NORMA
BRASILEIRA

ABNT NBR 15606-2

Primeira edição
30.11.2007

Váilda a partir de
01.12.2007

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

Terrestrial digital television - Data coding and transmission specification for digital broadcasting
Part 2: Ginga-NCL for fixed and mobile receivers - XML application language for application coding

Palavras-chave: Televisão digital terrestre. *Middleware*. Ginga. NCL. Receptores fixos e móveis. Perfil *Full-seg.*
Descriptors: Terrestrial digital televison. Middleware. Ginga. NCL. Mobile and fixed receivers. Full-seg profile.

ICS 33.160.01

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS

Número de referência
ABNT NBR 15606-2:2006
291 páginas

Sede da ABNT
Av.Treze de Maio, 13 - 28º andar
20031-901 - Rio de Janeiro - RJ
Tel.: + 55 21 3974-2300
Fax: + 55 21 2220-1762
abnt@abnt.org.br
www.abnt.org.br

Impresso no Brasil

Prefácio

A Associação Brasileira de Normas Técnicas (ABNT) é o Foro Nacional de Normalização. As Normas Brasileiras, cujo conteúdo é de responsabilidade dos Comitês Brasileiros (ABNT/CB), dos Organismos de Normalização Setorial (ABNT/ONS) e das Comissões de Estudo Especiais Temporárias (ABNT/CEET), são elaboradas por Comissões de Estudo (CE), formadas por representantes dos setores envolvidos, delas fazendo parte: produtores, consumidores e neutros (universidades, laboratórios e outros).

Os Documentos Técnicos ABNT são elaborados conforme as regras da Diretivas ABNT, Parte 2.

A Associação Brasileira de Normas Técnicas (ABNT) chama atenção para a possibilidade de que alguns dos elementos deste documento podem ser objeto de direito de patente. A ABNT não deve ser considerada responsável pela identificação de quaisquer direitos de patentes.

A ABNT NBR 15606-2 foi elaborada pela Comissão de Estudo Especial Temporária de Televisão Digital (ABNT/CEET-00:001.85). O Projeto circulou em Consulta Nacional conforme Edital n° 09, de 21.08.2007 a 20.09.2007, com o número de Projeto 00:001.85-006/2.

Esta Norma é baseada nos trabalhos do Fórum do Sistema Brasileiro de Televisão Digital Terrestre, conforme estabelecido no decreto presidencial n° 5.820, de 29.06.2006.

A ABNT NBR 15606, sob o título geral "*Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiodifusão digital*", tem previsão de conter as seguintes partes:

— Parte 1: Codificação de dados;

— Parte 2: Ginga-NCL para receptores fixos e móveis – Linguagem de aplicação XML para codificação de aplicações;

— Parte 3: Especificação de transmissão de dados;

— Parte 4: Ginga-J – Ambiente para a execução de aplicações procedurais;

— Parte 5: Ginga-NCL para receptores portáteis – Linguagem de aplicação XML para codificação de aplicações.

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

1 Escopo

Esta parte da ABNT NBR 15606 especifica uma linguagem de aplicação XML denominada NCL (*Nested Context Language*), a linguagem declarativa do *middleware* Ginga, a codificação e a transmissão de dados para radiodifusão digital.

2 Referências normativas

Os documentos relacionados a seguir são indispensáveis à aplicação deste documento. Para referências datadas, aplicam-se somente as edições citadas. Para referências não datadas, aplicam-se as edições mais recentes do referido documento (incluindo emendas).

ABNT NBR 15601, *Televisão digital terrestre – Padrão de transmissão*

ABNT NBR 15606-1, *Televisão digital terrestre – Codificação de dados e especificações de transmissão para radiodifusão digital – Parte 1: Codificação de dados*

ABNT NBR 15606-3, *Televisão digital terrestre – Codificação de dados e especificações de transmissão para radiodifusão digital – Parte 3: Especificação de transmissão de dados*

ISO 639-1, *Codes for the representation of names of languages - Part 1: Alpha-2 code*

ISO 8859-1, *Information technology - 8-bit single-byte coded graphic character sets - Part 1: Latin alphabet N° 1*

ISO/IEC 11172-1*, Coding of moving pictures and associated audio for digital storage mediaat up to about 1,5 Mbit/s Part 1 - Systems*

ISO/IEC 11172-2*, Coding of moving pictures and associated audio for digital storage mediaat up to about 1,5 Mbit/s Part 2 - Video*

ISO/IEC 11172-3*, Coding of moving pictures and associated audio for digital storage mediaat up to about 1,5 Mbit/s Part 3 - Audio*

ISO/IEC 13818-1, *Information technology - Generic coding of moving pictures and associated audio information - Part 1:Systems*

ISO/IEC 13818-2, *Information technology - Generic coding of moving pictures and associated audio information- Part 2:Video*

ISO/IEC 13818-3, *Information technology - Generic coding of moving pictures and associated audio information - Part 3: Audio*

ISO/IEC 13818-6, *Information technology - Generic coding of moving pictures and associated audio information - Part 6: Extensions for DSM-CC*

ISO/IEC 13818-7, *Information technology - Generic coding of moving pictures and associated audio information - Part 7: Advanced Audio Coding (AAC)*

ISO/IEC 14496-3, *Information technology - Coding of audio-visual objects - Part 3: Audio*

ECMA 262, *ECMAScript language specification*

3 Termos e definições

Para os efeitos desta Norma, aplicam-se os seguintes termos e definições.

3.1
ambiente de aplicação
contexto ou ambiente de *software* no qual uma aplicação é processada

3.2
ambiente de aplicação declarativa
ambiente que suporta o processamento de aplicações declarativas

NOTA Um formatador (*user agent*) NCL é um exemplo de ambiente de aplicação declarativa.

3.3
ambiente de aplicação procedural
ambiente que suporta o processamento de aplicações procedurais

3.4
API DOM
API que define a estrutura lógica de um documento XML e a forma de acessar, ou manipular, um documento XML

NOTA Esta API é uma interface independente de plataformas e linguagens e segue o Modelo DOM (*Document Object Model*).

3.5
aplicação
informação que expressa um conjunto específico de comportamentos observáveis

3.6
aplicação declarativa
aplicação que utiliza principalmente, e como ponto de partida, informação declarativa para expressar seu comportamento

NOTA Uma instância de documento NCL é um exemplo de aplicação declarativa.

3.7
aplicação híbrida
aplicação híbrida declarativa ou aplicação híbrida procedural

3.8
aplicação híbrida declarativa
aplicação declarativa que contém conteúdo de objeto ativo

NOTA Um documento NCL com um Java Xlet embutido é um exemplo de aplicação híbrida declarativa.

3.9
aplicação híbrida procedural
aplicação procedural com conteúdo declarativo

NOTA Um Java Xlet que cria e causa a exibição de uma instância de documento NCL é um exemplo de aplicação híbrida procedural.

3.10
aplicação nativa
função intrínseca implementada por uma plataforma receptora

NOTA Uma exibição em *closed caption* é um exemplo de aplicação nativa.

3.11
aplicação procedural
aplicação que utiliza principalmente, e como ponto de partida, informações procedurais para expressar o seu comportamento

NOTA Um programa em Java é um exemplo de uma aplicação procedural.

3.12
armazenamento persistente
memória disponível que pode ser lida ou escrita por uma aplicação e pode ser mantida por mais tempo do que o tempo de vida da própria aplicação

NOTA O armazenamento persistente pode ser volátil ou não-volátil.

3.13
atributo
parâmetro para representar a natureza de uma propriedade

3.14
atributo de um elemento
propriedade de um elemento XML

3.15
autor
pessoa que escreve documentos NCL

3.16
canal de interatividade
canal de retorno
mecanismo de comunicação que fornece conexão entre o receptor e um servidor remoto

3.17
caractere
"letra" específica ou outro símbolo identificável

EXEMPLO “A”

3.18
carrossel de dados
método que envia qualquer conjunto de dados ciclicamente, para que esses dados possam ser obtidos, via radiodifusão, em um intervalo de tempo tão longo quanto necessário

[ISO/IEC 13818-6:2001]

3.19
codificação de caracteres
mapeamento entre um valor de entrada inteiro e o caractere textual, representado por esse mapeamento

3.20
conteúdo de objeto ativo
tipo de conteúdo que toma a forma de um programa executável

NOTA Um Xlet Java compilado é um exemplo de conteúdo de objeto ativo.

3.21
conteúdo NCL
conjunto de informações que consiste em um documento NCL e em um grupo de dados, incluindo objetos (de mídia ou de execução), que acompanham o documento NCL

3.22
digital storage media command and control
DSM-CC
método de controle que fornece acesso a um arquivo ou fluxo em serviços digitais interativos

[ISO/IEC 13818-6:2001]

3.23
document type definition
DTD
declaração que descreve um tipo de documento XML

3.24
ECMA*Script*
linguagem de programação definida na ECMA 262

3.25
elemento
unidade de estruturação do documento delimitada por *tags*

NOTA Um elemento é usualmente delimitado por uma *tag* inicial e uma *tag* final, exceto um elemento vazio que é delimitado por uma *tag* de elemento vazio.

3.26
elemento *property*
elemento NCL que define um nome de propriedade e seu valor associado

3.27
entidade da aplicação
unidade de informação que expressa alguma parte de uma aplicação

3.28
evento
ocorrência no tempo que pode ser instantânea ou ter duração mensurável

3.29
exibidor de mídia
media player
componente identificável de um ambiente de aplicação que decodifica ou executa um tipo específico de conteúdo

3.30
eXtensible HTML
XHTML
versão estendida do HTML como aplicação XML

NOTA Na especificação XHTML, um documento HTML é reconhecido como aplicação XML.

3.31
ferramenta de autoria
ferramenta para auxiliar os autores a criar documentos NCL

3.32
fonte
mecanismo que permite a renderização específica de um caractere

EXEMPLO Tiresias, 12 pontos.

NOTA Na prática, um formato de fonte incorpora aspectos da codificação de um caractere.

3.33
formatador NCL
componente de *software* responsável por receber a especificação de um documento NCL e controlar sua apresentação, tentando garantir que os relacionamentos entre os objetos de mídia, especificados pelo autor, sejam respeitados

NOTA Renderizador (*renderer*) de documentos, agente do usuário (*user agent*) e exibidor são outros nomes usados com o mesmo significado do formatador de documentos.

3.34
fluxo de transporte
refere-se à sintaxe do fluxo de transporte MPEG-2 para empacotamento e multiplexação de vídeo, áudio e sinais de dados em sistemas de radiodifusão digital

3.35
fluxo elementar
elementary stream
ES
fluxo básico que contém dados de vídeo, áudio, ou dados privados

NOTA Um único fluxo elementar é transportado em uma seqüência de pacotes PES com um e apenas um identificador (*stream_id*).

3.36
gerenciador de aplicações

entidade responsável por administrar o ciclo de vida das aplicações e que gerencia as aplicações, rodando tanto na máquina de apresentação quanto na máquina de execução

3.37
identificador de pacote
PID

valor inteiro único utilizado para associar os fluxos elementares de um programa, tanto em um fluxo de transporte único ou multiprograma

3.38
informação de serviço
SI

dados que descrevem programas e serviços

3.39
informações específicas do programa
program specific information
PSI

dados normativos necessários para demultiplexar os fluxos de transporte e regenerar os programas

3.40
interface de programação da aplicação
API

bibliotecas de *software* que oferecem acesso uniforme aos serviços do sistema

3.41
linguagem de marcação

formalismo que descreve uma classe de documentos que empregam marcação para delinear a estrutura, aparência ou outros aspectos do documento

3.42
linguagem de *script*

linguagem utilizada para descrever um conteúdo de objeto ativo embutido em documentos NCL e em documentos HTML

3.43
localizador

identificador que fornece uma referência a uma aplicação ou recurso

3.44
máquina de apresentação

subsistema em um receptor que analisa e apresenta aplicações declarativas, com conteúdos como áudio, vídeo, gráficos e texto, baseadas em regras definidas na máquina de apresentação

NOTA Uma máquina de apresentação é responsável pelo controle do comportamento da apresentação e por iniciar outros processos em resposta a entradas do usuário e outros eventos.

EXEMPLO Navegador HTML e formatador NCL.

3.45
máquina de execução

subsistema em um receptor que avalia e executa aplicações procedurais, consistindo em instruções em linguagem de computador, conteúdo de mídia associados e outros dados

NOTA Uma máquina de execução pode ser implementada com um sistema operacional, compiladores de linguagem de computador, interpretadores e interfaces de programação de aplicações (API), que uma aplicação procedural pode utilizar para apresentar conteúdo audiovisual, interagir com o usuário ou executar outras tarefas que não sejam evidentes ao usuário.

EXEMPLO Ambiente de *software* JavaTV, utilizando linguagem de programação Java e interpretador *bytecode*, API JavaTV e máquina virtual Java para execução do programa.

3.46
método

função associada a um objeto que tem permissão de manipular os dados do objeto

3.47
nó NCL

elemento <media>, <context>, <body> ou <switch> de NCL

3.48
normal play time
NPT

coordenada temporal absoluta que representa a posição em um fluxo

3.49
objeto de mídia

coleção de pedaços de dados identificados por nome que pode representar um conteúdo de mídia ou um programa escrito em linguagem específica

3.50
perfil

especificação de uma classe de capacidades, oferecendo diferentes níveis de funcionalidades em um receptor

3.51
perfil *one-seg*

caracteriza o serviço que pode ser recebido por um sintonizador de banda estreita (430 KHz) e portanto com economia no consumo de bateria

NOTA O perfil *one-seg* também é conhecido como perfil portátil.

3.52
perfil *full-seg*

caracteriza o serviço que precisa necessariamente de um demodulador de faixa larga (5,7 MHz) para ser recebido

NOTA Dependendo das configurações de transmissão e de funcionalidade específicas do receptor, pode ser recebido em movimento ou apenas por receptores fixos, porém sem o benefício da economia de energia. A resolução do vídeo transmitido pode ser ou não de alta definição.

3.53
plug-in
conjunto de funcionalidades que pode ser adicionado a uma plataforma genérica para fornecer funcionalidade adicional

3.54
plataforma receptora
plataforma
hardware, sistema operacional e bibliotecas de *software* nativas do receptor, escolhidos pelo fabricante

3.55
recurso
objeto de dados ou um serviço da rede que é identificado univocamente

3.56
sistema de arquivos local
sistema de arquivos fornecido pela plataforma receptora local

3.57
tempo de vida de uma aplicação
período de tempo entre o momento em que uma aplicação é carregada e o momento em que ela é destruída

3.58
uniform resource identifier
URI
método de endereçamento que permite o acesso a objetos em uma rede

3.59
user agent
qualquer programa que interpreta um documento NCL

NOTA Um *user agent* pode exibir um documento, tentando garantir que as relações especificadas pelo autor entre objetos de mídia sejam respeitadas, pronunciá-lo em áudio sintetizado, convertê-lo para um outro formato etc.

3.60
usuário
pessoa que interage com um formatador para visualizar, ouvir ou utilizar de outra forma um documento NCL

3.61
usuário final
indivíduo que opera ou interage com um receptor

4 Abreviaturas

Para os efeitos deste documento, aplicam-se as seguintes abreviaturas.

API	*Application Programming Interface*
BML	*Broadcast Markup Language*
CLUT	*Color Look-up Table*
CSS	*Cascading Style Sheets*
DOM	*Document Object Model*
DSM-CC	*Digital Storage Media Command and Control*
DTD	*Document Type Definition*
DTV	*Digital Television*
DVB	*Digital Video Broadcasting*
GIF	*Graphics Interchange Format*
HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
JPEG	*Joint Photographic Expert Group*
MIME	*Multipurpose Internet Mail Extensions*
MNG	*Multiple Network Graphics*
MPEG	*Moving Picture Expert Group*
NCL	*Nested Context Language*
NCM	*Nested Context Model*
NPT	*Normal Play Time*
OS	*Operating System*
PAT	*Program Association Table*
PES	*Packetized Elementary Stream*
PID	*Packet Identifier*
PMT	*Program Map Table*
PNG	*Portable Network Graphics*
PSI	*Program Specific Information*
SBTVD	Sistema Brasileiro de Televisão Digital Terrestre
SMIL	*Synchronized Multimedia Integration Language*
TS	*Transport Stream*
UCS	*Universal (Coded) Character Set*
URI	*Universal Resource Identifier*
URL	*Universal Resource Locator*
XHTML	*eXtensible HTML*
XML	*Extensible Markup Language*
W3C	*World-Wide Web Consortium*

5 Arquitetura Ginga

5.1 Ginga *main modules*

O universo das aplicações Ginga pode ser particionado em um conjunto de aplicações declarativas e um conjunto de aplicações procedurais. Uma aplicação declarativa é aquela onde o tipo do conteúdo da entidade inicial é declarativo. Por outro lado, uma aplicação procedural é aquela cujo tipo do conteúdo da entidade inicial é procedural. Uma aplicação declarativa pura é aquela na qual o conteúdo de todas as entidades é do tipo declarativo. Uma aplicação procedural pura é aquela na qual o conteúdo de todas as entidades é do tipo procedural. Uma aplicação híbrida é aquela cujo conjunto de entidades possui tanto conteúdo do tipo declarativo quanto procedural. Uma aplicação Ginga não necessita ser puramente declarativa ou procedural.

