Quanto às perdas, as taxas de erros de bits ou de pacotes podem ser relativamente altas, devido ao alto grau de redundância presente nos sinais de áudio. O único requisito é que os pacotes não sejam muito grandes (no caso da voz, menores que uma sílaba), o que normalmente já é satisfeito para não se perder tempo no empacotamento e, assim, não aumentar o retardo de transferência. Perdas da ordem de 1% da informação de voz são toleráveis[11] [Gopal, 1984; Gruber, 1985]. Uma vez que as redes de comunicação utilizam, hoje em dia, meios físicos de alta confiabilidade, a detecção de erros para a voz nessas redes pode ser tranquilamente dispensada, em benefício de um maior desempenho. Apesar da baixa taxa de erros das redes, por exemplo em fibra ótica, nas mídias gráfica e de texto a detecção de erros ainda é, na maioria das vezes, necessária, e em alguns casos até a detecção e a correção. Um cuidado adicional deve ser tomado quando, devido às técnicas de compressão utilizadas no áudio, um erro pode se propagar para outros bits. Nesse caso, o erro pode ser intolerável. Ainda com respeito ao áudio, porções da informação podem ser diferenciadas quanto à tolerância às perdas. No caso da voz, por exemplo, perdas nos intervalos de silêncio são muito mais

[11] Na realidade, como já comentamos, a porcentagem de perda depende do tamanho do surto de voz e se a perda ocorre no início ou no meio do surto.

toleráveis do que perdas durante os surtos de voz. Um sistema de comunicação deve poder identificar as porções mais sensíveis a perdas, caso seja necessário o descarte de dados.

No caso da utilização de sistemas de comunicação que apresentam variação estatística do retardo (como as redes comutadas por pacotes em um sistema de WebTV), tal variação deve ser compensada. Para entendermos melhor o problema, analisemos a Figura A.23.

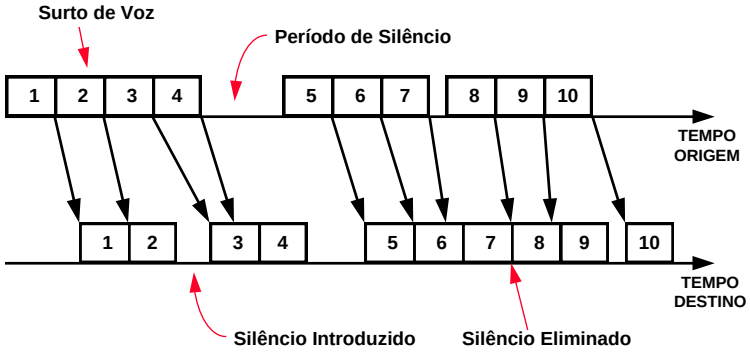


Figura A.23 Efeito da variação estatística do retardo na comunicação de voz.

Na linha horizontal superior vemos os surtos de voz e de silêncio sendo gerados na fonte a uma taxa constante. Os surtos de voz são divididos em pacotes, que são as unidades que transitarão no sistema de comunicação (os surtos de silêncio não são transmitidos). Uma vez que o pacote é gerado, ele é imediatamente entregue para transmissão (veremos a seguir que o retardo máximo é um requisito importante). Se os pacotes sofrerem retardos variáveis, chegarão ao destino não mais preservando a continuidade, conforme mostra a linha horizontal inferior da figura, podendo gerar intervalos de silêncio dentro de um surto de voz ou diminuir, e até mesmo eliminar, intervalos de silêncio, o que pode causar a perda da inteligibilidade da informação no destino. Alguma forma de compensação dessa variação estatística do retardo deve ser realizada.[12] A estratégia utilizada pelos algoritmos de compensação baseia-se fundamentalmente em assegurar uma

[12] Note que a variação estatística do retardo não é necessariamente introduzida só pela rede de comunicação, mas por todo o sistema. Ela é introduzida desde a interação da placa de áudio com o sistema operacional da estação, passando pelos protocolos de comunicação (sistema operacional de rede), até chegar ao sistema de transmissão. No destino, o caminho semelhante, mas em ordem inversa, também pode introduzir aleatoriedade no retardo antes da reprodução. Assim, embora muitas vezes o sistema de transmissão não introduza aleatoriedade no retardo, a compensação ainda deve ser feita.

reserva de pacotes antes de dar início ao processo de reprodução, introduzindo um retardo inicial a cada surto de voz. Aparentemente, o problema estaria resolvido se escolhêssemos o retardo inicial bem grande; entretanto, o valor desse retardo está limitado pelo máximo retardo de transferência (desde a geração até a reprodução) permitido para o sinal de voz, sem que haja perda da interatividade da comunicação [Bastos *et al.*, 1992]. As referências [Soares *et al.*, 1991; Bastos *et al*.. 1992; Soares *et al*.. 1992; Gopal. 1984; Adams *et al*.. 1985; Faria. 1992] discutem com algum detalhe a análise de desempenho de vários algoritmos para compensação da variação estatística de retardo, com o objetivo de manter a continuidade em sinais de voz. Embora apresentados para sinais de voz, os algoritmos podem ser facilmente estendidos para qualquer sinal contínuo (com taxa constante ou variável).

O retardo de transferência máximo é crítico para o áudio, principalmente no caso de conversações. Um dos motivos é o problema do eco, mas, mesmo nos casos em que o eco não causa problemas ou que canceladores de eco sejam utilizados, o retardo de transferência máximo pode ser crítico. Cada interlocutor, em uma conversação, normalmente espera o fim do discurso do outro para dar início à sua fala; se o retardo de transferência for muito grande, a conversação começa a sentir um efeito de ruptura, podendo até se tornar inviável (se o leitor já utilizou a rede telefônica via satélite, deve ter sentido esse efeito). Um retardo de transferência maior que 200 ms já começa a incomodar os interlocutores [Bastos *et al.*, 1992]. Os padrões de telefonia estipulam 40 ms para distâncias continentais e 80 ms para distâncias intercontinentais como limites para o retardo máximo de transferência. É bom frisarmos novamente que os problemas de retardo só são críticos em aplicações que exigem comunicação interativa em tempo real. Nesse caso, como não podemos introduzir um retardo inicial muito grande para compensarmos a variação estatística do retardo, a compensação só poderá ser efetiva se a variação estatística apresentada for pequena.

A.4.4 Vídeo

Tal qual a mídia de áudio, a mídia de vídeo se caracteriza por gerar um tráfego contínuo com taxa constante. Da mesma forma que no áudio, mesmo quando no sinal é realizada alguma técnica de compactação ou compressão e o tráfego gerado para comunicação se caracterizar como um tráfego com taxas variáveis, o sinal deve ser reproduzido no destino a uma taxa constante. Como na mídia de áudio, o retardo de transferência máximo tem grande importância, e a variação estatística do retardo deve ser compensada. Normalmente, como o vídeo vem acompanhado de áudio (sincronizado com

áudio), uma vez obedecidos os requisitos de retardo deste, são obedecidos os daquele.

A vazão média gerada por uma fonte de vídeo varia com a qualidade do sinal e os algoritmos de codificação, compactação e compressão empregados, conforme discutido na Seção A.3.8.

Em vídeo, a taxa de erro de bit pode ser maior que a taxa de erro de pacote, pelos mesmos motivos explicitados para as imagens gráficas no formato matricial. No entanto, no vídeo, como a imagem não é estática e devem ser gerados vários quadros por segundo, a taxa de erro de pacote não é tão crítica. Mesmo a taxa de erro de bit tolerável é maior do que aquela para imagens estáticas [Hehmann *et al*., 1990]. Na verdade, a discussão sobre a taxa de erro aceitável não é tão simples. Quando utilizamos técnicas de compressão, um erro pode se propagar. Dessa forma, alguns quadros em que o erro não se propaga podem tolerar erros de bits e de pacotes. Naqueles em que o erro se propaga, às vezes até um único erro de bit pode ser intolerável. Tal qual nas imagens no formato matricial, quando se utilizam técnicas de compressão ou compactação, a tolerância à perda depende muito da aplicação e de seus usuários. Como discutimos na Seção A.3.8, existem métodos de compressão que identificam a porção mais importante dos dados de um vídeo. Para esses dados, deve-se evitar ao máximo as perdas, as porções menos importantes podem ser descartadas, se necessário (por erro na transmissão ou por congestionamento no sistema de comunicação, ou mesmo porque o usuário final não necessita delas para obter a informação que deseja). Mais uma vez, um sistema de comunicação deve poder identificar as porções em que deve minimizar as perdas.

Bibliografia:

ABNT NBR 15602-1 (2007). Associação Brasileira de Normas Técnicas, "Televisão digital terrestre — codificação de vídeo, áudio e multiplexação, Parte 1: Codificação de vídeo", Sistema Brasileiro de TV Digital Terrestre, *NBR 15602-1*.

Adam, C. e Ades, S. "Voice Experiments in the Universe Project." Proceedings of International Conference on Communications, 29.4.1–29.4.9, 1985.

Bastos, T.L.P. e Soares, L.F.G. "Análise de algoritmos para reprodução em tempo real de voz em redes de pacotes." *Relatório Técnico IBM CCR-141*, Rio de Janeiro. Janeiro, 1992.

Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. *Algoritmos*. Tradução da 2.ª ed. americana. Teoria e Prática. 2002.

Faria, A.L.A. "Implementação do mecanismo de controle de acesso por detecção de silêncio em um sistema de teleconferência." *Dissertação de Mestrado, Depto. de Engenharia Elétrica, PUC-Rio*. Março, 1992.

Gruber, J.G. "A Comparison of Mesure and Calculated Speech Temporal Parameters Relevant to Speech Activity Detection." *IEEE Transactions on Communications*, vol. com-30, n.4. Abril, 1982.

Gruber, J.G. "Subjective Effects of Variable Delay and Speech Loss in Dinamically Managed Voice Systems." *IEEE Transactions on Communications*, vol. com-33. Agosto, 1985.

Gopal, P.M., Wong, J.W. e Majithia, J.C. "Analysis of Playout Strategies for Voice Transmission Using Packet Switching Techniques." *Performance Evaluation*, n.4. Fevereiro, 1984.

Hehmann, D. B., M.G. Salmony, and H.J. Stuttgen. "Transport services for multimedia applications on broadband networks." *Computer Communications* Vol. 13 N.º 4, 1990, pp. 197-203.

ISO/IEC 10918-1 (1994). International Organization for Standardization/International Eletrotecnical Committee, "Digital Compression and Coding of Continuous-Tone Still Images, Part 1: Requirements and Guidelines", *ISO/IEC IS 10918-1*.

ISO/IEC 11172-1 (1993). International Organization for Standardization/International Eletrotecnical Committee. "Information Technology — Coding of Moving Pictures and Associated Digital Storage Media at up to About 1.5 Mbits/s, Part 1: Systems." *ISO/IEC IS 11172-1*.

ISO/IEC 11172-2 (1993). International Organization for Standardization/International Eletrotecnical Committee. "Information Technology — Coding of Moving Pictures and Associated Digital Storage Media at up to About 1.5 Mbits/s, Part 2: Video", *ISO/IEC IS 11172-2*.

ISO/IEC 11172-3 (1993). International Organization for Standardization/International Eletrotecnical Committee, "Information Technology — Coding of Moving Pictures and Associated Digital Storage Media at up to About 1.5 Mbits/s, Part 3: Audio", *ISO/IEC 11172-3*.

ISO/IEC 13818-1 (2000). International Organization for Standardization/International Eletrotecnical Committee, "Information Technology — Generic coding of moving pictures and associated audio information, Part 1: Systems", *ISO/IEC 13818-1*.

ISO/IEC 13818-2 (2000). International Organization for Standardization/International Eletrotecnical Committee, "Information

Technology — Generic coding of moving pictures and associated information, Part 2: Video", *ISO/IEC 13818-2*.

ISO/IEC 13818-3 (1998). International Organization for Standardization/International Eletrotecnical Committee. "Information Technology — Generic Coding of Moving Pictures and Associated Audio: Audio." *ISO/IEC JTC1/SC29/WG11 13818-3*.

ISO/IEC 13818-7 (1997). International Organization for Standardization/International Eletrotecnical Committee, "Information Technology — Generic coding of moving pictures and associated audio information, Part 7: Advanced Audio Coding (AAC)", *ISO/IEC 13818-7*.

ISO/IEC 14496-1 (2001). International Organization for Standardization/International Eletrotecnical Committee. "Coding of Audio-Visual Objects, Part 1: Systems." *ISO/IEC JTC1/SC29/WG11 14496-1*. 2001.

ISO/IEC 14496-2 (2001). International Organization for Standardization/International Eletrotecnical Committee. "Coding of Audio-Visual Objects, Part 2: Visual."*ISO/IEC JTC1/SC29/WG11 14496-2*. 2001.

ISO/IEC 14496-3 (2004). International Organization for Standardization/International Eletrotecnical Committee, "Information Technology — Coding of Audio-Visual Objects, Part 3: Audio", *ISO/IEC 14496-3*.

ISO/IEC 14496-10 (2005). International Organization for Standardization/International Eletrotecnical Committee, "Information Technology — Coding of Audio-Visual Objects, Part 10: Advanced Video", *ISO/IEC 14496-10*.

ISO/IEC 14496-20 (2005). International Organization for Standardization/International Eletrotecnical Committee, "Information Technology — Coding of Audio-Visual Objects, Part 20: Lightweight Application Scene Representation (LASeR) and Simple Aggregation Format (SAF)", *ISO/IEC 14496-20*.

ITU-R BT.601-4 (1994). International Communication Union "Encoding Parameters of Digital Television for Studios", *Recommendation BT.601-4*, BT Series Volume, Geneva.

ITU-T G.711 (1988). International Communication Union. "Pulse Code Modulation of Voice Frequencies." *Recommendation G.711*. Geneva.

ITU-T G.722 (1988). International Communication Union. "7 khz Audio-Coding Within 64 kbit/s." *Recommendation G.722*. Geneva.

ITU-T G.723.1 (1996). International Communication Union. “Dual Rate Speech Coder for Multimedia Communications Transmitting at 5.3 and 6.3 kbit/s.” *Recommendation G.723.1*. Geneva.

ITU-T G.726 (1990). ITU-T. “40, 32, 24, 16 kbit/s Adaptive Differential Pulse Code Modulation (ADPCM)”. ITU-T G.726.

ITU-T G.728 (1992). International Communication Union. “Coding of Speech at 16 kbit/s Using Low-Delay Code Excited Linear Prediction.” *Recommendation G.728*. Geneva.

ITU-T G.729 (1996). International Communication Union. “Coding of Speech at 8kbit/s Using Conjugate-Structure Algebraic-Code-Excited Linear-Prediction (CS-ACELP).” *Recommendation G.729*. Geneva.

ITU-T H.261 (1993). International Communication Union. “Video Codec for Audiovisual Services at p*64 kbit/s.” *Recommendation H.261*. Geneva.

ITU-T H.263(2005). International Communication Union. “Video Coding for Low Bit Rate Communication.” *Recommendation H.263*. Geneva.

Netravali, A.N.; Haskell, B.G. “*Digital Pictures*”. 2.ª ed. Plenum Press, 1995.

Soares, L.F.G., Martins, S.L. e Bastos, T.L.P. “Um algoritmo para compensação da variação estatística do retardo em redes comutadas por pacotes.” *Anais do 8.º Simpósio Brasileiro de Redes de Computadores,* 1991.

Soares, L.F.G. e Bastos, T.L.P. “Análise de algoritmos para reprodução em tempo real de voz em redes depPacotes.” *Anais do 10.º Simpósio Brasileiro de Redes de Computadores*. Recife. 1992.

Soares, L.F.G.; Colcher, S. e Souza, G.L. “Redes de computadores: das LANs, MANs e WANs às redes ATM.” Rio de Janeiro: Campus, 1995.

Apêndice B

DSM-CC — Digital Storage Media — Command and Control

O DSM-CC (*Digital Storage Media — Command and Control*) tem uma especificação extremamente complexa, e grande parte dela não está diretamente relacionada a nenhum dos padrões dos sistemas de TV digital. Como consequência, neste apêndice ignoraremos grande parte do padrão, simplificaremos em muito a discussão de outras partes e consideraremos apenas os casos necessários ao entendimento de como o DSM-CC atua na difusão de dados nos principais sistemas de TV digital.

B.1 Introdução

O DSM-CC [ISSO/IEC 13818-6, 1998] foi originalmente projetado para ser implementado usando algum tipo de mecanismo RPC (*Remote Procedure Call*) — em outras palavras, onde os dados seriam buscados (*pulled from*) de um provedor de conteúdos. Sistemas de difusão, como a TV digital terrestre, são, no entanto, de natureza diferente. Neles, os dados são enviados sem requisição (*pushed to*), do provedor para o cliente consumidor. Uma outra solução deve então ser encontrada para o acesso aos dados.

A solução encontrada é bastante simples, mas nem um pouco eficiente. Arquivos de dados devem ser periodicamente transmitidos pelo provedor de conteúdos, devendo o cliente receptor aguardar pelo arquivo que deseja. Esse tipo de solução é chamada de *carrossel*.

O DSM-CC dá suporte a dois tipos de carrossel: carrossel de dados e carrossel de objetos, assunto das Seções B.2 e B.3, respectivamente.

B.2 Carrossel de Dados

O carrossel de dados é a forma mais simples de transmissão de dados DSM-CC. Nele não existe qualquer indicação sobre em que consistem os dados. Cabe ao receptor analisar os dados de um modo que faça sentido para ele. As especificações ATSC (padrão americano) [ATSC A/90, 2001] e ARIB (padrão japonês) [ARIB STB-B24 V 4.0, 2004] fazem uso dessa modalidade de transmissão.

Um carrossel de dados consiste em um número de módulos, em que cada módulo contém um item de dados como, por exemplo, um arquivo. Não existe nenhuma estruturação de mais alto nível acima do módulo.

Cada módulo pode, por sua vez, ser quebrado em blocos, como mostra a Figura B.1.

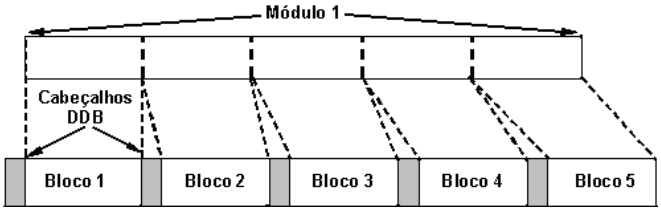


Figura B.1 Módulo de um carrossel**.**

Elementos de um carrossel DSM-CC são inseridos em um conjunto de mensagens DSM-CC:

- *DSM-CC DownloadDataBlock message* (DDB), que contém os dados dos módulos de um carrossel. Cada mensagem contém um bloco, todos do mesmo tamanho, exceto o último bloco de um módulo, o que torna mais fácil o parser de mensagens. A Figura B1 ilustra tais mensagens.
- DSM-CC *download control messages*, que especificam, para o receptor, como as mensagens DSM-CC DownloadDataBlock são organizadas em módulos, isto é, como é a estrutura do módulo.

Existe apenas um tipo de mensagem DDB. Cada mensagem contém uma identificação do módulo a que pertence e da versão do módulo, além da numeração do bloco correspondente ao módulo.

Como mencionamos, a estrutura de um módulo é definida em uma ou mais mensagens de controle (*download control messages*), que podem ser de vários tipos.

Como módulos grandes podem necessitar de mais de um bloco, e um carrossel pode ter mais de um módulo, é necessária alguma informação adicional para descrever como os blocos são agrupados em módulos. Isso é feito dentro da mensagem *DownloadInfoIndication* (DII).

Cada DII contém uma descrição para um conjunto de módulos e os parâmetros usados para suas transmissões. Todos os módulos em uma DII são ditos pertencerem ao mesmo grupo.

A DII especifica o tamanho das mensagens DownloadDataBlock usadas na transmissão dos módulos, o ID de cada um dos módulos, seus tamanhos e suas versões, assim como vários descritores que podem dar mais informações de cada módulo.

Sabendo o tamanho de cada módulo e o tamanho de cada um de seus blocos, um receptor pode calcular quantos blocos são necessários para a recepção completa. O número de versão permite ao receptor saber quando o conteúdo de um módulo mudou.

Mensagens DDB e DII são transportadas em seções privadas MPEG-2, por nós apresentadas no Capítulo 1. Assim, uma limitação que o MPEG impõe ao DSM-CC é o tamanho máximo de suas mensagens, igual ao tamanho máximo de uma seção, isto é, 4 KBytes.

Carrosséis que podem ser descritos por uma única mensagem DII (de valor menor que 4 KBytes) são chamados *carrosséis de uma camada*.

Para carrosséis maiores, mais de uma mensagem DII pode ser necessária. Nesse caso, uma mensagem *DownloadServerInitiate* (DSI) atua

como uma mensagem de alto nível para as demais mensagens DII. Ela agrupa mensagens DII e os grupos de módulos de cada uma delas em um "supergrupo".Carrosséis em que várias mensagens DII são reunidas em um supergrupo são chamados de *carrosséis de duas camadas*.

Como mencionamos, uma aplicação que se utiliza do carrossel de dados tem de saber o formato dos dados contidos e como manipulá-los. Em estruturações de dados mais complexas, o carrossel de dados não é muito útil. Como uma estrutura de arquivos e diretórios é uma forma muito importante de organização, o carrossel de objetos oferece uma solução melhor para esse caso, como discutido na próxima seção.

B.3 Carrossel de Objetos

Carrosséis de objetos são construídos tendo por base o modelo de carrossel de dados, adicionando a ele o conceito de arquivos, diretório e fluxos. As especificações SBTVD-T (padrão brasileiro) [ABNT NBR 15606-3, 2007] e DVB (padrão europeu) [ETSI TS 102 812 v1.2.1, 2003] fazem uso dessa modalidade de transmissão, que também foi adotado pelo ACAP [ATSC A/90, 2001].

Nos carrosséis de objetos, os dados são representados por *objetos* (objeto de diretório, objeto de arquivo, objetos de fluxo, objetos de eventos de fluxo e objeto Service Gateway), que contêm atributos (nome, tipo e, possivelmente, conteúdo).

Objetos de fluxo permitem a um carrossel de objeto referenciar fluxos elementares que são parte da difusão. Pode haver um único *tap* (uma única referência), que se refere a um fluxo completo, ou pode haver vários *taps*, que se referem a vários fluxos que formam um único fluxo lógico. Por exemplo, um programa de TV pode conter duas trilhas de áudio para línguas diferentes. Um único objeto de fluxo DSM-CC pode referenciar a todo o serviço, mas pode também haver dois objetos de fluxo, cada um referenciando o vídeo e um dos dois fluxos de áudio.

Embora a forma mais óbvia de se referenciar a um fluxo seja através do seu PID, isso traz duas desvantagens: PIDs só se referem a fluxos elementares e não a programas inteiros; o PID de um fluxo pode mudar quando ele é remultiplexado, o que implicaria atualizar cada referência feita pelo carrossel.

Para referências, o DSM-CC introduz outro identificador chamado *association tag* (ou *component tag*, no SBTVD e no DVB). Ele provê uma identificação única de um fluxo que não é afetada pela remultiplexação. O *association tag* é ligado a um fluxo por meio de um *descritor de association*

tag associado ao fluxo pela PMT (*Programming Mapping Table*). Cada programa tem uma PMT, e cada PMT tem uma lista de fluxos de um programa; para cada elemento da lista, vários descritores podem ser associados, incluindo o descritor de *association tag*.

Objetos de eventos de fluxo permitem a um receptor saber a respeito de um ponto específico de sincronização dentro desse fluxo elementar, como veremos na Seção B.4.

Além dos objetos de fluxos e objetos de eventos de fluxo, cada carrossel de objetos contém uma árvore de diretórios, que é quebrada em uma série de módulos, que podem conter um ou mais arquivos (objetos de arquivo) ou diretórios (objetos de diretórios e objetos Service Gateway). Cada módulo pode conter vários objetos, desde que o tamanho seja menor que 64 Kbytes. Não é possível dividir um arquivo em mais de um módulo. Dessa forma, arquivos com tamanho maior que 64 Kbytes devem ser transportados em um único módulo. Esse é o único caso em que um módulo pode exceder o tamanho de 64 Kbytes. Arquivos em um módulo podem vir de qualquer parte da árvore de diretórios porque eles não têm necessidade de pertencer ao mesmo diretório.

Objetos Service Gateway representam um conceito similar a um diretório. A principal diferença é que um objeto Service Gateway identifica o diretório raiz da árvore de diretórios de um carrossel de objetos. Isso significa que existirá um e apenas um objeto Service Gateway em um carrossel de objetos.

Quando quiser ter acesso a um determinado arquivo, o receptor deve esperar pelo módulo que contém o arquivo, quando poderá analisar os dados recebidos e delimitar o arquivo.

Um carrossel de objetos é também chamado de *service domain*. Em sistemas de difusão, não há distinção entre os dois termos.

Segundo as especificações DSM-CC, que são compatíveis com o *framework ORB* (Object Request Broker) definido pelas especificações CORBA (*Common Object Request Broker Architecture*) [OMG, 2004], cada objeto deve ser encapsulado em uma mensagem BIOP (*Broadcast Inter ORB Protocol*) que é transmitida em um módulo.

Uma mensagem BIOP deve ser transmitida em um único módulo, mas um módulo pode conter mais de uma mensagem, como consequência do que mencionamos anteriormente sobre o encapsulamento de objetos em módulos.

A Figura B.2 ilustra o processo de quebra dos dados em um carrossel de objetos DSM-CC, passando pela estrutura do carrossel de dados até a geração final de seções privadas DSM-CC. Cada seção DSM-CC de um

carrossel é transmitida no fluxo de transporte MPEG, como um fluxo elementar de dados, uma após a outra.

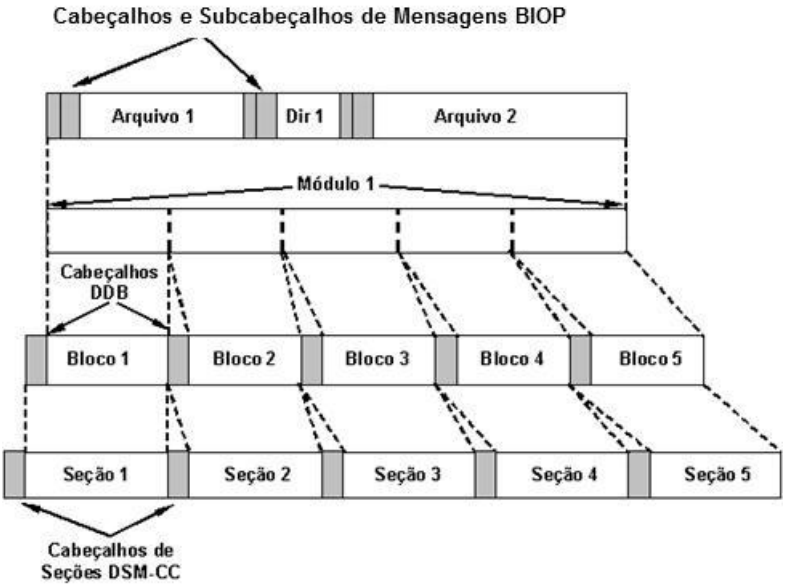


Figura B.2 Módulo de um carrossel**.**

Um módulo pode ser inserido mais de uma vez em um ciclo do carrossel (isso é verdade também para os carrosséis de dados). Como é usual nos carrosséis, os módulos são transmitidos um após o outro até que todos os módulos de um ciclo tenham sido transmitidos, quando todo o processo recomeça, repetidamente.

Vamos tomar como exemplo a árvore de diretórios da Figura B.3, representando uma aplicação NCL, contendo um objeto de mídia vídeo, dois objetos de imagem e um objeto de mídia contendo código Lua. Para criar o carrossel para difusão, vamos adicionar arquivos em módulos. Adicionar os dois primeiros arquivos (joão.ncl e foto.png) no mesmo módulo (Módulo 1) é possível, mas não podemos adicionar o arquivo chuteira.png nesse mesmo módulo porque ultrapassaria o tamanho máximo de 64 Kbytes. Assim, o arquivo chuteira.png deve ir para o Módulo 2, no qual poderemos também adicionar o arquivo contendo o código Lua (gols.lua). O arquivo drible.mp4 deve ocupar um único módulo (Módulo3), por ser maior que 64 Kbytes. Vamos, então, adicionar as informações do diretório imagens no Módulo 1, as do diretório códigos lua no Módulo 2, e as informações do diretório raiz no objeto Service Gateway também no Módulo1.

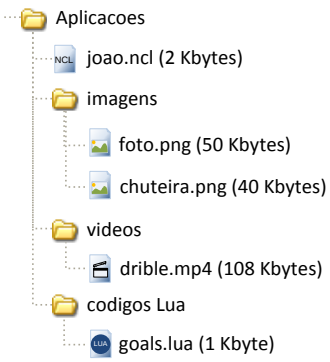


Figura B.3 Árvore de diretórios para um carrossel**.**

A divisão anterior pode não ser a melhor divisão dos arquivos em módulos. A divisão depende de quando os arquivos serão necessários, os relacionamentos entre eles etc.

Vamos agora colocar os módulos no carrossel. Como o Módulo 1 contém a aplicação e ela deve ser carregada antes de tudo, para diminuir o seu retardo de acesso vamos inserir o Módulo 1 mais de uma vez no carrossel, como ilustra a Figura B.4.

Módulo 1
Módulo 2
Módulo 1
Módulo 3

Figura B.4 Carrossel de objetos para o exemplo da Figura B.3.