Em particular, as aplicações declarativas freqüentemente fazem uso de *scripts*, cujo conteúdo é de natureza procedural. Além disso, uma aplicação declarativa pode fazer referência a um código Java TV Xlet embutido. Da mesma forma, uma aplicação procedural pode fazer referência a uma aplicação declarativa, contendo, por exemplo, conteúdo gráfico, ou pode construir e iniciar a apresentação de aplicações com conteúdo declarativo. Portanto, ambos os tipos de aplicação Ginga podem utilizar as facilidades dos ambientes de aplicação declarativo e procedural.

Ginga-NCL é um subsistema lógico do sistema Ginga responsável pelo processamento de documentos NCL[1)]. Um componente-chave do Ginga-NCL é a máquina de interpretação do conteúdo declarativo (formatador NCL). Outros módulos importantes são o exibidor (*user agent*) XHTML, que inclui interpretadores CSS e ECMAScript, e a máquina de apresentação Lua, que é responsável pela interpretação dos *scripts* Lua (ver Anexo B).

Ginga-J é um subsistema lógico do sistema Ginga responsável pelo processamento de conteúdos ativos. Um componente-chave do ambiente de aplicação procedural é a máquina de execução do conteúdo procedural, composta por uma máquina virtual Java.

Decodificadores de conteúdo comuns servem tanto às aplicações procedurais quanto às declarativas que necessitam decodificar e apresentar tipos comuns de conteúdo como PNG, JPEG, MPEG e outros formatos. O Núcleo Comum Ginga (Ginga *Common Core*) é composto pelos decodificadores de conteúdo comuns e por procedimentos para obter conteúdos transportados em fluxos de transporte (*transport streams*) MPEG-2 e através do canal de interatividade. O Núcleo Comum Ginga também deve obrigatoriamente suportar o modelo conceitual de exibição, conforme descrito na ABNT NBR 15606-1.

A arquitetura (ver Figura 1) e facilidades Ginga foram projetadas para serem aplicadas a sistemas de radiodifusão e receptores terrestres de radiodifusão. Adicionalmente, a mesma arquitetura e facilidades podem ser aplicadas a sistemas que utilizam outros mecanismos de transporte de dados (como sistemas de televisão via satélite ou a cabo).

---

1) NCL é marca registrada e sua especificação é propriedade intelectual da PUC-Rio (INPI Departamento de Transferência Tecnológica - No. 0007162-5; 20/12/2005).

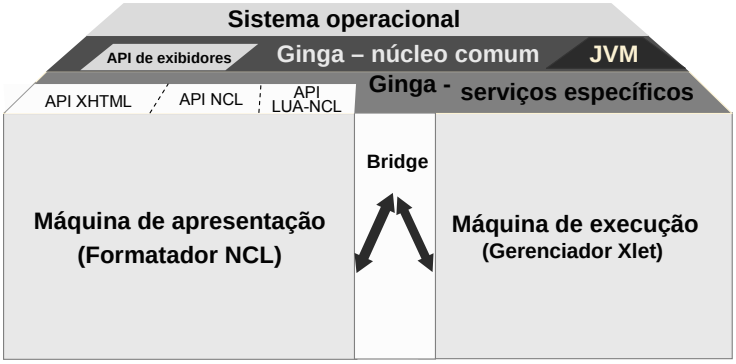

Figura 1 — Arquitetura Ginga

5.2 Interação com o ambiente nativo

Em geral, o Ginga é alheio a quaisquer aplicações nativas que podem também optar por utilizar o plano gráfico. Isso inclui, mas não se limita a aplicações como: *closed caption*, mensagens do sistema de acesso condicional (CA), menus do receptor e guias de programação nativos.

As aplicações nativas podem ter prioridade sobre as aplicações Ginga. O *closed caption* e as mensagens de emergência devem obrigatoriamente ter prioridade sobre o sistema Ginga.

Algumas aplicações nativas, como o *closed caption*, representam um caso especial no qual a aplicação nativa pode estar ativa por longos períodos juntamente com as aplicações Ginga.

6 Interoperabilidade com ambientes declarativos definidos em outros sistemas de televisão digital - Objetos XHTML embutidos em apresentações NCL

6.1 NCL como linguagem cola

Todas as máquinas de apresentação dos três principais sistemas de televisão digital utilizam uma linguagem baseada em XHTML.

XHTML é uma linguagem declarativa baseada em mídias, o que significa que a sua estrutura é definida pelos relacionamentos entre objetos XHTML (documentos XHTML ou objetos inseridos em documentos XHTML) que estão embutidos no conteúdo das mídias do documento. XHTML pode então ser classificada como linguagem de marcação: um formalismo que descreve uma classe de documentos que empregam marcação para delinear a estrutura, aparência e outros aspectos dos documentos.

Os relacionamentos de referência definidos pelos *links* XHTML são o foco dessa linguagem declarativa. Outros tipos de relacionamentos, como relacionamentos de sincronização espaço-temporal e relacionamentos alternativos (adaptação da mídia), são usualmente definidos através de uma linguagem imperativa (por exemplo, ECMAScript).

Diferentemente de XHTML ou HTML, NCL define uma separação bem demarcada entre o conteúdo e a estrutura de um documento (ou aplicativo), provendo um controle não invasivo da ligação entre o conteúdo e sua apresentação e leiaute.

O foco da linguagem declarativa NCL é mais amplo do que o oferecido pela XHTML. A sincronização espaço-temporal, definida genericamente pelos *links* NCL; adaptabilidade, definida pelos elementos *switch* e *descriptor*

switch da NCL; e suporte a múltiplos dispositivos de exibição, definidos por regiões NCL, é o foco dessa linguagem declarativa. A interação do usuário é tratada apenas como caso particular de sincronização temporal.

Como a NCL tem uma separação mais acurada entre o conteúdo e a estrutura, ela não define nenhuma mídia em si. Ao contrário, ela define a cola que prende as mídias em apresentações multimídia.

Um documento NCL apenas define como os objetos de mídia são estruturados e relacionados no tempo e espaço. Como uma linguagem de cola, ela não restringe ou prescreve os tipos de conteúdo dos objetos de mídia. Nesse sentido, pode-se ter objetos de imagem (GIF, JPEG etc.), de vídeo (MPEG, MOV etc.), de áudio (MP3, WMA etc.), de texto (TXT, PDF etc.), de execução (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). Um desses exibidores é o decodificador/exibidor MPEG-4, normalmente implementado em *hardware* no receptor de televisão digital. Dessa forma, o vídeo e o áudio MPEG-4 principal são tratados como todos os demais objetos de mídia que podem estar relacionados utilizando NCL.

Outro objeto de mídia NCL que deve obrigatoriamente ser suportado é o objeto de mídia baseado em XHTML. A NCL não substitui, mas embute documentos (ou objetos) baseados em XHTML. Como acontece com outros objetos de mídia, qual linguagem baseada em XHTML tem suporte em um formatador NCL é uma escolha de implementação e, portanto, depende de qual navegador XHTML, incorporado no formatador NCL, atua como exibidor dessa mídia.

Como conseqüência, é possível ter navegadores BML, DVB-HTML e ACAP-X individualmente embutidos em um exibidor de documento NCL. É possível, ainda, ter todos eles. É igualmente possível receber o código de um programa navegador através da difusão de dados e instalá-lo como *plug-in* (normalmente um *plug-in* Java).

Também é possível ter um navegador genérico implementado e, se necessário, receber a parte complementar (específica) como um *plug-in*, para converter o exibidor XHTML genérico em um exibidor específico de um dos diversos padrões de navegador DTV.

Em último caso, um documento NCL pode ser reduzido para conter apenas um objeto de mídia XHTML. Nesse caso, o exibidor do documento NCL atua quase como um navegador XHTML, isto é, como qualquer outro navegador dos padrões supracitados.

Não importa o caso, a implementação do navegador XHTML deve ser uma conseqüência das seguintes exigências:

- interoperabilidade;
- robustez;
- conformidade com as normas do W3C;
- rejeição de conteúdo não conforme;
- compatibilidade com o modelo de segurança Ginga;
- minimização da redundância com a tecnologia Ginga-J existente;
- minimização da redundância com as facilidades NCL existentes;
- mecanismos precisos de controle do leiaute do conteúdo;
- suporte a diferentes razões de aspecto das unidades de exibição (*pixels*).

Para suportar as facilidades do navegador XHTML definidas por outros padrões DTV, recomenda-se que todas as especificações SBTVD relacionadas à difusão de dados suportem também as facilidades definidas para tais navegadores, como o transporte de eventos de fluxos (*stream events*), por exemplo.

Embora um navegador XHTML deva obrigatoriamente ser suportado, recomenda-se que a utilização de elementos XHTML para definir relacionamentos (inclusive *links* XHTML) seja evitada na autoria de documentos NCL. Recomenda-se que a autoria baseada na estrutura seja priorizada por razões conhecidas e amplamente divulgadas na literatura.

Durante a exibição do conteúdo de objetos de mídia são gerados vários eventos (ver 7.2.8). Alguns exemplos são a apresentação de parte do conteúdo de um objeto de mídia, a seleção de parte do conteúdo de um objeto etc. Os eventos podem gerar ações sobre outros objetos de mídia, como iniciar ou terminar suas apresentações. Portanto, os eventos devem obrigatoriamente ser relatados pelos exibidores de mídia ao formatador NCL que, por sua vez, pode gerar ações a serem aplicadas a esses ou outros exibidores. Ginga-NCL define a API (ver Seção 8) de um adaptador com o objetivo de padronizar a interface entre o formatador Ginga-NCL e cada exibidor específico.

Para que qualquer exibidor de mídia, em particular um navegador XHTML, seja acoplado ao formatador Ginga-NCL, ele deve obrigatoriamente suportar a API dos adaptadores. Assim, para alguns exibidores de mídia, inclusive navegadores XHTML, um módulo adaptador pode ser necessário para que a integração seja alcançada.

Para edição ao vivo, o Ginga-NCL também define eventos de fluxo NCL para oferecer suporte aos eventos gerados ao vivo sobre fluxos de mídia, em particular sobre o fluxo de vídeo do programa principal. Esses eventos são uma generalização do mesmo conceito encontrado em outras normas, como, por exemplo, os *bevents* de BML. Embora um navegador XHTML deva obrigatoriamente ser suportado, recomenda-se que a utilização de elementos XHTML para definir relacionamentos (inclusive eventos de fluxo) seja evitada quando da criação de documentos NCL, pela mesma razão, isto é, recomenda-se que a autoria baseada na estrutura seja priorizada por razões conhecidas e amplamente divulgadas na literatura.

6.2 Formato de conteúdo XHTML

Formatos comuns de conteúdo devem obrigatoriamente ser adotados para a produção e intercâmbio de conteúdo multimídia, como definido na ABNT NBR 15606-1. Além disso, no ambiente de aplicação declarativa também é exigida a especificação de formatos comuns de conteúdos XHTML para as aplicações de televisão interativa.

NOTA Esta Norma segue a ITU Recommendation J.201 para identificar as funcionalidades comuns entre os ambientes de aplicação declarativa para aplicações de televisão interativa especificadas por DVB-HTML, ACAP-X e BML.

Convém que os elementos comuns e API no nível sintático de objetos de mídia XHTML embutidos em aplicações NCL sejam especificados, para auxiliar os autores na criação de conteúdo XHTML.

Qualquer implementação de objeto de mídia XHTML de acordo com esta Norma deve obrigatoriamente dar suporte a pelo menos todas as marcações XML e propriedades de folhas de estilo em comum aos serviços básicos BML ("perfil terminal fixo"), ACAP-X e DVB-HTML, como definido em 6.3. É recomendado que facilidades de objetos nativos ECMAScript e API DOM, em comum aos serviços básicos BML ("perfil terminal fixo"), ACAP-X e DVB-HTML, também tenham suporte.

6.3 Harmonização do formato de conteúdo XHTML

6.3.1 Marcações XML

NOTA Objetos de mídia NCL baseados em XHTML seguem a recomendação W3C "*Modularization of XHTML*" e suas marcações XML são definidas na ITU Recommendation J.201.

Os módulos em comum de marcações XML podem ser:

— *structure;*

— *text;*

— *hypertext;*

— *list;*

— *presentation;*

— *bidirectional text;*

— *forms;*

— *image;*

— *client-side image map;*

— *object;*

— *frames;*

— *target;*

— *meta information;*

— *scripting;*

— *stylesheet;*

— *style attribute;*

— *link;*

— *base.*

As coleções de atributo XHTML são definidas de acordo com a Tabela 1. As marcações XML em comum dos padrões serviços básicos BML (“perfil de terminal fixo”), ACAP-X e DVB-HTML, que devem obrigatoriamente ser suportadas por qualquer implementação, são listadas na Tabela 2, em conjunto com as extensões Ginga obrigatórias.

Tabela 1 — Coleções de atributos

Nome da coleção	Atributos na coleção	Condição do atributo
Core	class (NMTOKENS)	Requerido
	Id (ID),	Requerido
	title (CDATA)	–
I18N	xml:lang (CDATA)	Requerido
Events	onclick (Script)	Requerido
	ondblclick (Script)	–
	onmousedown (Script)	–
	onmouseup (Script)	–
	onmouseover (Script)	–
	onmousemove (Script)	–
	onmouseout (Script)	–
	onkeypress (Script)	–
	onkeydown (Script)	Requerido
	onkeyup (Script)	Requerido
Style	style (CDATA)	Requerido
Common	Core + Events + I18N + Style	

Tabela 2 — Elementos de marcação XML em comum

Módulo		Elemento	Condição do elemento	Atributo	Condição do atributo
Core	Structure	body	Requerido	%Common.attrib	
				%Core.attrib	Requerido
				%I18n.attrib	Requerido
				%Events.attrib	–
		head	Requerido	%I18n.attrib	Requerido
				profile	–
		html	Requerido		
		title	Requerido	%I18n.attrib	Requerido
	Text	abbr	–		
		acronym	–		
		address	–		
		blockquote	–		
		br	Requerido	%Core.attrib	Requerido
		cite	–		
		code	–		
		dfn	–		
		div	Requerido	%Common.attrib	Requerido
		em	–		
		h1	Requerido	%Common.attrib	Requerido
		h2	Requerido	%Common.attrib	Requerido
		h3	Requerido	%Common.attrib	Requerido
		h4	Requerido	%Common.attrib	Requerido
		h5	Requerido	%Common.attrib	Requerido
		h6	Requerido	%Common.attrib	Requerido
		kbd	–		
		p	Requerido	%Common.attrib	Requerido
		pre	–		
		q	–		
		samp	–		
		span	Requerido	%Common.attrib	Requerido
		strong	–		
		var	–		
	Hypertext	a	Requerido	%Common.attrib	Requerido
				accesskey	Requerido
				charset	Requerido
				href	Requerido
				hreflang	–
				rel	–
				rev	–
				tabindex	–
				type	–
	List	dl	–		
		dt	–		
		dd	–		
		ol	–		
		ul	–		
		li	–		

Tabela 2 (continuação)

Módulo		Elemento	Condição do elemento	Atributo	Condição do atributo
Applet		applet	–		
		param	–		
Text extension	Presentation	b	–		
		big	–		
		hr	–		
		i	–		
		small	–		
		sub	–		
		sup	–		
		tt	–		
	Edit	del	–		
		ins	–		
	Bi-directional text	bdo	–		
Forms	Basic forms	form	–		
		input	–		
		label	–		
		select	–		
		option	–		
		textarea	–		
	Forms	form	Requerido	%Common.attrib	Requerido
				action	Requerido
				method	Requerido
				enctype	Requerido
				accept-charset	Requerido
				accept	Requerido
				name	Requerido
		input	Requerido	%Common.attrib	Requerido
				accesskey	Requerido
				checked	–
				disabled	Requerido
				readonly	Requerido
				maxlength	Requerido
				alt	–
				name	–
				size	Requerido
				src	–
				tabindex	–
				accept	–
				type	Requerido
				value	Requerido
		select	–		
		option	–		
		textarea	–		
		button	–		
		fieldset	–		
		label	–		
		legend	–		
		optgroup	–		

Tabela 2 (continuação)

Módulo		Elemento	Condição do elemento	Atributo	Condição do atributo
Table	Basic tables	caption	–		
		table	–		
		td	–		
		th	–		
		tr	–		
	Tables	caption	–		
		table	–		
		td	–		
		th	–		
		tr	–		
		col	–		
		colgroup	–		
		tbody	–		
		thead	–		
		tfoot	–		
Image		img	–		
Client side map		a&	–		
		area	–		
		img&	–		
		input&	–		
		map	–		
		object&	–		
Server side image map		img&	–		
		Input&	–		
Object		object	Requerido	%Common.attrib	Requerido
				archive	–
				classid	–
				codebase	–
				codetype	–
				data	Requerido
				declare	–
				height	Requerido
				name	–
				standby	–
				tabindex	–
				type	Requerido
				width	Requerido
		param	–		
Frames		frameset	–		
		frame	–		
		noframe	–		
Target		a&	–		
		area&	–		
		base&	–		
		link&	–		
		form&	–		
IFrame		iframe	–		

Tabela 2 (continuação)

Módulo	Elemento	Condição do elemento	Atributo	Condição do atributo
Intrinsic events	a&	Requerido		
	area&	–		
	frameset&	–		
	form&	–		
	body&	–		
	label&	–		
	input&	–		
	select&	–		
	textarea&	–		
	button&	–		
Metainformation	meta	Requerido	%I18n.attrib	–
			http-equiv	–
			name	Requerido
			content	Requerido
			scheme	–
Scripting	noscript			
	script	Requerido	charset	Requerido
			type	Requerido
			src	–
			defer	–
Stylesheet	style	Requerido	%I18n.attrib	Requerido
			id	–
			type	Requerido
			media	Requerido
			title	–
Style attribute		Requerido		
Link	link	Requerido		
Base	base	–		

6.3.2 Folhas de estilo

As propriedades de folhas de estilo (CSS) são listadas na Tabela 3.

Tabela 3 — Propriedades de folhas de estilo em comum

background	clear	outline-color
background-attachment	clip	outline-style
background-color	color	outline-width
background-image	content	overflow
background-position	counter-increment	padding
background-repeat	counter-reset	padding-bottom
border	display	padding-left
border-bottom	float	padding-right
border-bottom-color	font	padding-top
border-bottom-style	font-family	position
border-bottom-width	font-size	right
border-color	font-style	text-align
border-left	font-variant	text-decoration
border-left-color	font-weight	text-indent
border-left-style	height	text-transform
border-left-width	left	top
border-right	letter-spacing	vertical-align
border-right-color	line-height	visibility
border-right-style	list-style	white-space
border-right-width	list-style-image	width
border-style	list-style-position	word-spacing
border-top	list-style-type	z-index
border-top-color	margin	nav-down
border-top-style	margin-bottom	nav-index
border-top-width	margin-left	nav-left
border-width	margin-right	nav-right
bottom	margin-top	nav-up
caption-side	outline	----

As propriedades de folhas de estilo em comum aos padrões serviços básicos BML, ACAP-X e DVB-HTML, que devem obrigatoriamente ser suportadas por qualquer implementação, são listadas na Tabela 4.

Tabela 4 — Propriedades de folhas de estilo CSS 2 em comum

Propriedade	Condição da propriedade
Value assignment/Inheritance	
@import	–
!important	–
Media type	
@media	Requerido
box model	
margin-top	–
margin-right	–
margin-bottom	–
margin-left	–
margin	Requerido
padding-top	Requerido
padding-right	Requerido
padding-bottom	Requerido
padding-left	Requerido
padding	Requerido
border-top-width	–
border-right-width	–
border-bottom-width	–
border-left-width	–
border-width	Requerido
border-top-color	–
border-right-color	–
border-bottom-color	–
border-left-color	–
border-color	Requerido
border-top-style	–
border-right-style	–
border-bottom-style	–
border-left-style	–
border-style	Requerido
border-top	–
border-right	–
border-bottom	–
border-left	–
border	Requerido
Visual formatting model	
position	Requerido
left	Requerido
top	Requerido
width	Requerido
height	Requerido
z-index	Requerido
line-height	Requerido
vertical-align	–
display	Requerido
bottom	–
right	–
float	–
clear	–
direction	–

Tabela 4 (continuação)

Propriedade	Condição da propriedade
unicode-bidi	–
min-width	–
max-width	–
min-height	–
max-height	–
Other visual effects	
visibility	Requerido
overflow	Requerido
clip	–
Generated content/Auto numbering/List	
content	–
quotes	–
counter-reset	–
counter-increment	–
marker-offset	–
list-style-type	–
list-style-image	–
list-style-position	–
list-style	–
Page media	
"@page"	–
size	–
marks	–
page-break-before	–
page-break-after	–
page-break-inside	–
page	–
orphans	–
widows	–
Background	
background	–
background-color	–
background-image	Requerido
background-repeat	Requerido
background-position	–
background-attachment	–
Font	
color	Requerido
font-family	Requerido
font-style	Requerido
font-size	Requerido
font-variant	Requerido
font-weight	Requerido
font	Requerido
font-stretch	–
font-adjust	–
Text	
text-indent	–
text-align	Requerido
text-decoration	–

Tabela 4 (continuação)

Propriedade	Condição da propriedade
text-shadow	–
letter-spacing	Requerido
word-spacing	–
text-transform	–
white-space	Requerido
Pseudo class/ Pseudo element	
:link	–
:visited	–
:active	Requerido
:hover	–
:focus	Requerido
:lang	–
:first-child	–
:first-line	–
:first-letter	–
:before	–
:after	–
Table	
caption-side	–
border-collapse	–
border-spacing	–
table-layout	–
empty-cells	–
speak-header	–
User interface	
outline-color	–
outline-width	–
outline-style	–
outline	–
cursor	–
Voice style sheet	
volume	–
speak	–
pause-before	–
pause-after	–
pause	–
cue-before	–
cue-after	–
cue	–
play-during	–
azimuth	–
elevation	–
speech-rate	–
voice-family	–
pitch	–
pitch-range	–
stress	–
richness	–
speak-punctuation	–
peak-numeral	–

Tabela 4 (continuação)

Propriedade	Condição da propriedade
Extended property	
clut	–
color-index	–
background-color-index	–
border-color-index	–
border-top-color-index	–
border-right-color-index	–
border-bottom-color-index	–
border-left-color-index	–
outline-color-index	–
resolution	–
display-aspect-ratio	–
grayscale-color-index	–
nav-index	–
nav-up	–
nav-down	–
nav-left	–
nav-right	–
used-key-list	–

As seguintes restrições devem obrigatoriamente ser aplicadas às propriedades de exibição:

— somente elementos de bloco podem ser aplicados para <p>, , <body>, <input> e <object>;

— somente valores definidos no próprio elemento HTML podem ser aplicados para
, <a> e .

Além disso, as seguintes restrições devem obrigatoriamente ser aplicadas às propriedades de posição:

— somente valores absolutos podem ser aplicados para <p>, , <input> e <object>;

— somente valores estáticos podem ser aplicados para
, e <a>.

Os seletores CSS em comum dos padrões serviços básicos BML, ACAP-X e DVB-HTML, que devem obrigatoriamente ser suportados por qualquer implementação, são os seguintes:

— *universal;*

— *type;*

— *class;*

— *id;*

— *dynamic* (:active and :focus).

6.3.3 ECMAScript

Quando implementada, a máquina ECMAScript deve obrigatoriamente dar suporte aos objetos nativos em comum dos padrões de serviços básicos BML, ACAP-X e DVB-HTML, listados na Tabela 5. Como restrição, os tipos numéricos suportam apenas operações inteiras.

Tabela 5 — Objetos nativos em comum

Object	*Method, properties*	*Operation condition*
(global)		
	NaN	Requerido
	Infinity	–
	eval(x)	–
	parseInt(string, radix)	Requerido
	parseFloat(string)	–
	escape(string)	–
	unescape(string)	–
	isNaN(number) O	Requerido
	isFinite(number)	–
Object		Todos requeridos
	prototype	Requerido
	Object([value])	Requerido
	new Object([value])	Requerido
Object.prototype		Todos requeridos
	constructor	Requerido
	toString()	Requerido
	valueOf()	Requerido
Function		
	prototype	Requerido
	Length	Requerido
	Function(p1, p2, . . . , pn, body)	–
	new Function(p1, p2, . . . , pn, body)	–
Function.prototype		Todos requeridos
	constructor	Requerido
	toString()	Requerido
Array		Todos requeridos
	prototype	Requerido
	Length	Requerido
	Array(item0, item1, . . .)	Requerido
	new Array(item0, item1, . . .)	Requerido
	new Array([len])	Requerido
Array.prototype		Todos requeridos
	constructor	Requerido
	toString()	Requerido
	join([separator])	Requerido
	reverse()	Requerido
	sort([comparefn])	Requerido
	constructor	Requerido
String		Todos requeridos
	prototype	Requerido
	Length	Requerido
	String([value])	Requerido
	new String([value])	Requerido
	String.fromCharCode(char0[, char1, . . .])	Requerido

Tabela 5 (continuação)

Object	***Method, properties***	***Operation condition***
String.prototype		Todos requeridos
	constructor	Requerido
	toString()	Requerido
	valueOf()	Requerido
	charAt(pos)	Requerido
	charCodeAt(pos)	Requerido
	indexOf(searchString, position)	Requerido
	lastIndexOf(searchString, position)	Requerido
	split(separator)	Requerido
	substring(start [,end])	Requerido
	toLowerCase()	Requerido
	toUpperCase()	Requerido
Boolean		Todos requeridos
	prototype	Requerido
	Boolean([value])	Requerido
	new Boolean([value])	Requerido
Boolean.prototype		Todos requeridos
	constructor	Requerido
	toString()	Requerido
	valueOf()	Requerido
Number		
	prototype	Requerido
	MAX_VALUE	Requerido
	MIN_VALUE	Requerido
	NaN	Requerido
	NEGATIVE_INFINITY	–
	POSITIVE_INFINITY	–
	Number([value])	Requerido
	new Number([value])	Requerido
Number.prototype		Todos requeridos
	constructor	Requerido
	toString([radix])	Requerido
	valueOf()	Requerido
Math		
	E	–
	LN10	–
	LN2	–
	LOG2E	–
	LOG10E	–
	PI	–
	SQRT1_2	–
	SQRT2	–
	abs(x)	–
	acos(x)	–
	asin(x)	–
	atan(x)	–
	atan2(y, x)	–
	cos(x)	–

Tabela 5 (continuação)

Object	*Method, properties*	*Operation condition*
Math		
	exp(x)	–
	floor(x)	–
	log(x)	–
	max(x, y)	–
	min(x, y)	–
	pow(x, y)	–
	random()	–
	round(x)	–
	sin(x)	–
	sqrt(x)	–
	tan(x)	–
Date		
	prototype	Requerido
	Date([year, month [, date [, hours [, minutes [,seconds [, ms]]]]]])	Requerido
	new Date([year, month [, date [, hours [, minutes[, seconds [, ms]]]]]])	Requerido
	Date(value)	–
	new Date(value)	–
	Date.parse(string)	–
	Date.UTC([year [, month [, date [, hours [,minutes [, seconds [, ms]]]]]]])	–
Date.prototype		
	constructor	Requerido
	toString()	Requerido
	valueOf()	–
	getTime()	–
	getYear())	–
	getFullYear()	Requerido
	getUTCFullYear()	Requerido
	getMonth()	Requerido
	getUTCMonth()	Requerido
	getDate()	Requerido
	getUTCDate()	Requerido
	getDay()	Requerido
	getUTCDay()	Requerido
	getHours()	Requerido
	getUTCHours()	Requerido
	getMinutes()	Requerido
	getUTCMinutes()	Requerido
	getSeconds()	Requerido
	getUTCSeconds()	Requerido
	getMilliseconds()	Requerido
	getUTCMilliseconds()	Requerido
	getTimezoneOffset()	Requerido
	setTime(time)	–
	setMilliseconds(ms)	Requerido
	setUTCMilliseconds(ms)	Requerido

Tabela 5 (continuação)

Object	*Method, properties*	*Operation condition*
Date.prototype		
	setSeconds(sec [, ms])	Requerido
	setUTCSeconds(sec [, ms])	Requerido
	setMinutes(min [, sec [, ms]])	Requerido
	setUTCMinutes(min [, sec [, ms]])	Requerido
	setHours(hour [, min [, sec [, ms]]])	Requerido
	setUTCHours(hour [, min [, sec [, ms]]])	Requerido
	setDate(date)	Requerido
	setMonth(mon [, date])	Requerido
	setUTCMonth(mon [, date])	Requerido
	setFullYear(year [, mon [, date]])	Requerido
	setUTCFullYear(year [, mon [, date]])	Requerido
	setYear(year)	–
	toLocaleString()	Requerido
	toUTCString()	Requerido
	toGMTString()	–

Dependendo da implementação do *middleware*, é possível ter funções ECMAScript mapeadas para as API fornecidas pelo Ginga-J, obtendo acesso a alguns recursos da plataforma receptora e facilidades Ginga. Nesse caso, recomenda-se que a API fornecida no ECMAScript siga a mesma especificação apresentada para o ambiente procedural do Ginga-J.

6.3.4 API DOM

As API DOM nível 1 são as seguintes:

- DOMException;
- DOMImplementation;
- DocumentFragment;
- Document;
- Node;
- NodeList;
- NamedNodeMap;
- CharacterData;
- Attr;
- Element;
- Text;
- Comment.

As API DOM nível 1, quando implementadas, devem obrigatoriamente seguir as API comuns do DOM nível 1 para serviços básicos BML, ACAP-X e DVB-HTML, listadas na Tabela 6.

Tabela 6 — API DOM nível 1 em comum

Interface	Atributo/Método	Condição da operação
DOMImplementation		
	hasFeature()	Requerido
Document		
	doctype	–
	implementation	Requerido
	documentElement	Requerido
	createElement()	–
	createDocumentFragment()	–
	createTextNode()	–
	createComment()	–
	createCDATASection()	–
	createProcessingInstruction()	–
	createAttribute()	–
	createEntityReference()	–
	getElementsByTagName()	–
Node		
	nodeName	–
	nodeValue	–
	nodeType	–
	parentNode	Requerido
	childNodes	–
	firstChild	Requerido
	lastChild	Requerido
	previousSibling	Requerido
	nextSibling	Requerido
	Attributes	–
	ownerDocument	–
	insertBefore()	–
	replaceChild()	–
	removeChild()	–
	appendChild()	–
	hasChildNodes()	–
	cloneNode()	–
CharacterData		
	data	Requerido
	length	Requerido
	substringData()	–
	appendData()	–
	insertData()	–
	deleteData()	–
	replaceData()	–
Element		
	tagName	Requerido
	getAttribute()	–
	setAttribute()	–
	removeAttribute()	–
	getAttributeNode()	–
	setAttributeNode()	–
	removeAttributeNode()	–
	getElementsByTagName()	–
	normalize()	–
Text		
	splitText	–

7 NCL - Linguagem declarativa XML para especificação de apresentações multimídia interativas

7.1 Linguagens modulares e perfis de linguagens

7.1.1 Módulos NCL

A abordagem modular tem sido utilizada em várias linguagens recomendadas pelo W3C.

Módulos são coleções de elementos, atributos e valores de atributos XML semanticamente relacionados que representam uma unidade de funcionalidade. Módulos são definidos em conjuntos coerentes. Essa coerência é expressa por meio da associação de um mesmo *namespace* aos elementos desses módulos.

NOTA *Namespaces* são discutidos em Namespaces in XML:1999.

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.

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*.

Outros perfis de linguagem podem ser especificados como subconjuntos de um perfil maior ou incorporar uma combinação de módulos associados a diferentes *namespaces*. Um exemplo desse último caso é a utilização de módulos SMIL (por exemplo, o módulo *SMIL Transition*) em perfis NCL. Exemplos do primeiro caso são os perfis TVD Básico (perfil BDTV) e TVD Avançado (perfil EDTV) da NCL.

Subconjuntos dos módulos do perfil Linguagem NCL utilizados na definição dos perfis TVD Básico e TVD Avançado são definidos, para ajustar a linguagem às características do ambiente de radiodifusão de televisão, com seus vários dispositivos de apresentação: aparelho de televisão, dispositivos móveis etc.

NOTA Uma abordagem similar também é encontrada em outras linguagens (SMIL 2.1 Specification:2005 e XHTML 1.0:2002).

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.

NOTA Renderizador de documentos, agente do usuário e exibidor são outros nomes atribuídos ao formatador de documentos.

A versão NCL 3.0 revisa as funcionalidades contidas na NCL 2.3 (NCL Main Profile:2005) e é particionada em 13 áreas funcionais, que são novamente particionadas em módulos. A partir das 13 áreas funcionais, 12 são utilizadas para definir os perfis TVD Avançado e TVD Básico. Além das 12 áreas funcionais de NCL 3.0, 2 áreas funcionais foram importadas de SMIL 2.0 para compor o perfil TVD Avançado. As 14 áreas funcionais utilizadas e seus módulos correnspondentes são:

1) *Structure*

Módulo *Structure*

2) *Layout*

Módulo *Layout*

3) *Components*

Módulo *Media*

Módulo *Context*

4) Interfaces

Módulo *MediaContentAnchor*

Módulo *CompositeNodeInterface*

Módulo *PropertyAnchor*

Módulo *SwitchInterface*

5) *Presentation Specification*

Módulo *Descriptor*

6) *Linking*

Módulo *Linking*

7) *Connectors*

Módulo *ConnectorCommonPart*

Módulo *ConnectorAssessmentExpression*

Módulo *ConnectorCausalExpression*

Módulo *ConnectorTransitionAssessment*

Módulo *CausalConnector*

Módulo *CausalConnectorFunctionality*

Módulo *ConnectorBase*

8) *Presentation Control*

Módulo *TestRule*

Módulo *TestRuleUse*

Módulo *ContentControl*

Módulo *DescriptorControl*

9) *Timing*

Módulo *Timing*

10) *Reuse*

Módulo *Import*

Módulo *EntityReuse*

Módulo *ExtendedEntityReuse*

11) *Navigational Key*

Módulo *KeyNavigation*

12) *Animation*

Módulo *Animation*

13) *SMIL Transition Effects*

Módulo *TransitionBase*

Módulo *BasicTransition*

Módulo *TransitionModifiers*

NOTA O módulo *TransitionBase* é um módulo definido pela NCL 3.0; não existe na SMIL 2.0.

14) *SMIL Meta-Information*

Módulo *Metainformation*

7.1.2 Identificadores para módulos e perfis de linguagem da NCL 3.0

Recomenda-se que cada perfil NCL declare explicitamente o URI do *namespace* que será usado para identificá-lo.

Documentos criados em perfis de linguagem que incluem o módulo *Structure* de NCL podem ser associados com o tipo *MIME* “application/x-ncl+xml”. Os documentos utilizando o tipo *MIME* “application/x-ncl+xml” devem obrigatoriamente estar em conformidade com a linguagem hospedeira.