Como mencionamos, transmitir um módulo mais de uma vez em um carrossel diminui seu tempo de acesso, mas aumenta o tamanho do carrossel e o tempo de acesso dos outros módulos. Mais ainda, como a banda passante de difusão é constante (6 MHz no caso do Brasil), aumentar o tamanho do carrossel é diminuir a banda dos outros fluxos que irão no mesmo sinal TS (*Transpor Stream*), incluindo o áudio e o vídeo principal de um programa de TV. Diminuir a banda desses sinais é diminuir sua qualidade. Alternativamente, poderíamos ter optado por colocar o arquivo joão.ncl nos módulos 1 e 2, pois nada impede que um arquivo seja transmitido mais de uma vez, em mais de um módulo. Entretanto, os mesmos cuidados devem ser tomados. De fato, a otimização de um carrossel é um problema extremamente complexo. Não existe maneira de saber qual é o carrossel de objetos mais eficiente para todos os casos. É necessário saber a estrutura e o projeto de cada aplicação específica carregada pelo carrossel e, mesmo assim, não é nada fácil chegar à solução mais eficiente.

Como já mencionamos, cada instância do carrossel de objetos é representada por um Service Domain, que consiste em uma identificação única do carrossel de objetos. Todo Service Domain possui um objeto Service Gateway, que contém referências para todos os objetos dispostos na raiz do carrossel de objetos.

O Service Gateway é um objeto do carrossel de objetos cuja localização é transmitida em uma mensagem *DownloadServerInitiate* (DSI), que apresentamos na Seção B.2 (um carrossel de objetos será sempre transportado em um carrossel de duas camadas).

O padrão DSM-CC utiliza a mesma estrutura de referências IOR (*Interoperable Object Reference*), definidas nas especificações CORBA para referenciar um objeto no carrossel. No contexto do protocolo carrossel de objetos, uma IOR é normalmente composta pelo identificador do carrossel, seguido do identificador do módulo e pelo identificador do objeto.

Vamos a um exemplo um pouco mais simples que o anterior para tornar mais claro o processo. Considere a árvore de diretórios apresentada na Figura B.5 e o carrossel de objetos gerado a partir dessa árvore, com o *Service Domain* de valor hexadecimal "1". Nesse *Service Domain*, dois módulos são gerados e identificados com valores em hexadecimal "1" e "2". O identificador de cada objeto é apresentado através do campo "objectKey". O objeto que representa o *Service Gateway* (na Figura B.5: objeto do tipo "srg") é identificado com o valor "1" e é encapsulado no Módulo "1". Como consequência, a IOR do objeto *Service Gateway* transmitida na mensagem *DownloadServerInitiate* é definida por "1,1,1" (Service Domain = 1, id do módulo = 1 e id do objeto = 1). Os objetos que representam o arquivo "weatherConditions.ncl" e o diretório "images" são identificados pelos valores hexadecimais "2" e "3", respectivamente, e também são encapsulados no Módulo "1". Já o objeto que representa o arquivo contendo o conteúdo da imagem é identificado com o valor "1", mas encapsulado no Módulo "2".

O objeto *Service Gateway* possui, por sua vez, duas IORs. Para relacionar uma IOR ao nome do objeto que a mesma referencia, bem como ao tipo desse objeto (arquivo, diretório etc.), as especificações DSM-CC utilizam o conceito de *binding*. Assim, no exemplo, o objeto *Service Gateway* possui dois *bindings* para os dois objetos do Módulo "1" que são filhos diretos da raiz do sistema de arquivos representado pelo carrossel.

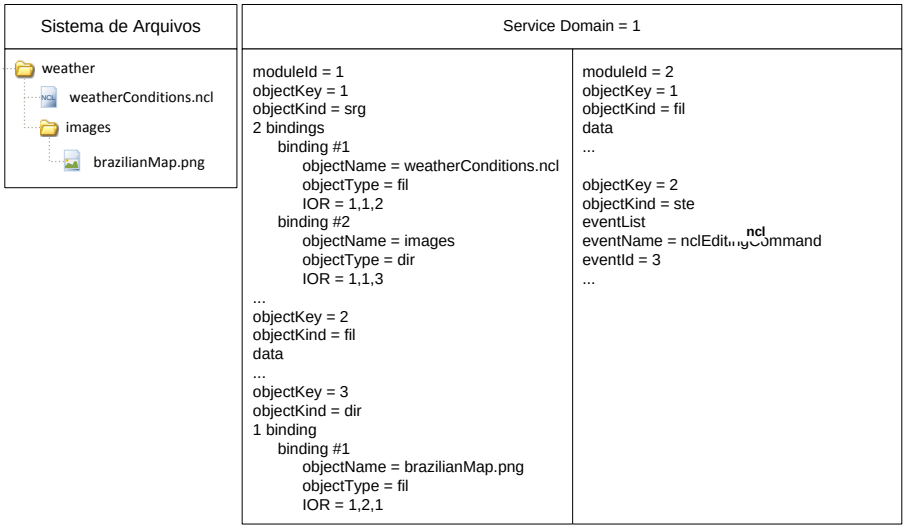


Figura B.5 Árvore de diretórios e carrossel de objetos correspondente.

Os objetos do tipo diretório (“dir”) possuem a mesma sintaxe e semântica dos objetos do tipo *Service Gateway*. Os objetos do tipo arquivo (“fil”) possuem como atributo, além da identificação do seu tipo, os dados relativos ao seu conteúdo.

A Figura B.5 apresenta também um objeto do tipo eventos de fluxo (“ste”). Esse objeto é utilizado para definir tipos de eventos DSM-CC possíveis de serem descritos no fluxo de transporte. Para isso, o objeto relaciona identificadores de descritores de eventos DSM-CC a uma string (na figura, conteúdo do campo “eventName”). Vamos nos aprofundar um pouco mais nesse tópico na Seção B.4.

Finalmente, é importante notar, no exemplo da Figura B.5, que a identificação da raiz do sistema de arquivos (diretório “weather”) é perdida quando da geração do carrossel. A consequência dessa perda e a solução para o problema são discutidos no Capítulo 16.

B.4 Eventos de Fluxo

Eventos de fluxo são descritores embutidos em um fluxo elementar DSM-CC. Esses descritores vão fornecer um modo de sincronizar eventos com um fluxo de mídia. Assim como os objetos de fluxo, os eventos de fluxo contêm uma lista de *taps* que se referem a fluxos elementares.

Eventos de fluxo são bastante úteis para especificar eventos não-previsíveis. Por exemplo, em uma partida de futebol, o momento de um gol que se quer sincronizar com outro objeto de mídia qualquer, como um *ranking* dos artilheiros do campeonato.

Do ponto de vista da especificação DSM-CC, o tratamento de eventos de fluxo são divididos em duas partes:

1. *Objetos de eventos de fluxo*, transportados em carrosséis DSM-CC.
2. *Descritores de eventos de fluxo*, transportados em seções privadas DSM-CC.

Um descritor de evento de fluxo determina o disparo de um evento e pode ser referido por um objeto de eventos de fluxo, que descreve em mais alto nível o que significa o evento. Mais de um descritor de evento de fluxo pode ser referido por um mesmo objeto de evento de fluxo.

Um objeto de eventos de fluxo possui um identificador (eventId) que deve ser único dentro de um carrossel e um nome legível para o ser humano, por exemplo, "nclEditingCommand". Uma aplicação pode se registrar para receber eventos por esse nome legível. Por exemplo, o Gerenciador de Base Privada do ambiente declarativo Ginga-NCL se registra para receber eventos nclEditingCommand que correspondem a comandos de edição de documentos NCL, como discutido no Capítulo 16. O exemplo da Figura B.5 apresenta um desses objetos do evento (tipo "ste"), identificando no campo "eventName" a string nclEditingCommand associada ao evento "3".

Descritores de eventos são transportados em fluxos listados na tabela PMT com o tipo igual a 0x0C. Cada descritor de evento possui um identificador numérico único (que o associa ao objeto de eventos de fluxo) e uma referência temporal, que indica ao receptor em qual instante o evento deverá ocorrer (usualmente baseado em um fluxo denominado *Normal Play Time* — NPT, como discutido no Apêndice E).

Como caso particular, um descritor de eventos pode informar ao sistema receptor que o evento deve ocorrer imediatamente; esse tipo de evento é chamado de evento *do it now*.

O SBTVD especifica que, para a maioria dos eventos de fluxo, um descritor de evento deve ser enviado uma vez a cada segundo, pelo menos cinco vezes, antes do tempo de disparo do evento. Uma exceção, claro, é o evento *do it now*, que é enviado apenas uma vez.

Descritores de evento de fluxo com valores de NPT permitem ao receptor saber antecipadamente o momento exato de ocorrência do evento, permitindo maior previsibilidade, uma vez que não é possível precisar o momento da chegada de um descritor no receptor. Eles também são mais confiáveis, uma vez que são enviados mais de uma vez.

Além do identificador e da referência temporal, o descritor de evento possui também um campo para dados específicos das aplicações, que pode ser utilizado de acordo com sintaxes e semânticas a serem tratadas pelas próprias aplicações, como discutido na Seção 16 para os comandos de edição para o Ginga-NCL.

Resumindo, no middleware Ginga, quando um descritor de evento de fluxo chega em um receptor, ele deve checar se um objeto de eventos de fluxo de mesmo *eventID* está presente no carrossel de objetos associado. Se não estiver, o descritor é ignorado. Se o evento é do tipo *do it now*, o evento é disparado imediatamente. Se não for *do it now*, o receptor verifica se o evento já está agendado para disparar. Se estiver, o descritor é ignorado; em caso contrário, o evento é agendado, dependendo do tempo NPT.

B.5 MPE: Multi-protocol Encapsulation

Para alguns tipos de dados, os carrosséis DSM-CC podem não ser adequados como estrutura de transmissão. Isso pode ser particularmente verdade para dados provenientes do mundo Internet. Assim, o DSM-CC também provê um meio para transporte de dados IP (em uma única direção) diretamente em seções privadas MPEG-2, chamadas *Datagram Sections*. Essas seções suportam qualquer protocolo do nível 3, mas o alvo é, em geral, o protocolo IP.

Cada receptor deve ter atribuído um endereço MAC de 48 bits, usado para identificá-lo como destino de um datagrama. Contudo, o DSM-CC não especifica como um endereço MAC é atribuído a um receptor.

Cada *Datagram Section* carrega um único datagrama para um único endereço MAC (*Medium Access Control*), embora este possa ser um endereço de multicast. Embora um datagrama IP possa ter tamanho maior do que uma seção MPEG-2, isso não é permitido pelo DSM-CC.

Bibliografia

ABNT NBR 15606-3 (2007). Associação Brasileira de Normas Técnicas, "Televisão digital terrestre — codificação de dados e especificações de transmissão para radiodifusão digital, Parte 3: Especificação de transmissão de dados", Sistema Brasileiro de TV Digital Terrestre, *NBR 15606-3*.

ARIB STB-B24 V 4.0 (2004). Association of Radio Industries and Business, “Data Coding and Transmission Specifications for Digital Broadcasting”, ARIB Standard. Fevereiro de 2004.

ATSC A/90 (2001). Advanced Television Systems Committee, “Implementation Guidelines for ATSC Data Broadcast Standard”, ATSC Recommended Practice. Junho de 2001.

ETSI TS 102 812 v1.2.1 (2003). European Broadcasting Union, “Digital Bideo Broadcasting (DVB); Multimedia Home Platform (MHP) Specification 1.1.1”. Technical Specification.

ISO/IEC 13818-1 (2000). International Organization for Standardization/International Eletrotecnical Committee, “Information Technology — Generic coding of moving pictures and associated audio information, Part 1: Systems”, *ISO/IEC 13818-1*.

ISO/IEC 13818-6 (1998). International Organization for Standardization/International Eletrotecnical Committee, “Information Technology — Generic coding of moving pictures and associated audio information, Part 6: Extensions for DSM-CC”, *ISO/IEC 13818-6*.

OMG Specification (2004). “Common Object Request Broker Architecture” (CORBA/IIOP).

Steven Morris (2004) Interactive TV Web. “A technical (and non-technical) guide to DSM-CC.” Disponível em http://interactivetvweb.org/tutorial/dtv-intro/dsm-cc/index.shtml. Acesso em 21 de março de 2008.

Balabanian, Casey and Greene (1996). Vahe Balabanian; Liam Casey; Nancy Greene. An Introduction to Digital Storage Media — Command and Control (DSM-CC). Nortel, 1996.

Apêndice C

Modelo de Contextos Aninhados NCM 3.0 Básico

Este apêndice descreve as entidades básicas da versão 3.0 do NCM (*Nested Context Model*). O NCM é um modelo conceitual centrado na representação e tratamento de documentos hipermídia. A linguagem NCL do middleware Ginga do sistema brasileiro de TV digital tem por base o modelo NCM.[1]

[1] Este capítulo foi baseado em Soares *et al*. (2005). O uso do material foi gentilmente cedido pelo Departamento de Informática da PUC-Rio.

C.1 Introdução

A definição de documentos hipermídia[2] no NCM [Soares *et al.*, 2005] é baseada nos conceitos usuais de nós e elos. *Nós* (*nodes*) (ou objetos) são fragmentos de informação, e *elos* (*links*) são usados para a definição de relacionamentos entre os nós. No entanto, os elos não são a única forma de definição de relacionamentos, como ficará evidente a seguir.

O modelo distingue duas classes básicas de nós, chamados de nós de conteúdo (*content nodes*) (ou objetos de mídia) e nós de composição (*composite nodes*), sendo estes últimos o ponto central do modelo. A Figura C.1 ilustra a visão geral da hierarquia de classes do modelo,[3] que será detalhada ao longo deste apêndice, seguindo uma abordagem *top-down*.

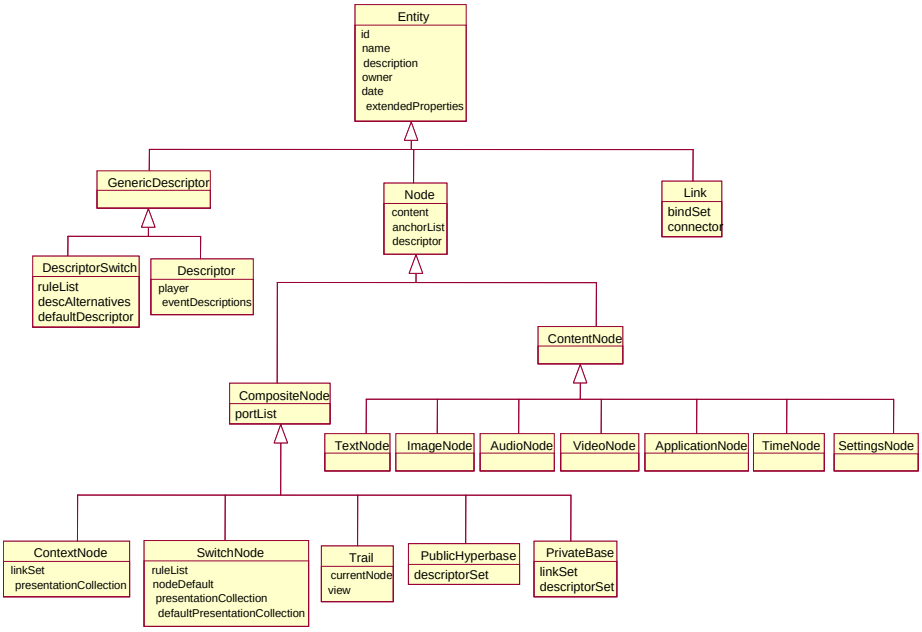


Figura C.1 Visão geral da hierarquia de classes do NCM.[4]

[2] Uma especificaçõ de aplicação para TV digital é um caso particular de documento hipermídia.

[3] É necessário salientar que, embora seguindo uma especificação orientada a objetos, o NCM não obriga que sua implementação seja orientada a objetos. O NCM é apenas um modelo hipermídia.

[4] Com o objetivo de não poluir visualmente as figuras, os métodos das classes foram omitidos nos diagramas.

C.2 Entidades e Propriedades

Toda *entidade* (*entity*) do modelo possui como atributos: um *identificador único* (ID), um *nome*, uma *descrição*, a *data de criação* e um *autor*.[5] Além dessa coleção básica de atributos, uma entidade NCM mantém uma lista de atributos estendidos, para permitir extensões que não se restrinjam apenas a heranças de classes. No NCM, a maioria dos atributos é chamada *propriedade* e eles devem ser envolvidos (*wrapped*) por uma classe do modelo chamada *propriedade* (*property*). Isso permite ao NCM o suporte para manutenção, para cada propriedade da entidade (*básica* ou *estendida*), de informações acerca de direitos de acesso, do último usuário que modificou o seu valor, da data dessa modificação, se a mudança deve ocasionar ou não um versionamento da entidade etc. Em outras palavras, o NCM prevê um controle granular bastante fino quando da implementação de suporte a controle de versões e controle de acesso das entidades, obrigando que as propriedades mantenham outros atributos. No entanto, sistemas que não tenham interesse em explorar todas as capacidades do modelo podem optar por representar os campos das classes como atributos tradicionais, em vez de utilizar o *wrapper* propriedade oferecido pelo modelo. Mesmo para aqueles sistemas que implementam controles de acesso e/ou de versões, campos de classes que não necessitem ser monitorados com tal granularidade podem ser representados sem a utilização dos *wrappers*.

Entidades do modelo devem oferecer métodos *get* e *set* para cada propriedade básica (por exemplo, *getId*, *setId*, *getName*, *setName*, etc.),[6] métodos para adicionar/remover propriedades estendidas, e dois métodos genéricos para consultar (*get*) e modificar (*set*) valores das propriedades estendidas.

C.3 Nós e Âncoras

Um *nó* (*node*) é uma entidade NCM que tem como propriedades básicas adicionais: um conteúdo, um descritor genérico (propriedade opcional) e uma lista ordenada de âncoras.

O *conteúdo* de um nó é composto por uma coleção de unidades de informação. A noção exata do que constitui uma unidade de informação é

[5] O NCM define um tipo *usuário* cuja implementação fica a cargo dos sistemas hipermídia que utilizem as classes do modelo.

[6] Deste ponto em diante, será assumido que as subclasses deverão especificar métodos do tipo *get* e *set* para manipular cada uma de suas propriedades.

parte da definição do nó e depende de sua especialização, conforme será exemplificado mais adiante.

Descritores NCM serão explicados nas Seções C.10 e C.11. A definição de um descritor como propriedade do nó é opcional. Quando especificado, ele conterá informações (proprieadades) determinando como o nó deve ser exibido no tempo e no espaço.

Cada elemento da *lista ordenada de âncoras* é chamado âncora do nó, ou simplesmente âncora. Toda *âncora* (*anchor*) é uma entidade NCM, que pode ser especializada, conforme ilustrado no diagrama de classes da Figura C.2.[7] O modelo define dois tipos de âncora (ou interfaces). O primeiro tipo é chamado de *âncora de conteúdo* (*content anchor* ou *area anchor*), que possui um atributo denominado *região* (*region*). A *região* da âncora define uma coleção de unidades de informação do conteúdo do nó. Qualquer subconjunto de unidades de informação do conteúdo de um nó pode definir uma âncora. Tem-se, assim, que a definição exata da região da âncora é dependente do tipo de conteúdo do nó. No entanto, o modelo requer que todo nó contenha uma âncora de conteúdo com uma região representando a marcação de todo o conteúdo do nó. Essa âncora é chamada de *âncora conteúdo total* e sua região correspondente é representada pelo símbolo λ. A âncora conteúdo total deve sempre ser a primeira âncora na lista de âncoras do nó. Âncoras serão extremamente importantes na especificação de relacionamentos entre nós.

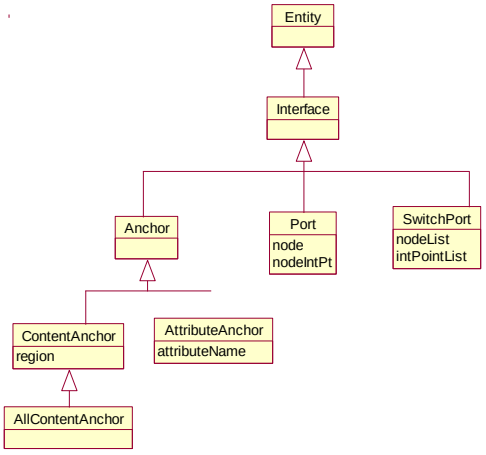


Figura C.2 Hierarquia de classes das interfaces de nós NCM.

[7] Na verdade, âncoras NCM herdam da classe ponto de interface, que por sua vez herda da classe entidade.

O segundo tipo de âncora definido no modelo é a *âncora de atributo* (*attribute anchor*). A âncora de atributo aponta para um atributo (ou propriedade) do nó. Como será explicado na Seção C.10, durante a apresentação de documentos NCM, nós são associados a descritores. Na especificação das âncoras de atributo do nó, os autores podem referenciar não apenas atributos do nó, mas também atributos definidos no descritor que venha a ser associado ao nó. Na verdade, a âncora de atributo pode identificar qualquer atributo recursivamente contido no nó, através de referências qualificadas. Como exemplo, um autor pode usar o nome qualificado *descritorSelecionado.posicaoX* para criar uma âncora de atributo que aponte para um atributo *posicaoX* definido em um descritor selecionado para apresentar o nó.

Além dos já mencionados métodos para consultar e modificar os valores dos atributos, os nós devem oferecer métodos para manipulação de suas listas de âncoras (por exemplo, adicionar, remover, recuperar uma âncora, percorrer a lista etc.).

C.4 Nós de Mídia

Um *nó de mídia*, também chamado *nó de conteúdo* (*content node*) ou *objeto de mídia*, é um nó que representa um objeto em uma mídia qualquer. Nós de conteúdo devem ser especializados em subclasses para melhor definir a interpretação do conteúdo (texto, áudio, imagem, vídeo, aplicação etc.). Conforme mencionado anteriormente, a noção exata do que constitui uma unidade de informação do conteúdo é parte da definição da classe do nó. Por exemplo, uma unidade de informação do conteúdo de um nó vídeo pode ser um quadro, enquanto uma unidade de informação do conteúdo de um nó texto pode ser um caractere ou uma palavra. O conteúdo de um nó de mídia pode ser definido como uma referência (por exemplo, um URI) para o conteúdo propriamente dito ou como uma sequência de bytes (*raw data*). Além disso, cada subclasse de nó de conteúdo pode ser refinada. Como exemplo, nós texto podem ser especializados em nós HTML, nós PDF etc.

Um tipo especial de nó de conteúdo definido pelo modelo é o *nó de tempo* (*time node*). Como o próprio nome sugere, esse nó representa o tempo. Esse nó representa tempos absolutos (baseados, por exemplo, na hora UTC) ou um tempo relativo (baseado em eventos, como por exemplo o início da apresentação de um documento). Cada instante de tempo é considerado uma unidade de informação para o seu conteúdo. Dessa forma, intervalos podem ser definidos como âncoras de um nó de tempo. O nó de tempo vai, assim, permitir acionar eventos (Seção C.7) baseados em instantes de tempo específicos.

Outro tipo especial de nó de conteúdo é o *nó de ambiente* (*settings node*). Esse nó representa todas as variáveis controladas diretamente pelo formatador (ferramenta responsável pela apresentação de um documento NCM). As variáveis são representadas pelos atributos (propriedades) do nó. Como discutido na Seção C.8, esses atributos podem ter seus valores modificados por ações dos elos. Como discutido nas Seções C.9 e C.11, esses atributos podem também ter seus valores testados pelas regras de nós *switch* e de *switch de descritores*, a fim de realizar a escolha entre alternativas.

C.5 Nós de Composição

Um *nó de composição* (*composite node*) *C*, ou simplesmente *composição*, é um nó cujo conteúdo é uma coleção C_L de nós (de conteúdo ou de composição, recursivamente) que constituem suas unidades de informação. Diz-se que um nó *N* em C_L é um *componente de C* e que *N está contido em C*. Diz-se também que um nó *A* está *recursivamente contido em C* se e somente se *A* está contido em *C* ou *A* está contido em um nó recursivamente contido em *C*. Devemos também notar que os componentes de *C* podem ser ordenados (lista ordenada), o que será útil na definição de operações de navegação. Note também que um nó pode estar contido mais de uma vez em C_L. Uma restrição importante é feita, no entanto: um nó não pode estar recursivamente contido em si mesmo.

Nós de composição diferentes podem conter um mesmo nó e ser aninhados em qualquer profundidade, desde que seja obedecida a restrição de um nó não conter recursivamente a si mesmo. Para identificar através de que sequência de nós de composição aninhados uma dada instância de um nó *N* está sendo observada, é introduzida a noção de perspectiva de um nó. A *perspectiva* de um nó *N* é uma sequência $P = (N_m,....,N_1)$, com $m \geq 1$, tal que $N_1 = N$, N_{i+1} é um nó de composição, N_i está contido em N_{i+1}, para $i \in [1,m)$ e N_m não está contido em qualquer nó. Note que pode haver várias perspectivas diferentes para um mesmo nó *N*, se esse nó estiver contido em mais de uma composição. A *perspectiva corrente* de um nó é aquela percorrida pela última navegação ao nó (as diversas formas de navegação serão definidas *a posteriori*). Dada a perspectiva $P = (N_m,....,N_1)$, o nó N_1 é chamado *nó-base da perspectiva*.

Nós de composição são objetos cuja semântica é bem conhecida pelo modelo. Um modelo conceitual deve representar não apenas os conceitos estruturais dos dados, mas também definir operações sobre os dados para manipulação e atualização das estruturas. Assim, todo nó *C* de composição deve possuir os seguintes métodos:

- Insere nó: deferido (implementação dependente da subclasse de composição).
- Retira nó: retira um nó da coleção de nós da composição.

Com base nas definições de conteúdo de composição e regiões de âncoras de conteúdo (Seção C.3), conclui-se que a região de uma âncora de conteúdo de uma composição deve especificar um subconjunto dos componentes da composição. Um subconjunto especial é aquele com todos os nós da composição (âncora conteúdo total [λ] da composição).

Além da lista ordenada de âncoras, nós de composição têm uma outra propriedade chamada *lista ordenada de portas*. Portas e âncoras possuem propósito similar e estendem um tipo comum denominado *interface*. Uma porta (*port*) de uma composição *C* é uma entidade NCM que possui dois atributos adicionais: um nó *N* e uma interface *ip*, onde *N* deve estar obrigatoriamente contido em *C* e *ip* deve ser uma interface definida em *N*, ou seja, contido em sua lista de âncoras ou de portas.[8] Como pode ser percebido, a porta de um nó de composição permite definir mapeamentos entre a composição e seus nós internos. Como consequência, o nó de composição pode tornar a interface de um nó constituinte visível para referências externas (elos hipermídia, por exemplo). O conjunto de interfaces (âncoras ou portas) age como uma proteção para referências a um nó, no sentido de que qualquer entidade só pode ter acesso a segmentos do conteúdo ou atributos de um nó se eles estiverem disponíveis na interface do nó (lista de âncoras ou de portas). Dessa forma, as interfaces podem impedir que modificações internas em um nó se reflitam em outras entidades que o referenciam. Tome como exemplo um nó texto com uma âncora de conteúdo cuja região especifique seu segundo parágrafo. Qualquer mudança no texto (por exemplo, a eliminação do primeiro parágrafo) deve se refletir na região da âncora, mas não deverá afetar as demais entidades que a referenciam, assim mantendo as referências para a parte correta do conteúdo (isto é, o antigo segundo parágrafo agora posicionado como sendo o primeiro). A proteção do nó através de interfaces também trará ao modelo o conceito de composicionalidade, permitindo provas formais de propriedades dos documentos, como a consistência temporal.

Define-se como *sequência de mapeamentos da porta* p_k de uma composição N_k a sequência de nós e interfaces (N_k, ip_k....,N_1, ip_1) com $k > 1$, tal que para $i \in [1,k]$:

i) N_{i+1} é um nó de composição, e N_i está contido em N_{i+1},

ii) ip_i é uma interface do nó N_i na sequência de nós da porta, e N_i e ip_i são os valores dos atributos nó e interface, respectivamente, da porta ip_{i+1}. Diz-se que ip_i *pertence* à sequência de mapeamentos da porta *p*.

[8] Evidentemente, *ip* estará contido na lista de portas de *N* apenas se *N* for uma composição.

Note que a definição de dois tipos de interfaces de composições (âncoras e portas) permite dois tipos de tratamento para ações de apresentação, muitas vezes desejável na construção de um documento hipermídia. Um autor pode desejar apresentar uma composição para que a estrutura da composição seja visualizada (por exemplo, um desenho mostrando o grafo estrutural da composição). Âncoras de composições são interfaces que a princípio expressam esse tipo de comportamento, e a região da âncora enumerará os componentes que devem ser desenhados. Todavia, apresentar uma composição algumas vezes pode significar apresentar seus constituintes a partir de um ponto de entrada, sem que a visão estrutural da composição seja exibida. Portas servem exatamente para fornecer pontos de acesso, permitindo que referências externas toquem em nós contidos dentro de um nó de composição, sem se perder a propriedade de composicionalidade do modelo (isto é, pontos de entrada e saída das composições devem ser explicitamente definidos).

Uma ação sobre um nó de composição deve especificar a interface onde se aplica. Caso não especifique, deve ser considerada como sendo aplicada em todas as suas portas.

Subclasses de composição irão definir semânticas para coleções específicas de nós. Cinco importantes subclasses de composição definidas pelo modelo são: nó de contexto, nó switch, trilha, hiperbase pública e base privada.

C.6 Nós de Contexto

Um *nó de contexto* (*context node*), ou objeto de contexto, ou simplesmente contexto, é um nó de composição tal que seu conteúdo contém um conjunto de nós de conteúdo, nós de contexto ou nós *switch*.[9] Os nós de contexto possuem como atributos adicionais um conjunto de elos e uma coleção de apresentação.

Cada *elo l* contido no conjunto de elos de um nó de contexto *C* define um relacionamento entre nós recursivamente contidos em *C*[10] ou o próprio nó de contexto *C*. Diz-se que um elo *l* é um *componente de um nó de contexto C* e que *l está contido em C*. Diz-se também que um elo *l* está *recursivamente contido em C* se e somente se *l* está contido em *C* ou *l* está contido em um nó de contexto recursivamente contido em *C*. Elos são o assunto da Seção C.8.

---

[9] Nós *switch* serão apresentados na Seção C.9. Nó switch é uma especialização de composição e permite definir alternativas de nós para documentos adaptativos.

[10] Como será discutido na Seção C.8, relacionamentos podem ter seus participantes definidos através de mapeamentos para nós recursivamente contidos na composição *C*.

Uma *coleção de apresentação* contém, para cada nó contido em um nó de contexto *C*, um grupo de descritores genéricos.[11] Como mencionado na Seção C.3, descritores reúnem as informações referentes às características de apresentação do nó e serão tratados em detalhes nas Seções C.10 e C.11. Cada grupo de descritores genéricos deve necessariamente formar um conjunto, ou seja, não pode haver repetição de descritores no grupo.[12] No caso de o nó contido em *C* ser um nó de contexto, o grupo de descritores deve conter no máximo um descritor genérico.

Nós de contexto vão servir, entre outras coisas, para definir uma estrutura lógica, hierárquica ou não, para documentos hipermídia. Essa estruturação permitirá a definição de diferentes visões de um mesmo documento e também melhorará a orientação do usuário na navegação sobre um documento.

Um nó de contexto *C* deve ter definido o método deferido de nó de composição:

- Insere nó: insere um nó de conteúdo, um nó *switch* ou um outro nó de contexto no conjunto de nós de *C*.

Além dos métodos para consultar e modificar os valores dos atributos, dos métodos para manipular as listas de portas e de âncoras, e dos métodos para manipular o conjunto de nós, nós de contexto devem também prover métodos para manipular seus conjuntos de elos e suas coleções de apresentação (por exemplo, inserir, remover, consultar, percorrer etc.).

C.7 Nós Switch (Alternativas de Nós)

O NCM possui várias características que oferecem suporte à adaptação de documentos. Uma importante facilidade é o grupo de nós alternativos, cuja seleção é feita com base em *regras* (*rules*) associadas ao documento. A Figura C.3 descreve o diagrama de classes para regras NCM.

[11] O grupo pode estar vazio para qualquer constituinte do contexto.

[12] A semântica por trás da definição do grupo de descritores selecionados para cada nó *N* contido em um nó de contexto é permitir uma navegação em profundidade para *N* especificando várias exibições diferentes simultâneas, como será explicado melhor na Seção C.10. Além disso, como será comentado na Seção C.11, um descritor do grupo pode ser o resultado de uma seleção entre alternativas de descritores (*switch de descritores*), cuja escolha poderá depender de parâmetros da plataforma ou do próprio usuário.

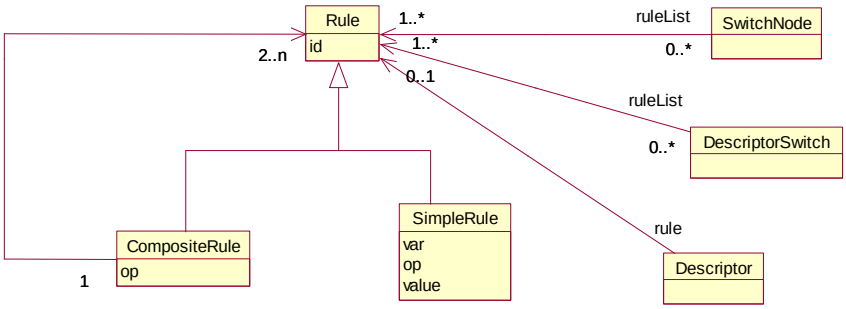


Figura C.3 Diagrama de classes para regras.

Baseado nas informações contextuais (por exemplo, preferências do usuário, características da plataforma de exibição etc.),[13] o formatador de documentos NCM deve avaliar cada regra para descobrir se uma determinada entidade associada à regra deve ou não ser considerada na apresentação do documento. A forma como entidades e regras são associadas será explicada adiante.

Uma regra pode ser simples ou composta. Uma *regra simples* (*simple rule*) é análoga à expressão assertiva do conector, que compara uma avaliação com um valor (Seção C.8.1) e possui três atributos: um identificador (*var*) da variável a ser testada, um operador de comparação (=, ≠, <, ≤, >, ≥) e um valor. A *regra composta* (*compound rule*) é uma expressão lógica compreendendo duas ou mais regras (simples ou compostas) relacionadas através de operadores lógicos *AND* e *OR*.

Com o objetivo de permitir que um autor especifique alternativas de nós dependendo da informação contextual (atributos do contexto de exibição), o NCM define uma entidade chamada nó switch. O *nó switch* (*switch node*) é uma especialização de nós de composição. O conteúdo de um nó switch é um conjunto que pode incluir nós de contexto e de conteúdo. O nó switch tem um atributo adicional que define, para cada nó contido no seu conjunto de nós, uma regra associada. As regras são definidas em uma lista ordenada. O formatador de documentos deve avaliar cada uma das regras conforme a ordem na lista. O primeiro nó que tiver a sua regra avaliada como verdadeira deve ser eleito a *alternativa selecionada*.

[13] Como anteriormente mencionado, as informações contextuais podem ser representadas por atributos (propriedades) do nó de ambiente (*settings node*).

O nó switch pode conter um nó default. Durante a apresentação do documento, esse nó deve ser eleito a alternativa selecionada do switch se nenhuma das regras for avaliada como verdadeira. Na ausência de um nó default e de uma regra avaliada como verdadeira, nenhum dos nós contidos no nó switch deve ser exibido.

Além da mencionada propriedade de lista ordenada de portas da composição (Seção C.5), os nós switch têm uma propriedade adicional chamada *lista ordenada de portas switch* (*ordered switch port list*). Cada *porta switch* (*switch port*) é um ponto de interface (Figura C.2) que define um conjunto de mapeamentos para interfaces de nós contidos no nó switch.

As portas de composição tradicionais permitem que um elo seja associado a uma alternativa específica. Se essa alternativa não for selecionada, o ponto terminal do elo será simplesmente ignorado durante a apresentação do documento. A definição de portas switch permite que elos (relacionamentos discutidos na Seção C.9) sejam definidos ancorando em nós switch, e sejam associados a mais de uma alternativa específica. Se nenhuma das alternativas for selecionada, o ponto terminal do elo será simplesmente ignorado durante a apresentação do documento.

Como os nós de contexto (Seção C.6), um nó switch tem uma *coleção de apresentação*, cujo objetivo é permitir que sejam definidos grupos de descritores para cada nó contido no nó switch.[14] Na verdade, o grupo deve formar um conjunto de descritores, pois não pode haver mais de uma instância de um mesmo descritor em um grupo. Se o nó contido no nó switch for um nó de contexto, o grupo de descritores deverá ser composto de, no máximo, um descritor genérico. Se o nó switch contiver um nó default, também pode ser especificado um grupo de descritores para esse nó, desde que as regras mencionadas neste parágrafo sejam seguidas.

O formatador de documentos NCM deve decidir quando avaliar as regras a fim de resolver os nós switch.

C.8 Eventos

Seguindo a definição de Pérez-Luque e Little [Pérez-Luque, 1996], um evento é uma ocorrência no tempo que pode ser instantânea ou ter uma duração mensurável. O NCM define algumas classes básicas de evento que

[14] O grupo pode ser vazio para qualquer constituinte do nó switch.

podem ser estendidas: evento de exibição, evento de composição, evento de seleção, evento de superposição, evento de proximidade, evento de colisão, evento de arraste, evento de foco e evento de atribuição.

Um *evento de exibição* (*presentation event*), também chamado *evento de apresentação*, representa a exibição de uma âncora de conteúdo (segmento de mídia) de um nó de conteúdo em uma dada perspectiva seguindo as diretrizes de um dado descritor. Dessa forma, perspectivas distintas ou descritores diferentes para um mesmo nó geram diferentes eventos de apresentação NCM. Os eventos de apresentação também podem ser definidos sobre nós de contexto e switch, representando a apresentação das unidades de informação de qualquer nó dentro desses nós de composição.

Eventos de composição são definidos pela apresentação da estrutura de um nó de composição. Os eventos de composição são utilizados para apresentar o mapa da composição (organização da composição).

Um *evento de seleção* (*selection event*) representa a seleção de uma âncora de conteúdo de um nó em uma dada perspectiva seguindo as diretrizes de um dado descritor. A forma mais comum para o usuário selecionar uma âncora é através de dispositivos de entrada como mouse e teclado, no entanto outros dispositivos também podem ser utilizados na geração desse evento, como um controle remoto de TV. Os eventos de proximidade, de colisão, de superposição (*hovering event*), de arraste (*drag event*) e de foco (*focus event*) também representam a ação de interação correspondente sobre uma âncora de conteúdo de um nó em uma dada perspectiva seguindo as diretrizes de um dado descritor.

Um *evento de atribuição* (*attribution event*) refere-se a uma âncora de atributo de um nó, dada uma perspectiva e um descritor específico.[15]

No NCM, cada evento define uma máquina de estados que deve ser mantida pela máquina de controle da apresentação dos documentos, denominada formatador [Soares *et al*., 2006]. A Figura C.4 mostra a máquina de estados geral dos eventos NCM.

[15] É importante lembrar que, como definido na Seção C.3, uma âncora de atributo do nó pode referenciar tanto um atributo do nó propriamente dito como um atributo do descritor associado ao nó para apresentá-lo, como discutido na Seção C.10.

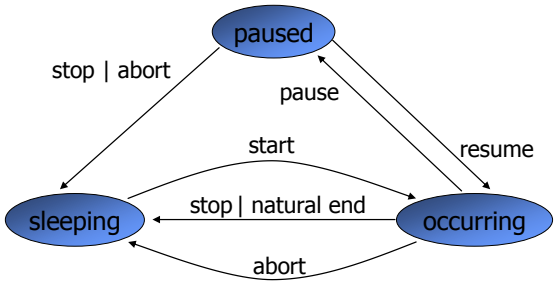


Figura C.4 Máquina de estados dos eventos NCM.

Um evento NCM pode estar em um dos seguintes estados: *dormindo* (*sleeping*), *ocorrendo* (*occurring*) ou *pausado* (*paused*). Todo evento possui um atributo denominado *ocorrências* (*occurrences*), que conta o número de vezes que o mesmo muda do estado *ocorrendo* para o estado *dormindo* durante a apresentação de um documento. Eventos como os de exibição e de atribuição também possuem um atributo denominado *repetições* (*repetitions*), que determina o número de vezes seguidas que o mesmo deve ocorrer. Esse atributo pode conter um valor finito ou o valor *indefinido*, que levará a uma execução em loop do evento, até que a mesma seja interrompida.

Intuitivamente, considerando um evento de exibição como exemplo (Figura C.4), o evento inicia no estado *dormindo*. Ao iniciar a exibição de suas unidades de informação, o evento passa para o estado *ocorrendo*. Se a apresentação for temporariamente suspensa, o evento vai para o estado *pausado* e no mesmo permanece enquanto a situação durar. Ao final da apresentação, o evento retorna para o estado *dormindo*, seu atributo ocorrências é incrementado de uma unidade, e o atributo repetições é decrementado de uma unidade. Se, após ser decrementado, o atributo repetições possuir um valor maior que zero, a apresentação do evento será reiniciada automaticamente. Quando uma apresentação de um evento é interrompida abruptamente, através de um comando de aborto da exibição, o evento passa para o estado *dormindo*, sem que o atributo ocorrências seja incrementado e tornando zero o valor do atributo repetições. Eventos de seleção permanecem no estado *ocorrendo* enquanto a âncora correspondente estiver sendo selecionada. De modo similar, eventos de arraste, foco e superposição permanecem no estado *ocorrendo* enquanto a respectiva operação sobre a âncora durar. Já os eventos de atribuição permanecem no estado *ocorrendo* enquanto os valores dos atributos estiverem sendo modificados. Evidentemente, eventos instantâneos, como uma simples atribuição de valor, podem permanecer por um tempo infinitesimal no estado *ocorrendo*.

Um evento de apresentação pode mudar do estado ocorrendo para dormindo em duas situações: como consequência de um término natural da exibição de suas unidades de informação ou devido a uma ação que force o término do evento.

A duração de um evento é o tempo que ele permanece no estado ocorrendo. No caso de um evento de apresentação, essa duração pode ser intrínseca ao objeto de mídia ou especificada pelo descritor do evento. A duração de um evento de apresentação será escolhida pelo formatador de documentos levando em consideração parâmetros intrínsecos ao conteúdo, parâmetros do descritor, relacionamentos do documento (principalmente os elos) e outras informações externas, como características da plataforma de exibição.

Um evento de apresentação associado com um nó de composição permanece no estado ocorrendo enquanto pelo menos um evento de apresentação associado com qualquer um dos nós filhos dessa composição estiver no estado ocorrendo ou enquanto pelo menos um elo filho do nó de composição estiver sendo avaliado.

Um evento de apresentação associado com um nó de composição está no estado pausado se pelo menos um evento de apresentação associado com qualquer um dos nós filhos da composição estiver no estado pausado e todos os outros eventos de apresentação associados com os nós filhos da composição estiverem no estado preparado ou pausado. Do contrário, o evento de apresentação está no estado dormindo.

Um evento de apresentação associado com um nó *switch* permanece no estado ocorrendo enquanto um elemento filho do *switch*, escolhido (nó selecionado) através das regras de ligação (*bind rules*), estiver no estado ocorrendo. Ele está no estado pausado se o nó selecionado estiver no estado pausado. Do contrário, o evento de apresentação está no estado dormindo.

Um evento de composição permanece no estado ocorrendo enquanto o mapa da composição estiver sendo apresentado.

Elos definidos nos nós de contexto, na verdade, especificam relacionamentos entre eventos definidos nas âncoras dos nós, mais precisamente entre máquinas de estados dos eventos, como será discutido na próxima seção. Com o objetivo de facilitar a explicação dos elos NCM, a Tabela C.1 define nomes para as transições de estados e também para as ações que produzem uma determinada transição de estado nas máquinas de estados dos eventos NCM.

Tabela C.1 Nomes de transições e ações para a máquina de estados dos eventos NCM

Transição (Causada pela Ação)	Nome da Transição
sleeping→ occurring (start)	*starts*
occurring→ sleeping (stop)	*stops*
occurring→ sleeping (abort)	*aborts*
occurring→ paused (pause)	*pauses*
paused→ occurring (resume)	*resumes*
paused→ sleeping (stop)	*stops*
paused→ sleeping (abort)	*aborts*

C.9 Elos

Um *elo* (*link*) é uma entidade NCM que possui duas propriedades adicionais: um conector e um conjunto de associações (*binds*) a esse conector.

O conector é uma entidade cujo objetivo é definir as semânticas das relações hipermídia, independentemente dos participantes que irão efetivamente interagir [Muchaluat *et al*., 2002]. Conectores recebem o *status* de entidades de primeira classe no modelo [Muchaluat *et al.*, 2001], isto é, os conectores podem ser definidos independentemente de outras entidades do modelo.

Elos representando o mesmo tipo de relação, mas interconectando participantes (nós) diferentes, podem reusar o mesmo conector.

Um conector especifica um conjunto de pontos de acesso de interface chamados papéis. Um elo NCM referencia um conector e define um conjunto de *binds* que relacionam cada extremidade do elo (ponto de interface de um nó) com um papel do conector referenciado.

A Figura C.5 apresenta a hierarquia de classes do elo NCM, discutida nas subseções seguintes.

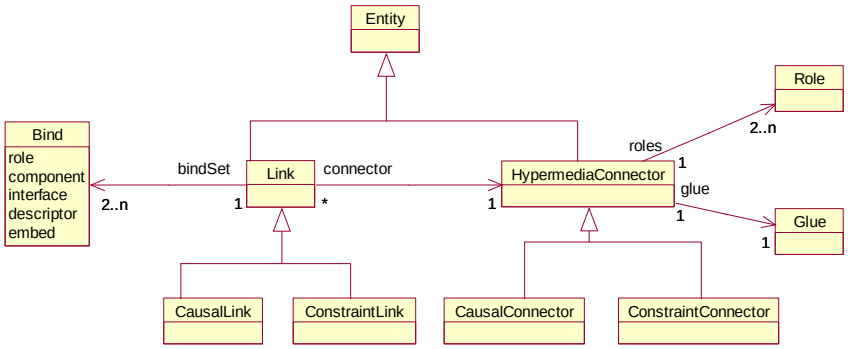


Figura C.5 Hierarquia de classes do elo NCM.

C.9.1 Conectores

A Figura C.6 ilustra um conector *R* representando uma relação com três papéis distintos, que significam três tipos de participantes da relação. A figura também mostra dois elos diferentes, l_1 e l_2, reusando *R*. Enquanto o conector define o tipo de relação, o conjunto de binds de um elo define os participantes. O elo l_1 especifica três binds, interligando os nós *A*, *B* e *C* aos papéis de *R*. O elo l_2 também especifica três binds, só que interligando um conjunto diferente de nós (*B*, *C* e *D*). Os elos l_1 e l_2 definem relacionamentos diferentes, já que interligam conjuntos distintos de nós, mas representam o mesmo tipo de relação, pois usam o mesmo conector. Na especificação de um documento NCM, um elo deve fazer referência a uma instância de conector.

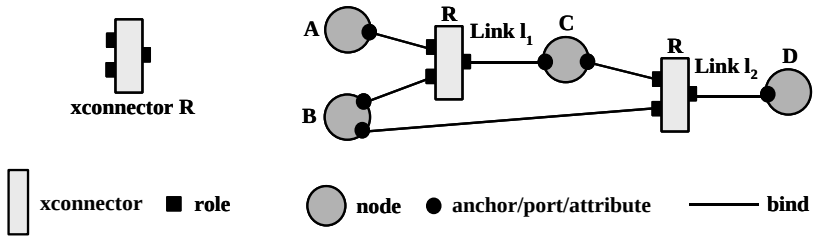


Figura C.6 Exemplos de elos que utilizam o mesmo conector *R*.

Conceitualmente, conectores podem representar qualquer tipo de relação hipermídia, tal como relações de referência, relações de sincronização, relações semânticas, relações de derivação etc. Essa versão 3.0 do NCM concentra seus esforços na especificação de relações de sincronização espaço-temporal, assim como relações de referência,[16] oferecendo o suporte necessário para a criação de documentos hipermídia.

O *conector* (*connector*) NCM permite a definição de relações multiponto com semântica causal ou de restrição. Em uma relação causal, uma condição deve ser satisfeita para que uma ação seja executada. Um exemplo de relação causal é a tradicional relação de referência hipermídia, que causa a navegação para um nó de destino quando uma âncora de um nó de origem for selecionada pelo usuário. Um outro exemplo de relação causal pode iniciar a apresentação de um nó quando a apresentação de outro terminar. Além de relações causais, relações de restrição, sem nenhuma causalidade envolvida, também podem ser especificadas por conectores

[16] Na realidade, as relações de referência são tratadas como casos particulares de relações de sincronização espaço-temporal.

NCM. Considere, por exemplo, uma restrição especificando que um nó deve terminar sua apresentação ao mesmo tempo que outro começa a dele. A ocorrência de uma apresentação sem a ocorrência da outra também satisfaz a restrição, que especifica que, se e somente se esses dois nós forem apresentados, seus tempos de fim e início, respectivamente, devem coincidir.

Para capturar relações causais e de restrição, conectores são especializados em conectores causais e conectores de restrição. Em ambos os tipos, a definição de um conector é feita por um conjunto de papéis (*roles*) que determinam a função dos participantes da relação e um atributo *glue*, que descreve como os papéis interagem. A definição de papéis é baseada no conceito de evento (Seção C.8). Cada papel descreve um evento associado a um participante da relação e o *glue* descreve a combinação entre os eventos de acordo com a semântica de causalidade ou de restrição.

Cada papel de um conector define um identificador único (*id*) no conjunto de papéis do conector, um tipo de evento e sua cardinalidade. O tipo de evento (*event type*) especifica o nome de uma das especializações da classe evento. A Tabela C.2 descreve os nomes para os tipos de evento NCM. A cardinalidade de um papel indica o número mínimo e máximo de participantes que podem desempenhar o papel (número de binds) quando esse conector for usado na criação de um elo, como será definido posteriormente.

Tabela C.2 Definições dos nomes para os tipos de evento dos conectores NCM

Especialização do Evento	Nome para o Tipo de Evento
Evento de apresentação	*presentation*
Evento de seleção	*selection*
Evento de superposição do dispositivo apontador	*pointOver*
Evento de arraste	*drag*
Evento de atribuição	*attribution*
Evento de foco	*focus*
Evento de colisão	*collision*
Evento de proximidade	*proximity*

Papéis são especializados em condição (*condition*), ação (*action*) e avaliação (*assessment*). Diferentes tipos de papéis são usados de acordo com o tipo de conector. Em conectores de restrição, somente papéis do tipo avaliação devem ser usados. Em conectores causais, qualquer tipo de papel pode ser usado. A Figura C.7 apresenta a hierarquia de classe dos papéis de conectores NCM.

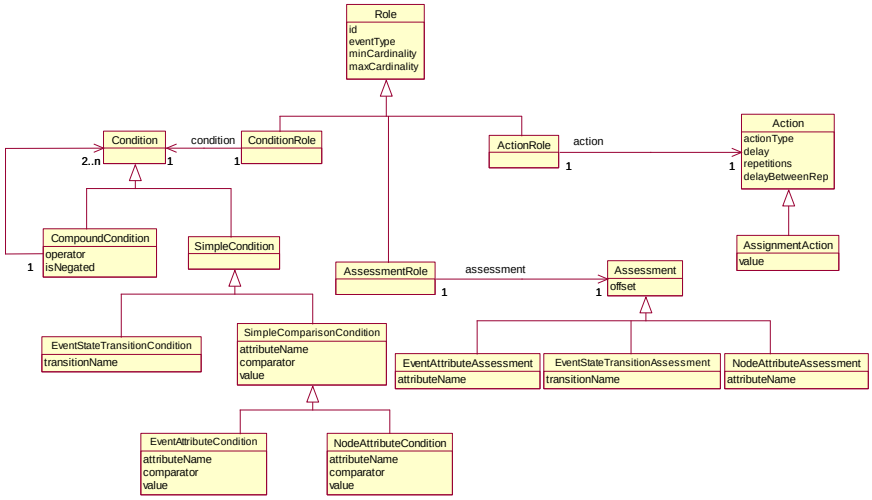


Figura C.7 Hierarquia de classes dos papéis de conectores NCM.

Papéis do tipo ação (*action roles*) capturam ações que devem ser executadas em uma relação causal. Os tipos de ação estão ilustrados na Figura C.4 por arcos rotulados, que causam transições na máquina de estados. Além de seu tipo, a ação de um papel pode definir um valor (*value*) a ser imposto a um atributo participante (no caso de eventos de atribuição). Um exemplo de papel do tipo ação é "**pause** a exibição de um evento de apresentação".

Ações NCM podem ser estendidas. Por exemplo, ações de animação podem ser especificadas definindo um valor inicial, um valor final e uma duração para que uma atribuição seja realizada (por exemplo, a posição *x* do objeto na tela, em um eixo de coordenadas *x* e *y*).

Em conectores causais, condições devem ser satisfeitas para disparar as ações. As condições são capturadas por papéis do tipo condição (*condition role*), que definem expressões lógicas avaliando transições nos estados dos eventos, valores de atributos dos eventos ou valores de atributos dos nós. Quando uma condição é avaliada, ela retorna um valor booleano.

As condições podem ser simples ou compostas. Uma condição simples (*simple condition*) pode testar uma transição de estado de um evento, o valor de um atributo de um evento ou o valor de um atributo de um nó. No caso da *condição sobre transição de estado do evento* (chamada *event state-transition condition*), o resultado do teste é considerado verdadeiro apenas no momento em que a transição ocorre, especificada em seu atributo *nome da transição* (*transition name*), conforme definido na Tabela C.3. A *condição sobre atributo* (*attribute condition*) deve especificar o tipo do atributo a ser

testado (*attibuteType*): estado de um evento ou atributo de um evento (occurences ou repetitions), associado por um elo a uma âncora de um nó, ou atributo de um nó (*nodeAttibute*), associado por um elo a um atributo qualquer de um nó. O atributo referenciado será comparado com o valor (*value*) especificado na condição, usando um dos seguintes comparadores: =, ≠, <, ≤, >, ≥. A *condição sobre atributo do nó* obriga que o tipo do evento definido no papel seja de atribuição, conforme discutido na Seção C.8. Para os eventos de seleção, o papel condição pode especificar, adicionalmente, a que dispositivo de seleção ele se refere (por exemplo, teclado ou teclas de um controle remoto), através do attributo *Key*.

Uma condição composta (*compound condition*) de um papel do tipo condição consiste em uma expressão lógica baseada nos operadores *and* ou *or* envolvendo duas ou mais condições que irão atuar sobre o mesmo evento. Um exemplo de papel com condição composta é "a apresentação de um participante terminou pela segunda vez", que seria especificada como "[(*eventType* = "presentation"), ((*transition* = "stops") AND (*occurrences* = "2"))]".[17] Note que qualquer condição composta pode ser negada utilizando o atributo booleano *está negada* (*isNegated*).

Enquanto uma condição sempre retorna um valor booleano, um papel de avaliação (*assessment role*) contém uma avaliação que retorna qualquer tipo de valor, dependendo do tipo do alvo da avaliação. Uma *avaliação de atributo* (*attribute assessment*) retorna o valor de um atributo do evento (*attributeType* igual a um dos atributos de evento: *occurences* ou *repetitions*) ou o valor de um estado de evento (*attributeType* igual a *state*), quando associado por um elo a uma âncora de um nó, ou retorna um valor de atributo de um nó (*attributeType* igual a *nodeAttribute*), associado por um elo. Uma *avaliação de transição de estado do evento* (*event-state transition assessment*) retorna o instante de tempo em que uma transição de estado do evento, especificada no atributo *nome da transição* (*transitionName*), ocorre. Ao se referir a um evento de seleção, um papel de avaliação pode especificar, adicionalmente, a que dispositivo de seleção ele se refere, através do atributo *key*.

Como mencionado anteriormente, um conector é definido por um conjunto de papéis e um *glue*, que especifica como os papéis interagem. Todo papel de um conector deve ser usado em seu *glue*. Um conector de restrição tem um *glue de restrição* (*constraint glue*), que define uma *expressão assertiva* (*statement expression*) relacionando papéis do tipo avaliação. Um conector causal tem um *glue causal* (*causal glue*), que define tanto uma *expressão de disparo* (*trigger expression*), relacionando papéis do tipo

[17] Operadores de condições compostas podem ser estendidos com outros tipos, como operadores de lógica temporal. Evidentemente, esses operadores terão de ser corretamente interpretados pelos formatadores dos documentos.

condição ou avaliação, quanto uma *expressão de ações* (*action expression*), relacionando papéis do tipo ação. Quando a expressão de disparo for satisfeita, a expressão de ações deverá ser executada. A Figura C.8 ilustra a hierarquia de classes definida pelo modelo para as expressões dos conectores NCM.

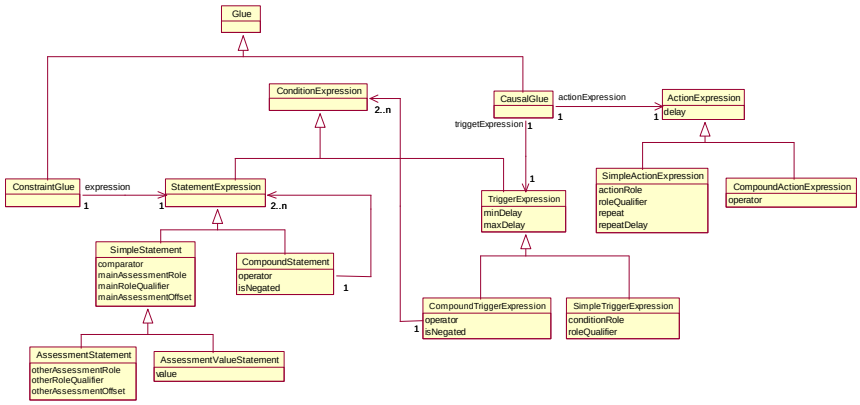


Figura C.8 Hierarquia de classes das expressões nos conectores NCM.

A expressão assertiva do *glue* de restrição pode ser simples ou composta. Uma *assertiva simples* (*simple statement*) pode comparar papéis de avaliação do mesmo tipo (*assertiva entre avaliações — assessment statement*) ou um papel de avaliação com um valor, do mesmo tipo, do resultado da avaliação (*assertiva de valor de avaliação — assessment value statement*). Um valor de deslocamento (*offset*) pode ser adicionado a um papel de avaliação antes da comparação. Por exemplo, um deslocamento pode ser adicionado a uma avaliação especificando: "5 segundos após o instante de tempo em que um evento de apresentação termina" ou, ainda, "a posição vertical na tela mais 50 pixels". A comparação pode usar os mesmos comparadores definidos para as condições simples. Por exemplo, suponha que um papel de avaliação de transição de estado de evento *P* especifique o tipo de evento como apresentação e a transição "starts" (início da ocorrência do evento), e que um outro papel de avaliação de transição de estado de evento *Q* especifique o tipo de evento como apresentação e a transição "stops" (término da ocorrência do evento). Se uma assertiva entre avaliações S_1 definir que "*P* = *Q*", S_1 será avaliada como verdadeira se um evento de apresentação associado a *P* iniciar ao mesmo tempo que um outro evento de apresentação

associado a *Q* terminar.[18] Como outro exemplo, suponha que um papel de avaliação *H* contenha uma avaliação de atributo do nó que especifique a posição horizontal na tela. Se uma assertiva de valor de avaliação S_2 definir que "$H \geq 100$", S_2 será avaliada como verdadeira se a posição horizontal de um participante desempenhando o papel *H* for maior que "100". Quando o valor da cardinalidade máxima de um papel é maior do que um, vários participantes podem desempenhar o mesmo papel. Nesse caso, um *qualificador* (*qualifier*) precisará ser definido cada vez que o papel for utilizado nas expressões do *glue*, como será explicado na descrição da Tabela C.4.