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 devem obrigatoriamente estar de acordo com a Tabela 7.

Cada módulo SMIL possui um identificador único a ele associado. Os identificadores dos módulos SMIL 2.0 devem obrigatoriamente estar de acordo com a Tabela 8.

Módulos também podem ser identificados coletivamente. As seguintes coleções de módulos são definidas:

— módulos utilizados pelo perfil Linguagem NCL 3.0: http://www.ncl.org.br/NCL3.0/LanguageProfile;

— módulos utilizados pelo perfil Conector Causal NCL 3.0: http://www.ncl.org.br/NCL3.0/CausalConnectorProfile;

— módulos utilizados pelo perfil DTV Avançado NCL 3.0: http://www.ncl.org.br/NCL3.0/EDTVProfile;

— módulos utilizados pelo perfil DTV Básico NCL 3.0: http://www.ncl.org.br/NCL3.0/BDTVProfile.

Tabela 7 — 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
ConnectorTransitionAssessment	http://www.ncl.org.br/NCL3.0/ConnectorTransitionAssessment
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

Tabela 8 — Identificadores dos módulos de SMIL 2.0 utilizados em perfis NCL

Módulos	Identificadores
BasicTransitions	http://www.w3.org/2001/SMIL20/BasicTransitions
Metainformation	http://www.w3.org/2001/SMIL20/Metainformation

7.1.3 Informações sobre versões da NCL

As seguintes instruções de processamento devem obrigatoriamente ser incluídas em um documento NCL. Elas identificam documentos NCL que contenham apenas os elementos definidos nesta Norma, e a 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 é 3.0.

Novas versões da NCL devem obrigatoriamente ser publicadas 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 está definida sob o URI http://www.ncl.org.br/NCL3.0/profileName, onde o número da versão é escrito imediatamente após a sigla “NCL”.

O nome do perfil (*profileName*) no URI deve obrigatoriamente ser *EDTVProfile* (Perfil TVD Avançado) ou *BDTVProfile* (Perfil TVD Básico).

7.2 Módulos NCL

7.2.1 Observações gerais

As principais definições de cada um dos módulos NCL 3.0 presentes nos perfis NCL TVD Básico e TVD Avançado são dadas em 7.2.2 a 7.2.13.

A definição completa dos módulos NCL 3.0, utilizando XML *Schema*, é apresentada no Anexo A. Qualquer ambigüidade encontrada neste texto pode ser esclarecida por meio da consulta aos esquemas XML.

As principais definições de cada um dos módulos SMIL 2.0 presentes no perfil TVD Avançado são dadas em 7.2.14 e 7.2.15.

NOTA A definição completa dos módulos importados da especificação SMIL 2.0 pode ser encontrada em SMIL 2.1 Specification: 2005.

Após discutir cada módulo, uma tabela é apresentada para indicar os elementos do módulo e seus atributos. Para um dado perfil, os atributos e conteúdos (elementos filhos) dos elementos podem ser definidos no próprio módulo ou no perfil da linguagem que agrupa os módulos.

As tabelas descritas em 7.2.2 a 7.2.15 mostram os atributos e conteúdos que vêm do perfil NCL DTV Avançado, além dos definidos nos próprios módulos. As tabelas descritas em 7.3.3 mostram os atributos e conteúdos que vêm do perfil NCL TVD Básico, além dos definidos nos próprios módulos. Os atributos de elementos que são obrigatórios estão sublinhados. Nas tabelas, os seguintes símbolos são empregados: (?) opcional (zero ou

uma ocorrência), (|) ou, (*) zero ou mais ocorrências, (+) uma ou mais ocorrências. A ordem dos elementos filhos não é especificada nas tabelas.

7.2.2 Área funcional *Structure*

A área funcional *Structure* tem apenas um módulo, chamado *Structure*, que define a estrutura básica de um documento NCL. Essa área define o elemento raiz, chamado <ncl>, o elemento <head> e o elemento <body>, seguindo a terminologia adotada por outros padrões W3C. O elemento <body> de um documento NCL é tratado como um nó de contexto NCM (NCMCore:2005).

No NCM, o modelo conceitual de dados da NCL, um nó pode ser um contexto, um switch ou um objeto de mídia. Todos os nós NCM são representados por elementos NCL correspondentes. Os nós de contexto, conforme definido em 7.2.4, contêm outros nós e elos NCM.

Os elementos <ncl> e <body> podem definir um atributo *id*. O atributo *id* identifica univocamente um elemento dentro de um documento. Seu valor é um identificador XML.

O atributo *title* de <ncl> oferece informação adicional sobre o elemento. Os valores do atributo *title* podem ser utilizados por agentes de usuários de várias formas.

O atributo *xmlns* declara um namespace XML, isto é, ele declara a coleção primária de construções XML utilizada pelo documento. O valor do atributo é o URL (*Uniform Resource Locator*), que identifica onde o *namespace* está oficialmente definido. Três valores são permitidos para o atributo *xmlns*: http://www.ncl.org.br/NCL3.0/EDTVProfile e http://www.ncl.org.br/NCL3.0/BDTVProfile, para os perfis TVD Avançado e Básico, respectivamente, e http://www.ncl.org.br/NCL3.0/CausalConnectorProfile, para o perfil Conector Causal. Um formatador NCL deve obrigatoriamente saber que a localização dos esquemas para tais *namespaces* é, por *default*, respectivamente:
http://www.ncl.org.br/NCL3.0/profiles/NCL30EDTV.xsd,
http://www.ncl.org.br/NCL3.0/profiles/NCL30BDTV.xsd, e
http://www.ncl.org.br/NCL3.0/profiles/NCL30CausalConnector.xsd

Os elementos filhos de <head> e <body> são definidos em outros módulos NCL. A ordem na qual os elementos filhos de <head> devem obrigatoriamente ser declarados é: importedDocumentBase?, ruleBase?, transitionBase?, regionBase*, descriptorBase?, connectorBase?, meta*, metadata*.

Os elementos deste módulo, seus elementos filhos e seus atributos devem obrigatoriamente estar de acordo com a Tabela 9.

Tabela 9 — Módulo *Structure* estendido

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)*

7.2.3 Área funcional *Layout*

A área funcional *Layout* tem um único módulo, chamado *Layout*, o qual especifica elementos e atributos que definem como os objetos serão inicialmente apresentados dentro de regiões de dispositivos de saída. De fato, este módulo define valores iniciais para propriedades NCL homônimas definidas nos elementos <media>, <body> e <context> (ver 7.2.4).

Um elemento <regionBase>, que deve obrigatoriamente ser declarado no elemento <head> do documento NCL, define um conjunto de elementos <region>, cada qual podendo conter outro conjunto de elementos <region> aninhados, e assim por diante, recursivamente.

O elemento <regionBase> pode ter um atributo *id*. Elementos <region> devem obrigatoriamente ter um atributo *id*. Como esperado, o atributo *id* identifica univocamente um elemento dentro de um documento.

Cada elemento <regionBase> está associado a um dispositivo onde acontecerá a apresentação. Para identificar a associação, o elemento <regionBase> define o atributo *device*, que pode ter os valores: “systemScreen (i)” ou “systemAudio(i)”. O dispositivo escolhido define as variáveis globais do ambiente: systemScreenSize(i), systemScreenGraphicSize(i) e systemAudioType(i), como definido na Tabela 12 (ver 7.2.4).

Recomenda-se que a interpretação do aninhamento das regiões dentro de um <regionBase> seja feita pelo *software* responsável pela orquestração da apresentação do documento (isto é, o formatador NCL). Para os efeitos desta Norma, um primeiro nível de aninhamento deve obrigatoriamente ser interpretado como se fosse a área do dispositivo onde a apresentação ocorrerá; o segundo nível como janelas (por exemplo, áreas de apresentação na tela) da área-pai; e os outros níveis como regiões dentro dessas janelas.

Uma <region> pode também definir os seguintes atributos: *title*, *left*, *right, top, bottom*, *height*, *width* e *zIndex*. Todos esses atributos possuem o significado usual W3C.

A posição de uma região, conforme especificada por seus atributos *top*, *bottom*, *left* e *right*, é sempre relativa à geometria-pai, que é definida pelo elemento <region> pai ou pela área total do dispositivo, no caso das regiões no primeiro nível de aninhamento. Os valores dos atributos podem ser valores percentuais não-negativos, ou unidades de pixels. Para valores em pixels, o autor pode omitir o qualificador de unidade “px” (por exemplo, “100”). Para valores percentuais, por outro lado, o símbolo “%” deve obrigatoriamente ser indicado (por exemplo, “50%”). O percentual é sempre relativo à largura do pai, no caso das definições dos atributos *right*, *left* and *width*, e à altura do pai, para as definições dos atributos *bottom*, *top* e *height*.

Os atributos *top* e *left* são os atributos primários de posicionamento da região. Eles posicionam o canto superior esquerdo da região na distância especificada a partir da margem superior esquerda da região-pai (ou margem superior esquerda do dispositivo, no caso da região no primeiro nível de aninhamento). Algumas vezes, ajustar explicitamente os atributos *bottom* e *right* pode ser útil. Seus valores estabelecem a distância entre o canto inferior direito da região e o canto inferior direito da região-pai (ou a margem inferior direita do dispositivo, no caso da região no primeiro nível de aninhamento) (ver Figura 2).

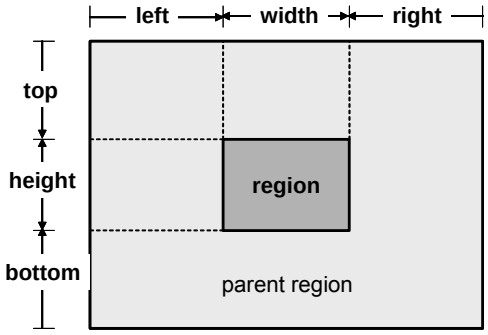

Figura 2 — Atributos de posicionamento da região

Com relação aos tamanhos da região, quando eles são especificados declarando os atributos *width* e *height* usando a notação "%", o tamanho da região é relativo ao tamanho da geometria de seu pai, como mencionado anteriormente. Os tamanhos declarados como valores absolutos em pixels mantêm tais valores absolutos. O tamanho intrínseco de uma região é igual ao tamanho da geometria lógica do pai. Isso significa que, se uma região aninhada não especificar qualquer posicionamento ou valores de tamanho, deve ser obrigatoriamente assumido que ela tem a mesma posição e valores de tamanho que sua região-pai. Em particular, quando uma região de primeiro nível não especifica qualquer posicionamento ou valores de tamanho, deve ser obrigatoriamente assumida como tendo toda a área de apresentação do dispositivo.

Quando o usuário especifica informações sobre *top*, *bottom* e *height* para uma mesma <region>, podem ocorrer inconsistências espaciais. Neste caso, os valores de *top* e *height* devem obrigatoriamente preceder o valor de *bottom*. De forma análoga, quando o usuário especifica valores inconsistentes para os atributos *left*, *right* e *width* da <region>, os valores de *left* e *width* devem obrigatoriamente ser utilizados para calcular um novo valor de *right*. Quando qualquer um desses atributos não é especificado e não pode ter seu valor calculado a partir de outros atributos, esse valor deve obrigatoriamente ser herdado do valor correspondente definido no pai dessa <region>. Outra restrição é que as regiões-filhas não podem ficar fora da área estabelecida por suas regiões-pais.

O atributo *zIndex* especifica a precedência de sobreposição da região. Regiões com maiores valores de *zIndex* devem ser obrigatoriamente empilhadas no topo de regiões com valores de *zIndex* menores. Se duas apresentações geradas pelos elementos A e B tiverem o mesmo nível de empilhamento, caso a exibição de um elemento B comece depois da exibição de um elemento A, a apresentação de B deve obrigatoriamente ser empilhada no topo da apresentação de A (ordem temporal); por outro lado, se a exibição dos elementos começar ao mesmo tempo, a ordem empilhada é escolhida arbitrariamente pelo formatador. Quando omitido, o valor default do zIndex é igual a 0 (zero).

O módulo *Layout* também define o atributo *region*, que é utilizado por um elemento <descriptor> (ver 7.2.6) para referir-se a um elemento <region> de *Layout*.

Os elementos deste módulo, seus elementos-filhos e seus atributos devem obrigatoriamente estar de acordo com a Tabela 10.

Tabela 10 — Módulo *Layout* estendido

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

7.2.4 Área funcional *Components*

A área funcional *Components* é particionada em dois módulos, chamados *Media* e *Context*.

O módulo Media define os tipos básicos de objetos de mídia. Para definir objetos de mídia, este módulo define o elemento <media>. Cada objeto de mídia tem dois atributos principais, além do atributo *id*: *src*, que define um URI do conteúdo do objeto, e *type*, que define o tipo de objeto.

Os URI (*Uniform Resource Identifier*) devem obrigatoriamente estar de acordo com a Tabela 11.

Tabela 11 — URI permitidos

Esquema	Parte específica do esquema	Uso
file:	///file_path/#fragment_identifier	Para arquivos locais
http:	//server_identifier/file_path/#fragment_identifier	Para arquivos remotos buscados pelo canal de interatividade usando o protocolo http
rstp:	//server_identifier/file_path/#fragment_identifier	Para fluxos (*streams*) obtidos pelo canal de interatividade usando o protocolo rstp
rtp:	//server_identifier/file_path/#fragment_identifier	Para fluxos (*streams*) obtidos pelo canal de interatividade usando o protocolo rtp
sbtvd-ts:	//program_id	Para fluxos elementares recebidos pelo fluxo de transporte (TS)

Um URI absoluto contém todas as informações necessárias para localizar seu recurso. Os URI relativos também são permitidos. URI relativos são endereços incompletos que são aplicados a um URI base para completar a localização. As partes omitidas são o esquema URI, o servidor e, também, em alguns casos, parte do caminho do URI.

O benefício principal de utilizar URI relativas 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 URI dentro dos documentos. Isso é especialmente interessante quando se transportam documentos do servidor (normalmente radiodifusores) 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. Eles frequentemente consistem apenas no nome do arquivo (opcionalmente com um identificador de fragmento dentro do arquivo). Eles também podem ter um caminho relativo de diretório antes do nome do arquivo.

Convém enfatizar que as referências para os recursos de fluxos de vídeo ou áudio não devem, obrigatoriamente, causar a ocorrência de sintonização (*tuning*). As referências que implicam sintonização para acessar um recurso devem obrigatoriamente se comportar como se o recurso estivesse indisponível.

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 em comum acordo; eles normalmente têm um “x-“ antes do nome do tipo de mídia.

Quatro tipos especiais são definidos: application/x-ginga-NCLua, application/x-ginga-NCLet, application/x-ginga-settings e application/x-ginga-time.

O tipo application/x-ginga-NCLua deve obrigatoriamente ser aplicado a elementos <media> com código procedural Lua como conteúdo (ver Seção 10). O tipo application/x-ginga-NCLet deve obrigatoriamente ser aplicado a elementos <media> com código procedural Xlet como conteúdo (ver Seção 11).

O tipo application/x-ginga-settings deve obrigatoriamente ser aplicado a um elemento <media> especial (pode existir somente um em um documento NCL) cujas propriedades são variáveis globais definidas pelo autor do documento ou variáveis de ambiente reservadas, que podem ser manipuladas pelo processamento do documento NCL. A Tabela 12 estabelece as variáveis já definidas e sua semântica.