Uma *assertiva composta* (*compound statement*) consiste em uma expressão lógica, baseada nos operadores *and* ou *or*, envolvendo duas ou mais outras expressões assertivas. Assertivas compostas podem opcionalmente ser negadas.

Apesar de expressões assertivas poderem ser utilizadas em conectores causais, sua principal utilidade está na especificação de conectores de restrição. A semântica de um conector de restrição é a de que a expressão assertiva deve ser mantida verdadeira durante a apresentação. A Tabela C.3 ilustra um exemplo de conector de restrição expressando uma relação de sincronização espacial especificando que "dois nós devem ser alinhados pelo topo (o atributo *top* de seus descritores deve ser idêntico)".

Tabela C.3 Exemplo de conector de restrição

Tipo de Papel e Id	Tipo do Evento	Cardinalidade (min, max)	Nome do Atributo
Avaliação P_1	*attribuição*	*(1,1)*	*descriptor.top*
Avaliação P_2	*attribuição*	*(1,1)*	*descriptor.top*

Tipo do Glue	Expressão Assertiva
Restrição	$P_1 = P_2$

Uma expressão de disparo de um *glue* causal também pode ser simples ou composta. Uma *expressão de disparo simples* (*simple trigger expression*) se refere a um papel do tipo condição. Uma *expressão de disparo composta* (*compound trigger expression*) consiste em uma expressão lógica, baseada nos operadores *and* ou *or*, envolvendo duas ou mais outras expressões de disparo ou de assertiva. Uma expressão de disparo composta pode ser

[18] Como comentado, se o primeiro evento de apresentação não for iniciado ou o segundo evento de apresentação não terminar, a expressão permanece verdadeira.

opcionalmente negada. Qualquer expressão de disparo (simples ou composta) pode especificar retardos mínimo (*minimum delay*) e máximo (*maximum delay*) para sua avaliação. Por exemplo, dado que uma expressão de disparo *C* é verdadeira no instante *t*, *C'* definida com *minimum delay*="t_1" e *maximum delay*="t_2" é verdadeira no intervalo [$t+t_1$, $t+t_2$].[19]

Expressões de disparo compostas podem relacionar qualquer número de papéis de condição e de avaliação (através das expressões assertivas e expressões de condição). Entretanto, uma restrição é necessária para garantir a consistência de relações causais. Toda expressão de disparo associada a um conector causal deve ser satisfeita somente em um instante de tempo infinitesimal, exigindo que pelo menos um papel de condição de cada conector causal defina uma condição sobre uma transição de estado de um evento.

Uma *expressão de ações* (*action expression*) também pode ser simples ou composta. Uma *expressão de ações simples* (*simple action expression*) refere-se a um papel do tipo ação. Se o papel do tipo ação de uma expressão de ações simples é exercido por um evento do tipo de apresentação ou de atribuição, um atributo *repeat* pode ter seu valor imposto ao atributo repetições (*repetitions*) do evento. Um retardo entre repetições da ação (*repeat delay*) também pode ser especificado. Uma *expressão de ações composta* (*compound action expression*) consiste em uma expressão, baseada nos operadores *par*, *seq* ou *excl*, envolvendo duas ou mais outras expressões de ações. Expressões compostas de ações parelelas (*par*) ou sequenciais (*seq*) especificam que a execução das ações deve ser feita em qualquer ordem ou em uma ordem específica, respectivamente. Expressões compostas de ações exclusivas (*excl*) especificam que somente uma das ações deve ser executada. No último caso, o formatador do documento deve decidir qual das ações deve ser disparada ou por sua conta ou com o auxílio do usuário.

Uma *expressão de ações* pode também especificar um retardo (*delay*) que deve ser respeitado antes que a ação seja efetivamente executada.[20]

Como dito anteriormente, quando a cardinalidade máxima de um papel for maior que um, vários participantes podem desempenhar o mesmo papel. Nesse caso, um qualificador (*qualifier*) deve ser especificado toda vez que esse papel for usado em expressões do *glue*. A Tabela C.4 apresenta os possíveis valores para qualificadores.

---

[19] Note que o comportamento temporal das relações NCM também pode ser obtido utilizando o nó de tempo (Seção C.4), em vez de explorar os parâmetros de retardo do conector.

[20] Uma aplicação baseada no NCM pode permitir a parametrização desse valor e de outros atributos presentes nas expressões de um *glue* e nos papéis. Na linguagem NCL, por exemplo, uma mesma especificação de conector pode ser reusada, com valores diferentes de parâmetros derivando conectores NCM diferentes. De fato, a parametrização NCL é usada não apenas para atributos de ações, mas também para atributos de condições e de avaliações, especificados tanto nos papéis do conector quanto nas expressões de um *glue*. Parametrização, contudo, é uma questão de implementação e não de modelo. Por isso, ela é deixada para as definições das aplicações.

Tabela C.4 Valores para os qualificadores dos papéis com cardinalidade máxima maior que um

Tipo do Papel	Qualificador	Semântica
Condição	*all*	Todas as condições devem ser verdadeiras
Condição	*any*	Ao menos uma condição deve ser verdadeira
Avaliação	*all*	Todas as avaliações devem ser consideradas
Avaliação	*any*	Ao menos uma avaliação deve ser considerada
Ação	*par*	Todas as ações devem executar em paralelo
Ação	*seq*	Todas as ações devem executar em paralelo, mas respeitando a ordem em que os participantes foram associados ao papel
Ação	*excl*	Apenas uma das ações deve ser executada

A Tabela C.5 ilustra um exemplo de conector causal expressando uma relação de sincronização temporal. A especificação do conector pode ser interpretada como "se um grupo de participantes estiver sendo apresentado (C_1) e outro participante for selecionado (C_2), pare a apresentação do grupo de participantes (A_1) e inicie a apresentação de outro participante (A_2)". Para parar a apresentação do mesmo grupo de participantes que desempenhou o papel C_1, um elo usando esse conector deve criar dois binds para cada participante do grupo, um para o papel C_1 e outro para o papel A_1.

Tabela C.5 Exemplo de conector causal

Tipo do Papel e Id	Tipo do Evento	Cardinalidade (min, max)	Condição	Ação
Condição C_1	*apresentação*	*(1, unbounded)*	*state=occurring*	
Condição C_2	*seleção*	*(1, 1)*	*transition=stops*	
Ação A_1	*apresentação*	*(1, unbounded)*		*stop*
Ação A_2	*apresentação*	*(1, 1)*		*start*

Tipo do Glue	Expressão de Disparo	Expressão de Ações
Causal	*all(C_1) AND C_2*	*seq(par(A_1), A_2)*

Como a definição de conectores não é simples de ser feita por um usuário leigo, pois ele precisaria conhecer os conceitos de estados e transições de estados de eventos, a ideia é fazer com que usuários experientes definam conectores, os armazenem em bibliotecas, chamadas de *bases de conectores* (*connector bases*), e as tornem disponíveis para a criação de elos.

C.9.2. Binds de Elos

Conforme já mencionado, elos são definidos em uma composição (na realidade, em um nó de contexto). Um elo referencia um conector e define um conjunto de binds que associam cada extremidade do elo (interface dos nós) a um papel do conector referenciado. Binds são limitados a conectar interfaces de nós que estejam diretamente contidos em uma composição *C* onde o elo é definido ou âncoras da própria composição *C*. No entanto, uma vez que a porta de um nó de composição pode ser mapeada para a porta de um outro nó de composição interno, e assim por diante, até que a âncora de um nó seja alcançada, elos definidos em uma composição *C* podem indiretamente associar aos papéis do seu conector eventos definidos em qualquer nó recursivamente contido em *C*. Isso traz a noção de elos visíveis e elos contextuais, definidos a seguir.

Lembramos que, no NCM, nós de composição diferentes podem conter o mesmo nó, e os elos podem ser definidos em qualquer composição da perspectiva de um nó, sendo necessário identificar quais elos efetivamente ancoram em um nó ou passam por um nó (no caso de nós de composição), em uma dada perspectiva. Ao conjunto de elos que ancoram em um nó, em uma dada perspectiva *P*, chamamos de *elos contextuais de P*; ao conjunto de elos que ancoram ou passam por um nó de composição, em uma dada perspectiva, chamamos de *elos visíveis de P*.

Mais precisamente, dado um nó N_1 e uma perspectiva $P = (N_m,....,N_1)$, com $m>0$:

1. Um elo *l* é *visível em P* se e somente se existe um nó de composição N_i, para $i \in [1,m]$, tal que N_i ocorre em *P*, N_i contém *l* e:

   i) se $i>1$, então $(N_{i-1}, p...,N_1 ,p_1)$ define uma sequência de mapeamentos de uma porta (Seção C.5) *p* da composição N_{i-1} e *p* é diretamente associada a um papel do conector usado por *l*;

   ii) senão, N_1 é um nó que possui uma interface diretamente associada a um papel do conector usado por *l*.

2. Um elo *l* é *contextual em P* se e somente se existe um nó de composição N_i, para $i \in [1,m]$, tal que N_i ocorre em *P*, N_i contém *l* e:

i) se $i>1$, então $(N_{i-1}, p...,N_1, p_1)$ define uma sequência de mapeamentos de uma porta p da composição N_{i-1}, N_1 contém uma âncora que pertence à sequência de mapeamentos da porta p e p é diretamente associada a um papel do conector usado por l;

ii) senão, N_1 é um nó que possui uma âncora diretamente associada a um papel do conector usado por l.

Por exemplo, suponha que os nós *A* e *Z* contenham o nó *B*, que por sua vez contém os nós *C*, *D*, *E* e *F*, com elos ilustrados na Figura C.9. Então, a exibição do nó *C*, através da perspectiva (*A*, *B*, *C*), vai mostrar um elo da âncora *i* de *C* para a âncora *j* de *E* e um elo da âncora *m* de *C* para âncora *n* de *F*, definidos em *A* e *B*, respectivamente. Ambos são elos contextuais e visíveis em (*A*, *B*, *C*). A exibição do nó *B*, através da perspectiva (*A*,*B*), vai mostrar um elo de *B* para *B*, que é definido no nó *A*. Esse é o único elo visível em (*A*, *B*); não há elos contextuais em (*A*, *B*).

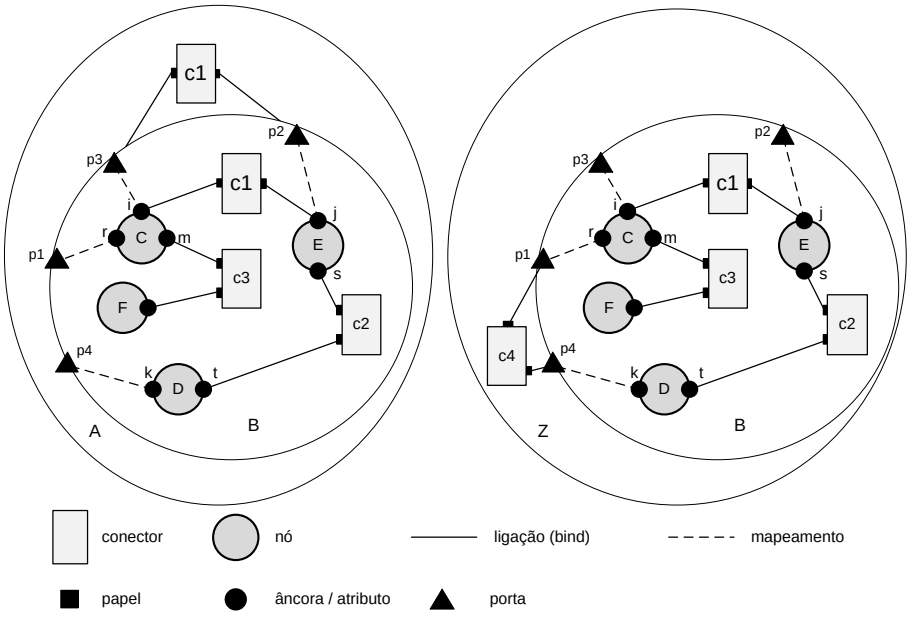


Figura C.9 Exemplos de elos visíveis em contextuais no NCM.

Por outro lado, a exibição do nó *C*, através da perspectiva (*Z, B, C*), vai mostrar um elo a partir da âncora *r* de *C* para a âncora *k* de *D* e um elo a partir da âncora *m* de *C* para a âncora *n* de *F*, definidos em *Z* e *B*, respectivamente. Ambos são elos contextuais e visíveis em (*Z, B, C*). A exibição do nó *B*, através da perspectiva (*Z, B*), vai mostrar um elo de *B* para *B*, definido no nó *Z*. Esse é o único elo visível em (*Z, B*), não havendo elos contextuais nessa perspectiva.

Binds de um elo possuem outros atributos além daqueles utilizados para associar uma interface a um papel: *descritor* e *embed*. Um atributo *descritor* é opcional e especifica um descritor genérico para o nó associado com o papel do conector. Se o nó associado for um nó de composição *N*, o descritor deve ser nulo. Note que vários binds para o mesmo nó de conteúdo com diferentes descritores levam a apresentações simultâneas do mesmo nó com diferentes características de exibição, similar ao que é proporcionado com o grupo de descritores e a navegação em profundidade discutida na Seção C.5.

O atributo *embed* é um atributo booleano que só deve ser usado na associação entre um papel de ação (somente para as ações *start* ou *prepare*) com um evento de apresentação *E*. Se o descritor do bind referenciar uma ferramenta de exibição (Seção C.11) que já esteja sendo utilizada para controlar a apresentação de um outro evento *F*, a ferramenta de exibição é solicitada a também controlar *E*, sem parar *F*, se o atributo *embed* for verdadeiro; caso contrário, se embed for igual a falso, a ferramenta de exibição deverá ser solicitada a substituir (parar) *F* por *E*. Quando não especificado, esse atributo deve ser considerado como igual a falso.

A definição de portas switch apresentada na Seção C.7 permite que elos (relacionamentos) sejam definidos ancorando em nós switch, independentemente do nó que será selecionado, como ilustrado na Figura C.10. As portas de composição tradicionais permitem que um elo seja associado a uma alternativa específica. Se essa alternativa não for selecionada, o elo será simplesmente ignorado durante a apresentação do documento.

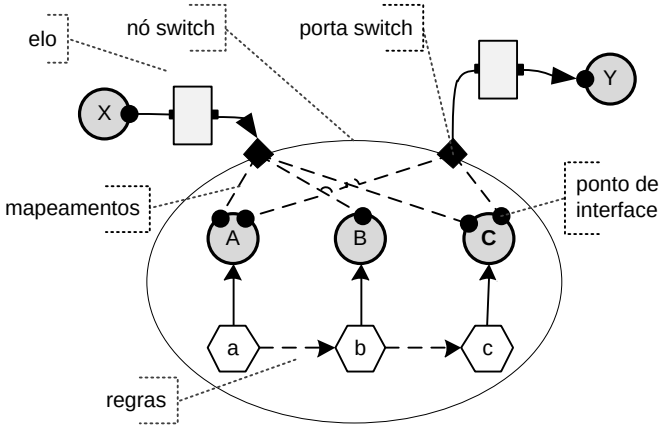


Figura C.10 Nós switch, portas switch e elos.

Com base nas entidades do modelo nós switch, portas switch e binds de elos, um autor pode especificar documentos que contemplem adaptação de elos. Para isso, basta colocar em um mesmo nó switch cópias de um mesmo nó, com diferentes elos associados às várias ocorrências do nó. A Figura C.11 oferece um exemplo de uso das alternativas para adaptar os relacionamentos. No exemplo, o elo que inicia a apresentação do nó *Z* só estará habilitado no documento se a regra *a* for avaliada como falsa e a regra *b* for avaliada como verdadeira.

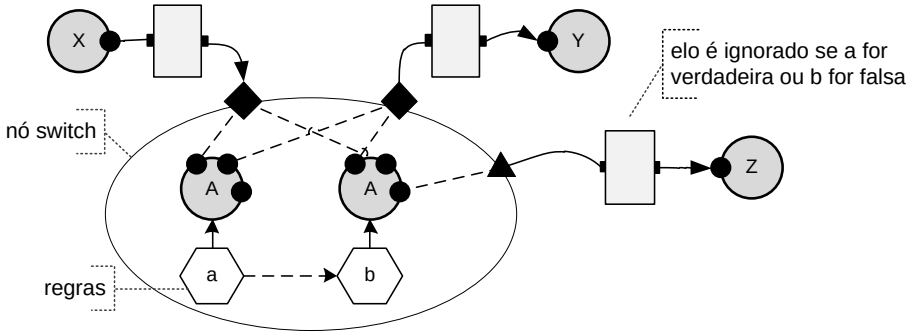


Figura C.11. Nós switch, portas switch e elos.

C.10 Objetos de Dados X Objetos de Representação

O NCM define um *objeto de dados* (*data object*) como uma entidade que compreende um nó NCM e todas as operações para manipulação desse nó, exceto as operações relacionadas com a apresentação do seu conteúdo (suas unidades de informação, conforme definido na Seção C.3). A funcionalidade de apresentar o conteúdo do nó é dada por uma instância de descritor que deve ser associada ao nó (objeto de dados).

A agregação de um objeto de dados e um descritor NCM é chamada de *objeto de representação*. A associação entre objetos de dados e descritores é representada na Figura C.12 por linhas conectando os objetos de dados no plano intermediário com os objetos de representação, desenhados no plano superior. Na figura, os nós são representados por círculos, e os elos por arcos e composições, pela inclusão de círculos e arcos em círculos maiores.

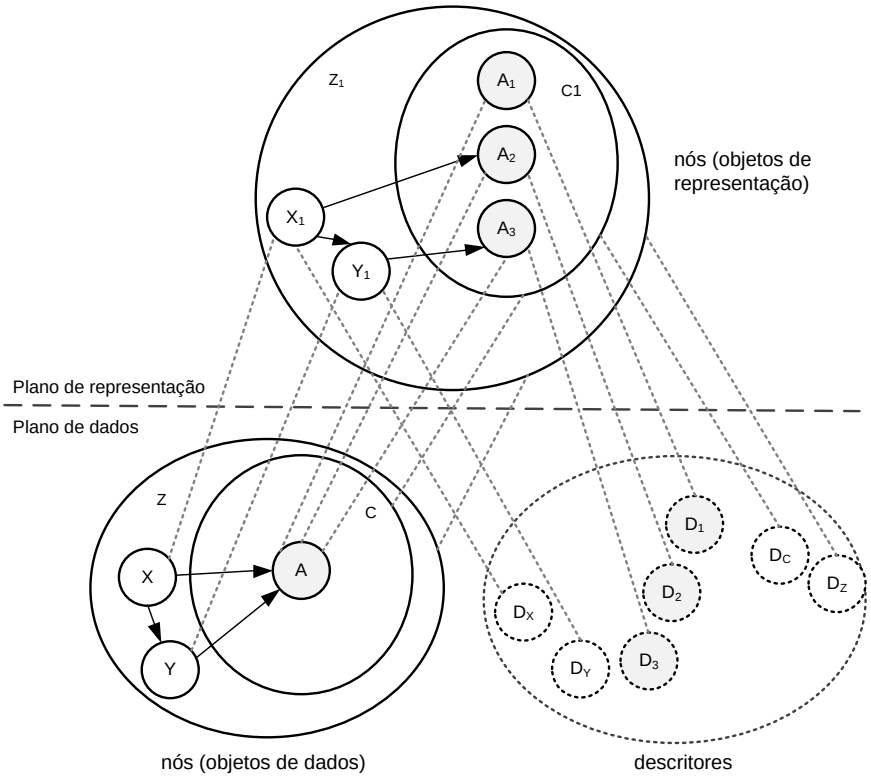


Figura C.12 Associação entre objetos de dados e descritores, gerando objetos de representação.

Note que um nó (objeto de dados) pode ser combinado a diferentes descritores, originando diferentes representações (objetos de representação) da mesma entidade. A figura mostra essa característica com a associação dos descritores D_1, D_2 e D_3 ao objeto de dados A, criando os objetos de representação A_1, A_2 e A_3. O nó A possui três diferentes representações porque existem, por exemplo, três diferentes formas de navegação até ele, através dos dois elos ou através da hierarquia de composições. Note, assim, que, devido ao fato de um mesmo objeto de dados poder gerar vários objetos de representação, um contexto objeto de representação pode conter um número de elementos diferente do contexto objeto de dados — por exemplo, o contexto C_1 da figura possui três nós, ao passo que o nó objeto de dados correspondente (contexto C) só tem um.

Pelas definições do NCM, um descritor pode ser definido em uma propriedade do nó, em um grupo de descritores da composição que contém o nó ou como um atributo de binds entre o nó e os elos. Além disso, descritores default podem ser especificados para as classes dos nós (texto, imagem etc.) ou explicitamente sugeridos pelos usuário leitor do documento. Se um nó possuir mais de um desses descritores especificados, o sistema de apresentação (formatador de documentos NCM) deverá construir um descritor resultante baseado na regra de cascateamento definida a seguir.

Suponha um nó N da classe C com a perspectiva corrente (C_k, …, C_1, N) alcançada através de uma navegação por elo (elo l). Seja D_1 um descritor definido para a classe do nó C, D_2 o descritor definido pela propriedade descritor de N, D_3 o único membro do grupo de descritores especificado para N em C_1, e D_4 um descritor especificado no bind feito para associar N ao conector referenciado pelo elo l. O descritor resultante será formado pela soma de todos os atributos/propriedades dos seguintes descritores: D_1, D_2, D_3, D_4. Se dois ou mais descritores definirem o mesmo atributo/propriedade com diferentes valores, D_4 (descritor do elo) terá prioridade sobre D_3 (descritor da composição), que terá prioridade sobre D_2 (descritor do nó), que terá prioridade sobre D_1 (descritor da classe). Se o usuário especificar um quinto descritor ao navegar para N, esse descritor será incluído na lista de cascateamento com precedência sobre D_4.

C.11 Descritores Genéricos, Descriptores e Switches de Descritores

Conforme mencionado anteriormente, descritores NCM agrupam informações das características de apresentação, objetivando separar essas informações do conteúdo do documento e de sua estrutura.

O NCM define uma classe *descritor genérico* (*generic descriptor*) que é especializada nas classes *descritor* (*descriptor*) e *switch de descritores* (*descriptor switch*). O descritor NCM é uma entidade que possui como propriedades adicionais: uma *especificação de início de apresentação*, uma *especificação de fim de apresentação* e uma *coleção de descrições de eventos*.

A propriedade *especificação de início de apresentação* deve conter um identificador da ferramenta de exibição (*player*) utilizada pelo formatador. Essa ferramenta será responsável por processar e exibir o conteúdo do nó durante a apresentação. Quando essa propriedade não estiver definida ou o seu valor apontar para uma ferramenta inexistente no formatador, o formatador de documentos deverá escolher um exibidor default com base no tipo de conteúdo do nó.

A especificação de início de apresentação pode também conter um atributo especificando se uma nova ferramenta de exibição deve ser instanciada ou se uma ferramenta já instanciada pode ser usada. Além disso, a especificação de início de apresentação pode conter um *conjunto de atributos particulares* que será interpretado pelo exibidor selecionado para controlar a apresentação do nó. Alguns exemplos desses parâmetros são:

- para nós com apresentações visíveis, como nós de texto, vídeo e imagem, um parâmetro *dispositivo* (*device*) especifica o dispositivo onde a apresentação ocorrerá; um parâmetro *região espacial* (*spatial region*) especifica uma localização para apresentar o conteúdo do nó, identificando a posição na tela do dispositivo especificado etc. Quando esses parâmetros não forem especificados, o formatador deverá escolher um dispositivo e uma área default para apresentação do nó;

- para nós de áudio, um parâmetro *dispositivo* (*device*) especifica o dispositivo de áudio onde o som deve ser apresentado; um parâmetro *volume* especifica o volume inicial de apresentação etc.[21] Quando esses parâmetros não forem especificados, o formatador deverá escolher um dispositivo e uma área default para apresentação do nó;

- para um nó de texto que deva ser exibido por uma ferramenta TtS (*text to speech*), parâmetros podem especificar a voz desejada, o sotaque etc.

A propriedade *especificação de fim de apresentação* define ações que devem ser executadas ao final de uma apresentação. Ela deve conter um

[21] O autor pode escolher exibir alguma informação visual associada ao áudio (por exemplo, uma barra de progressão temporal). Nesse caso, os parâmetros descritos no primeiro item também poderiam ser especificados.

atributo especificando o que acontecerá com a ferramenta de exibição ao final da apresentação, isto é, se a ferramenta será fechada ou permanecerá aberta, pronta para uma próxima apresentação. Neste último caso, deve também ser especificado se a ferramenta de exibição permanecerá escondida ou não.

Uma *descrição de um evento*, em um descritor, por sua vez, consiste na tupla <*IdÂncora*, *DurExplícita*, *FunçãoCusto*, *Rep*>. O parâmetro *IdÂncora* é um identificador de uma âncora do objeto de dados (nó) ao qual o descritor será associado para a formação do objeto de representação (essa âncora pode ser genericamente identificada por um rótulo ou por sua posição na lista de âncoras para permitir que um mesmo descritor seja reusado por mais de um nó). *DurExplícita* é opcional, somente podendo ser utilizada para eventos do tipo exibição, e sobrescreve a duração de apresentação ideal intrínseca ao conteúdo do nó. *FunçãoCusto* também é opcional, também só podendo ser usada em eventos do tipo exibição, e especifica uma métrica para guiar o formatador em ajustes que precisem ser feitos na duração de apresentação da âncora do nó. *Rep* especifica um valor para iniciar o atributo repetições do evento (Seção C.8). Em particular, todo descritor contém ao menos a descrição de evento <λ, *DurExplícita*, *FunçãoCusto*, *Rep*> associada à sua âncora de conteúdo total, onde, como já mencionado, *DurExplícita*, *FunçãoCusto* são opcionais.

De forma similar à entidade nó switch (Seção C.7), o switch de descritores contém uma lista ordenada de regras, estando cada regra associada a um descritor. O formatador de documentos deve percorrer a lista avaliando cada regra e selecionando o primeiro descritor cuja regra tiver sido satisfeita. Se nenhuma das regras for verdadeira, o switch de descritores pode especificar um *descritor default* para ser selecionado.

A entidade switch de descritores permite que formatadores de documentos NCM adaptem características da apresentação dos nós independentemente das informações estruturais e dos conteúdos. Obviamente, ambos os tipos de flexibilidade (adaptação de apresentação e adaptação de conteúdo) podem ser combinadas. Juntas com os ajustes de duração, essas características do modelo proporcionam um suporte flexível para que os autores especifiquem documentos e apresentações de documentos hipermídia adaptativos.

C.12 Trilhas

Características como grande número de nós e grande número de elos, muitas mudanças no documento, tempo de resposta ruim para as ações do usuário, diferenças visuais insuficientes entre elos e nós, e usuários não-orientados visualmente se combinam para dificultar os mecanismos de

navegação em um documento. Usuários desorientados precisam de informações de escopo para restabelecerem a noção de localização. Em particular, informações do escopo temporal são necessárias para responder a questões do tipo "como eu cheguei aqui?". Essas questões encontram suas respostas com a introdução do conceito de trilha.

Dada uma composição *C*, do tipo nó de contexto, base privada ou hiperbase pública,[22] uma *trilha T para C* é um nó de composição cujo conteúdo é uma lista ordenada de nós de conteúdo, nós de contexto ou outros nós de trilha, tais que todos os nós, que não são trilhas, estão recursivamente contidos em *C* e todos os seus nós de trilhas são trilhas para *C*. Mais ainda, *T* tem um atributo básico adicional denominado *nó corrente*, cujo valor indica a posição de um nó na lista ordenada de *T*, chamado de *entidade corrente de T*. Trilhas possuem um outro atributo básico adicional denominado *visão*, cujo valor associa a cada ocorrência de um nó *N*, na lista de *T*, um aninhamento de nós (N_m,....,N_1), com $m \geq 1$, e um descritor *D*, tal que $N_1 = N$, $N_m = C$, N_{i+1} é um nó de composição, N_i está contido em N_{i+1}, para $i \in [1,m)$. Diz-se que a trilha *T* é *associada* a *C*.

Toda trilha deve implementar o método deferido da classe composição:

- Insere nó: insere um nó na lista de nós da trilha, em uma posição especificada, com uma visão associada.

Adicionalmente, toda trilha deve implementar os seguintes métodos:

- *próximo*: se o atributo nó corrente não apontar para o último nó, incrementa o atributo nó corrente;
- *anterior*: se o atributo nó corrente não apontar para o primeiro nó, decrementa o atributo nó corrente;
- *primeiro*: coloca o atributo nó corrente apontando para o primeiro nó da lista;
- *último*: coloca o atributo nó corrente apontando para o último nó da lista;
- *ativa*: habilita os métodos próximo, anterior, primeiro e último, e inibe os métodos, insere nó e retira nó;
- *desativa*: desabilita os métodos próximo, anterior, primeiro e último, e habilita os métodos insere nó e retira nó.

A razão dos métodos ativa e desativa é não permitir que uma trilha seja usada ao mesmo tempo para navegação (pela trilha) e para manter o histórico da navegação. Ou ela é usada para a realização de uma tarefa ou da outra.

[22] Bases privadas e hiperbase pública serão definidas na próxima seção.

Note que um nó pode aparecer mais de uma vez na lista ordenada de *T*. Além disso, cada ocorrência do nó é associada com um aninhamento de nós relativos a *C*. Dessa forma, trilhas são úteis para linearização de documentos hipermídia e para implementação de *viagens guiadas*. Uma *trilha especial do sistema* pode manter a história de navegação do usuário durante uma sessão de trabalho, de tal modo que o usuário possa mover-se aleatoriamente entre os nós e depois recordar a navegação realizada. No entanto, se um nó *N* contido em um nó switch *S* pertencer a uma trilha, esse nó deverá obrigatoriamente ser a alternativa selecionada de *S* quando o usuário navegar pela trilha, independentemente de a regra associada ao nó não for mais verdadeira no momento da navegação através da trilha.

A definição de trilhas é importante para sistemas hipermídia, pois elas representam travessias ordenadas. Através delas, os autores podem fornecer ordens de leitura, que auxiliam leitores não-familiarizados com o material informativo a navegar pelo hiperdocumento ou determinar uma ordem apropriada de apresentação para uma certa audiência. Os usuários sentem-se menos desorientados quando seguem uma trilha já definida, pois é limitado o número de opções que possuem para percorrer o documento.

Seguindo o modelo NCM, uma implementação possível para trilhas que mantenham o histórico de navegação é criar uma *trilha principal* (ou *trilha do sistema*). Toda vez que um usuário navegar para um nó, esse nó, sua perspectiva corrente e o descritor resultante (descritor cascateado) são inseridos na trilha principal pelo sistema. Se o usuário decidir navegar através da trilha, é criada uma cópia da trilha principal, e o método ativa é chamado sobre essa cópia. Mesmo durante a navegação nessa cópia, a trilha especial do sistema é atualizada, de forma a sempre manter o histórico da navegação. Se o usuário fizer qualquer navegação fora da dada pela trilha cópia, a cópia é destruída.

C.13 Hiperbase Pública e Bases Privadas

A hiperbase pública é um conceito do NCM que representa o repositório público global das entidades disponíveis em um sistema hipermídia. A *hiperbase pública* (*public hyperbase*) é um nó de composição único que, estando um nó de composição *C* nele contido, então todos os nós recursivamente contidos em *C* devem também pertencer à hiperbase. A composição hiperbase pública tem como propriedade básica adicional um conjunto de descritores. Os descritores desse conjunto são aqueles utilizados para criação dos objetos de representação (veja Seção C.10), a partir do conjunto de nós contidos na hiperbase pública.

Uma *base privada* é um tipo especial de nó de composição, tal que:

i) ela pode conter nós de conteúdo, nós de contexto, nós switch, trilhas e outras bases privadas;

ii) uma base privada pode pertencer a, no máximo, uma base privada;

iii) se um nó de composição estiver contido em uma base privada *PB*, seus componentes podem estar contidos em *PB*, contidos na hiperbase pública ou contidos em alguma base privada recursivamente contida em *PB*.

Bases privadas possuem como propriedades básicas adicionais um conjunto de elos e um conjunto de descritores. Cada elo contido no conjunto de elos de um nó-base privada *PB* define um relacionamento entre nós recursivamente contidos em *PB*. Os descritores no conjunto de descritores de uma base privada são aqueles utilizados para criar os objetos de representação (veja Seção C.10) a partir dos nós recursivamente contidos na base privada.

Intuitivamente, uma base privada reúne todas as entidades utilizadas durante uma sessão de trabalho do usuário.

Bibliografia

Muchaluat-Saade D.C., Soares L.F.G. Hypermedia Spatio-Temporal Synchronization Relations Also Deserve First Class Status. VIII Multimedia Modeling Conference. MMM'2001, Amsterdam, novembro de 2001.

Muchaluat-Saade D.C., Rodrigues R.F., Soares L.F.G. "XConnector: Extending XLink to Provide Multimedia Synchronization." ACM Symposium on Document Engineering. DocEng'02, Virginia,. novembro de 2002.

Pérez-Luque M.J., Little T.D.C. "A Temporal Reference Framework for Multimedia Synchronization." IEEE Journal on Selected Areas in Communications (Special Issue: Synchronization Issues in Multimedia Communication), 14(1). Janeiro de 1996, pp. 36-51.

Soares, L.F.S. e Rodrigues, R.F. "Nested Context Model 3.0 Part 1 – NCM Core," Monografias em Ciência da Computação do Departamento de Informática, PUC-Rio, N.º 18/05. Rio de Janeiro, maio de 2005. ISSN 0103-9741.

Soares, L.F.S. e Rodrigues, R.F. "Nested Context Model 3.0 Part 8 – NCL (Nested Context Language) Digital TV Profiles." Monografias em Ciência da Computação do Departamento de Informática, PUC-Rio, N.º 35/06. Rio de Janeiro, outubro de 2006. ISSN 0103-9741.

Apêndice D

Exemplo de
Base de Conectores

Este apêndice descreve os conectores causais usados nos exemplos do Capítulo 3, *O Primeiro João*, e do Capítulo 15, sobre a utilização de múltiplos dispositivos de exibição.[1]

[1] O conteúdo deste capítulo está disponível sob a Licença Creative Commons Atribuição — Uso Não-Comercial — Compartilhamento pela mesma Licença 3.0 Unported, conforme publicado em clube.ncl.org.br.

D.1 Conectores Causais

No Capítulo 3 tivemos a oportunidade de discutir a concepção de alguns dos conectores causais apresentados no documento NCL da Listagem D.1. Outros conectores, contudo, foram apenas referenciados naquele capítulo e no Capítulo 15.

Neste apêndice, nos limitaremos apenas a apresentar as definições dos conectores usados nos Capítulos 3 e 15, sem discutir os detalhes de suas concepções. No Capítulo 10 o leitor pode encontrar uma discussão geral e mais aprofundada sobre a concepção de conectores causais.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="causalConnBase"
                xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
   <head>
      <connectorBase>

         <causalConnector id="onBeginStart">
           <simpleCondition role="onBegin"/>
           <simpleAction role="start" max="unbounded" qualifier="par"/>
         </causalConnector>

         <causalConnector id="onBeginStart_delay">
           <connectorParam name="delay"/>
           <simpleCondition role="onBegin"/>
           <simpleAction role="start" delay="$delay" max="unbounded"
                                                     qualifier="par"/>
         </causalConnector>

         <causalConnector id="onEndStop">
           <simpleCondition role="onEnd"/>
           <simpleAction role="stop" max="unbounded" qualifier="par"/>
         </causalConnector>

         <causalConnector id="onBeginSet_varStart">
           <connectorParam name="var"/>
           <simpleCondition role="onBegin"/>
           <compoundAction operator="seq">
             <simpleAction role="set" value="$var"/>
             <simpleAction role="start" max="unbounded"
                                                     qualifier="par"/>
           </compoundAction>
         </causalConnector>

         <causalConnector id="onKeySelectionStopSet_varStart">
           <connectorParam name="var"/>
           <connectorParam name="keyCode"/>
           <simpleCondition role="onSelection" key="$keyCode"/>
           <compoundAction operator="seq">
             <simpleAction role="stop" max="unbounded"
                                                     qualifier="par"/>
             <simpleAction role="set" value="$var"/>
             <simpleAction role="start" max="unbounded"
                                                     qualifier="par"/>
           </compoundAction>
         </causalConnector>

         <causalConnector id="onEndSet_var">
           <connectorParam name="var"/>
```

```
    <simpleCondition role="onEnd"/>
    <simpleAction role="set" value="$var"/>
  </causalConnector>

  <causalConnector id="onKeySelectionStopStart">
    <connectorParam name="keyCode"/>
    <simpleCondition role="onSelection" key="$keyCode"/>
    <compoundAction operator="seq">
      <simpleAction role="stop" max="unbounded"
                                              qualifier="par"/>
      <simpleAction role="start" max="unbounded"
                                              qualifier="par"/>
    </compoundAction>
  </causalConnector>

  <causalConnector id="onKeySelectionSet_var">
    <connectorParam name="keyCode"/>
    <connectorParam name="var"/>
    <simpleCondition role="onSelection" key="$keyCode"/>
    <simpleAction role="set" value="$var"/>
  </causalConnector>

  <causalConnector id="onBeginVarStart">
    <compoundCondition operator="and">
      <simpleCondition role="onBegin"/>
      <assessmentStatement comparator="eq">
        <attributeAssessment role="var"
          attributeType="nodeProperty" eventType="attribution"/>
        <valueAssessment value="true"/>
      </assessmentStatement>
    </compoundCondition>
    <simpleAction role="start"/>
  </causalConnector>

  <causalConnector id="onBeginStartSet_var_delay_duration">
    <connectorParam name="var"/>
    <connectorParam name="delay"/>
    <connectorParam name="duration"/>
    <simpleCondition role="onBegin"/>
    <compoundAction operator="seq">
      <simpleAction role="start"/>
      <simpleAction role="set" value="$var" delay="$delay"
                                        duration="$duration"/>
    </compoundAction>
  </causalConnector>

  <causalConnector id="onSelectionSet_varStop">
    <connectorParam name="var"/>
    <simpleCondition role="onSelection"/>
    <compoundAction operator="seq">
      <simpleAction role="set" value="$var" max="unbounded"
                                              qualifier="par"/>
      <simpleAction role="stop"/>
    </compoundAction>
  </causalConnector>

  <causalConnector id="onSelection_orSet_varStopStart">
    <connectorParam name="var"/>
    <simpleCondition role="onSelection" qualifier="or"
                                              max="unbounded"/>
    <compoundAction operator="seq">
      <simpleAction role="set" value="$var" max="unbounded"
                                              qualifier="par"/>
      <simpleAction role="stop"/>
      <simpleAction role="start"/>
    </compoundAction>
```

```
        </causalConnector>

        <causalConnector id="onBeginSet_var">
          <connectorParam name="var"/>
          <simpleCondition role="onBegin"/>
          <simpleAction role="set" value="$var"/>
        </causalConnector>

        <causalConnector id="onEndSet_varStop_delay">
          <connectorParam name="var"/>
          <simpleCondition role="onEnd"/>
          <compoundAction operator="seq">
            <simpleAction role="set" value="$var"/>
            <simpleAction role="stop" max="unbounded" qualifier="par"
                                                      delay="3s"/>
          </compoundAction>
        </causalConnector>
      </connectorBase>
   </head>
</ncl>
```

Listagem D.1 Exemplo de base de conectores causais.

Apêndice E

NPT
Normal Play Time

Quando se trata de um fluxo elementar enviado por difusão sem solicitação, o conceito de tempo de exibição de um objeto de mídia não pode ser estabelecido, uma vez que podemos sintonizar (começar a receber) o fluxo em qualquer ponto e, portanto, não há como saber quanto tempo já se passou desde seu início (ou até mesmo o que seja o início). Para possibilitar uma solução para o problema, o padrão MPEG-2 criou o conceito de *Normal Play Time*, ou *NPT*, que é o assunto deste apêndice.

E.1 Introdução

Um receptor pode começar a receber um fluxo elementar enviado sem solicitação a partir de qualquer ponto temporal do mesmo, uma vez que o instante de sintonização é qualquer um. Mais ainda, é usual que um fluxo elementar carregue mais de um conteúdo de objeto de mídia. Por exemplo, um fluxo elementar de uma determinada emissora de TV carrega um programa em sequência do outro. Pior ainda, um conteúdo pode ser entremeado por outro. Por exemplo, quando uma propaganda é inserida no meio de um programa de TV. Tudo isso torna impossível saber em que instante de tempo estamos em um determinado conteúdo, se não houver algum auxílio extra.

Para possibilitar a solução do problema, o padrão MPEG-2 permite ao difusor especificar, para o receptor, em que ponto uma mídia está sendo apresentada, por meio da definição de um código de tempo associado a fluxos elementares. Denominado *Normal Play Time*, ou *NPT*, esse código de tempo é transportado em um tipo especial de descritor em uma seção privada MPEG-2, discutida no Capítulo 1.

NPT é uma linha de tempo sobre a duração de um evento, definido pelo padrão MPEG-2 [ISO/IEC 13818-11998] como sendo uma coleção de fluxos elementares com a mesma base temporal, com um tempo de início e um tempo de fim associado.

Um NPT pode começar de qualquer valor e provê uma referência de tempo para um segmento de mídia. Embora o NPT cresça no decorrer de um fluxo de mídia, seus valores podem apresentar descontinuidades. Isso significa que, mesmo se um fluxo de mídia for editado, seu valores NPT não necessitam mudar. A Figura E.1 ilustra um exemplo. Na figura, o NPT original é apresentado, bem como o NPT final, após os cortes em cinza-claro no vídeo e a inserção de um comercial em cinza-escuro.

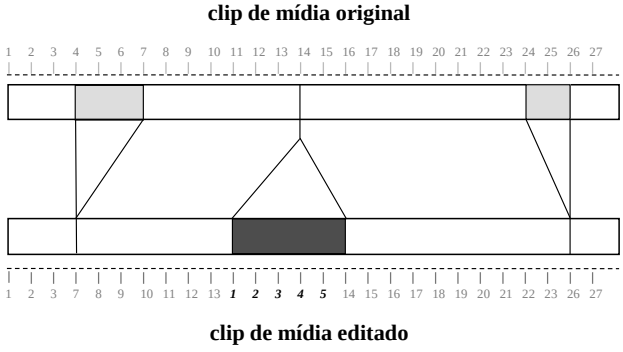


Figura E.1 NPT em uma mídia editada.

Valores NPT podem ser aninhados. Para identificar a que conteúdo o NPT se refere, os *descritores de referência NPT* têm dois campos: o valor de NPT (NPT_Reference) e um identificador (contentId) do conteúdo a que ele se refere. Por exemplo, na Figura E.1, o *contentId* do comercial inserido (cinza-escuro na figura) é diferente do restante do vídeo.

Assim, uma descontinuidade em um NPT pode ser reconhecida como sendo uma simples edição no fluxo original ou uma fronteira entre dois diferentes segmentos de mídia.

Cada descritor de referência NPT também inclui um valor de taxa, especificando para o receptor quantas pulsações do relógio STC (System Time Clock) do fluxo associado correspondem a uma pulsação do NPT. Essa taxa não precisa ser constante durante toda uma transmissão de um segmento de mídia. A taxa é especificada por dois campos do descritor: scaleNumerator e scaleDenominator. Quando os dois campos são iguais a 1, significa que o NPT está mudando a uma taxa equivalente ao STC. Se o campo scaleNumerator é igual a 0 e o scaleDenominator é diferente de zero, isso indica que o NPT não está mudando em relação ao STC, ou seja, tem um valor constante. Se os dois campos têm o valor zero, é indicado que o descritor não carrega os dois campos mencionados. Um scaleNumerator diferente de zero com scaleDenominator igual a zero não é permitido.

A Tabela E.1 ilustra a sintaxe de um descritor de referência NPT. Nela, dois campos adicionais devem ser observados: postDiscontinuityIndicator e STC_Reference. O primeiro, se receber o valor "1", indica que o descritor de referência NPT será válido na próxima descontinuidade da base temporal STC; se receber o valor "0", indica que o descritor é válido no momento de sua recepção. O segundo campo indica o valor de STC quando o valor de NPT é aquele dado no campo NPT_Reference.

Tabela E.1 Descritor de Referência NPT

Sintaxe	N.º. de Bits
NPT ReferenceDescriptor(){	
descriptorTag (igual a 0x01)	8
descriptorLength	8
postDiscontinuityIndicator	1
contentId	7
reserved	7
STC_Reference	33
reserved	31
NPT_Reference	33
scaleNumerator	16
scaleDenominator	16
}	

Como mencionamos, um NPT pode começar (e obviamente terminar) em qualquer valor. Para informar a um cliente receptor (que, como vimos, pode sintonizar um fluxo com base temporal NPT em qualquer ponto do tempo) os valores inicial e final do NPT de um evento corrente, o MPEG define um outro descritor chamado *NPT Endpoint Descriptor*, ilustrado na Tabela E.2.

Tabela E.2 Descritor de Endpoint NPT

Sintaxe	N.º de Bits
NPTEndpointDescriptor(){	
descriptorTag (igual a 0x02)	8
descriptorLength	8
reserved	15
StartNPT	33
reserved	31
stopNPT	33
}	

O Capítulo 10 discute como o NPT é usado para definir pontos de sincronização entre objetos de mídia de um documento NCL e como o *contentId* de um descritor NPT é associado a um trecho do fluxo.

Bibliografia

ISO/IEC 13818-1 (2000). International Organization for Standardization/International Eletrotecnical Committee, "Information Technology — Generic coding of moving pictures and associated audio information, Part 1: Systems", *ISO/IEC 13818-1*.

ISO/IEC 13818-6 (1998). International Organization for Standardization/International Eletrotecnical Committee, "Information Technology — Generic coding of moving pictures and associated audio information, Part 6: Extensions for DSM-CC", *ISO/IEC 13818-6*.

Steven Morris (2004). Interactive TV Web. "A technical (and non-technical) guide to DSM-CC." http://interactivetvweb.org/tutorial/dtv-intro/dsm-cc/index.shtml. Acesso em 21 de março de 2008.

Balabanian, Casey and Greene (1996). Vahe Balabanian; Liam Casey; Nancy Greene. *An Introduction to Digital Storage Media — Command and Control (DSM-CC)*. Nortel, 1996.

Apêndice F

Transporte de Comandos de Edição em Seções MPEG-2

Comandos de Edição NCL podem vir de diferentes modos, como discutido no Capítulo 16, que se dedica à descrição desses comandos.

Neste apêndice nos centraremos na discussão das diversas formas como esses comandos podem ser transportados, tanto através de estruturas de dados recebidas sem solicitação como através de estruturas de dados recebidas sob demanda, mas em especial usando Seções MPEG-2.[1]

[1] Este capítulo foi baseado em Soares *et al.*, (2006). O uso do material foi gentilmente cedido pelo Departamento de Informática da PUC-Rio.

F.1 Introdução

Como mencionamos no Capítulo 16, os parâmetros dos comandos de edição podem vir embutidos no descritor de evento de comando ou podem vir em estruturas de dados externas, referenciadas pelos descritores de evento. Esse é sempre o caso dos comandos de edição *addDocument* e *addNode*. Nesta seção vamos ver alguns exemplos dessas estruturas de dados e como elas são transportadas. Vamos ver, também, como os descritores de evento podem ser "envelopados" e transportados.

F.2 Transporte de Comandos por Descritores de Eventos de Fluxo e Carrossel de Objetos DSM-CC

Em sistemas de TV digital terrestre é comum o uso do protocolo DSM-CC para transporte de comandos de edição em fluxos elementares de transporte MPEG-2.

Comandos de edição NCL podem ser transportados usando descritores de evento de fluxo DSM-CC. Como especificado em ISO/IEC 13818-6 [1998], descritores de evento de fluxo DSM-CC têm uma estrutura muito parecida com a dos descritores de evento NCL apresentados no Capítulo 16 (ver Figura 16.1). Na verdade, a estrutura dos descritores de evento NCL foi definida para tornar o mais fácil possível esse mapeamento (no caso, tornou trivial). A Tabela F.1 apresenta a sintaxe de um descritor de evento de fluxo DSM-CC para transporte de comandos de edição NCL, salientando em itálico os campos adicionados ao descritor de evento definido no Capítulo 16.

Tabela F.1 Descritor de Eventos de Fluxo para Comandos de Edição

Sintaxe	N.º de Bits
StreamEventDescriptor ( ) {	
descriptorTag	8
descriptorLenght	8
eventId	16
reserved	31
eventNPT	33
privateDataLength	8
commandTag	8
sequenceNumber	7
finalFlag	1
privateDataPayload	8 a 2008
FCS	8
}	

Para transmissão dos parâmetros dos comandos de edição, o carrossel de objetos DSM-CC pode ser usado [ISO/IEC 13818-6, 1998]. O protocolo de carrossel de objetos DSM-CC permite a transmissão cíclica de objetos de eventos de fluxo e sistemas de arquivos, como vimos no Apêndice B.

Em um carrossel de objetos, os objetos de eventos de fluxo são utilizados para mapear nomes de eventos de fluxo em *ids* de eventos de fluxo, definidos pelos descritores de evento. No caso dos eventos de comando NCL, ele é responsável por mapear o nome *nclEditingCommand* em *eventId*s definidos nos descritores de evento. Os nomes dos eventos permitem especificar tipos de eventos, oferecendo maior nível de abstração às aplicações. O gerenciador da base privada, definido no Capítulo 16, deve se registrar como observador dos eventos com os quais lida, utilizando nomes de evento; no caso de comandos de edição, o nome "*nclEditingCommand*".

Como mencionamos no Apêndice B, além dos objetos de eventos de fluxos, os carrosséis de objetos DSM-CC podem também ser usados para o transporte de arquivos organizados em diretórios. Um demultiplexador DSM-CC é responsável por montar a estrutura recebida no dispositivo receptor. Parâmetros dos comandos de edição especificados como documentos XML (documentos NCL ou entidades NCL que se quer adicionar) podem, assim, ser organizados em sistemas de arquivos a serem transportados nesses carrosséis, como alternativa ao transporte dentro dos próprios descritores de evento que definem os comandos. O gerador de carrossel de objetos é o responsável por juntar esses arquivos e os objetos de eventos de fluxo em fluxos elementares de dados MPEG-2.

Assim, quando um comando de edição NCL precisa ser enviado, um objeto de eventos DSM-CC deve ser criado, mapeando a *string* "*nclEditingCommand*" em uma *id* de evento de fluxo, e então colocado em um carrossel de objetos DSM-CC, que é enviado em um fluxo elementar (o tipo do fluxo deve ter o valor de 0x0B). Um ou mais descritores de evento de fluxo DSM-CC com a *id* previamente selecionada podem ser então criados e enviados em outro fluxo elementar MPEG-2 TS. Esses eventos de fluxo podem ter sua referência de tempo colocadas em zero, mas também podem ser adiados para serem executados em um tempo específico. O gerenciador da base privada deve se registrar como um ouvinte "nclEditingCommand" e será notificado quando esses eventos de fluxo chegarem. O *commandTag* recebido é então utilizado pelo gerenciador da base privada para interpretar a semântica da *command string*.

Se, no descritor de evento de comando de edição, o *command parameter* baseado em XML for curto o suficiente, ele pode ser transportado diretamente no *payload* dos descritores de evento. Se não for, o *privateDatePayload* transportará um conjunto de pares de referência. Nesse caso, a especificação XML deve ser enviada no mesmo carrossel de objetos que carrega o objeto de eventos. O parâmetro *uri* do primeiro par de referências deve ter o caminho absoluto da especificação XML (o caminho no servidor de dados). O parâmetro *id* correspondente no par deve fazer referência ao IOR da especificação XML (*carouselId*, *moduleId*, *objectKey*) [ISO/IEC 13818-6, 1998] no carrossel de objetos. Em comandos de edição diferentes de addDocument e addNode, o parâmetro *uri* pode ser omitido. Se outros sistemas de arquivos precisarem ser transmitidos usando outros carrosséis de objeto a fim de completar o comando de edição (como é usual nos comandos addDocument ou addNode) com conteúdo de mídia, outros pares {*uri*, *id*} devem estar presentes no comando. Nesse caso, o parâmetro *uri* deve ter o caminho absoluto da raiz do sistema de arquivos (o caminho no servidor de transmissão de dados), e o respectivo parâmetro *ior* no par deve fazer referência ao IOR (*carouselId*, *moduleId*, *objectKey*) de qualquer arquivo ou diretório filho da raiz no carrossel de objetos (o *service gateway* do carrossel, conforme apresentado no Apêndice B).

A Figura F.1 ilustra um exemplo de transmissão de documento NCL, por meio de um carrossel de objetos, a ser adicionado por meio de um comando *addDocument*. Nesse exemplo, um provedor de conteúdo quer transmitir o programa interativo chamado "weatherConditions.ncl", armazenado em um de seus servidores de dados (sistema local de arquivos, de acordo com a Figura F.1).

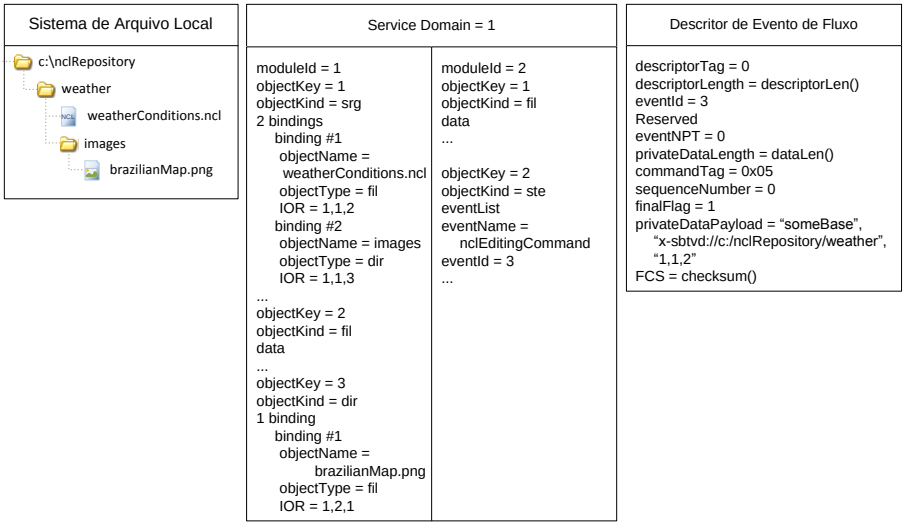


Figura F.1 Exemplo de uma transmissão de documento NCL em carrossel DSM-CC.

Um carrossel de objetos deve então ser gerado (domínio de serviço = 1, na Figura F.1), carregando todo o conteúdo do programa interativo (arquivo .ncl e todos os arquivos de mídia) e também um objeto de eventos (*moduleId* = 2 e *objectKey* = 2, de acordo com a Figura F.1) mapeando o nome “nclEditingCommand” para o valor de *eventId* (valor “3” na Figura F.1).

Um descritor de evento de fluxo também deve ser transmitido com o valor de *eventId* apropriado, no exemplo “3”, e o valor “0x05” de *commandTag*, que indica um comando addDocument (ver Capítulo 16). O parâmetro *uri* conterá opcionalmente o esquema (no caso “x-sbtvd”, indicando o recebimento no carrossel) e o caminho absoluto do documento NCL (“C:\nclRepository\weather” de acordo com Figura F.1). Finalmente, o IOR do documento NCL no carrossel de objetos é transportado no parâmetro *xmlDocument* (*carouselId* = 1, *moduleId* = 1, *objectKey* = 2, também de acordo com a Figura F.1).

Note que os pares {uri, id} transportados nos descritores de evento são suficientes para que a identificação dos recursos da aplicação no dispositivo receptor possa ser feita de forma idêntica àquela realizada no ambiente de autoria do documento, a despeito de os identificadores dos recursos usarem URLs absolutas ou relativas e de um carrossel de objetos usados no transporte fazer referência a recursos transportados em outro carrossel. Por exemplo, a aplicação “weatherConditions.ncl” poderia se referir ao arquivo “brazilianMap.jpg” de forma absoluta ou relativa. Mais ainda, a aplicação poderia se referir a um conteúdo armazenado em um provedor de conteúdos diferente do da aplicação, que deverá, então, ser transportado em um

carrossel de objetos diferente daquele que transporta a aplicação. O mapeamento para a localização dos recursos é realizado automaticamente pelo gerenciador de bases privadas.

F.3 Transporte de Comandos de Edição Usando Estruturas Específicas Definidas pelo Ginga-NCL

Na Seção F.2 discutimos o transporte de parâmetros de comandos de edição em seções MPEG-2 transportando carrosséis de objetos. Nesta seção discutiremos o uso de outras estruturas para transporte desses parâmetros [ABNT, NBR 15606-2, 2011].

Descritores de evento podem ser enviados em fluxo elementar MPEG-2 TS usando eventos de fluxo DSM-CC, como vimos na seção anterior, como também usando qualquer protocolo para transmissão de dados sem solicitação ("pushed data").

Três tipos de estrutura de dados são definidos [Soares *et al*., 2006] para dar suporte à transmissão de parâmetros dos comandos de edição NCL, além da estrutura de descritor de evento já definida: mapa, metadados e arquivos de dados.

Para estruturas de *mapa*, o campo *mappingType* identifica o tipo do mapa. Se o valor de mappingType for igual a "0x01" ("events"), um mapa de eventos é caracterizado. Nesse caso, depois do campo mappingType, vem uma lista de identificadores de eventos, como definido na Tabela F.2. Outros valores para o campo mappingType podem ser definidos, mas não são relevantes para nossa discussão.

Tabela F.2 Lista de Identificadores de Eventos Definidos pela Estrutura de Mapa

Sintaxe	Número de Bits
mappingStructure ( ) {	
mappingType	8
for (i=1; i<N; i++){	
eventId	8
eventNameLength	8
eventName	8 to 255
}	
}	

Mapas do tipo “events” (mapas de eventos) são usados para mapear nomes de eventos em *eventIds* dos descritores de evento. Mapas de eventos são usados para indicar quais eventos devem ser recebidos (qualquer semelhança com a função dos objetos de eventos de fluxo DSM-CC não é mera coincidência). Nomes de eventos permitem especificar tipos de eventos, oferecendo maior nível de abstração às aplicações. Assim, quando precisamos enviar um comando de edição NCL, devemos criar um mapa de eventos, mapeando a string “*nclEditingCommand*” em um *eventId* previamente selecionado de um descritor de evento. Um ou mais descritores de evento com o *eventId* previamente selecionado deverão ser criados e enviados. Esses descritores de evento podem ter os seus tempos de referência como zero ou podem ter sua execução postergada para um tempo especificado. O gerenciador de bases privadas de um sistema receptor deve se registrar como ouvinte de um evento “*nclEditingCommand*” para ser notificado da chegada desse tipo de evento.

Cada estrutura *arquivos de dados* é de fato o conteúdo de um arquivo que compõe uma aplicação NCL ou um documento XML definindo uma entidade NCL: um arquivo contendo a especificação XML da aplicação ou entidade NCL, ou um dos arquivos com conteúdo de mídia da aplicação ou de um nó NCL (vídeo, áudio, texto, imagem, ncl, lua etc.).

Uma estrutura de *metadados* é um documento XML, como definido no esquema a seguir. Note que o esquema define, para cada dado entregue sem solicitação (*pushed file*), uma associação entre sua localização no sistema de transporte (identificação do sistema de transporte — atributo *component_tag* — e a identificação do arquivo no sistema de transporte — atributo *structureId*) e seu identificador de recurso universal (atributo *uri*).