Tabela 12 — Variáveis de ambiente

Grupo	Variável	Semântica	Valores possíveis
system • grupo de variáveis mantidas pelo sistema receptor; • podem ser lidas, mas não podem ter seus valores alterados por um aplicativo NCL, um procedimento Lua ou um procedimento Xlet; • programas nativos no receptor podem alterar os valores das variáveis; • persistem durante todo o tempo de vida de um receptor	system.language	Linguagem de áudio	ISO 639-1 code
	system.caption	Linguagem de caption	ISO 639-1
	system.subtitle	Linguagem de legenda	ISO 639-1
	system.returnBitRate(i)	Taxa de bits do canal de interatividade (i) em Kbps	real
	system.screenSize (i)	Tamanho da tela do dispositivo (i) em (linhas, pixels/linha)	(inteiro, inteiro)
	system.screenGraphicSize (i)	Resolução configurada para o plano gráfico da tela do dispositivo (i) em (linhas, pixels/linha)	(inteiro, inteiro)
	system.audioType(i)	Tipo de áudio do dispositivo (i)	“mono” | “stereo” | “5.1”
	system.CPU	Desempenho da CPU em MIPS	real
	system.memory	Espaço da memória em Mbytes	inteiro
	system.operatingSystem	Tipo de sistema operacional	string a ser definida
	system.javaConfiguration	Tipo e versão da configuração suportada pela JVM do receptor.	string (tipo seguido da versão, sem espaço, por exemplo: “CLDC1.1”)
	system.javaProfile	Tipo e versão do perfil suportado pela JVM do receptor	string (tipo seguido da versão, sem espaço – ex.: “MIDP2.0”
	system.luaVersion	Versão da máquina Lua do receptor	string
	system.xxx	Qualquer variável prefixada por “system” deve obrigatoriamente ser reservada para uso futuro	
user • grupo de variáveis mantidas pelo sistema receptor; • podem ser lidas, mas não podem ter seus valores alterados por um aplicativo NCL, um procedimento Lua ou um procedimento Xlet; • programas nativos no receptor podem alterar os valores das variáveis; • persistem durante todo o tempo de vida de um receptor	user.age	Idade do usuário	Inteiro
	user.location	Localização do usuário (CEP)	String
	user.genre	Gênero do usuário	“m”| “f”
	user.xxx	Qualquer variável prefixada por “user” deve obrigatoriamente ser reservada para uso futuro	

Tabela 12 (continuação)

Grupo	Variável	Semântica	Valores possíveis
default • grupo de variáveis mantidas pelo sistema receptor; • podem ser lidas e ter seus valores alterados por um aplicativo NCL ou por um procedimento Lua ou Xlet; • programas nativos no receptor podem alterar os valores das variáveis; • persistem durante todo o tempo de vida de um receptor, no entanto, voltam ao seu valor inicial quando da troca de canais	default.focusBorderColor	Cor padrão aplicada à margem de um elemento em foco	"white" | "black" | "silver" | "gray" | "red" | "maroon" | "fuchsia" | "purple" | "lime" | "green" | "yellow" | "olive" | "blue" | "navy" | "aqua" | "teal"
	default.selBorderColor	Cor padrão aplicada à margem de um elemento em foco quando ativado	"white" | "black" | "silver" | "gray" | "red" | "maroon" | "fuchsia" | "purple" | "lime" | "green" | "yellow" | "olive" | "blue" | "navy" | "aqua" | "teal"
	default.focusBorderWidth	Largura padrão (em pixels) aplicada à margem de um elemento em foco	inteiro
	default.focusTransparency	Transparência padrão aplicada à borda de um elemento em foco	valor real entre 0 e 1, ou valor real na faixa [0,100] terminando com o caractere "%" (por exemplo, 30 %), com "1", ou "100 %" significando máxima transparência e "0", ou "0 %" significando nenhuma transparência
	default.xxx	Qualquer variável prefixada por "default" deve obrigatoriamente ser reservada para uso futuro	
service • grupo de variáveis mantidas pelo formatador NCL; • podem ser lidas e ter seus valores alterados por um aplicativo NCL do mesmo serviço; • podem apenas ser lidas por um programa Lua ou Xlet do mesmo serviço; a escrita deve ser através de comandos NCL; • persistem, no mínimo, durante todo o tempo de duração de um serviço	service.currentFocus	Valor do atributo *focusIndex* do elemento <media> em foco	inteiro
	service.xxx	Qualquer variável prefixada por "service" deve obrigatoriamente seguir as regras especificadas para o grupo	
channel • grupo de variáveis mantidas pelo formatador NCL; • podem ser lidas e ter seus valores alterados por um aplicativo NCL do mesmo canal; • podem apenas ser lidas por um programa Lua ou Xlet do mesmo canal; a escrita deve ser através de comandos NCL; • persistem, no mínimo, durante todo o tempo de sintonia de um canal	channel.xxx	Qualquer variável prefixada por "channel" deve obrigatoriamente seguir as regras especificadas para o grupo	

Tabela 12 (continuação)

Grupo	Variável	Semântica	Valores possíveis
shared • grupo de variáveis mantidas pelo formatador NCL; • podem ser lidas e ter seus valores alterados por um programa NCL; • podem apenas ser lidas por um programa Lua ou Xlet; a escrita deve ser através de comandos NCL; • persistem, no mínimo, durante todo o tempo do serviço que a definiu; • não são prefixadas	shared.xxx	Qualquer variável prefixada por "shared" deve obrigatoriamente seguir as regras especificadas para o grupo	

O tipo application/x-ginga-time deve obrigatoriamente ser aplicado a um elemento <media> especial (pode existir somente um em um documento NCL) cujo conteúdo é o horário de Greenwich (GMT). Qualquer elemento <media> contínuo sem fonte pode ser utilizado para definir um relógio, relativo ao tempo de início desse elemento <media>.

A Tabela 13 mostra alguns valores possíveis do atributo *type* para os perfis TVD Avançado e Básico e as extensões de arquivos associadas. Os tipos obrigatórios são definidos na ABNT NBR 15601. O atributo *type* é opcional (exceto para os elementos <media> sem atributo *src* definido) e recomenda-se ser usado para guiar a escolha do exibidor de mídia (ferramenta de apresentação) pelo formatador. Quando o atributo *type* não é especificado, recomenda-se que o formatador use a extensão do conteúdo especificado no atributo *src* para fazer a escolha do exibidor.

Quando há mais que um exibidor para o tipo suportado pelo formatador, o elemento <descriptor> pode especificar qual será utilizado para a apresentação. Caso contrário, o formatador deve obrigatoriamente utilizar um exibidor *default* para aquele tipo de mídia.

Tabela 13 — Tipos de mídias MIME para formatadores Ginga-NCL

Tipo de mídia	Extensão de arquivo
text/html	htm, html
text/plain	txt
text/css	css
text/xml	xml
image/bmp	bmp
image/png	png
image/gif	gif
image/jpeg	jpg, jpeg
audio/basic	wav
audio/mp3	mp3
audio/mp2	mp2
audio/mpeg	mpeg, mpg
audio/mpeg4	mp4, mpg4
video/mpeg	mpeg, mpg
application/x-ginga-NCLua	lua
application/x-ginga-NCLet	class, jar
application/x-ginga-settings	*no src (source)*
application/x-ginga-time	*no src (source)*

O módulo *Context* é responsável pela definição de nós de contexto através de elementos <context>. Um nó de contexto NCM é um tipo particular de nó de composição NCM e é definido contendo um conjunto de nós e um conjunto de elos. Como normalmente ocorre, o atributo *id* identifica univocamente cada elemento <context> e <media> dentro de um documento.

Os atributos *newInstance*, *refer* e *descriptor* são extensões definidas em outros módulos e são discutidos na definição de tais módulos.

Os elementos dos dois módulos da funcionalidade *Components*, seus elementos-filhos e seus atributos devem obrigatoriamente estar de acordo com as Tabelas 14 e 15.

Tabela 14 — Módulo *Media* estendido

Elements	Attributes	Content
media	*id, src, refer, newInstance, type, descriptor*	(area|property)*

Tabela 15 — Módulo *Context* estendido

Elements	Attributes	Content
context	*id, refer*	(port|property|media|context|link|switch)*

7.2.5 Área funcional *Interfaces*

A área funcional *Interfaces* permite a definição de interfaces de nós (objetos de mídia ou nós de composição) que serão utilizadas em relacionamentos com outras interfaces de nós. Esta área funcional é particionada em quatro módulos:

— *MediaContentAnchor*, que permite definições de âncoras de conteúdo (ou área) para nós de mídia (elementos <media>);

— *CompositeNodeInterface*, que permite definições de portas para nós de composição (elementos <context> e <switch>);

— *PropertyAnchor*, que permite a definição de propriedades de nós como interfaces de nós; e

— *SwitchInterface*, que permite a definição de interfaces especiais para elementos <switch>.

O módulo *MediaContentAnchor* define o elemento <area>, que estende a sintaxe e semântica do elemento homônimo definido por SMIL e XHTML. Assim, ele permite a definição de âncoras de conteúdo representando porções espaciais, através de atributo *coords* (como em XHTML); a definição de âncoras de conteúdo representando porções temporais, através dos atributos *begin* e *end*; e a definição de âncoras de conteúdo representando porções espaço-temporais através dos atributos *coords*, *begin* e *end* (como em SMIL). Além disso, o elemento <area> permite a definição de âncoras textuais, através dos atributos *text* e *position*, que definem uma cadeia de caracteres e a ocorrência dessa cadeia no texto, respectivamente. Adicionalmente, o elemento <area> pode também definir uma âncora de conteúdo com base no número de amostras de áudio ou frames de vídeo, através dos atributos *first* e *last*, que devem obrigatoriamente indicar a amostra/frame inicial e final. Finalmente, o elemento <area> também pode definir uma âncora de conteúdo baseada no atributo *label*, que especifica uma cadeia de caracteres que deve ser utilizada pelo exibidor de mídias para identificar uma região de conteúdo.

Se o atributo *begin* for definido, mas o atributo *end* não for especificado, o final de toda apresentação do conteúdo de mídia deve obrigatoriamente ser 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 deve obrigatoriamente ser considerado como o início da âncora. Comportamento semelhante é esperado com relação aos atributos *first* e *last*. No caso de um elemento <media> do tipo application/x-ginga-time, os atributos *begin* e *end* devem obrigatoriamente ser sempre especificados e correspondem a um tempo absoluto do horário de Greenwich (GMT).

Como normalmente ocorre, os elementos <area> devem obrigatoriamente ter um atributo *id*, que identifica univocamente o elemento dentro de um documento.

O elemento <area> e seus atributos devem obrigatoriamente estar de acordo com a Tabela 16.

Tabela 16 — Módulo *MediaContentAnchor* estendido

Elementos	Atributos	Conteúdo
area	*id, coords, begin, end, text, position, first, last, label*	vazio

O módulo *CompositeNodeInterface* define o elemento <port>, que especifica uma porta de um nó de composição com seu respectivo mapeamento para uma interface (atributo *interface*) de um de seus componentes (especificado pelo atributo *component*).

No NCM, todo nó (mídia ou contexto) deve obrigatoriamente possuir uma âncora com uma região representando o conteúdo total do nó. Essa âncora é chamada de âncora de conteúdo total e é declarada por omissão (default) em NCL. Cada vez que um componente NCL é referenciado sem especificar uma de suas âncoras, deve-se obrigatoriamente assumir a âncora de conteúdo total.

O elemento <port> e seus atributos devem obrigatoriamente estar de acordo com a Tabela 17.

Tabela 17 — Módulo *CompositeNodeInterface* estendido

Elementos	Atributos	Conteúdo
port	*id, component, interface*	vazio

O módulo PropertyAnchor define um elemento chamado <property>, que pode ser utilizado para definir uma propriedade ou grupo de propriedades de um nó, como uma de suas interfaces (âncora). O elemento <property> define o atributo *name*, que indica o nome da propriedade ou grupo de propriedades, e o atributo *value*, atributo opcional que define um valor inicial para a propriedade *name*. O elemento pai não pode ter elementos <property> com os mesmos valores para o atributo *name*.

É possível ter exibidores de documentos NCL (formatadores) que definem algumas propriedades de nós e interfaces de nós, implicitamente. Entretanto, em geral, é de boa prática definir explicitamente as interfaces. Assim, todas as interfaces devem obrigatoriamente ser explicitamente definidas.

Os elementos <body>, <context> e <media> podem ter várias propriedades embutidas. Exemplos dessas propriedades podem ser encontrados entre aquelas que definem o local a ser colocado o objeto de mídia durante uma apresentação, a duração da apresentação e outras que definem características adicionais da apresentação: *top*, *left*, *bottom*, *right*, *width*, *height*, *explicitDur*, *background*, *transparency*, *visible*, *fit*, *scroll*, *style*, *soundLevel*, *balanceLevel*, *trebleLevel*, *bassLevel*, *fontColor, fontFamily, fontStyle, fontSize, fontVariant, fontWeight, reusePlayer, playerLife* etc. Tais propriedades assumem como seus valores iniciais os definidos em atributos homônimos do descritor e região associado ao nó (ver 7.2.3 e 7.2.6). Entretanto, quando uma propriedade embutida é utilizada em um relacionamento, ela deve obrigatoriamente ser explicitamente declarada como elemento <property> (interface).

Um grupo de propriedades do nó também pode ser explicitamente declarado como um elemento único <property> (interface), permitindo que os autores especifiquem o valor de várias propriedades com uma propriedade única. Os seguintes grupos devem obrigatoriamente ser reconhecidos por um formatador NCL: *location*, grouping (*left*, *top*), nessa ordem; *size*, grouping (*width*, *height*), nessa ordem; e *bounds*, grouping (left, top, *width*, *height*), nessa ordem.

Quando um formatador trata uma alteração em um grupo de propriedade, deve obrigatoriamente apenas testar a consistência do processo ao seu final. As palavras *top*, *left*, *bottom*, *right*, *width*, *height*, *explicitDur*, *background*, *transparency*, *visible*, *fit*, *scroll*, *style*, *soundLevel*, *balanceLevel*, *trebleLevel*, *bassLevel*, *fontColor, fontFamily, fontStyle, fontSize, fontVariant, fontWeight, reusePlayer, playerLife*, *location*, *size* e *bounds* são palavras reservadas para valores do mesmo atributo *name* do elemento <property>.

O elemento <property> e seus atributos devem obrigatoriamente estar de acordo com a Tabela 18.

Tabela 18 — Módulo *PropertyAnchor* estendido

Elementos	Atributos	Conteúdo
property	*name, value*	vazio

O módulo *SwitchInterface* permite a criação de interfaces de elemento <switch> (ver 7.2.4), que podem ser mapeadas a um conjunto de interfaces alternativas de nós internos, permitindo a um elo ancorar no componente escolhido quando o <switch> é processado (NCM Core:2005). Esse módulo introduz o elemento <switchPort>, que contém um conjunto de elementos de mapeamento. Um elemento de mapeamento define um caminho a partir do <switchPort> para uma interface (atributo *interface*) de um dos componentes do <switch> (especificados por seu atributo *component*).

É importante mencionar que cada elemento representando uma interface de objeto (<area>, <port>, <property> e <switchPort>) deve obrigatoriamente ter um identificador (atributo id).

O elemento <switchPort>, seus elementos-filhos e seus atributos devem obrigatoriamente estar de acordo com a Tabela 19.

Tabela 19 — Módulo *SwitchInterface* estendido

Elementos	Atributos	Conteúdo
switchPort	*id*	mapping+
mapping	*component, interface*	vazio

7.2.6 Área funcional *Presentation Specification*

A área funcional *Presentation Specification* tem um único módulo chamado *Descriptor*. O objetivo desse módulo é especificar informações espaço-temporais necessárias para a apresentação de cada componente do documento. Essas informações são modeladas pelos objetos descritores.

O módulo *Descriptor* permite a definição de elementos <descriptor>, que contêm um conjunto de atributos opcionais, agrupando todas as definições espaço-temporais a serem usadas de acordo com o tipo de objeto a ser apresentado. A definição de elementos <descriptor> deve obrigatoriamente ser incluída no cabeçalho

do documento, dentro do elemento <descriptorBase>, que especifica o conjunto de descritores de um documento. O elemento <descriptor> deve obrigatoriamente ter o atributo *id*; e o elemento <descriptorBase> pode ter o atributo *id*, que, como normalmente ocorre, identifica univocamente os elementos dentro de um documento.

Um elemento <descriptor> pode ter atributos temporais: *explicitDur* e *freeze*, definidos pelo módulo *Timing* (ver 7.2.10); um atributo chamado *player*, que identifica a ferramenta de apresentação a ser utilizada; um atributo chamado *region*, que refere-se à região definida pelos elementos do módulo *Layout* (ver 7.2.3); atributos para navegação: *moveLeft*, *moveRight*, *moveUp*; *moveDown*, *focusIndex*, *focusBorderColor*; *focusBorderWidth*; *focusBorderTransparency*, *focusSrc*, *selBorderColor*, e *focusSelSrc*, definidos pelo módulo KeyNavigation (ver 7.2.12); e atributos de transição: *transIn* e *transOut* (ver 7.2.14).

Um elemento <descriptor> pode também ter elementos <descriptorParam> como elementos filho, que são utilizados para parametrizar o controle da apresentação do objeto associado com o elemento descritor. Esses parâmetros podem, por exemplo, redefinir alguns valores de atributos definidos pelos atributos da região. Eles também podem definir novos atributos, tais como *background*, especificando a cor de fundo utilizada para preencher a área de uma região de exibição da mídia, quando toda a região não é preenchida pela mídia em si; *visible*, permitindo que a apresentação do objeto seja visualizada ou ocultada; *fit*, indicando como um objeto será apresentado; *scroll*, que permite a especificação de como um autor gostaria de configurar a rolagem em uma região; *transparency*, indicando o grau de transparência da apresentação de um objeto; *style*, que refere-se a uma folha de estilo [Cascading Style Sheets, 1998] com informações, por exemplo, para apresentação do texto; além de especificar atributos para objetos de áudio, tais como *soundLevel*, *balanceLevel*, *trebleLevel* e *bassLevel*. Além disso, os elementos-filhos <descriptorParam> podem determinar se um novo exibidor deve obrigatoriamente ser instanciado, ou se um exibidor já instanciado deve obrigatoriamente ser utilizado (*reusePlayer*), e especificar o que acontecerá à instância do exibidor no final da apresentação (*playerLife*). As palavras top, left, bottom, right, width, height, explicitDur, location, size, bounds, background, visible, fit, scroll, style, soundLevel, balanceLevel, trebleLevel, bassLevel, reusePlayer e playerLife são palavras reservadas para valores do atributo *name* do elemento <descriptorParam>. Os valores possíveis para os nomes reservados de parâmetros/atributos devem obrigatoriamente estar de acordo com a Tabela 20.