```
<!--
XML Schema for NCL Section Metadata File

This is NCL
Copyright: 2000-2005 PUC-RIO/LABORATORIO TELEMIDIA, All Rights
Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCLSectionMetadataFile.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Section Metadata File namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:NCLSectionMetadataFile="http://www.ncl.org.br/
                                          NCLSectionMetadataFile"
  targetNamespace="http://www.ncl.org.br/NCL3.0/
                                          NCLSectionMetadataFile"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <complexType name="NCLSectionMetadataType">
    <sequence>
```

```
        <sequence>
          <element ref="NCLSectionMetadataFile:baseData"
                              minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
        <element ref="NCLSectionMetadataFile:pushedRoot"
                                    minOccurs="0" maxOccurs="1"/>
        <sequence>
          <element ref="NCLSectionMetadataFile:pushedData"
                              minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
    </sequence>
    <attribute name="name" type="string" use="optional"/>
    <attribute name="size" type="positiveInteger" use="optional"/>
  </complexType>

  <complexType name="baseDataType">
    <sequence>
        <element ref="NCLSectionMetadataFile:pushedRoot"
                                    minOccurs="0" maxOccurs="1"/>
        <sequence>
          <element ref="NCLSectionMetadataFile:pushedData"
                              minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
    </sequence>
    <attribute name="uri" type="anyURI" use="required"/>
  </complexType>

  <complexType name="pushedRootType">
    <attribute name="component_tag" type="positiveInteger"
                                                  use="optional"/>
    <attribute name="structureId" type="string" use="required"/>
    <attribute name="uri" type="anyURI" use="required"/>
    <attribute name="size" type="positiveInteger" use="optional"/>
  </complexType>

  <complexType name="pushedDataType">
    <attribute name="component_tag" type="positiveInteger"
                                                   use="optional"/>
    <attribute name="structureId" type="string" use="required"/>
    <attribute name="uri" type="anyURI" use="required"/>
    <attribute name="size" type="positiveInteger" use="optional"/>
  </complexType

  <!-- declare global elements in this module -->
  <element name="metadata"
              type="NCLSectionMetadataFile:NCLSectionMetadataType"/>
  <element name="baseData"
                      type="NCLSectionMetadataFile:baseDataType"/>
  <element name="pushedRoot"
                      type="NCLSectionMetadataFile:pushedRootType"/>
  <element name="pushedData"
                      type="NCLSectionMetadataFile:pushedDataType"/>