Tabela 20 — Parâmetros/atributos reservados e valores possíveis

Nome do parâmetro/atributo	Valor
top, left, bottom, right, width, height	Número real na faixa [0,100] terminando com o caractere "%" (por exemplo, 30 %), ou um valor inteiro especificando o atributo em pixels (no caso de *weight* e *height*, um inteiro não negativo)
location	Dois números separados por vírgula, cada um seguindo as regras de valor especificadas para parâmetros *left* e *top*, respectivamente
size	Dois valores separados por vírgula. Cada valor deve obrigatoriamente seguir as mesmas regras especificadas para parâmetros de *width* e *height*, respectivamente
bounds	Quatro valores separados por vírgula. Cada valor deve obrigatoriamente seguir as mesmas regras especificadas para parâmetros *left*, *top*, *width* e *height*, respectivamente
background	Nomes de cores reservadas: "white", "black", "silver", "gray", red", "maroon", fuchsia", "purple", "lime", "green", "yellow", "olive", "blue", "navy", "aqua", ou "teal". Uma outra opção para especificar o valor da cor é especificada na ABNT NBR 15606-1. O valor da cor de fundo pode também ter valor reservado "transparent". Isso pode ser útil para apresentar imagens transparentes, como GIF transparentes, sobrepostos sobre outras imagens ou vídeos. Quando não especificado, o atributo de cor de fundo toma o valor *default* "transparent"
visible	"true" or "false"
transparency	Um número real na faixa [0,1], ou um número real na faixa [0,100] terminando com o caractere "%" (por exemplo, 30 %), especificando o grau de transparência de uma apresentação de objeto ("1" ou "100 %" significa transparência total e "0" ou "0 %" significa opaco)
fit	"fill", "hidden", "meet", "meetBest", "slice". "fill": redimensiona o conteúdo do objeto de mídia para que toque todas as bordas da caixa definida pelos atributos de largura e altura do objeto. "hidden": se a altura/largura intrínseca do conteúdo de mídia for menor que o atributo de altura/largura, o objeto deve obrigatoriamente ser criado iniciando da margem superior/esquerda e ter a altura/largura remanescente preenchida com a cor de fundo; se a altura/largura intrínseca do conteúdo de mídia for maior do que o atributo altura/largura, o objeto deve obrigatoriamente ser criado iniciando pela margem superior/esquerda até que a altura/largura definida no atributo seja alcançada, e ter a parte do conteúdo de mídia abaixo/a direita da altura/largura cortada. "meet": redimensiona o objeto de mídia visual enquanto preserva sua relação de aspecto até que sua altura ou largura seja igual ao valor especificado pelos atributos height ou width. O canto superior esquerdo do conteúdo de mídia é posicionado nas coordenadas superiores esquerdas da caixa, o espaço vazio à direita ou na base deve obrigatoriamente ser preenchido com a cor de pano de fundo. "meetBest": a semântica é identica à do "meet", exceto que a imagem não é redimensionada em mais de 100 % em qualquer dimensão. "slice": redimensiona o conteúo de mídia visual enquanto preserva sua relação de aspecto, até que sua altura ou largura seja igual ao valor especificado nos atributos de altura e largura, e que a caixa de apresentação definida esteja completamente preenchida. Algumas partes do conteúdo podem ser cortadas. O excesso de largura é cortado a partir da direita do objeto de mídia. O excesso de altura é cortado a partir da base do objeto de mídia
scroll	"none", "horizontal", "vertical", "both", ou "automatic"
style	Localizador de um arquivo de folha de estilo
soundLevel, balanceLevel, trebleLevel, bassLevel	Um número real na faixa [0, 1], ou um número real na faixa [0,100] terminando com o caractere "%" (por exemplo, 30 %)
zIndex	Um número inteiro na faixa [0, 255], sendo que regiões com maior valor de *zIndex* são posicionadas sobre regiões com menor valor de *zIndex*
fontFamily	Uma lista priorizada de nomes de família de fonts e/ou nomes genéricos de famílias
fontStyle	Estilo da fonte ("normal",ou "italic")
fontSize	Tamanho da fonte
fontVariant	Forma de exibição do texto: fonte em "small-caps" ou "normal"
fontWeight	Peso da fonte ("normal", ou "bold")
fontColor	Cor da fonte ("white", "black", "silver", "gray", red", "maroon", fuchsia", "purple", "lime", "green", "yellow", "olive", "blue", "navy", "aqua", ou "teal")
reusePlayer	Valor booleano: "false", "true". Valor default = "false"
playerLife	"keep", "close". Valor default = "close"

Além de todos os atributos mencionados, o elemento <descriptor> também pode ter atributos definidos na área funcional *Transition Effects* de SMIL (ver 7.2.14).

NOTA Se forem especificadols vários valores para um mesmo atributo, o valor definido no elemento <property> tem precedência sobre o valor definido em um elemento <descriptorParam>, que, por sua vez, tem precedência sobre o valor definido em um atributo do elemento <descriptor> (incluindo o atributo *region*).

Além do elemento <descriptor>, o módulo *Descriptor* define um atributo homônimo, que refere-se a um elemento do conjunto de descritores do documento. Quando um perfil de linguagem utiliza o módulo *Descriptor*, ele deve obrigatoriamente determinar como os descritores estarão associados com os componentes do documento. Seguindo as diretivas NCM, esta Norma estabelece que o atributo *descriptor* está associado com qualquer nó de mídia através de elementos <media> e através das extremidades dos elos (elementos <bind>) (ver 8.2.1).

O conjunto de descritores de um documento pode conter elementos <descriptor> ou elementos <descriptorSwitch>, que permitem especificar descritores alternativos (ver 7.2.9).

Os elementos do módulo *Descriptor*, seus elementos-filhos e seus atributos devem obrigatoriamente estar de acordo com a Tabela 21.

Tabela 21 — Módulo *Descriptor* estendido

Elementos	**Atributos**	**Conteúdo**
descriptor	*id, player, explicitDur, region, freeze, moveLeft, moveRight, moveUp, moveDown, focusIndex, focusBorderColor, focusBorderWidth, focusBorderTransparency, focusSrc,focusSelSrc, selBorderColor, transIn, transOut*	(descriptorParam)*
descriptorParam	*name, value*	vazio
descriptorBase	*id*	(importBase|descriptor|descriptorSwitch)+

7.2.7 Área funcional *Linking*

A área funcional *Linking* define o módulo *Linking*, responsável por definir os elos, que utilizam conectores. Um elemento <link> pode ter um atributo *id*, que identifica univocamente o elemento dentro de um documento e deve obrigatoriamente ter um atributo *xconnector*, que refere-se ao URI de um conector hipermídia. A referência deve obrigatoriamente ter o formato *alias#connector_id*, ou *documentURI_value#connector_id*, para conectores definidos em um documento externo (ver 7.2.11), ou simplesmente *connector_id*, para conectores definidos no próprio documento.

O elemento <link> contém elementos-filhos chamados <bind>, que permitem associar nós a papéis (*roles*) do conector (ver 7.2.8). Para fazer esta associação, um elemento <bind> tem quatro atributos básicos. O primeiro é chamado *role*, que é usado para fazer referência a um papel do conector. O segundo é chamado *component*, que é usado para identificar o nó. O terceiro é um atributo opcional chamado *interface*, usado para fazer referência a uma interface do nó. O quarto é um atributo opcional chamado *descriptor*, usado para fazer referência a um descritor a ser associado com o nó, conforme definido pelo módulo *Descriptor* (ver 7.2.6).

NOTA O atributo *interface* pode referir-se a qualquer interface do nó, isto é, uma âncora, uma propriedade ou uma porta, se for um nó de composição. O atributo *interface* é opcional. Quando não é especificado, a associação é feita com todo o conteúdo do nó (ver 7.2.5).

Se o elemento conector definir parâmetros (ver 7.2.8), convém aos elementos <bind> ou <link> definirem valores para esses parâmetros, através de seus elementos-filhos chamados <bindParam> e <linkParam>, respectivamente, ambos com atributos *name* e *value*. Nesse caso, o atributo *name* deve obrigatoriamente fazer referência ao nome de um parâmetro do conector, enquanto o atributo *value* deve obrigatoriamente definir um valor a ser atribuído ao respectivo parâmetro.

Os elementos do módulo *Linking*, seus atributos e seus elementos-filhos devem obrigatoriamente estar de acordo com a Tabela 22.

Tabela 22 — Módulo *Linking* estendido

Elementos	Atributos	Conteúdo
bind	*role, component, interface, descriptor*	(bindParam)*
bindParam	*name, value*	vazio
linkParam	*name, value*	vazio
link	*id, xconnector*	(linkParam*, bind+)

7.2.8 Área funcional *Connectors*

A área funcional *Connectors* da NCL 3.0 é particionada em oito módulos básicos: ConnectorCommonPart, ConnectorAssessmentExpression, ConnectorCausalExpression, ConnectorTransitionAssessment, CausalConnector, ConstraintConnector (não considerado nesta Norma), ConnectorBase e CompositeConnector (também não considerado nesta Norma).

Os módulos da área funcional *Connectors* são totalmente independentes dos outros módulos NCL. Esses módulos formam o núcleo de uma linguagem nova de aplicação XML (de fato, outros perfis NCL 3.0) para a definição de conectores, que podem ser utilizados para especificar relações de sincronização espaço-temporais, tratando relações de referência (de interação com usuário) como um caso particular de relações de sincronização temporal.

Além dos módulos básicos, a área funcional *Connectors* também define módulos que agrupam conjuntos de módulos básicos, para facilitar a definição do perfil de linguagem. Esse é o caso do módulo CausalConnectorFunctionality, utilizado na definição dos perfis EDTV, BDTV e CausalConnector. O módulo CausalConnectorFunctionality agrupa os seguintes módulos: ConnectorCommonPart, ConnectorAssessmentExpression, ConnectorCausalExpression, ConnectorTransitionAssessment e CausalConnector.

Um elemento <causalConnector> representa uma relação causal que pode ser utilizada por elementos <link> em documentos. Em uma relação causal, uma condição deve obrigatoriamente ser satisfeita para disparar uma ação.

Um <causalConnector> especifica uma relação independentemente dos relacionamentos, isto é, ele não especifica quais nós (representados por elementos <media>, <context>, <body> e <switch>) interagirem através da relação. Um elemento <link>, por sua vez, representa um relacionamento, do tipo definido por seu conector, interligando diferentes nós. Os elos representando o mesmo tipo de relação, mas interligando diferentes nós, podem reutilizar o mesmo conector, reutilizando todas as especificações. Um <causalConnector> especifica, através de seus elementos-filhos, um conjunto de pontos da interface, chamados papéis. Um elemento <link> refere-se a um <causalConnector> e define um conjunto de mapeamentos (elementos <bind> filhos do elemento <link>), que associam cada extremidade do elo (interface de nó) a um papel do conector utilizado.

As relações em NCL são baseadas em eventos. Um evento é uma ocorrência no tempo que pode ser instantânea ou ter duração mensurável. A NCL 3.0 define os seguintes tipos de eventos:

— evento de apresentação, que é definido pela apresentação de um subconjunto das unidades de informação de um objeto de mídia, especificado na NCL pelo elemento <area>, ou pelo nó de mídia em si (apresentação de todo o conteúdo). Os 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, que é definido pela seleção de um subconjunto das unidades de informação de um objeto de mídia, especificado na NCL pelo elemento <area>, ou pelo próprio nó de mídia (apresentação do conteúdo total);

— evento de atribuição, que é definido pela atribuição de um valor a uma propriedade de um nó (representado por um elemento <media>, <body>, <context> ou <switch>), que deve obrigatoriamente ser declarado em um elemento <property>, filho do nó; e

— evento de composição, que é definido pela 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).

Cada evento define uma máquina de estados que recomenda-se ser controlada pelo formatador NCL, como demonstrado na Figura 3. Além disso, todo evento tem um atributo associado, denominado *occurrences*, que conta quantas vezes o evento vai do estado ocorrendo (*occurring*) ao estado preparado (*sleeping*), durante a apresentação de um documento. Eventos como apresentação e atribuição também têm um atributo denominado *repetitions*, que conta quantas vezes o evento deve obrigatoriamente ser reiniciado (transitou do estado ocorrendo para o preparado) pelo formatador. Esse atributo pode conter o valor “indefinite”, levando a uma repetição infinita (loop) das ocorrências do evento até que ocorra alguma interrupção external.

Os nomes de transição para a máquina de estado de evento devem obrigatoriamente estar de acordo com a Tabela 23.

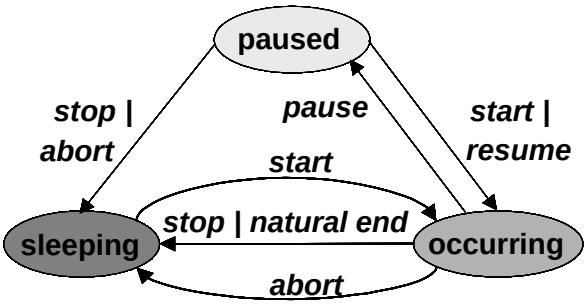

Figura 3 — Máquina de estado de evento

Tabela 23 — Nomes das transições para uma máquina de estado de evento

Transição (causada por ação)	Nome da transição
sleeping → occurring (start)	*starts*
occurring → sleeping (stop or natural end)	*stops*
occurring → sleeping (abort)	*aborts*
occurring → paused (pause)	*pauses*
paused → occurring (resume or start)	*resumes*

paused → sleeping (stop)	*stops*
paused → sleeping (abort)	*aborts*

Um evento de apresentação associado com um nó de mídia, representado por um elemento <media>, inicia no estado preparado. No início da exibição de suas unidades de informação, o evento vai para o estado ocorrendo. Se a exibição for temporariamente suspensa, o evento permanece no estado pausado (*paused*), enquanto durar essa situação.

Um evento de apresentação pode mudar de ocorrendo para preparado como conseqüência do término natural da duração da apresentação, ou devido a uma ação que termina o evento. Em ambos os casos, o atributo *occurrences* é incrementado, e o atributo *repetitions* é diminuído de um. Se, após diminuído, o valor do atributo *repetitions* for maior que zero, o evento é automaticamente reiniciado (volta novamente para o estado ocorrendo).

Quando a apresentação de um evento é interrompida bruscamente, através de um comando para abortar a apresentação, o evento também vai para o estado preparado, mas sem incrementar o atributo *occurrences* e atualizando o valor do atributo *repetitions* para zero. A duração de um evento é o tempo que ele permanece no estado ocorrendo. Tal duração pode ser intrínseca ao objeto de mídia, explicitamente especificada por um autor (atributo *explicitDur* de um elemento <descriptor>), ou derivado de um relacionamento.

Um evento de apresentação associado com um nó de composição representado por um elemento <body> ou <context> 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 representado por um elemento <body> ou <context> 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 de composição estiverem no estado preparado ou pausado. Do contrário, o evento de apresentação está no estado preparado.

NOTA Outros detalhes sobre o comportamento das máquinas de estado de evento de apresentação para nós de mídia e de composição são fornecidos na Seção 8.

Um evento de apresentação associado com um nó *switch*, representado por um elemento <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 preparado.

Um evento de seleção é iniciado no estado preparado. Ele permanence no estado ocorrendo enquanto a âncora correspondente (subconjunto das unidades de informação do objeto de mídia) estiver sendo selecionada.

Os eventos de atribuição permanecem no estado ocorrendo enquanto os valores de propriedade correspondentes estiverem sendo modificados. Obviamente, eventos instantâneos, como eventos para simples atribuição de valor, permanecem no estado ocorrendo apenas durante um período infinitesimal de tempo.

Um evento de composição (associado a um nó de composição representado por um elemento <body>, <context> ou <switch>) permanece no estado ocorrendo enquanto o mapa da composição estiver sendo apresentado.

As relações são definidas com base nos estados dos eventos, nas alterações sobre as máquinas de estado de evento, sobre os valores de atributos dos eventos, e sobre os valores de propriedade dos nós (elemento <media>, <body>, <context> ou <switch>). O módulo *CausalConnectorFunctionality* permite apenas a definição de relações causais, definidas pelo elemento <causalConnector> do módulo *CausalConnector*.

Um elemento <causalConnector> tem uma expressão de cola (*glue expression*), que define uma expressão de condição e uma de ação. Quando a expressão de condição é satisfeita, a expressão de ação deve obrigatoriamente ser executada. O elemento <causalConnector> deve obrigatoriamente ter um atributo *id*, que identifica unicamente o elemento dentro de um documento.

Uma expressão de condição pode ser simples (elemento <simpleCondition>) ou composta (elemento <compoundCondition>), ambos elementos definidos pelo módulo *ConnectorCausalExpression*.

O elemento <simpleCondition> tem um atributo *role*, cujo valor deve obrigatoriamente ser único no conjunto de *roles* do conector. Um *role* (papel) é um ponto de interface do conector, que pode ser associado a interfaces de nós por um elo que referencia o conector. Um elemento <simpleCondition> também define um tipo de evento (atributo *eventType*) e à qual transição a condição se refere (atributo *transition*). Os atributos *eventType* e *transition* são opcionais. Eles podem ser inferidos pelo valor do atributo *role* se forem utilizados valores reservados. Do contrário, atributos *eventType* e *transition* são obrigatórios.

Os valores reservados utilizados para definir os *roles* em <simpleCondition> são estabelecidos na Tabela 24. Se um valor de *eventType* for “selection”, o *role* pode também definir sobre qual dispositivo a seleção se refere (por exemplo, teclas de um teclado ou controle remoto), através do seu atributo *key*. Pelo menos os seguintes valores (que são sensíveis ao tipo maiúscula ou minúscula) devem obrigatoriamente ser aceitos pelo atributo *key*: “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”.

Tabela 24 — Valores reservados para especificação da condição do role associados às máquinas de estado de evento

Valor de *role*	Valor da transição	Tipo de evento
onBegin	*starts*	*presentation*
onEnd	*stops*	*presentation*
onAbort	*aborts*	*presentation*
onPause	*pauses*	*presentation*
onResume	*resumes*	*presentation*
onSelection	*stops*	*selection*
onBeginAttribution	*starts*	*attribution*
onEndAttribution	*stops*	*attribution*

A cardinalidade do *role* especifica o número mínimo (atributo *min*) e máximo (atributo *max*) dos participantes que podem exercer o *papel* (número de *binds*) quando o <causalConnector> é utilizado para criar um <link>. O valor mínimo da cardinalidade deve obrigatoriamente ser sempre um valor finito positivo, maior que zero e menor ou igual ao valor máximo da cardinalidade. Se a cardinalidade mínima e a máxima não forem informadas, “1” deve obrigatoriamente ser assumido como valor (default) para ambos os parâmetros. Quando o valor máximo de cardinalidade for maior que um, vários participantes podem executar o mesmo *papel (role)*, isto é, pode haver várias *ligações* (*binds*) conectando diversos nós ao mesmo *papel*. O valor “unbounded” pode ser dado ao atributo *max*, se o *role* puder ter *binds* ilimitados associados a ele. Nos dois últimos casos, convém que um atributo *qualifier* seja especificado para informar a relação lógica entre os *binds* de condição simples. Como descrito na Tabela 25, os valores possíveis para o atributo *qualifier* são: “or” (ou) ou “and” (e). Se o qualificador (atributo *qualifier*) estabelecer um operador lógico “or”, o elo será acionado quando da ocorrência de qualquer condição.

Se o qualificador estabelecer um operador lógico “and”, o elo será acionado após a ocorrência de todas as condições simples. Se não especificado, deve-se obrigatoriamente assumir o valor (default) “or”.

Tabela 25 — Valores do qualificador para condições simples

Elemento role	Qualificador	Semântica
<simpleCondition>	“or”	Verdadeiro sempre que ocorre qualquer condição simples associada
<simpleCondition>	“and”	Verdadeiro imediatamente após a ocorrência de todas as condições simples associadas

Um atributo de retardo (*delay*) pode também ser definido para uma <simpleCondition>, especificando que a condição é verdadeira após um período de retardo a partir do momento em que a transição ocorre.

O elemento <compoundCondition> tem um atributo *operator* com o valor Booleano (“and” ou “or”) relacionando seus elementos filhos: <simpleCondition>, <compoundCondition>, <assessmentStatement> e <compoundStatement>. Um atributo *delay* pode também ser definido, especificando que a condição composta é verdadeira depois que um tempo de retardo do momento em que a expressão, relacionada a seus elementos filhos, for verdadeira. Os elementos <assessmentStatement> e <compoundStatement> são definidos pelo módulo *ConnectorAssessmentExpression*.

NOTA Quando uma condição “and” composta relaciona-se a mais de uma condição de disparo (isto é, uma condição somente satisfeita em um instante de tempo infinitesimal – como, por exemplo, o final da apresentação de um objeto), a condição composta deve obrigatoriamente ser considerada verdadeira no instante imediatamente após a satisfação de todas as condições de disparo.

Uma expressão de ação capta ações que podem ser executadas em relações causais, podendo ser compostas de um elemento <simpleAction> ou <compoundAction>, também definido pelo módulo *ConnectorCausalExpression*.

O elemento <simpleAction> tem um atributo *role*, que deve obrigatoriamente ser único no conjunto de *papéis* (*roles)* do conector. Como sempre, um papel é um ponto de interface do conector, que está associado às interfaces de nós por um <link> que se refere ao conector. Um elemento <simpleAction> também define um tipo de evento (atributo *eventType*) e qual transição de estado de evento ele dispara (*actionType*).

Os atributos *eventType* e *actionType* são opcionais. Eles podem ser inferidos pelo valor de *role*, se forem utilizados valores reservados. Do contrário, atributos *eventType* e *actionType* são obrigatórios. Os valores reservados utilizados para definir os *roles* de um elemento <simpleAction> são estabelecidos na Tabela 26.

Tabela 26 — Valores de role de ação reservados associados às máquinas de estado de evento

Role value	***Action type***	***Event type***
start	*start*	*presentation*
stop	*stop*	*presentation*
abort	*abort*	*presentation*
pause	*pause*	*presentation*
resume	*resume*	*presentation*
set	*set*	*attribution*

Se um valor *eventType* for “attribution”, o elemento <simpleAction> também deve obrigatoriamente definir o valor a ser atribuído, através de seu atributo *value*. Se esse valor for especificado como “&got”, o valor a ser atribuído

deve ser obtido da propriedade ligada ao elemento <simpleCondition> cujo valor de seu atributo *role* é “onBeginAttribution” ou “onEndAttribution”. Se esse valor não puder ser obtido, nenhuma atribuição deve ser realizada.

Como no caso dos elementos <simpleCondition>, a cardinalidade do *role* especifica o número mínimo (atributo *min*) e máximo (atributo *max*) dos participantes de um *role* (número de *binds*), quando o <causalConnector> é utilizado para criar um elo. Quando o valor máximo da cardinalidade é maior que um, vários participantes podem participar de um mesmo role. Quando tem valor “unbounded”, o número de *binds* é ilimitado. Nos últimos dois casos, um qualificador deve obrigatoriamente ser especificado. A Tabela 27 apresenta os possíveis valores de qualificador.

Tabela 27 — Valores do qualificador de ação

Elemento role	Qualificador	Semântica
<simpleAction>	“par”	Todas as ações devem obrigatoriamente ser executadas em paralelo
<simpleAction>	“seq”	Todas as ações devem obrigatoriamente ser executadas na seqüência das declarações dada pelo *bind*

Um atributo *delay* pode também ser definido por um <simpleAction>, especificando, quando definido, que a ação deve obrigatoriamente ser disparada apenas após esperar pelo tempo especificado. Além disso, o <simpleAction> também pode definir um atributo *repeat* para ser aplicado ao atributo *repetitions* do evento, e um atributo *repeatDelay*, para ser aguardado antes da repetição da ação.

Além de todos os atributos acima citados, o elemento <simpleAction> também pode ter atributos definidos na área funcional *Animation* (atributos *duration* e *by*), se seu valor de *eventType* for “attribution” (ver 7.2.13).

O elemento <compoundAction> tem um atributo *operator* (“par” ou “seq”) relacionando seus elementos-filhos: <simpleAction> e <compoundAction>. Ações compostas paralelas ("par") e sequenciais ("seq") especificam que a execução das ações deve obrigatoriamente ser feita em qualquer ordem ou em uma ordem específica, respectivamente. Um atributo de *delay* pode também ser definido, especificando que a ação composta deve obrigatoriamente ser aplicada após o retardo especificado.

Quando o operador sequencial é utilizado, as ações devem obrigatoriamente ser iniciadas na ordem especificada. Entretanto, uma ação não precisa esperar até que a anterior seja completada para então ser iniciada.

O módulo ConnectorAssessmentExpression define quatro elementos: <assessmentStatement>, <attributeAssessment>, <valueAssessment> e <compoundStatement>.

O <attributeAssessment> tem um atributo *role*, que deve obrigatoriamente ser único no conjunto de *roles* do conector. Como normalmente ocorre, um *role* é um ponto de interface do conector, que é associado às interfaces dos nós por um <link> que referencia o conector. Um <attributeAssessment> também define um tipo de evento (atributo *eventType*).

Se o valor de *eventType* for “selection” (seleção), convém ao <attibuteAssessment> também definir sobre qual equipamento a seleção se refere (por exemplo, teclas de um teclado ou controle remoto), através do seu atributo *key*. Se o valor *eventType* for “presentation”, o atributo *attributeType* especifica o atributo de evento (“occurrences” ou “repetitions”) ou o estado do evento (“state”); se o valor *eventType* for “selection”, o atributo *attributeType* é opcional e, se presente, pode ter o valor “occurrences” (default) ou “state”; se o *eventType* for “attribution” o *attributeType* é opcional e pode ter o valor “nodeProperty” (default), “occurrences”, “repetition” ou “state”. No primeiro caso, o evento representa uma propriedade do nó a ser avaliada, nos outros, o evento representa a avaliação da propriedade de evento de atribuição correspondente ou o estado do evento de atribuição. Um valor de compensação (*offset*) pode ser adicionado a um <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, ou qualquer valor a ser comparado com uma propriedade do nó ou atributo de evento.

O elemento <assessmentStatement> tem um atributo *comparator* que compara os valores inferidos a partir dos seus elementos-filhos (elementos <attributeAssessment> e <valueAssessment>):

a) no caso de <attibuteAssessment>: um valor de propriedade do nó [*eventType* = “attribution” e o *attributeType* = “nodeProperty”]; ou um valor de atributo de evento [*eventType* = (“presentation”, “attribution” ou “selection”) e o *attributeType* = (“occurrences” ou “repetition”)]; ou um estado de evento [*eventType* = (“presentation”, “attribution” ou “selection”) e o *attributeType* = “state”];

b) no caso do <valueAssessment>: um valor de seu atributo *value*.

O elemento <compoundStatement> tem um atributo *operator* com um valor Booleano (“and” ou “or”) relacionando seus elementos-filhos: <assessmentStatement> ou <compoundStatement>. Um atributo *isNegated* também pode ser definido para especificar se o elemento-filho do <compoundStatement> deve obrigatoriamente ser negado antes que a operação Booleana seja avaliada.

O elemento <causalConnector> pode ter elementos-filhos (elementos <connectorParam>), que são utilizados para parametrizar valores dos atributos dos conectores. O módulo *ConnectorCommonPart* define o tipo de elemento <connectorParam>, que tem atributos *name* e *type*.

Para especificar quais atributos recebem valores de parâmetro definidos pelo conector, seus valores são especificados como o nome do parâmetro, precedido pelo símbolo $.

EXEMPLO Para parametrizar o atributo *delay*, um parâmetro chamado *actionDelay* é definido (<connectorParam name="actionDelay" type=”unsignedLong”/>) e o valor “$actionDelay” é utilizado no atributo (delay=”$actionDelay”).

Os elementos do módulo CausalConnectorFunctionality, seus elementos-filhos e seus atributos devem obrigatoriamente estar de acordo com a Tabela 28.

Tabela 28 — Módulo *CausalConnectorFunctionality* estendido

Elementos	Atributos	Conteúdo
causalConnector	*id*	(connectorParam*, (simpleCondition | compoundCondition), (simpleAction | compoundAction))
connectorParam	*name, type*	vazio
simpleCondition	*role, delay, eventType, key, transition, min, max, qualifier*	vazio
compoundCondition	*operator, delay*	((simpleCondition | compoundCondition)+, (assessmentStatement | compoundStatement)*)
simpleAction	*role, delay, eventType, actionType, value, min, max, qualifier, repeat, repeatDelay, duration, by*	vazio
compoundAction	*operator, delay*	(simpleAction | compoundAction)+
assessmentStatement	*comparator*	(attributeAssessment, (attributeAssessment |

		valueAssessment))
attributeAssessment	*role*, *eventType*, *key*, *attributeType*, *offset*	vazio
valueAssessment	*value*	vazio
compoundStatement	*operator*, *isNegated*	(assessmentStatement | compoundStatement)+

O módulo *ConnectorBase* define um elemento chamado <connectorBase>, que permite o agrupamento de conectores. Como normalmente ocorre, recomenda-se que o elemento <connectorBase> tenha um atributo *id*, que identifica unicamente o elemento dentro de um documento.

O conteúdo exato de uma base de conectores é especificado por um perfil de linguagem que utiliza as facilidades oferecidas pelos conectores. Entretanto, como a definição de conectores não é facilmente realizada por usuários inexperientes, a idéia é ter usuários experientes definindo os conectores, armazenando-os em bibliotecas (bases de connectores) que possam ser importadas, tornando-as disponíveis a outros usuários para a criação de elos. O Anexo C fornece um exemplo de definições de conectores que podem ser importadas.

Os elementos do módulo *ConnectorBase*, seus elementos-filhos, e seus atributos devem obrigatoriamente estar de acordo com a Tabela 29.

Tabela 29 — Módulo *ConnectorBase* estendido

Elementos	Atributos	Conteúdo
connectorBase	*id*	(importBase|causalConnector)*

7.2.9 Área funcional *Presentation Control*

O objetivo da área funcional *Presentation Control* é especificar alternativas de conteúdo e apresentação para um documento. Essa área functional é particionada em quatro módulos, chamados *TestRule*, *TestRuleUse*, *ContentControl* e *DescriptorControl*.

O módulo *TestRule* permite a definição de regras que, quando satisfeitas, selecionam alternativas para a apresentação do documento. A especificação de regras em NCL 3.0 é feita em um módulo separado, porque são úteis para definir tanto componentes quanto descritores alternativos.

O elemento <ruleBase> especifica um conjunto de regras e deve obrigatoriamente ser definido como elemento filho do elemento <head>. Essas regras podem ser simples, definidas pelo elemento <rule>, ou compostas, definidas pelo elemento <compositeRule>. As regras simples definem um identificador (atributo *id*), uma variável (atributo *var*), um valor (atributo *value*) e um comparador (atributo *comparator*) relacionando a variável a um valor.

A variável deve obrigatoriamente ser uma propriedade do nó *settings* (elemento <media> do tipo application/x-ginga-settings), ou seja, o atributo *var* deve obrigatoriamente possuir o mesmo valor do atributo *name* de um elemento <property>, definido como filho do elemento <media> do tipo application/x-ginga-settings. Regras compostas têm um identificador (atributo *id*) e um operador Booleano (“and” ou “or” – atributo *operator*) relacionando suas regras-filhas. Como normalmente ocorre, o atributo *id* identifica unicamente os elementos <rule> e <compositeRule> dentro de um documento.

Os elementos do módulo *TestRule*, seus elementos-filhos e seus atributos devem obrigatoriamente estar de acordo com a Tabela 30.

Tabela 30 — Módulo TestRule estendido

Elementos	Atributos	Conteúdo

ruleBase	*id*	(importBase|rule|compositeRule)+
Rule	*id*, *var*, *comparator*, *value*	vazio
compositeRule	*id*, *operator*	(rule | compositeRule)+

O *TestRuleUse* define o elemento <bindRule>, que é utilizado para associar regras com componentes de um elemento <switch> ou <descriptorSwitch>, através de seus atributos *rule* e *constituent*, respectivamente.

O elemento do módulo *TestRuleUse* e seus atributos devem obrigatoriamente estar de acordo com a Tabela 31.

Tabela 31 — Módulo TestRuleUse estendido

Elementos	Atributos	Conteúdo
bindRule	*constituent*, *rule*	vazio

O módulo *ContentControl* especifica o elemento <switch>, permitindo a definição de nós alternativos a serem escolhidos em tempo de apresentação do documento. As regras de teste utilizadas para escolher o componente do switch a ser apresentado são definidas pelo módulo *TestRule*, ou são regras de teste especificamente definidas e embutidas em uma implementação do formatador NCL. O módulo *ContentControl* também define o elemento <defaultComponent>, cujo atributo *component* (do tipo IDREF) identifica o elemento (*default*) que deve obrigatoriamente ser selecionado se nenhuma das regras *bindRule* for avaliada como verdadeira.

Para permitir a definição de elos que se ligam a âncoras do componente escolhido depois da avaliação das regras de um *switch*, recomenda-se que um perfil da linguagem também inclua o módulo *SwitchInterface*, que permite a definição de interfaces especiais, chamadas <switchPort>.

Os elementos <switch> devem obrigatoriamente ter um atributo *id*, que identifica unicamente o elemento dentro de um documento. O atributo *refer* é uma extensão definida no módulo *Reuse* (ver 7.2.11).

Quando um <context> é definido como elemento-filho de um <switch>, os elementos <link> recursivamente contidos no elemento <context> devem obrigatoriamente ser considerados por um exibidor NCL apenas se <context> for selecionado após a avaliação do switch. Caso contrário, recomenda-se que os elementos <link> sejam considerados desabilitados e obrigatoriamente não devem interferir na apresentação do documento.

Os elementos do módulo *ContentControl*, seus elementos-filhos e seus atributos devem obrigatoriamente estar de acordo com a Tabela 32.

Tabela 32 — Módulo *ContentControl* estendido

Elementos	Atributos	Conteúdo
switch	*id, refer*	defaultComponent?, (switchPort | bindRule | media | context | switch)*)
defaultComponent	*component*	vazio

O módulo *DescriptorControl* especifica o elemento <descriptorSwitch>, que contém um conjunto de descritores alternativos a ser associado a um objeto. Os elementos <descriptorSwitch> devem obrigatoriamente ter um

atributo *id*, que identifica unicamente o elemento dentro de um documento. Analogamente ao elemento <switch>, a escolha <descriptorSwitch> é feita em tempo de apresentação, utilizando regras de teste definidas pelo módulo *TestRule*, ou regras de teste especificamente definidas e embutidas em uma implementação do formatador NCL. O módulo *DescriptorControl* também define o elemento <defaultDescriptor>, cujo atributo *descriptor* (do tipo IDREF) identifica o elemento (default) que deve obrigatoriamente ser selecionado se nenhuma das regras *bindRule* for avaliada como verdadeira.

Os elementos do módulo *DescriptorControl*, seus elementos-filhos e seus atributos devem obrigatoriamente estar de acordo com a Tabela 33.