</schema>
```

Para cada arquivo de documento NCL ou outros arquivos de documento XML usados nos comandos de edição *addDocument* ou *addNode*, pelo menos uma estrutura de metadados deve ser definida. Apenas um arquivo de aplicação NCL ou um arquivo de documento XML representando um nó NCL a ser inserido pode ser definido por estrutura de metadados. Mais precisamente, pode haver apenas um elemento <pushedRoot> em um

documento XML representando o metadados. Contudo, uma aplicação NCL (e seus arquivos de conteúdo) ou um documento de especificação de um nó NCL (e seus arquivos de conteúdo) pode se estender por mais de uma estrutura de metadados. Mais ainda, podem existir estruturas de metadados sem qualquer aplicação NCL ou documento XML descritos em seus elementos <pushedRoot> e <pushedData>.

As três estruturas anteriormente definidas podem ser transmitidas usando sistemas de transporte diferentes, como exemplificado no que se segue.

F.3.1 Transporte em um Tipo Específico de Seção MPEG-2

O uso de um tipo específico de seção MPEG-2 (identificado por um valor específico do campo table_id de uma seção privada MPEG-2, conforme apresentado no Apêndice B), a partir de agora chamada de seção NCL, nos permite transmitir as três estruturas de dados anteriormente definidas: mapas, metadados e arquivos de dados.

Cada Seção NCL pode conter os dados de apenas uma estrutura. Contudo, uma estrutura pode se estender por várias seções. Estruturas de dados podem ser transmitidas em qualquer ordem e quantas vezes forem necessárias.

O começo de uma estrutura de dados é delimitada pelo campo payload_unit_start_indicator de um pacote TS. Depois dos quatro bytes do cabeçalho TS, a carga (*payload*) do pacote TS começa, com um campo ponteiro de um byte indicando o início de uma seção NCL [ISO/IEC 13818-1, 2000]. O cabeçalho da seção NCL é então definido como uma seção MPEG-2 [ISO/IEC 13818-1, 2000]. O primeiro byte da carga (*payload*) da seção NCL identifica o tipo da estrutura transportada (0x01 para metadatos; 0x02 para arquivos de dados e 0x03 para mapa de eventos). O segundo byte carrega um identificador único da estrutura (*structureId*) no fluxo elementar de transporte.[2]

Depois do segundo byte vem uma estrutura de dados serializada que pode ser a mappingStructure (como ilustrado pela Tabela F.2) ou a estrutura de matadados (um documento XML), ou uma estrutura de arquivos de dados

[2] O fluxo elementar e o identificador da estrutura são aqueles que devem ser associados pela estrutura de metadados, através dos atributos component_tag e structureId dos elementos <pushedRoot> e <pushedData>, a localizadores de arquivos (URL).

(um conteúdo de arquivo serializado). O demultiplexador de seções NCL é responsável por montar toda a estrutura da aplicação no dispositivo receptor.[3]

No mesmo fluxo elementar que carrega a especificação XML (o arquivo do documento NCL ou outro documento XML usado nos comandos de edição) é recomendado que um arquivo mapa de eventos seja transmitido, para que seja mapeado o nome "*nclEditingCommand*" no *eventId* do descritor de evento que deverá transportar os comandos de edição, como descrito no Capítulo 16. O campo *privateDataPayload* do descritor de evento deve carregar o par de referências {uri, id}, como sempre. Os parâmetros *uri* têm sempre o valor "null". No caso dos comandos *addDocument* e *addNode*, o parâmetro *id* do primeiro par deve identificar o fluxo elementar ("component_tag") e a estrutura de metadados que ele transporta ("structureId"). A estrutura de metadados, por sua vez, contém o caminho absoluto do documento NCL ou da especificação do nó NCL (o caminho no servidor de dados) e a estrutura arquivos de dados relacionada ("structureId") transportada em seções NCL do mesmo fluxo elementar. Se outras estruturas de metadados adicionais forem necessárias para completar os comandos de edição *addDocument* ou *addNode*, outros pares {uri, id} devem estar presentes no comando. Nesse caso, os parâmetros *uri* devem também ter o valor "null" e os parâmetros *id* correspondentes devem referir-se ao component_tag e à estrutura de metadados transportada (structureId) correspondente.

A Figura F.2 ilustra um exemplo de transmissão de um documento NCL através de seções NCL. Nesse exemplo, um provedor de conteúdo quer transmitir um programa interativo chamado "weatherConditions.ncl", armazenado em um de seus servidores de dados (sistema de arquivo local na Figura F.2). Um fluxo elementar MPEG-2 (component_tag= "0x09") deve então ser gerado, carregando todo o conteúdo do programa interativo (o arquivo ncl e todos os arquivos de conteúdo de mídia). Um mapa de eventos também deve ser gerado (structureType="0x03"; structureId="0x0C", na Figura F.2), mapeando o nome "*nclEditingCommand*" ao valor de eventId (valor "3", Figura F.2). Um descritor de evento também deve ser transmitido, com o valor apropriado para eventId, no exemplo o valor "3", e o valor de commandTag igual a "0x05", que indica um comando *addDocument* (veja Capítulo 16). O parâmetro uri deve ter o valor "null" e o parâmetro id deve ter o valor (component_tag= "0x09", structureId= "0x0B"), como na Figura F.2.

[3] É importante salientar que seções NCL podem também transportar as estruturas de dados definidas encapsuladas em outras estruturas de dados. Por exemplo, o MPE (Multi-protocol Encapsulation) pode ser usado. Nesse caso, seções NCL seriam seções MPEG-2 de datagrama. Mais ainda, todas as estruturas de dados mencionadas podem ser envelopadas em outro formato de dados de protocolo, como, por exemplo, pacotes FLUTE.

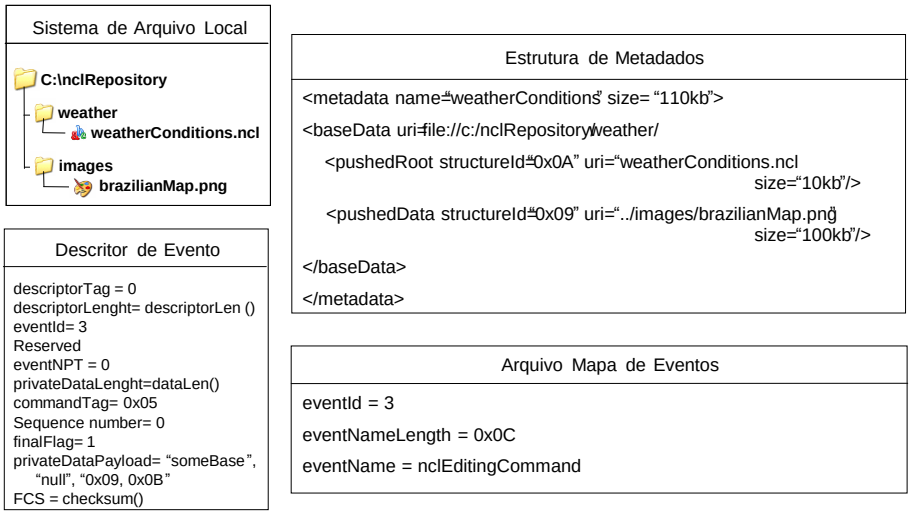


Figura F.2 Exemplo de transmissão de documento NCL usando seções NCL MPEG-2.

F.3.2 Transporte das Estruturas de Metadados como Parâmetro de Comandos de Edição

Uma alternativa ao transporte de estruturas de metadados diretamente em seções NCL é tratar essas estruturas como parâmetros dos comandos *addDocument* and *addNode*, transportados no campo *privateDataPayload* dos descritores de evento.

Nesse caso, o conjunto de pares {uri, id} dos comandos *addDocument* e *addNode* é substituído por parâmetros da estrutura de metadados, que definem um conjunto de pares {“uri”, “component_tag, structureId”} para cada arquivo transmitido sem solicitação (*pushed file*).

Voltando ao exemplo da Figura F.2, o novo transporte seria exatamente o mesmo, com exceção do descritor de evento. Em vez de esses descritores terem o par {uri; id} igual ao valor {“null”; “0x09, 0x0B”} como parâmetro, eles teriam a estrutura de metadados serializada. Na estrutura de metadados, os atributos *component-tag* dos elementos <pushedRoot> e <pushedData> devem, nesse caso, ser definidos, uma vez que a estrutura não é mais transportada no mesmo fluxo elementar que transporta os arquivos da aplicação NCL.

F.3.3 Transporte de Estruturas de Metadados em Seções de Metadados MPEG-2

Uma outra alternativa é transportar as estruturas de metadados em seções de metadados MPEG-2, transportadas em fluxos MPEG-2 do tipo "0x16" [ISO/IEC 13818-1, 2000]. Como sempre, cada seção de metadados MPEG-2 poderá conter dados de apenas uma estrutura de metadados. Contudo, uma estrutura de metadados pode se estender por várias seções de metadados.

A Tabela F.3 ilustra a sintaxe da seção de metadados para o transporte de estruturas de metadados, que devem estar de acordo com ISO/IEC 13818-1, 2000.

Tabela F.3 Sintaxe da Seção para o Transporte de Estruturas de Metadados

Sintaxe	N.º de Bits	Valor
Metadata section() {		
table_id	8	0x06
section_syntax_indicator	1	1
private_indicator	1	1
random_access_indicator	1	1
decoder_config_flag	1	0
metadata_section_length	12	Inteiro
metadata_service_id	8	Inteiro
reserved	8	
section_fragment_indication	2	De acordo com a Tabela F.4
version_number	5	Inteiro
current_next_indicator	1	1
section_number	8	Inteiro
last_section_number	8	Inteiro
structureId	8	Inteiro
For (i=1; i< N; i++) {		
serialized_metadata_structure_byte	8	
}		
CRC_32	32	
}		

Tabela F.4 Indicação de Fragmento de Seção

Valor	Descrição
1	Uma única seção de metadados carregando uma estrutura de metadados completa.
1	Primeira seção de metadados de uma série, com dados de uma mesma estrutura de metadados.
0	Última seção de metadados de uma série, com dados de uma mesma estrutura de metadados.
0	Uma seção de metadados de uma série, com dados de uma mesma estrutura de metadados, mas que não é nem a primeira nem a última seção.

Como anteriormente, no mesmo fluxo elementar que transporta a especificação XML (o arquivo do documento NCL ou um outro arquivo XML usados nos comandos de edição), devemos transmitir um arquivo mapa de eventos a fim de mapear o nome "*nclEditingCommand*" ao *eventId* do descritor de evento que carregará o comando de edição. No caso dos comandos *addDocument* e *addNode*, o campo *privateDataPayload* do descritor de evento deve transportar um conjunto de pares de referência {uri, id}. Os parâmetros *uri* devem ter o valor "null". No caso de comandos *addDocument* e *addNode*, o parâmetro *id* do primeiro par deve identificar o fluxo elementar ("component_tag") do tipo= "0x16" e a estrutura de metadados ("structureId") que carrega o caminho absoluto do documento NCL ou da especificação do nó NCL (o caminho no servidor de dados). Se outras estruturas de metadados forem usadas para relacionar arquivos presentes no documento NCL ou na especificação do nó NCL, a fim de completar os comandos *addDocument* ou *addNode* com conteúdos de mídia, outros pares de referência {uri, id} devem ser definidos no comando. Nesse caso, o parâmetro *uri* deve ter o valor "null", e o parâmetro *id* correspondente no par deve referir-se ao component_tag e ao structureId do metadado correspondente.

Voltando ao exemplo da Figura F.2, com a nova estrutura de transmissão tudo seria similar. Devemos fazer apenas pequenas mudanças, de forma que o descritor de evento refira o fluxo elementar e sua seção que carrega a estrutura de metadados ("component_tag= "0x08" e structureId= "0x0B"), e que a estrutura de metadados também refira o fluxo elementar onde os arquivos do documento serão transportados. A Figura F.3 ilustra a nova situação.

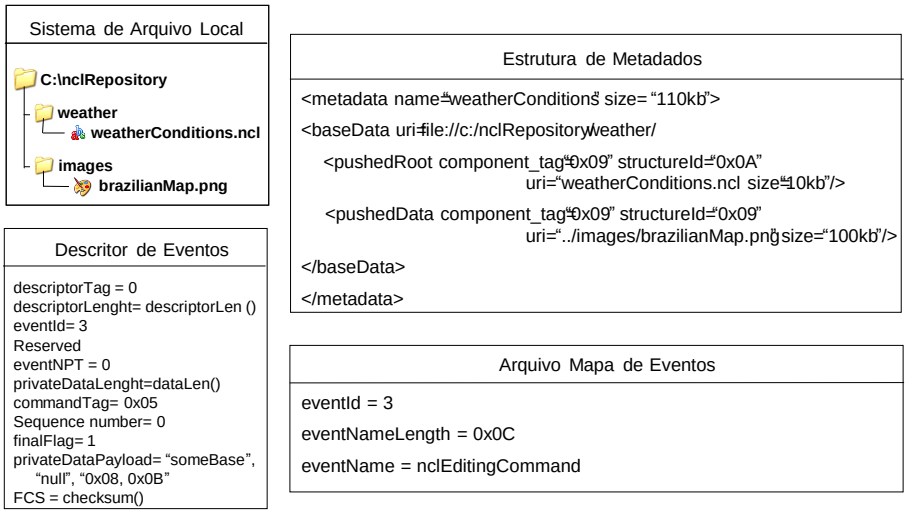


Figura F.3 Exemplo de transmissão de documento NCL usando seções de metadados MPEG-2.

Bibliografia

ABNT, NBR 15606-2 (2011). Associação Brasileira de Normas Técnicas, "Televisão digital terrestre — Codificação de dados e especificações de transmissão para radiodifusão digital, Parte 2: Ginga-NCL para receptores fixos e móveis — Linguagem de aplicação XML para codificação de aplicações", Sistema Brasileiro de TV Digital Terrestre, *NBR 15606-2*.

ISO/IEC 13818-1 (2000). International Organization for Standardization/International Eletrotecnical Committee, "Information Technology — Generic coding of moving pictures and associated audio information, Part 1: Systems", *ISO/IEC 13818-1*.

ISO/IEC 13818-6 (1998). International Organization for Standardizatio International Eletrotecnical Committee, "Information Technology — Generic coding of moving pictures and associated audio information, Part 6: Extensions for DSM-CC", *ISO/IEC 13818-6*.

Soares, L.F.S. e Rodrigues, R.F.; Costa, R.R; Moreno, M. (2006). "Nested Context Language 3.0 Part 9 — NCL Live Editing Commands." Monografias em Ciência da Computação do Departamento de Informática, PUC-Rio, N.º 35/06. Rio de Janeiro, dezembro de 2006. ISSN 0103-9741.

Apêndice G

HTG

A apresentação com qualidade de uma aplicação NCL depende de uma série de procedimentos de escalonamento que devem ser realizados tanto pelos receptores quanto pelos provedores de conteúdo das aplicações.

Para a realização desses procedimentos, estruturas de dados específicas devem ser computadas. Este apêndice traz uma discussão a respeito dessas estruturas de dados e de como elas podem ser deduzidas de uma outra estrutura de dados única chamada HTG (Hypermedia Temporal Graph).

G.1 Escalonadores

Como já discutimos em diversas parte do livro, nas aplicações em que o sincronismo depende da ocorrência de eventos com duração variável ou mesmo imprevisível no momento da especificação, é imperativo que essa especificação seja realizada de forma relativa à ocorrência desses eventos, isto é, independentemente do momento temporal em que eles ocorrem e se de fato eles ocorrerem. É nesse paradigma orientado a eventos que se baseia a linguagem NCL e a sua linguagem de script Lua.

Quando o sincronismo é especificado de forma relativa, os instantes temporais de ocorrência dos eventos somente serão conhecidos na fase de execução da aplicação.

O uso do paradigma orientado a eventos traz não só o problema de como especificar relacionamentos temporais e espaciais entre eventos, solucionados pelo uso de NCL, como dois outros problemas adicionais:
1) como controlar a execução de uma aplicação de forma a garantir que os relacionamentos especificados sejam respeitados;
2) como gerenciar as transmissões, dos servidores de conteúdos para os clientes receptores, mantendo uma qualidade de serviço tal que garanta que os conteúdos estejam presentes no receptor nos momentos necessários de suas apresentações.

O primeiro problema diz respeito apenas aos receptores das aplicações e vai exigir o trabalho complementar de dois escalonadores na implementação do formatador NCL.

O primeiro escalonador (“escalonador de exibidores”) é responsável por instanciar os vários exibidores de mídia, de forma que eles estejam prontos nos momentos necessários à apresentação dos vários objetos. Esse trabalho não é simples porque, devido à limitação de memória da maioria dos dispositivos receptores para TV, é importante manter o mínimo de componentes exibidores instanciados ou mesmo na memória principal (RAM) do receptor.

O segundo escalonador (“escalonador de apresentação”) é o núcleo central do formatador, sendo responsável por entregar os conteúdos a serem apresentados pelos exibidores de mídia em um dado instante. Em outras palavras, é esse escalonador que de fato resolve todos os relacionamentos de sincronismo temporal em tempo de execução, transformando os tempos relativos dos relacionamentos em tempos absolutos.

Entre os vários problemas que deve tratar, o escalonador de apresentação deve permitir que uma aplicação inicie a partir de qualquer ponto de sua cadeia temporal de exibição de objetos. Isso é muito importante em aplicações para TV digital, por exemplo, na qual um serviço ou canal

pode ser sintonizado em qualquer instante de tempo de um programa de TV. O escalonador deve também permitir que uma aplicação sofra operações de pausa e retomada, que exigem que ela prossiga, mesmo em um instante temporal posterior ao da pausa. Esse tipo de operação é comum quando o usuário troca e, em seguida, retorna ao mesmo canal (ou serviço) de TV.

Para enfrentar o segundo problema, o gerenciamento das transmissões, devemos definir procedimentos, tanto nos provedores de conteúdo quanto nos clientes receptores. Esses procedimentos dependem de os dados serem recebidos sem solicitação (*pushed data*) ou sob demanda (*pulled data*).

No caso de dados recebidos sem solicitação, eles são geralmente enviados ciclicamente (em carrossel), como discutido em detalhes no Apêndice F. A colocação e a retirada dos dados no carrossel e em qual posição do carrossel são fundamentais para diminuir a banda necessária para transmissão dos dados e para garantir que eles estarão presentes no receptor quando necessário. O "escalonador de carrossel" é o responsável por essas tarefas nos provedores de conteúdo.

Do lado do cliente receptor, um "escalonador de pré-busca" é necessário para retirar, nos momentos precisos, dados dos carrosséis e carregá-los na memória do receptor. Esse escalonador é também responsável por requisitar dados sob demanda, de forma a garantir sua presença no dispositivo receptor quando necessário. Devemos notar, mais uma vez, que devido à usual limitação de memória dos dispositivos receptores é importante manter o mínimo de conteúdos de mídia na memória principal. Para busca desses dados, pode ser necessária a negociação de QoS (*Quality of Service*) na rede. Nesse caso, um "escalonador de negociação de QoS" é responsável pela tarefa, garantindo a reserva de recursos na rede quando o escalonador de pré-busca requisitar os dados.

Para dar suporte à realização das tarefas executadas pelos diversos escalonadores mencionados, várias estruturas de dados são computadas a partir da especificação da aplicação NCL. Na implementação de referência do *middleware* Ginga, elas são baseadas e derivadas de uma estrutura de dados única, denominada *Hypermedia Temporal Graph* ou HTG.

G.2 Hypermedia Temporal Graph (HTG)

Para controlar o comportamento temporal das aplicações durante a sua execução, o Ginga-NCL adota uma estrutura formada por grafos direcionados (dígrafos). Essa estrutura, conhecida como *Hypermedia Temporal Graph* ou HTG[Costa *et al.*, 2008; Moreno *et al.*, 2008], oferece suporte ao início ou à retomada síncrona da apresentação a partir de um instante de tempo qualquer.

No HTG, cada vértice representa uma transição de estado de um evento. Os eventos representados nessa estrutura são os mesmos definidos na NCL, ou seja, os eventos de apresentação, seleção e atribuição, cada um controlado por uma máquina de estados, repetida por comodidade de leitura na Figura G.1.

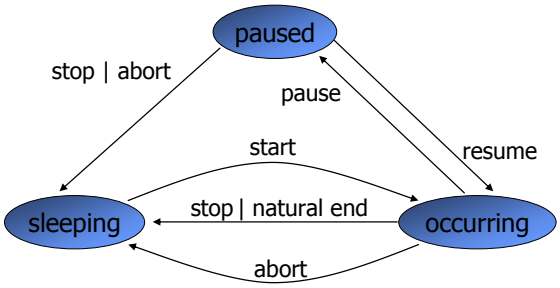


Figura G.1 Máquina de estado de um evento**.**

O grafo temporal obtido através da modelagem de uma aplicação deve preservar todos os relacionamentos entre os eventos, sejam eles previsíveis ou imprevisíveis, bem como o estado atual de cada máquina de estados associada a um evento.

No HTG, as arestas entre os vértices representam os "relacionamentos entre as transições de estados dos eventos". Cada aresta possui uma condição associada que deve ser satisfeita para que a transição, representada pelo seu vértice de destino, seja disparada. Em resumo, os grafos nessa estrutura são definidos por um conjunto de vértices, arestas e condições de caminhamento (V, A, C), onde:

- V = (v0, v1, v2, v3, ..., vn – 1) é um conjunto finito de vértices, em que cada vértice representa uma transição na máquina de estados associada a um evento. Cada vértice é representado por uma tripla: o nome da ação que dispara a transição, o tipo de evento correspondente e o identificador da âncora que identifica o objeto ou parte de seu conteúdo (se o evento é do tipo apresentação ou seleção) ou o identificador único da propriedade e seu valor (se for um evento de atribuição);
- A = (a0, a1, a2, a3, ..., am – 1) é um conjunto finito de arestas que representam, individualmente, um relacionamento entre transições. Para cada aresta "a", v e w são os vértices de origem e destino, respectivamente ((v, w) ∈ V x V);
- C = {cij} é um conjunto finito de condições associadas às arestas. Uma condição "cij" é associada com a aresta (vi, vj) ∈ A e deve ser satisfeita para que o disparo do vértice (disparo da transição) destino da aresta considerada seja realizado.

As condições podem ser simples ou compostas. Uma condição simples é definida por:

- um intervalo temporal que deve ser obedecido para que a transição, representada no vértice destino da aresta, seja executada;
- uma variável que pode ser testada em relação a um valor desejado, inclusive (e na maioria das vezes) variáveis cujo valor são intervalos temporais;
- ações externas à aplicação, como as interações do usuário.

Condições compostas são definidas através de operadores lógicos (OR, AND, NOT), relacionando duas ou mais condições (simples ou compostas).

Para exemplificar o uso dos grafos temporais, considere a mesma aplicação apresentada na Seção 3.6 do Capítulo 3, cuja visão estrutural é apresentada na Figura G.2.

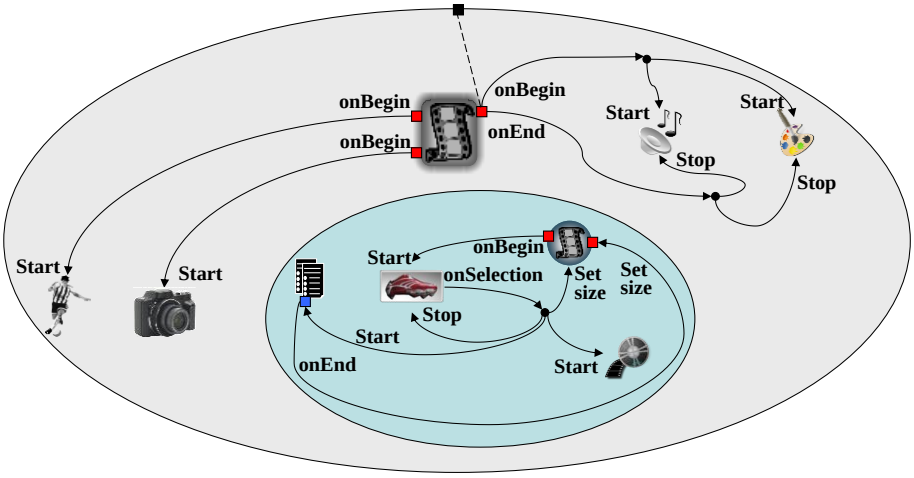


Figura G.2 Visão estrutural da aplicação *O Primeiro João*.

Recordando o Capítulo 3, ao iniciar a animação propriamente dita da aplicação *O Primeiro João*, logo após um pequeno trecho de prefácio do vídeo, é iniciada uma música de fundo e o pano de fundo de toda a exibição. Em um trecho da aplicação é apresentado um vídeo com o drible semelhante àquele em exibição no vídeo da animação. Mais adiante, uma foto de um jogador caído, após receber um drible, é apresentada, também sincronizada com o vídeo da animação. Um pouco mais à frente é exibido um ícone de uma chuteira que, se selecionado, exibe um vídeo de propaganda e uma página HTML para que a compra seja realizada. A ação interativa provoca também o redimensionamento do vídeo principal, que volta ao seu tamanho original após um certo tempo, em que tanto o vídeo da propaganda como o formulário

HTML já terminaram (na verdade, volta ao tamanho original ao final da apresentação do formulário). A aplicação NCL para esse exemplo pode ser revisitada na Listagem 3.22. Recomenda-se a leitura dessa listagem, principalmente para entender alguns dos tempos especificados no grafo.

A Figura G.3 apresenta o grafo resultante da modelagem do exemplo da Figura G.2. Para simplificar a figura, a descrição dos eventos de apresentação foi suprimida nas triplas que definem os vértices.

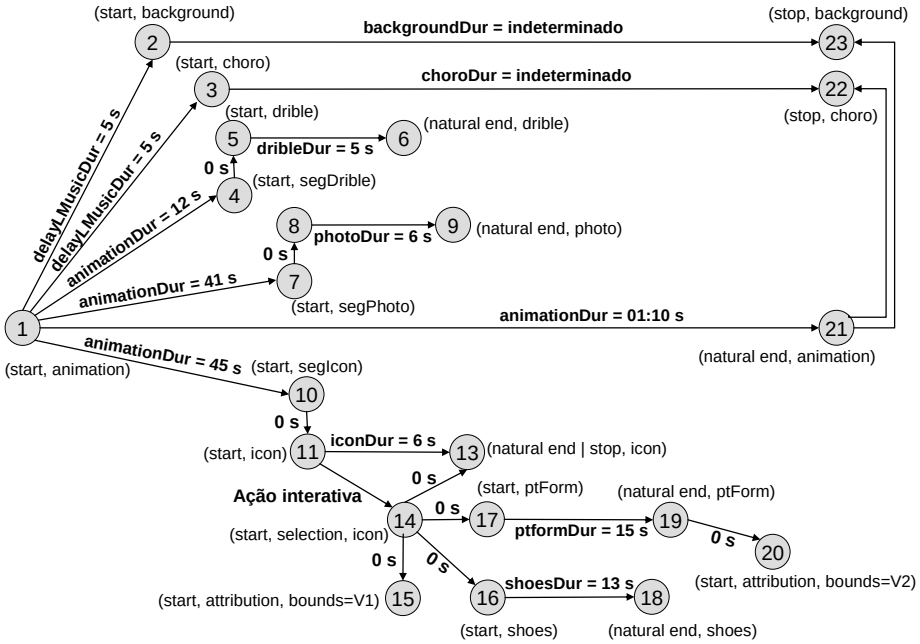


Figura G.3 **Grafo temporal da aplicação *O Primeiro João*.**

Na Figura G.3, o vértice 1 é o ponto de entrada da aplicação e representa a transição de início da apresentação do vídeo da animação. Passados cinco segundos do início da animação (*delay* especificado no elo "lMusic" na Listagem 3.22 do Capítulo 3), é disparada a transição de início dos objetos de mídia "background" (vértice 2) e "choro" (vértice 3).

Decorridos 12 segundos a partir do início do vídeo da animação, uma de suas âncoras (elemento <area id="segDrible">) é iniciada (vértice 4). A transição de início dessa âncora dispara a transição de início do objeto de mídia "drible", representada pelo vértice 5, cuja duração de apresentação é de cinco segundos.

Nesse ponto temos vários pontos a observar. Note que alguns intervalos de tempo (por exemplo, "0 segundos") foram especificados diretamente como

condição de percurso de uma aresta, e outros como valores de variáveis de condições para o percurso, por exemplo “delayLMusicDur=5s” e “animationDur=12s”. A diferença é que certos intervalos de tempo podem ser pausados, por efeito de ações de pausa em um objeto de mídia (caso do intervalo “animationDur”) ou em objetos de composição.

Voltando ao nosso exemplo, decorridos 41 segundos a partir do início do vídeo da animação, outra de suas âncoras (elemento <area id=“segPhoto”>) é iniciada (vértice 7). A transição de início dessa âncora dispara a transição de início do objeto de mídia “photo”, representada pelo vértice 8, cuja duração de apresentação é de seis segundos.

De forma análoga, decorridos 45 segundos a partir do início do vídeo da animação, outra de suas âncoras (elemento <area id=“segIcon”>) é iniciada (vértice 10), o que causa o disparo da transição de início do objeto de mídia “icon”, representada pelo vértice 11, indicando ao usuário a possibilidade de interação com a aplicação. Se a apresentação desse objeto de mídia não é interrompida, ele tem seu fim natural após decorridos seis segundos, conforme representado pelo vértice 13.

O vértice 14 representa a transição de início do evento de seleção. A aresta, que relaciona esse vértice com o vértice 11, tem como condição a ação de interatividade do usuário. O leitor deve notar que um único grafo é utilizado para representar toda a aplicação e que, durante a apresentação, podem ser retiradas partes desse grafo quando os eventos imprevisíveis que ligam essas partes ao restante do grafo não puderem mais ser disparados. No contexto do nosso exemplo, se o usuário não realizar a ação interativa até o fim da apresentação do objeto de mídia “icon”, representado pelo vértice 13, os eventos representados no grafo a partir do vértice 14 nunca irão ocorrer. Note, assim, que o HTG é um grafo dinâmico.

A partir da ocorrência do evento interativo (vértice 14), a disposição espacial do objeto de mídia “animation” é alterada (vértice 15, propriedade *bounds=V1*), e são iniciadas as exibições do vídeo de propaganda “shoes” (vértice 16) e do formulário XHTML (vértice 17). Note também que a ocorrência do evento interativo (vértice 14) leva à interrupção da apresentação do ícone de interação (vértice 13).

Nesse ponto devemos fazer uma nova observação: as ações *natural end* e *stop* são o único caso de ações que podem levar a um mesmo vértice, pois o comportamento decorrente da entrada no estado “sleeping” será indiferente a qualquer um dos dois tipos de ação mencionados que levou a esse estado.

O objeto de mídia “shoes” tem seu fim natural após 13 segundos de seu início (vértice 18). O objeto de mídia “ptForm” tem seu fim natural após 15 segundos de seu início (vértice 19). Ao término desse último objeto, a

disposição espacial inicial do objeto de mídia "animation" é restabelecida (vértice 20).

O leitor deve notar que foram criados dois vértices (V15 e V20) para a mesma propriedade (*bounds*), um para cada valor atribuído. Poderíamos ter criado apenas um vértice, mas isso tornaria mais confuso o gerenciamento do HTG, embora minimizasse o número de vértices e arestas do grafo.

Finalmente, o término natural do vídeo de animação (vértice 21) dispara as transições de término do áudio ("choro") e do pano de fundo ("background"), representadas pelos vértices 22 e 23, respectivamente.

Terminando esta seção, cabe ressaltar que no exemplo da Figura G.1 exploramos apenas um tipo de não-determinismo, a interação do usuário. Outros não-determinismos podem ocorrer; por exemplo, a resolução de uma regra de um elemento <switch> em tempo de execução. Note que, nesse caso, o não-determinismo pode ser representado por condições associadas às arestas, tendo como variáveis as regras.

G.2.1 Cálculo das Durações das Arestas

Durante a execução das aplicações, os exibidores de objetos de mídia devem reportar todos os eventos, incluindo os não-determinísticos, para que outros eventos relacionados possam ser disparados no HTG. O tempo de exibição (*media time*) individual de cada objeto de mídia deverá ser controlado, uma vez que os eventos podem estar associados a trechos específicos de seu conteúdo.

Para os conteúdos de mídia que estejam previamente disponíveis junto aos exibidores, o controle do tempo de exibição pode ser realizado simplesmente verificando o tempo transcorrido a partir do início de sua exibição. Além disso, estando os conteúdos previamente disponíveis, seu tempo total de duração (*total media time*) pode ser calculado. Ao contrário, para conteúdos entregues em tempo de exibição através de fluxos contínuos (*streaming*), determinar o instante inicial do conteúdo ou seu tempo total de exibição exige que o receptor esteja sintonizado desde o início do fluxo ou, então, que mecanismos adicionais estejam disponíveis.

Como discutido no Apêndice E, um receptor pode começar a receber um fluxo elementar, enviado sem solicitação, a partir de qualquer ponto temporal do mesmo, uma vez que o instante de sintonização é qualquer um. Mais ainda, é usual que um fluxo elementar carregue mais de um conteúdo de objeto de mídia. Por exemplo, um fluxo elementar de uma determinada emissora de TV carrega um programa em sequência do outro. Pior ainda, um conteúdo pode ser entremeado por outro; por exemplo, quando uma

propaganda é inserida no meio de um programa de TV. Tudo isso torna impossível saber em que instante de tempo estamos em um determinado conteúdo, sem nenhum auxílio extra.

O Apêndice E discute a solução proposta pelo padrão MPEG System para o problema, por meio do uso de "descritores de referência NPT (*Normal Play Time*). Através dos diferentes tipos de descritores, dos campos *NPT* e *contentId*, é possível determinar o ponto exato de uma mídia.

G.3 Planos de Escalonamento

Tendo por base o HTG, as estruturas de dados para dar suporte aos vários escalonadores discutidos na Seção G.1 podem ser calculadas.

O conjunto de transições de eventos disparados sobre os objetos de uma aplicação e os seus correspondentes momentos no tempo define o "plano de apresentação", a estrutura de suporte do escalonador de apresentação.

Após modelar uma aplicação através de seu HTG, como apresentado no exemplo da Figura G.3, os tempos para o disparo de seus eventos podem ser estimados através do caminhamento nas arestas do grafo. Para aplicações que contêm apenas eventos previsíveis, os instantes para o disparo das transições dos seus eventos podem ser totalmente calculados antes da execução da aplicação. Quando uma aplicação contém eventos imprevisíveis, o cálculo dos instantes para o disparo das transições dos eventos também pode ser realizado *a priori*; no entanto, nesse caso, alguns tempos de disparo serão computados de forma relativa à ocorrência de um evento imprevisível. Considerando o exemplo apresentado na Figura G.3, os tempos dos eventos representados pelos vértices numerados de 15 a 20 são calculados com base no tempo da ocorrência do vértice 14.

A Tabela G.1resume o plano de apresentação obtido a partir do grafo da Figura G.3. Na tabela, as duas primeiras colunas representam os eventos que não dependem da ação interativa, com exceção do evento de seleção. Nas duas últimas colunas encontram-se os eventos disparados a partir do evento de seleção representado pelo vértice 14. Como os tempos dessa coluna são relativos à ocorrência do evento imprevisível, seus tempos estimados estão vinculados ao tempo da ocorrência do evento imprevisível.

Tabela G.1 Eventos do Plano de Apresentação

Início, Apresentação, animation	**70 s**
Início, Apresentação, background	**5 s**
Início, Apresentação, choro	**5 s**
Início, Apresentação, segDrible	**12 s**
Início, Apresentação, drible	**12 s**
Fim, Apresentação, drible	**17 s**
Início, Apresentação, segPhoto	**41 s**
Início, Apresentação, Photo	**41 s**
Início, Apresentação, segIcon	**45 s**
Início, Apresentação, icon	**45 s**
Início, Seleção, icon	**X s**
Fim, Apresentação, photo	**47 s**
Fim, Apresentação, icon	**51 s**
Fim, Apresentação, animation	**70 s**
Fim, Apresentação, choro	**70 s**
Fim, Apresentação, background	**70 s**

Fim, Apresentação, icon	**(X+0)s**
Início, Atribuição, bounds=V1	**(X+0)s**
Inicio, Apresentação, shoes	**(X+0)s**
Inicio, Apresentação, form	**(X+0)s**
Fim, Apresentação, shoes	**(X+13)s**
Fim, Apresentação, form	**(X+15)s**
Início, Atribuição, bounds=V2	**(X+15)s**

Com base no plano construído, o escalonador de apresentação é agora capaz de entregar aos exibidores, no momento preciso, os conteúdos a serem exibidos.

A partir do plano de apresentação, as operações de início/retomada da exibição de uma aplicação NCL podem ser realizadas simplesmente desprezando-se os eventos não-determinísticos que poderiam ter ocorrido antes do tempo em que se pretende iniciar/retomar a apresentação, mas levando em consideração todos os que de fato ocorreram, no caso de retomada da exibição.

Voltemos ao nosso exemplo. Vamos considerar que estamos recebendo o vídeo da animação como o vídeo principal do programa da emissora e que estamos recebendo os outros objetos por *datacasting*. Conforme vimos no Capítulo 16, comandos de edição *addDocument* e *startDocument* são também enviados ciclicamente aos receptores. Assim, ao sintonizar um canal, o receptor passa a receber os fluxos elementares do fluxo de transporte, incluindo os objetos de mídia transmitidos por *datacasting*, os descritores NPT (ver Apêndice E) e os comandos de edição (ver Capítulo 16).

O comando *addDocument* será interpretado pelos receptores fazendo com que a especificação da aplicação seja adicionada às suas bases privadas. O comando *startDocument* gera uma notificação ao formatador NCL, que

passa a construir o grafo temporal (HTG). Com o grafo construído, o plano de apresentação é calculado.

Vamos agora supor que a sintonização se deu durante a exibição da animação, mas 43 segundos após o seu começo. Note que o *middleware* pode calcular esse valor pela monitoração de descritores NPT. O escalonador de apresentação é então informado desse tempo. De posse do valor, o escalonador posiciona a aplicação nesse instante temporal do plano de apresentação. No caso específico do exemplo, o usuário ainda poderia realizar a ação interativa e ser capaz de ter a foto (elemento de mídia "photo") exibida por quatro segundos (e não seis segundos, como na especificação para a aplicação começando do tempo zero).

O leitor deve notar que o mesmo procedimento pode ser utilizado para operações de pausa e retomada, com mudança intermediária de canal. Ao se mudar o canal, a aplicação que estava executando entra em modo de espera (*stand by*). Ao retornar ao canal, quando a base temporal for retomada[1] (quando voltar a receber o descritor NPT com o mesmo *contentId*, como discutimos no Apêndice E e no Capítulo 9), a aplicação continua sua execução, a partir do ponto informado pelo valor do descritor NPT, desconsiderando qualquer evento não-determinístico que poderia ter ocorrido desde o momento de sua pausa até o momento de sua retomada.

Com base no plano de apresentação e levando em conta o tempo de instanciação de cada exibidor, o plano de instanciação de exibidores pode ser construído. Esse plano é a base para a operação do escalonador de exibidores, conforme vimos na Seção G.1.

Também a partir do plano de apresentação, mas agora levando em conta os retardos introduzidos pelo transporte dos vários objetos de mídia até a memória do receptor (retardos na rede, retardos de acesso ao carrossel etc.), o plano de pré-busca pode ser construído. Fundamentado nesse plano, base de operação do escalonador de pré-busca e de retardos na reserva de recursos em redes com QoS, planos de negociação de QoS podem ser calculados.

Até agora focamos a construção do HTG do lado do cliente, isto é, do lado do receptor. Entretanto, para o controle da transmissão não-solicitada de objetos de mídia, isto é, para dar suporte ao escalonador de carrossel, apresentado na Seção G.1, é necessário que o HTG seja também construído pelo provedor de *datacasting*. Tendo por base o HTG e alguma heurística de otimização de banda passante e retardo de transmissão, o escalonador de carrossel saberá que objetos devem ser colocados em um carrossel para transmissão cíclica, quantas vezes deve ser colocado e em que posições.

---

[1] Note que a base temporal pode ser retomada, mas não imediatamente após o retorno ao canal, pois no momento pode estar sendo exibido outro conteúdo, por exemplo, uma propaganda.

Costa, R.R.; Moreno, M.F.; Soares, L.F.G. Intermedia Synchronization Management in DTV Systems. *Proceedings of the ACM Symposium on Document Engineering*. São Paulo, setembro de 2008, pp. 289-297. ISBN: 978-1-60558-081-4.

Moreno, M.F.; Costa, R.R.; Soares, L.F.G.. Sincronismo entre Fluxos de Mídia Contínua e Aplicações Multimídia em Redes por Difusão. *Anais do XIV Simpósio Brasileiro de Sistemas Multimídia e Hipermídia*, Vila Velha, outubro de 2008, pp. 202-209. ISBN: 857669198-1.

Apêndice H

Comportamento de Exibidores

Para que não haja surpresa na execução de uma aplicação declarativa NCL, é necessário que o autor da aplicação saiba com precisão o comportamento do formatador NCL e dos diversos exibidores de objetos de mídia envolvidos na apresentação de um documento.

Este apêndice apresenta a interpretação dada pelos vários exibidores de mídia, incluindo os objetos de mídia com código imperativo e declarativo, a comandos enviados pelo formatador NCL, a partir de ações determinadas nos elementos <link> e também em alguns comandos de edição NCL. O apêndice também apresenta a interpretação dada pelo formatador quando as ações são submetidas em nós de composição NCL (representados pelos elementos <body>, <context> e <switch>).[1]

[1] Este capítulo foi baseado em Soares *et al*. (2006). O uso do material foi gentilmente cedido pelo Departamento de Informática da PUC-Rio.

H.1 Introdução

A apresentação de um documento NCL requer o controle da sincronização de vários objetos de mídia (especificados através do elemento <media>). Para cada objeto de mídia, um *player* (exibidor de mídia) é carregado para o controle do objeto e de seus eventos NCL. Um exibidor de mídia (ou seu adaptador) deve ser capaz de receber os comandos de apresentação, controlar (notificar) as máquinas de estado dos eventos do objeto de mídia controlado e responder às demandas do formatador.

O comportamento dos exibidores de mídia e do formatador NCL são apresentados nas Seções H2 e H3, respectivamente.

H.2 Comportamento dos Exibidores de Objetos de Mídia Não-Declarativos

Nesta seção discutiremos o comportamento para exibidores de objetos de mídia, incluindo os objetos de mídia com código imperativo, mas excluindo os objetos de mídia cujo conteúdo é um código declarativo (objetos de mídia declarativos). Esses últimos são apresentados por exibidores de documentos, que têm um comportamento semelhante ao formatador NCL, por isso são discutidos em uma seção à parte (Seção H.4). Afinal, o próprio formatador NCL é um exibidor de mídia declarativo para a linguagem NCL.

Um objeto de mídia em apresentação é identificado pelo atributo *id* do elemento <media> correspondente, e os atributos *id* dos elementos <descriptor> que foram associados ao objeto de mídia. Por simplicidade, chamaremos essa identificação de *representationObjectId*.

H.2.1 Comportamento na Execução de Ações sobre Eventos de Apresentação

H.2.1.1 Instrução *start*

Antes de enviar a instrução *start*, o formatador NCL encontra o exibidor de mídia mais apropriado, com base no tipo de conteúdo a ser exibido. Para tanto, o formatador leva em consideração o atributo *player* associado com o objeto de mídia a ser exibido. Se esse atributo não for especificado, o formatador leva em conta o atributo *type* do elemento <media>. Se esse atributo também não for especificado, o formatador considera a extensão do arquivo especificado no atributo *src* do elemento <media>.

A instrução *start* emitida por um formatador NCL informa ao exibidor de mídia os seguintes parâmetros: o localizador do conteúdo do objeto de mídia a ser controlado, uma lista de todas as propriedades associadas ao objeto de mídia, a identificação do objeto de mídia em apresentação (*representationObjectId*), uma lista de eventos (apresentação, seleção ou atribuição, definidos pelos elementos <area>, <property> e pelas âncoras de conteúdo *default* do elemento <media>) que precisam ser monitorados pelo exibidor de mídia, o evento de apresentação a ser iniciado (chamado *evento principal*), um tempo de compensação (*offset-time*), opcional, e um tempo de retardo, opcional. No caso de objetos de mídia imperativo, o evento de apresentação a ser iniciado é identificado pelo atributo *label* de suas âncoras de conteúdo ou é, por omissão (*default*), o evento associado à âncora de conteúdo principal.

O atributo *src* do elemento <media> é usado, pelo exibidor de mídia, para localizar o conteúdo e iniciar sua apresentação. Se o conteúdo não puder ser localizado ou se o exibidor de mídia não souber como lidar com o tipo de conteúdo, o exibidor de mídia encerra a operação de iniciação sem realizar nenhuma ação.

Os descritores a serem utilizados são escolhidos pelo formatador seguindo as diretrizes especificadas no documento NCL. Se a instrução *start* resultar de uma ação de um elo que tenha um descritor explicitamente declarado em seu elemento <bind> (atributo *descriptor*), o descritor resultante informado pelo formatador mescla os atributos do descritor especificado pelo <bind> com os atributos do descritor especificado no elemento <media> correspondente, se esse atributo tiver sido especificado. Para atributos em comum, a informação do descritor do <bind> sobrepõe a do descritor do <media>. Se o elemento <bind> não contiver um descritor explícito, o descritor informado pelo formatador é o descritor especificado pelo elemento <media>, se o atributo tiver sido especificado. Caso contrário, um descritor *default* para o tipo de <media> específico pode ser escolhido pelo formatador. Baseado nesse procedimento, a lista dos *ids* dos descritores associados ao objeto de mídia é computada e, unificando as propriedades definidas no descritor resultante com as propriedades explicitamente declaradas no elemento <media> correspondente, a lista de propriedades associada ao objeto de mídia é avaliada.

A lista de eventos a serem monitorados por um exibidor de mídia é computada pelo formatador, levando em conta a especificação do documento NCL. Para tanto, ele checa todos os elos dos quais participa o objeto de mídia com o descritor resultante. Ao computar os eventos a serem monitorados, o formatador considera a perspectiva do objeto de mídia, isto é, o caminho a partir do elemento <body> por vários elementos <context> para alcançar em profundidade o elemento <media> correspondente. Apenas os elos contidos

nesses elementos <body> e <context> são considerados na computação dos eventos monitorados.

O parâmetro *offset-time* é opcional e tem "zero" como seu valor *default*. O parâmetro é significativo somente para mídia contínua, ou estática com duração explícita. Nesse caso, o parâmetro define um tempo de compensação, desde o início (*beginning-time*) do evento principal, a partir do qual a apresentação desse evento é imediatamente iniciada (isto é, ele comanda o exibidor para pular para o tempo de partida = *beginning-time* + *offset-time*). Obviamente, o valor do *offset-time* deve ser menor que a duração do evento principal. Caso contrário, a instrução *start* é ignorada.

Se o *offset-time* for maior que zero, o exibidor de mídia coloca o evento principal no estado ocorrendo (*occurring*), mas a transição de início (*starts*) do evento não é notificada. Se o *offset-time* for zero, o exibidor de mídia coloca o evento principal no estado *occurring* e notifica a ocorrência da transição de início. No caso de objetos de mídia que não são objetos imperativos, os eventos que teriam seus tempos de término anteriores ao tempo de início do evento principal e os eventos que teriam seus tempos de início após o tempo de término do evento principal não precisam ser monitorados pelo exibidor de mídia (o formatador faz essa verificação quando constrói a lista de eventos monitorados).

Os eventos monitorados que teriam seus tempos de término após o tempo de início do evento principal, mas antes do tempo de partida (*beginning-time* + *offset-time*), têm seu atributo *occurrences* incrementado, mas as transições de início e término (stops) não são notificadas. Os eventos monitorados que têm seu tempo de início antes do tempo de partida (*beginning time* + *offset-time*) e tempo de término após o tempo de partida são colocados no estado *occurring*, mas as transições de início correspondentes não são notificadas.

O tempo de retardo também é um parâmetro opcional, e seu valor *default* também é "zero". Se maior que zero, esse parâmetro contém um tempo a ser esperado pelo exibidor de mídia antes de iniciar sua apresentação.

Com exceção dos objetos de mídia imperativos, se o exibidor receber uma instrução de *start* para um objeto que já está sendo apresentado (pausado ou não), ele ignora a instrução e mantém o controle da apresentação em andamento. Nesse caso, o elemento <simpleAction> que causou a instrução *start* não causa qualquer transição na máquina de estados do evento a ele associado.

Diferentemente dos procedimentos realizados para os outros tipos de elementos <media>, se um exibidor de objeto de mídia imperativo receber uma instrução *start* para um evento associado a um elemento <area> e esse evento estiver no estado *sleeping*, ele inicia a execução do código imperativo

associado ao elemento, mesmo se outra parte do código imperativo do objeto de mídia estiver em execução (pausado ou não). Contudo, se o evento associado ao elemento-alvo <area> estiver no estado *occurring* ou *paused*, a instrução *start* é ignorada pelo exibidor imperativo, que continuará controlando a execução anteriormente iniciada.

H.2.1.2 Instrução *stop*

No caso de objetos de mídia com código imperativo, a instrução *stop* precisa identificar um trecho de código que já está sendo controlado. Identificar o trecho de código significa identificar o objeto de mídia sendo controlado (*representationObjectId*) e a interface que identifica o trecho de código. Se a interface não for especificada, a âncora de conteúdo total é assumida. Nesse caso, a ação *stop* é aplicada em todas as âncoras de conteúdo. Para os outros objetos de mídia comuns, a instrução *stop* não precisa especificar a interface; se um elemento <simpleAction> com o *actionType* igual a "stop" é ligado por um elo a uma interface desse nó, a interface é ignorada quando a ação for executada.

A instrução *stop* é ignorada pelo exibidor de objeto de mídia imperativo se o trecho de código associado com a interface especificada na instrução não estiver sendo executado (se o evento correspondente não estiver nos estados *occurring* ou *paused*) e se o exibidor do objeto imperativo não estiver esperando devido a uma instrução retardada de *start*. Se o código correspondente da interface especificada na instrução *start* estiver em execução, a execução é parada, o evento de apresentação correspondente transita para o estado *sleeping*, sua transição *stops* é notificada ao formatador e seu atributo *occurrences* não é incrementado. Se o atributo *repetitions* do evento for maior que zero, ele é diminuído em um, e a apresentação do evento associado à interface é reiniciada, após o tempo entre repetições (o tempo de retardo entre repetições é transmitido ao exibidor de mídia como parâmetro de retardo de início). Se um trecho de código do objeto de mídia imperativo estiver esperando para ser executado após uma instrução *start* atrasada, e uma instrução *stop* for emitida, a instrução de *start* anterior é removida. Todo esse procedimento, exceto para o evento associado à "âncora de conteúdo total", deve ser realizado por instruções programadas pelo autor (programador) do objeto imperativo para cada trecho de código que pode ser parado.

Ainda, apenas para os objetos de mídia com código imperativo, se qualquer âncora de conteúdo for parada e todos os outros eventos de apresentação estiverem no estado *sleeping*, a âncora de conteúdo total será colocada no estado *sleeping*. Se uma âncora de conteúdo for parada e pelo menos um outro evento de apresentação do objeto estiver no estado

occurring, a âncora de conteúdo total será mantida no estado *occurring*. Em todos os demais casos, se uma âncora de conteúdo for parada, a âncora de conteúdo total será colocada no estado *paused*. Novamente, todo esse procedimento deve ser realizado por instruções programadas pelo autor (programador) do objeto imperativo para cada trecho de código que pode ser parado.

Para objetos de mídia que não têm código imperativo, se o objeto não estiver sendo apresentado (isto é, se nenhum dos eventos na lista de eventos do objeto estiver no estado *occurring* ou *paused*) e o exibidor de mídia não estiver aguardando devido a uma instrução atrasada de *start*, a instrução *stop* é ignorada. Se o objeto estiver sendo apresentado, o evento principal (evento passado como parâmetro quando o objeto de mídia foi iniciado) e todos os eventos monitorados no estado *occurring* ou *paused* com tempo de término igual ou anterior ao tempo de término do evento principal transitam para o estado *sleeping* (mesmo que um efeito de transição esteja sendo aplicado ao objeto de mídia, isto é, o efeito de transição também deve ser parado), e suas transições *stops* são notificadas.

Ainda para objetos de mídia que não têm código imperativo declarativo, quando da aplicação de uma instrução *stop*, os eventos monitorados no estado *occurring* ou *paused* com tempo de término posterior ao tempo de término do evento principal são colocados no estado *sleeping*, mas suas transições *stops* não são notificadas e seus atributos *occurrences* não são incrementados. A apresentação do conteúdo do objeto é parada. Se o atributo *repetitions* do evento for maior que zero, ele é diminuído em um e a apresentação do evento principal é reiniciada após o tempo entre repetições (novamente, o tempo de retardo entre repetições é transmitido ao exibidor de mídia como parâmetro de retardo de início). Se o objeto de mídia estiver esperando para ser apresentado após uma instrução *start* atrasada e se uma instrução *stop* for emitida, a instrução de *start* anterior é removida.

NOTA: Na norma brasileira para o Ginga-NCL [ABNT, NBR 15606-2, 2007], quando todos os objetos de mídia que se referem ao fluxo elementar que transporta o vídeo principal de um programa estiverem no estado *sleeping*, a exibição do vídeo principal deve ocupar a tela inteira. Só por meio de um objeto de mídia em execução referindo ao vídeo principal, esse vídeo pode ser redimensionado. O mesmo acontece com o áudio principal de um programa: quando todos os objetos de mídia que se referem ao fluxo elementar que transporta o áudio principal de um programa estiverem no estado *sleeping*, a exibição do áudio principal deve ocorrer com 100% de seu volume original.

H.2.1.3 Instrução *abort*

No caso de objetos de mídia com código imperativo, a instrução *abort* precisa identificar um trecho de código que já está sendo controlado. Como sempre, identificar o trecho de código significa identificar o objeto de mídia sendo controlado (*representationObjectId*) e a interface que identifica o trecho de código. Mais uma vez, se a interface não for especificada, a âncora de conteúdo total é assumida. Nesse caso, a instrução *abort* é aplicada em todas as âncoras de conteúdo. Para os outros objetos de mídia comuns, a instrução *abort* precisa apenas identificar um objeto de mídia que já está sendo controlado; se um elemento <simpleAction> com o *actionType* igual a "abort" é ligado por um elo a uma interface de nó, a interface é ignorada quando a instrução for executada.

A instrução *abort* é ignorada pelo exibidor de objeto de mídia imperativo se o trecho de código associado com a interface especificada na instrução não estiver sendo executado (se o evento correspondente não estiver nos estados *occurring* ou *paused*) e se o exibidor do objeto imperativo não estiver esperando devido a uma instrução retardada de *start*. Se o código correspondente da interface especificada na instrução *start* estiver em execução, a execução é abortada, o evento de apresentação correspondente transita para o estado *sleeping*, sua transição *aborts* é notificada ao formatador e seu atributo *occurrences* não é incrementado. O atributo *repetitions* do evento é colocado em zero, e em nenhuma hipótese a apresentação do evento associado à interface é reiniciada. Se um trecho de código do objeto de mídia imperativo estiver esperando para ser executado após uma instrução *start* atrasada e uma instrução *abort* for emitida, a instrução de *start* anterior será removida. Todo esse procedimento, exceto para o evento associado à "âncora de conteúdo total", deve ser realizado por instruções programadas pelo autor (programador) do objeto imperativo para cada trecho de código que pode ser parado.

Ainda, apenas para os objetos de mídia com código imperativo, se a execução de qualquer âncora de conteúdo for abortada e todos os outros eventos de apresentação estiverem no estado *sleeping*, a âncora de conteúdo total é colocada no estado *sleeping*. Se uma âncora de conteúdo for abortada e pelo menos um outro evento de apresentação do objeto estiver no estado *occurring*, a âncora de conteúdo total será mantida no estado *occurring*. Em todos os demais casos, se uma âncora de conteúdo for abortada, a âncora de conteúdo total será colocada no estado *paused*. Novamente, todo esse procedimento deve ser realizado por instruções programadas pelo autor (programador) do objeto imperativo para cada trecho de código que pode ser parado.

Para objetos de mídia que não têm código imperativo, se o objeto não estiver sendo apresentado e não estiver esperando para ser apresentado após uma instrução de *start* atrasada, a instrução *abort* é ignorada. Se o objeto estiver sendo apresentado, o evento principal e todos os eventos monitorados no estado *occurring* ou *paused* transitam para o estado *sleeping* e suas transições *aborts* são notificadas. Qualquer apresentação de conteúdo é abortada.

Se o atributo *repetitions* do evento for maior que zero, ele é colocado em zero e a apresentação do objeto de mídia não é reiniciada. Se o objeto de mídia estiver esperando para ser apresentado após uma instrução *start* atrasada e uma instrução *abort* for emitida, a instrução *start* é removida.

H.2.1.4 Instrução *pause*

A instrução *pause* atua de forma semelhante à instrução anterior.

No caso de objetos de mídia com código imperativo, a instrução *pause* precisa identificar um trecho de código que já está sendo controlado. Como sempre, identificar o trecho de código significa identificar o objeto de mídia sendo controlado (*representationObjectId*) e a interface que identifica o trecho de código. Mais uma vez, se a interface não for especificada, a âncora de conteúdo total é assumida. Nesse caso, a instrução *pause* é aplicada em todas as âncoras de conteúdo. Para os outros objetos de mídia comuns, a instrução *pause* precisa apenas identificar um objeto de mídia que já está sendo controlado; se um elemento <simpleAction> com o *actionType* igual a "pause" é ligado por um elo a uma interface de nó, a interface é ignorada quando a instrução for executada.

Também semelhante aos casos anteriores, a instrução *pause* é ignorada pelo exibidor de objeto de mídia imperativo se o trecho de código associado com a interface especificada na instrução não estiver sendo executado (se o evento correspondente não estiver nos estados *occurring* ou *paused*) e se o exibidor do objeto imperativo não estiver esperando devido a uma instrução retardada de *start*. Se o código correspondente da interface especificada na instrução *start* estiver em execução, a execução é pausada, o evento de apresentação correspondente transita para o estado *paused* e sua transição *pauses* é notificada ao formatador. O tempo em que o código se encontra no estado *paused* não é considerado no cálculo da duração de sua execução. Se um trecho de código do objeto de mídia imperativo estiver esperando para ser executado após uma instrução *start* atrasada e uma instrução *pause* for emitida, a execução do código espera por uma instrução de *resume* para continuar esperando pelo retardo especificado na instrução *start*. Todo esse procedimento, exceto para o evento associado à "âncora de conteúdo total",

deve ser realizado por instruções programadas pelo autor (programador) do objeto imperativo para cada trecho de código que pode ser parado.

Ainda, apenas para os objetos de mídia com código imperativo, se qualquer âncora de conteúdo for pausada e todos os outros eventos de apresentação estiverem no estado *sleeping* ou *paused*, a âncora de conteúdo total é colocada no estado *paused*. Se uma âncora de conteúdo for pausada e pelo menos um outro evento de apresentação do objeto estiver no estado *occurring*, a âncora de conteúdo total é mantida no estado *occurring*. Novamente, todo esse procedimento deve ser realizado por instruções programadas pelo autor (programador) do objeto imperativo para cada trecho de código que pode ser parado.

Para objetos de mídia que não têm código imperativo, se o objeto não estiver sendo apresentado (se o evento principal, passado como parâmetro quando o objeto de mídia foi iniciado, não estiver no estado *occurring*) e o exibidor de mídia não estiver esperando pelo retardo de início, a instrução *pause* é ignorada. Se o objeto estiver sendo apresentado, o evento principal e todos os eventos monitorados no estado *occurring* transitam para o estado *paused* e suas transições *pauses* são notificadas. A apresentação do objeto é pausada e o tempo de pausa decorrido não é considerado como parte da duração do objeto. Como exemplo, se um objeto tiver duração explícita de 30 segundos e, após 25 segundos, for pausado, mesmo se o objeto permanecer pausado por, digamos, cinco minutos após o reinício o evento principal do objeto permanecerá ocorrendo por mais cinco segundos. Se o evento principal ainda não estiver ocorrendo porque o exibidor de mídia está esperando pelo retardo de início, o objeto de mídia espera por uma instrução *resume* para continuar aguardando o retardo de início.

H.2.1.5 Instrução *resume*

A instrução *resume* atua de forma semelhante às instruções anteriores.

No caso de objetos de mídia com código imperativo, a instrução *resume* precisa identificar um trecho de código que já está sendo controlado. Como sempre, identificar o trecho de código significa identificar o objeto de mídia sendo controlado (*representationObjectId*) e a interface que identifica o trecho de código. Mais uma vez, se a interface não for especificada, a âncora de conteúdo total é assumida. Se a âncora de conteúdo total não estiver no estado *paused*, a instrução *resume* é ignorada. Em caso contrário, a instrução *resume* é aplicada em todas as âncoras de conteúdo que estão no estado *paused*, exceto aquelas que já estavam pausadas quando a âncora de conteúdo total recebeu a instrução *pause*. Para os outros objetos de mídia comuns, a instrução *resume* precisa apenas identificar um objeto de mídia que já está sendo controlado; se um elemento <simpleAction> com o *actionType* igual a

"resume" é ligado por um elo a uma interface de nó, a interface é ignorada quando a instrução for executada.

A instrução *resume* é ignorada pelo exibidor de objeto de mídia imperativo se o trecho de código associado com a interface especificada na instrução não estiver pausado (se o evento correspondente não estiver no estado *paused*) e se o exibidor do objeto imperativo não estiver esperando devido a uma instrução retardada de *start*. Se o trecho de código do objeto de mídia imperativo estiver pausado na espera para ser executado após uma instrução *start* atrasada e ter sido submetido a uma instrução *pause*, ele retoma a espera especificada na instrução *start*. Se o código correspondente da interface especificada na instrução *start* estiver pausado, o evento de apresentação correspondente transita para o estado *occurring*, e a transição *resumes* é notificada ao formatador. Todo esse procedimento, exceto para o evento associado à "âncora de conteúdo total", deve ser realizado por instruções programadas pelo autor (programador) do objeto imperativo para cada trecho de código que pode ser parado.

Ainda apenas para os objetos de mídia com código imperativo, se qualquer âncora de conteúdo tiver sua execução retomada, a âncora de conteúdo total é colocada no estado *occurring*. Se uma âncora de conteúdo for pausada e pelo menos um outro evento de apresentação do objeto estiver no estado *occurring*, a âncora de conteúdo total é mantida no estado *occurring*. Novamente, todo esse procedimento deve ser realizado por instruções programadas pelo autor (programador) do objeto imperativo para cada trecho de código que pode ser parado.

Para objetos de mídia que não têm código imperativo, se o objeto não estiver pausado (se o evento principal, passado como parâmetro quando o objeto de mídia foi iniciado, não estiver no estado *paused*) e o exibidor de mídia não estiver pausado (esperando pelo retardo de início), a instrução é ignorada. Se o exibidor de mídia estiver pausado aguardando o retardo de início, ele retoma a espera da exibição a partir do instante em que foi pausado. Se o evento principal estiver no estado *paused*, o evento principal e todos os eventos monitorados no estado *paused* são colocados no estado *occurring* e suas transições *resumes* são notificadas.

H.2.1.6 Término Natural de uma Apresentação

Eventos de um objeto, com duração explícita ou intrínseca, normalmente terminam suas apresentações naturalmente, sem precisar de instruções externas. Nesse caso, o exibidor de mídia transita o evento para o estado *sleeping* e notifica a transição *stops*.

Para objetos de mídia com código imperativo, se o atributo *repetitions* do evento for maior que zero, ele é diminuído em um, e a apresentação do evento associado à interface é reiniciada após o tempo entre repetições (o tempo de retardo entre repetições é transmitido ao exibidor de mídia como parâmetro de retardo de início). Todo esse procedimento, exceto para o evento associado à "âncora de conteúdo total", deve ser realizado por instruções programadas pelo autor (programador) do objeto imperativo para cada trecho de código que pode ser parado.

Ainda, apenas para os objetos de mídia com código imperativo, se qualquer âncora de conteúdo tiver um fim natural, e todos os outros eventos de apresentação estiverem no estado *sleeping*, a âncora de conteúdo total é colocada no estado *sleeping*. Se uma âncora de conteúdo terminar e pelo menos um outro evento de apresentação do objeto estiver no estado *occurring*, a âncora de conteúdo total é mantida no estado *occurring*. Em todos os demais casos, se uma âncora de conteúdo terminar sua execução, a âncora de conteúdo total é colocada no estado *paused*. Novamente, todo esse procedimento deve ser realizado por instruções programadas pelo autor (programador) do objeto imperativo para cada trecho de código que pode ser parado.

Para objetos de mídia que não têm código imperativo, os eventos monitorados no estado *occurring* com o mesmo tempo de término do evento principal ou com tempo de término desconhecido, quando o evento principal termina, vão para o estado *sleeping* e suas *transições stops* são notificadas. Os eventos no estado *occurring* com tempo de término posterior ao tempo de término do evento principal são colocados no estado *sleeping* sem gerar a transição *stops* e sem incrementar o atributo *occurences*. É importante ressaltar que, se o evento principal corresponder a uma âncora temporal interna ao objeto, quando a apresentação dessa âncora terminar, toda a apresentação do objeto de mídia termina.

H.2.2 Comportamento na Execução de Ações sobre Eventos de Atribuição

Antes de iniciarmos a discussão de cada instrução, é importante salientar que, embora as propriedades de um nó de mídia com código imperativo possa estar associado com trechos de código, a execução desses trechos não afeta máquinas de estado de âncoras de conteúdo que porventura estejam associadas aos mesmos códigos.

H.2.2.1 Instrução *start*

A instrução *start* pode ser aplicada a um objeto independentemente do fato de ele estar sendo ou não apresentado (nesse último caso, embora o objeto não esteja sendo apresentado, seu exibidor de mídia já deve estar instanciado). No primeiro caso, a instrução *start* precisa identificar o objeto de mídia sendo controlado (*representatioObjectId*) e um evento de atribuição monitorado. Deve também especificar um valor a ser atribuído à propriedade que definiu o evento, a duração do processo de atribuição e o passo de atribuição..

Ao imputar um valor à propriedade, o exibidor de mídia transita a máquina de estado do evento de atribuição para o estado *occurring* e, depois de terminada a atribuição, novamente para o estado *sleeping*, gerando a transição *starts* e, em seguida, a transição *stops*. No caso de objetos de mídia com código imperativo, todo esse procedimento deve ser realizado por instruções programadas pelo autor (programador) do objeto imperativo para cada elemento <property> declarado.

Para cada evento de atribuição monitorado, se o exibidor de mídia alterar, por sua própria conta, o valor correspondente de um atributo, ele procede como se tivesse recebido uma instrução externa de *start*. Novamente, no caso de objetos de mídia com código imperativo, todo o procedimento deve ser realizado por instruções programadas pelo autor (programador) do objeto imperativo.

H.2.2.2 Instruções stop, abort, pause e resume

As instruções *stop, abort, pause e resume* precisam identificar o objeto de media em apresentação (*representation ObjectId*) e um eveno de atribuição sendo monitorado.

A instrução *stop* apenas cessa o procedimento de atribuição, trazendo o evento de atribuição para o estado *sleeping*.

A instrução *abort* aborta o procedimento de atribuição, trazendo o evento de atribuição para o estado *sleeping* e a propriedade para o seu valor inicial.

A instrução *pause* apenas pausa o procedimento de atribuição, trazendo o evento de atribuição para o estado *paused*.

Finalmente, a instrução *resume* retoma o procedimento de atribuição, trazendo o evento de atribuição para o estado *occurring*.

No caso de objetos de mídia com código imperativo, todos os procedimentos citados nos parágrafos anteriores devem ser realizados por instruções programadas pelo autor (programador) do objeto imperativo.

H.2.3 Comportamento na Execução de Comandos de Edição

H.2.3.1 Instrução *addEvent*

A instrução *addEvent* é emitida no caso de recepção de um comando de edição NCL *addInterface* (ver Capítulo 16). A instrução precisa apenas identificar um objeto de mídia que já esteja sendo controlado e um novo evento de interface a ser incluído e colocado em monitoramento.

No caso de objetos de mídia comuns, que não possuem código imperativo, todas as regras aplicadas à interseção de eventos monitorados com o evento principal são aplicadas ao novo evento. Se o tempo de início do novo evento for anterior ao tempo atual do objeto e o tempo de término do novo evento for posterior ao tempo atual do objeto, o novo evento é colocado no mesmo estado do evento principal (*occurring* ou *paused*), sem notificar a transição correspondente.

H.2.3.2 Instrução *removeEvent*

A instrução *removeEvent* é emitida no caso de recepção de um comando de edição NCL *removeInterface*. A instrução precisa identificar um objeto de mídia que já esteja sendo controlado e um evento de interface que não se quer mais controlar. O estado do evento da interface a ser removida é colocado em *sleeping*, sem gerar nenhuma transição.

H.3 Comportamento do Formatador NCL na Exibição de Composições

Um <simpleCondition> ou <simpleAction> com valor do atributo *eventType* igual a "presentation" pode ser associado por um elo a um nó de composição (representado por um elemento <context> ou <body>) como um todo (isto é, sem que uma de suas interfaces seja informada). Como normalmente ocorre, a máquina de estado do evento de apresentação definido pelo nó de composição deve ser controlada pelo formatador, como discutimos no Capítulo 10. De forma análoga, um <attributeAssessment>, com valor de

atributo *eventType* igual a “presentation” e *attributeType* igual a “state”, “occurrences” ou “repetitions” pode ser associado por um elo a um nó de composição (representado por um elemento <context> ou <body>) como um todo.

A particularidade do procedimento se aplica quando uma <simpleAction> com valor de atributo *eventType* igual a “presentation” for associada por um elo a um nó de composição (representado por um elemento <context> ou <body>) como um todo (ou seja, sem que uma de suas interfaces seja informada). Nesse caso, a instrução é refletida nas máquinas de estado de evento dos nós filhos da composição, como veremos a seguir.

H.3.1 Iniciando a Apresentação de um Contexto

Se um elemento <context> ou <body> participar de um papel (*role*) de ação (*action*) cujo tipo de ação é “start” quando essa ação for acionada, a instrução *start* também é aplicada a todos os eventos de apresentação mapeados pelas portas do elemento <context> ou <body>, quando nenhuma porta (elemento <port>) da composição for especificada na ação.

Se o autor quiser iniciar a apresentação a partir de uma porta específica, ele também deve indicar o *id* de <port> como valor do atributo *interface* do elemento <bind>.

H.3.2 Parando a Apresentação de um Contexto

Se um elemento <context> ou <body> participar de um papel (*role*) de ação (*action*) cujo tipo de ação é “stop” quando essa ação for acionada, a instrução *stop* também é aplicada a todos os eventos de apresentação dos nós filhos da composição, quando nenhuma porta (elemento <port>) da composição for especificada na ação.

Se a composição contiver elos sendo avaliados (ou com sua avaliação pausada), as avaliações são suspensas e nenhuma ação é acionada.

H.3.3 Abortando a Apresentação de um Contexto

Se um elemento <context> ou <body> participar de um papel (*role*) de ação (*action*) cujo tipo de ação é “abort” quando essa ação for acionada, a instrução *abort* também é aplicada a todos os eventos de apresentação dos nós filhos da composição quando nenhuma porta (elemento <port>) da composição for especificada na ação.

Se a composição contiver elos sendo avaliados (ou com sua avaliação pausada), as avaliações são suspensas e nenhuma ação é acionada.

H.3.4 Pausando a Apresentação de um Contexto

Se um elemento <context> ou <body> participar de um papel (*role*) de ação (*action*) cujo tipo de ação é "pause" quando essa ação for acionada, a instrução *pause* também é aplicada a todos os eventos de apresentação dos nós filhos da composição que estejam no estado *occurring* quando nenhuma porta (elemento <port>) da composição for especificada na ação.

Se a composição contiver elos sendo avaliados, todas as avaliações são suspensas até que uma ação *resume*, *stop* ou *abort* seja emitida.

Se a composição contiver nós filhos com eventos de apresentação no estado *paused* quando a ação "pause" na composição for emitida, esses nós são identificados porque, se a composição receber uma instrução *resume*, esses eventos não devem ser retomados.

H.3.5 Retomando a Apresentação de um Contexto

Se um elemento <context> ou <body> participar de um papel (*role*) de ação (*action*) cujo tipo de ação é "resume" e nenhuma porta (elemento <port>) da composição for especificada na ação quando essa ação for acionada, a instrução *resume* também é aplicada a todos os eventos de apresentação dos nós filhos da composição que estejam no estado *paused*, exceto aqueles que já estavam pausados antes de a composição ser pausada.

Se a composição contiver elos com avaliações pausadas, elas são retomadas.

H.3.6 Relação entre as Máquinas de Estado de Eventos de Apresentação de um Nó e a Máquina de Estado do Evento de Apresentação de seu Nó de Composição Pai

Sempre que o evento de apresentação de um nó (mídia ou composição) for para o estado *occurring*, o evento de apresentação do nó de composição que contém o nó também entra no estado *occurring*.

Quando todos os nós filhos de um nó de composição tiverem seus eventos de apresentação no estado *sleeping*, o evento de apresentação do nó de composição também vai para o estado *sleeping*.

Os nós de composição não inferem transições *aborts* a partir de seus nós filhos. Essas transições nos eventos de apresentação de nós de composição ocorrem apenas quando instruções são aplicadas diretamente ao seu evento de apresentação.

Quando todos os nós filhos de um nó de composição têm seus eventos de apresentação em um estado diferente de *occurring* e ao menos um dos nós tem seu evento principal no estado *paused*, o evento de apresentação do nó de composição deve também estar no estado *paused*.

Se um elemento <switch> for iniciado mas não definir um componente *default* e nenhuma das regras <bindRule> referenciadas for avaliada como verdadeira, a apresentação *switch* não vai para o estado *occurring*.

H.4 Comportamento de Exibidores de Objetos Hipermídia com Conteúdo Declarativo

Um exibidor de objeto hipermídia com conteúdo composto por um código declarativo (mesmo tendo conteúdos imperativos embutidos) tem um comportamento muito semelhante à apresentação de um nó de composição realizado pelo formatador NCL. Na verdade, o formatador NCL é o exibidor de um objeto hipermídia contendo código declarativo NCL. Em NCL, tais objetos têm seu atributo *type* especificados como application/x-ncl-NCL.

Embora a implementação de referência do *middleware* Ginga-NCL permita que um documento NCL contenha objetos hipermídia com código SMIL [W3C REC-SMIL2-20051213, 2008] e X3D, a norma para o Sistema Brasileiro de TV Digital só permite a documentos NCL embutirem outros documentos NCL ou documentos X-HTML. Vamos, então, neste apêndice, nos restringir a esses objetos. Contudo, como veremos, a discussão que aqui faremos é facilmente generalizada para objetos especificados em outras linguagens declarativas.

H.4.1 Iniciando a Apresentação de um Objeto Hipermídia com Conteúdo Declarativo

Se um elemento <media> com conteúdo declarativo (embutindo ou não códigos imperativos) participar de um papel (*role*) de ação (*action*) cujo tipo de ação é "start", quando essa ação for acionada, uma instrução *start* também é aplicada a todas as cadeias temporais definidas pelo objeto (veja o Capítulo 14 e o Apêndice G), quando nenhuma cadeia do objeto for especificada na ação. Por exemplo, para um objeto de mídia com tipo igual a "application/x-

ncl-NCL", todas as portas do documento NCL especificadas em seu elemento <body> sofrerão a ação "start".

Se o autor quiser iniciar a apresentação a partir de uma cadeia específica, ele também deve definir o identificador da cadeia. Por exemplo, para um objeto de mídia com tipo igual a "application/x-ncl-NCL", uma *interface* do elemento <bind>, que se liga ao objeto com código NCL, deve indicar uma das âncoras do objeto, a qual referencia o *id* de um elemento <port> que inicia a cadeia, conforme discutimos no Capítulo 14.

H.4.2 Parando a Apresentação de um Objeto Hipermídia com Conteúdo Declarativo

Se um elemento <media> com conteúdo declarativo (embutindo ou não códigos imperativos) participar de um papel (*role*) de ação (*action*) cujo tipo de ação é "stop", quando essa ação for acionada, uma instrução *stop* também é aplicada a todas as cadeias definidas internamente pelo objeto quando nenhuma cadeia específica do objeto for especificada na ação. Por exemplo, para um objeto de mídia com tipo igual a "application/x-ncl-NCL", a ação "stop" também é aplicada a todos os eventos de apresentação dos nós filhos do elemento <body> do objeto de mídia com código NCL.

Se o autor quiser parar a apresentação de uma cadeia específica, ele também deve indicar o identificador da cadeia. Por exemplo, para um objeto de mídia com tipo igual a "application/x-ncl-NCL", uma *interface* do elemento <bind>, que se liga ao objeto com código NCL, deve indicar uma das âncoras do objeto, a qual referencia o *id* de um elemento <port> que inicia a cadeia, conforme discutimos no Capítulo 14. Nesse caso, a instrução *stop* também é aplicada a todos os eventos de apresentação de todos os nós filhos do elemento <body>, do objeto de mídia com código NCL, que participam da cadeia e não estão no estado *sleeping*.

Se o objeto de mídia com código declarativo contiver alguma relação entre seus objetos internos sendo avaliada (ou com sua avaliação pausada), a avaliação é suspensa e nenhuma ação é acionada.

H.4.3 Abortando a Apresentação de um Objeto Hipermídia com Conteúdo Declarativo

Se um elemento <media> com conteúdo declarativo (embutindo ou não códigos imperativos) participar de um papel (*role*) de ação (*action*) cujo tipo de ação é "abort" quando essa ação for acionada, uma instrução *abort* também é aplicada a todas as cadeias definidas internamente pelo objeto

quando nenhuma cadeia específica do objeto for especificada na ação. Por exemplo, para um objeto de mídia com tipo igual a "application/x-ncl-NCL", a instrução *abort* também é aplicada a todos os eventos de apresentação dos nós filhos do elemento <body> do objeto de mídia com código NCL.

Se o autor quiser abortar a apresentação de uma cadeia específica, ele também deve indicar o identificador da cadeia. Por exemplo, para um objeto de mídia com tipo igual a "application/x-ncl-NCL", uma *interface* do elemento <bind>, que se liga ao objeto com código NCL, deve indicar uma das âncoras do objeto, a qual referencia o *id* de um elemento <port> que inicia a cadeia, conforme discutimos no Capítulo 14. Nesse caso, a instrução *abort* também é aplicada a todos os eventos de apresentação de todos os nós filhos do elemento <body>, do objeto de mídia com código NCL, que participam da cadeia e não estão no estado *sleeping*.

Se o objeto de mídia com código declarativo contiver alguma relação entre seus objetos internos sendo avaliada (ou com sua avaliação pausada), a avaliação é suspensa e nenhuma ação é acionada.

H.4.4 Pausando a Apresentação de um Objeto Hipermídia com Conteúdo Declarativo

Se um elemento <media> com conteúdo declarativo (embutindo ou não códigos imperativos) participar de um papel (*role*) de ação (*action*) cujo tipo de ação é "pause" quando essa ação for acionada, uma instrução *pause* também é aplicada a todas as cadeias definidas internamente pelo objeto que estejam em apresentação quando nenhuma cadeia específica do objeto for especificada na ação. Por exemplo, para um objeto de mídia com tipo igual a "application/x-ncl-NCL", a instrução *pause* também é aplicada a todos os eventos de apresentação dos nós filhos do elemento <body> do objeto de mídia com código NCL que estejam no estado *occurring*.

Se o autor quiser pausar a apresentação de uma cadeia específica, ele também deve indicar o identificador da cadeia. Por exemplo, para um objeto de mídia com tipo igual a "application/x-ncl-NCL", uma *interface* do elemento <bind>, que se liga ao objeto com código NCL, deve indicar uma das âncoras do objeto, a qual referencia o *id* de um elemento <port> que inicia a cadeia, conforme discutimos no Capítulo 14. Nesse caso, a instrução *pause* também é aplicada a todos os eventos de apresentação de todos os nós filhos do elemento <body>, do objeto de mídia com código NCL, que participam da cadeia e que estejam no estado *occurring*.

Se o objeto de mídia com código declarativo contiver alguma relação entre seus objetos internos sendo avaliada (ou com sua avaliação pausada), a avaliação é suspensa até que uma ação *resume*, *stop* ou *abort* seja emitida.

Se o objeto de mídia com código declarativo contiver cadeias temporais no estado *paused*, quando a ação "pause" no objeto for emitida, essas cadeias são identificadas porque, se o objeto receber uma instrução *resume*, essas cadeias não deverão ser retomadas.

H.4.5 Retomando a Apresentação de um Objeto Hipermídia com Conteúdo Declarativo

Se um elemento <media> com conteúdo declarativo (embutindo ou não códigos imperativos) participar se um papel (*role*) de ação (*action*) cujo tipo de ação é "resume" e nenhuma cadeia específica do objeto for especificada na ação quando essa ação for acionada, uma instrução *resume* também é aplicada a todas as cadeias definidas internamente pelo objeto que estejam no estado *paused*, exceto aquelas que já estavam pausadas antes de o objeto declarativo ser pausado. Por exemplo, para um objeto de mídia com tipo igual a "application/x-ncl-NCL", a instrução *resume* também é aplicada a todos os eventos de apresentação dos nós filhos do elemento <body>, do objeto de mídia com código NCL, que estejam no estado *paused*, exceto aqueles que já estavam pausados antes de o objeto de mídia declarativo ser pausado.

Se o autor quiser retomar a apresentação de uma cadeia específica, ele também deve indicar o identificador da cadeia. Por exemplo, para um objeto de mídia com tipo igual a "application/x-ncl-NCL", uma *interface* do elemento <bind>, que se liga ao objeto com código NCL, deve indicar uma das âncoras do objeto, a qual referencia o *id* de um elemento <port> que inicia a cadeia, conforme discutimos no Capítulo 14. Nesse caso, a instrução *resume* também é aplicada a todos os eventos de apresentação de todos os nós filhos do elemento <body> do objeto de mídia com código NCL que participam da cadeia e que estejam no estado *paused*, exceto aqueles que já estavam pausados antes de o objeto de mídia declarativo ser pausado.

Se o objeto de mídia com código declarativo contiver alguma relação entre seus objetos internos pausada, ela é retomada quando a cadeia da qual esses objetos participam são retomadas.

H.4.6 Comportamento na Execução de Ações sobre Eventos de Atribuição de um Objeto Hiperídia com Código Declarativo

As máquinas de estado associadas a propriedades de um objeto de mídia com código declarativo têm exatamente o mesmo comportamento das máquinas de estado associadas a um objeto de mídia não-imperativo, conforme descrito na Seção H.2.2.

H.4.7 Relação entre os Estados de uma Cadeia Temporal de um Objeto Hipermídia e suas Âncoras de Conteúdo

Sempre que qualquer cadeia temporal de um objeto de mídia com conteúdo declarativo estiver sendo apresentada, o evento de apresentação associado à "âncora de conteúdo total" estará no estado *occurring*.

Um evento associado a uma âncora específica de uma cadeia está no estado *occurring* quando o trecho da cadeia que ele especifica está sendo apresentado.

Quando todas as cadeias de um objeto de mídia com conteúdo declarativo ainda não foram iniciadas (ou foram paradas ou abortadas), o evento de apresentação associado à sua "âncora de conteúdo total" estará no estado *sleeping*.

Um evento associado a uma âncora específica de uma cadeia está no estado *sleeping* quando o trecho da cadeia que ele especifica ainda não foi iniciado ou foi parado ou abortado.

Quando todas as cadeias de um objeto de mídia com conteúdo declarativo não estiverem sendo apresentadas e ao menos uma das cadeias está pausada, o evento de apresentação associado à sua "âncora de conteúdo total" estará no estado *paused*.

Um evento associado a uma âncora específica de uma cadeia está no estado *paused* quando o trecho da cadeia que ele especifica está pausado.

Bibliografia

ABNT, NBR 15606-2 (2011). Associação Brasileira de Normas Técnicas, "Televisão digital terrestre — Codificação de dados e especificações de transmissão para radiodifusão digital, Parte 2: Ginga-NCL para receptores fixos e móveis — Linguagem de aplicação XML para codificação de aplicações", Sistema Brasileiro de TV Digital Terrestre, *NBR 15606-2*.

Soares, L.F.G. e Rodrigues, R.F. (2006). "Nested Context Model 3.0 Part 8 — NCL (Nested Context Language) Digital TV Profiles." Monografias em Ciência da Computação do Departamento de Informática, PUC-Rio, N.º 35/06. Rio de Janeiro, outubro de 2006. ISSN 0103-9741.

W3C, REC-SMIL2-20051213 (2008). World Wide Web Consortium, "Synchronized Multimedia Integration Language — SMIL 2.1 Specification", *W3C Recommendation SMIL2-20051213*.