Tabela 33 — Módulo *DescriptorControl* estendido

Elementos	Atributos	Conteúdo
descriptorSwitch	*id*	(defaultDescriptor?, (bindRule | descriptor)*)
defaultDescriptor	*descriptor*	vazio

Durante a apresentação de um documento, a partir do momento em que um <switch> é avaliado, ele é considerado resolvido até o final de sua apresentação, ou seja, enquanto seu evento de apresentação correspondente está no estado ocorrendo ou pausado. Durante a apresentação de um documento, a partir do momento em que um <descriptorSwitch> é avaliado, ele é considerado resolvido até o final da apresentação do elemento <media> que lhe foi associado, ou seja, enquanto qualquer evento de apresentação associado com o elemento <media> estiver no estado ocorrendo ou pausado.

NOTA Recomenda-se que os formatadores NCL posterguem a avaliação do switch até o momento que um elo ancorado ao switch precisar ser avaliado. Convém que a avaliação do descriptorSwitch seja atrasada até o momento em que o objeto referindo-se a ele precise estar preparado para ser apresentado.

7.2.10 Área funcional *Timing*

A área funcional *Timing* define o módulo *Timing*. O módulo *Timing* permite a definição de atributos temporais para componentes de um documento. Basicamente, esse módulo define atributos para especificar o que acontece com um objeto ao final de sua apresentação (*freeze*) e a duração ideal de um objeto (*explicitDur*). Esses atributos podem ser incorporados pelos elementos <descriptor>.

7.2.11 Área funcional *Reuse*

NCL permite uma grande reutilização de seus elementos. A área funcional *Reuse* da NCL é particionada em três módulos: *Import*, *EntityReuse* e *ExtendedEntityReuse*.

Para permitir que uma base de entidades seja incorporada a outra base já existente, o módulo *Import* define o elemento <importBase>, que tem dois atributos: *documentURI* e *alias*. O atributo *documentURI* refere-se a um URI correspondente ao documento NCL contendo a base a ser importada. O atributo *alias* especifica um nome a ser utilizado como prefixo quando for necessário referir-se a elementos dessa base importada.

O nome do atributo *alias* deve obrigatoriamente ser único em um documento e seu escopo é restrito ao documento que o definiu. A referência teria o formato: *alias#element_id*. A operação de importação é transitiva, ou seja, se a *baseA* importa a *baseB* que importa a *baseC*, então a *baseA* importa a *baseC*. Entretanto, o *alias* definido para a *baseC* dentro da *baseB* obrigatoriamente não deve ser considerado pela *baseA*.

Quando um perfil de linguagem utiliza o módulo *Import*, as seguintes especificações são permitidas:

— o elemento <descriptorBase> pode ter um elemento-filho <importBase> referindo-se a um URI correspondente a um outro documento NCL contendo a base de descritores a ser importada (na verdade seus elementos-filhos) e aninhada. Quando uma base de descritores é importada, a base de regiões e a base de regras, quando existentes no documento importado, são também automaticamente importadas para as bases de regiões e de regras do documento correspondente que realiza a importação;

— o elemento <connectorBase> pode ter um elemento-filho <importBase> referindo-se a um URI correspondente a uma outra base de conectores a ser importada (na verdade seus elementos-filhos) e aninhada;

— o elemento <transitionBase> pode ter um elemento-filho <importBase> referindo-se a um URI correspondente a uma outra base de transições a ser importada (na verdade seus elementos-filhos) e aninhada;

— o elemento <regionBase> pode ter um elemento-filho <importBase> referindo-se a um URI correspondente a um outro documento NCL contendo a base da regiões a ser importada (na verdade seus elementos-filhos) e aninhada. Embora NCL defina seu modelo de leiaute, nada impede que um documento NCL utilize outros modelos de leiaute, desde que eles definam regiões onde os objetos podem ser apresentados, como, por exemplo, modelos de leiaute SMIL 2.0 [13]. Ao importar uma <regionBase>, um atributo opcional denominado *region* pode ser especificado, dentro de um elemento <importBase>. Quando presente, o atributo deve obrigatoriamente identificar o *id* de um elemento <region> declarado no elemento <regionBase> do documento hospedeiro (documento que fez a operação de importação). Como conseqüência, todos os elementos <region>, filhos do <regionBase> importados, devem obrigatoriamente ser considerados elementos <region> filhos da região referida pelo atributo *region* do <importBase>. Se não especificado, os elementos <region>, filhos do <regionBase> importado, devem obrigatoriamente ser considerados filhos diretos do elemento <regionBase> do documento hospedeiro.

O elemento <importedDocumentBase> especifica um conjunto de documentos NCL importados e deve obrigatoriamente ser definido como elemento-filho do elemento <head>. O elemento <importedDocumentBase> deve obrigatoriamente ter um atributo *id*, que identifica unicamente o elemento dentro de um documento.

Um documento NCL pode ser importado através de um elemento <importNCL>. Todas as bases definidas dentro de um documento NCL, bem como o elemento <body> do documento, são importados todos de uma vez, através do elemento <importNCL>. As bases são tratadas como se cada uma tivesse sido importada por um elemento <importBase>. O elemento <body> importado será tratado como um elemento <context>. 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>. Assim, o <body> importado, bem como quaisquer de seus nós, pode ser reusado dentro do elemento <body> 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. Como no elemento <importBase>, o nome deve obrigatoriamente ser único (type=ID) e seu escopo é restrito ao documento que definiu o atributo *alias*. A referência teria o formato: *alias#element_id*. É importante notar que o mesmo *alias* convém ser utilizado quando é necessário referir-se a elementos definidos nas bases do documento importado (<regionBase>, <connectorBase>, <descriptorBase> etc).

A operação do elemento <importNCL> também tem propriedade transitiva, ou seja, se o *documentoA* importar o *documentoB* que importa *documentoC*, então o *documentoA* importa o *documentoC*. Entretanto, o *alias* definido para o *documentoC* dentro do *documentoB* obrigatoriamente não deve ser considerado pelo *documentoA*.

Os elementos do módulo *Import*, seus elementos-filhos e seus atributos devem obrigatoriamente estar de acordo com a Tabela 34.

Tabela 34 — Módulo *Import* estendido

Elementos	Atributos	Conteúdo
importBase	*alias*, *documentURI*, *region*	vazio
importedDocumentBase	*id*	(importNCL)+
importNCL	*alias*, *documentURI*	vazio

O módulo *EntityReuse* permite que um elemento NCL seja reusado. Esse módulo define o atributo *refer*, que referencia um elemento *id* que será reusado. Apenas <media>, <context>, <body> e <switch> podem ser reusados. Um elemento que referencia um outro elemento não pode ser reusado; isto é, seu *id* não pode ser o valor de um atributo *refer*.

Se o nó referenciado for definido dentro de um documento D importado, o valor do atributo *refer* deve obrigatoriamente ter o formato “alias#id”, onde “alias” é o valor do atributo *alias* associado ao documento D importado.

Quando um perfil da linguagem utiliza o módulo *EntityReuse*, ele pode adicionar o atributo *refer* a:

— um elemento <media> ou <switch>. Nesse caso, o elemento referenciado deve obrigatoriamente ser, respectivamente, um elemento <media> ou <switch>, que representa o mesmo nó previamente definido no próprio <body> do documento ou em um <body> externo importado. O elemento referenciado deve obrigatoriamente conter diretamente a definição de todos os seus atributos e elementos-filhos;

— um elemento <context>. Nesse caso, o elemento referenciado deve obrigatoriamente ser um elemento <context> ou <body>, que representa o mesmo contexto previamente definido no <body> do próprio documento ou em um <body> externo importado. O elemento referenciado deve obrigatoriamente conter diretamente a definição de todos os seus atributos e elementos-filhos.

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, devem obrigatoriamente ser ignorados pelo formatador, exceto o atributo *id* que deve obrigatoriamente ser definido. A única outra exceção é para elementos <media>, para os quais novos elementos-filhos <area> e <property> podem ser adicionados, e um novo atributo Booleano, *newInstance*, pode ser definido.

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 deve obrigatoriamente ser 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 deve obrigatoriamente ser ignorado. O atributo Booleano *newInstance* é definido no módulo *ExtendedEntityReuse* e tem “true” (verdadeiro) como seu valor *default*.

O elemento referenciado e o elemento que o referencia devem obrigatoriamente ser considerados o mesmo, com relação às suas estruturas de dados. Em outras palavras, isso significa que um nó NCM pode ser representado por mais que um elemento NCL. Como nós contidos em um nó de composição NCM definem um conjunto, um nó NCM pode ser representado por não mais que um elemento NCL dentro de uma composição. Isso significa que o atributo *id* de um elemento NCL representando um nó NCM não é apenas um identificador único para o elemento, mas também o único identificador correspondente ao nó NCM na composição.

NOTA Pode-se consultar a NCMCore:2005 para outras informações.

EXEMPLO Supondo que o elemento NCL (node1) representa um nó NCM, os elementos NCL que o referenciam (node1ReuseA, node1ReuseB) representam o mesmo nó NCM. Em outras palavras, o nó NCM único é representado por mais de um elemento NCL (node1, node1ReuseA, e node1ReuseB). Mais ainda, como os nós contidos em um nó de composição NCM definem um conjunto, cada um dos elementos NCL, node1, node1ReuseA e node1ReuseB, deve ser declarado dentro de uma composição diferente.

O elemento referenciado e o elemento que o referencia também devem obrigatoriamente ser considerados o mesmo nó em relação à sua estrutura de apresentação, se o atributo *newInstance* receber um valor “false”.

Portanto, a seguinte semântica deve obrigatoriamente ser respeitada. Considere o conjunto de elementos <media> composto pelo elemento <media> referenciado e todos os elementos <media> que o referenciam. Se qualquer elemento do subconjunto formado pelo elemento <media> referenciado e todos os outros elementos <media> que têm o atributo *newInstance* igual a “false” estiverem agendados para serem apresentados, assume-se que todos os outros elementos nesse subconjunto, que não são descendentes de um elemento <switch>, estão também agendados para apresentação e, mais que isso, quando eles forem apresentados, devem obrigatoriamente ser representados pela mesma instância de apresentação. Elementos descendentes de um elemento <switch> também devem obrigatoriamente ter o mesmo comportamento, se todas as regras necessárias para apresentá-los forem satisfeitas; caso contrário, eles não devem ser agendados para apresentação. A instância comum em apresentação deve obrigatoriamente notificar todos os eventos associados com os elementos <area> e <property> definidos em todos os elementos <media> do subconjunto que foram agendados para apresentação. Por outro lado, os elementos <media> no conjunto que tem valores de atributo *newInstance* iguais a “true” obrigatoriamente não devem ser agendados para apresentação. Quando são individualmente agendados para apresentação, novas instâncias independentes de apresentação devem obrigatoriamente ser criadas.

7.2.12 Área funcional *Navigational Key*

A área funcional *Navigational Key* define o módulo *KeyNavigation* que oferece as extensões necessárias para descrever as operações de movimentação do foco, definido por um dispositivo, como um controle remoto. Basicamente, o módulo define atributos que podem ser incorporados por elementos <descriptor>.

O atributo *focusIndex* especifica um índice para o elemento <media> sobre o qual o foco pode ser aplicado, quando esse elemento estiver em exibição, utilizando o elemento <descriptor> que definiu o atributo. Quando um elemento <descriptor> não definir esse atributo, ele é considerado como se não pudesse receber o foco. Em um determinado momento da apresentação, se o foco não tiver sido ainda definido, ou se estiver perdido, um foco será inicialmente aplicado ao elemento que estiver sendo apresentado cujo descritor tenha o menor valor de índice.

Os valores do atributo *focusIndex* devem obrigatoriamente ser únicos em um documento NCL. Caso contrário, atributos repetidos devem obrigatoriamente ser ignorados. Além disso, quando um elemento <media> referencia um outro elemento <media> (utilizando o atributo *refer* especificado em 7.2.11), ele deve obrigatoriamente ignorar o *focusIndex* especificado pelo elemento <descriptor> associado ao elemento <media> referido.

O atributo *moveUp* especifica um valor igual ao valor do *focusIndex* associado ao elemento sobre o qual o foco será aplicado quando a tecla “seta para cima” (“*up arrow key*”) for pressionada. O atributo *moveDown* especifica um valor igual ao valor do *focusIndex* associado ao elemento sobre o qual o foco será aplicado quando a tecla “seta para baixo” (“*down arrow key*”) for pressionada.

O atributo *moveRight* especifica um valor igual ao valor do *focusIndex* associado ao elemento sobre o qual o foco será aplicado quando a tecla “seta para a direita “(“*right arrow key*”) for pressionada. O atributo *moveLeft* especifica um valor igual ao valor do *focusIndex* associado ao elemento sobre o qual o foco será aplicado quando a tecla “seta para a esquerda” (“*left arrow key*”) for pressionada.

Quando o foco é aplicado a um elemento com a propriedade de visibilidade com o valor “false”, ou a um elemento que não estiver sendo apresentado, o foco atual não deve se mover.

O atributo *focusSrc* pode especificar um conteúdo alternativo a ser apresentado, ao invés do conteúdo da apresentação atual, se um elemento receber o foco. Esse atributo segue as mesmas regras do atributo *src* do elemento <media>.

Quando um elemento recebe um foco, a moldura (*box*) definida pelos atributos de posicionamento do elemento deve obrigatoriamente ser destacada. O atributo *focusBorderColor* define a cor de destaque e pode receber os nomes reservados de cor: “white”, “black”, “silver”, “gray”, “red”, “maroon”, “fuchsia”, “purple”, “lime”, “green”, “yellow”, “olive”, “blue”, “navy”, “aqua”, ou “teal”. O atributo *focusBorderWidth* define a largura em pixels da borda em destaque (0 significa que nenhuma borda aparece, valores positivos significam que a borda está fora do

conteúdo do objeto e valores negativos significam que a borda é desenhada sobre o conteúdo do objeto); o atributo *focusBorderTransparency* define a transparência da cor de destaque. O *focusBorderTransparency* deve obrigatoriamente ser um valor real entre 0 e 1, ou um valor real na faixa [0,100] terminando com o caractere "%" (por exemplo, 30 %), com "1" ou "100 %" significando transparência total e "0" ou "0 %" significando nenhuma transparência. Quando o *focusBorderColor*, o *focusBorderWidth* ou o *focusBorderTransparency* não estão definidos, valores *default* devem obrigatoriamente ser assumidos. Esses valores são especificados nas propriedades do elemento <media> do tipo *application/x-ginga-settings*: *defaultFocusBorderColor*, *defaultFocusBorderWidth* e *defaultFocusTransparency*, respectivamente.

Quando um elemento em foco é selecionado pressionando a tecla de ativação, o atributo *focusSelSrc* pode especificar um conteúdo de mídia alternativo a ser apresentado, ao invés da apresentação atual. Este atributo segue as mesmas regras do atributo *src* do elemento <media>. Quando selecionada, a caixa definida pelos atributos de posicionamento de elemento deve obrigatoriamente ser destacada com a cor definida pelo atributo *selBorderColor* (valor-*default* especificado pelo *defaultSelBorderColor* do elemento <media> do tipo *application/x-ginga-settings*), a largura da borda em destaque é definida pelo atributo *focusBorderWidth* e a transparência da cor da borda é definida pelo atributo *focusBorderTransparency*.

Quando um elemento em foco é selecionado pressionando a tecla de ativação, o controle do foco deve obrigatoriamente ser passado ao exibidor do elemento <media> (*player*). O exibidor pode então seguir suas próprias regras para navegação. O controle de foco deve obrigatoriamente ser devolvido ao formatador NCL quando a tecla "back" for pressionada. Nesse caso, o elemento ativado deve obrigatoriamente ser destacado como quando estava em foco, antes de ser ativado.

7.2.13 Área funcional *Animation*

Animação é na verdade uma combinação de dois fatores: suporte ao desenho do objeto e suporte ao movimento do objeto ou, mais propriamente, suporte para a alteração do objeto em função do tempo.

NCL não é um formato de conteúdo e, como tal, não tem suporte para a criação de objetos de mídia e não tem um método generalizado para alterar o conteúdo do objeto de mídia. Ao contrário, NCL é um formato de escalonamento e orquestração. Isso significa que NCL não pode ser utilizada para fazer desenhos animados, mas pode ser utilizada para exibir objetos de desenho animado no contexto de uma apresentação geral e para alterar as propriedades de sincronização e exibição de um objeto de um desenho animado (ou qualquer outro) como um todo, enquanto o objeto estiver sendo exibido.

As primitivas de animação NCL permitem que os valores de propriedades dos nós sejam alterados durante uma duração determinada. Como a animação NCL pode ser computacionalmente intensa, ela somente é suportada pelo perfil EDTV e apenas as propriedades que definem valores numéricos e cores podem ser animadas.

A área funcional *Animation* define o módulo *Animation* que fornece as extensões necessárias para descrever o que acontece quando o valor de uma propriedade de um nó é alterado. Basicamente, o módulo define atributos que podem ser incorporados pelos elementos <simpleAction> de um conector, se seu valor *eventType* for "attribution". Dois novos atributos são definidos: *duration* e *by*.

Ao atribuir um novo valor para uma propriedade, a alteração é instantânea por *default* (*duration*="0"), mas a alteração também pode ser feita durante um período explicitamente declarado, especificado pelo atributo *duration*.

Além disso, ao atribuir um novo valor a uma propriedade, a alteração do valor antigo para o novo pode ser linear por *default* (*by*="indefinite"), ou feita passo a passo, com o passo especificado pelo atributo *by*.

A combinação das definições dos atributos *duration* e *by* oferece a definição de como (de forma discreta ou linear) a alteração deve obrigatoriamente ser realizada e o seu intervalo de transformação.

7.2.14 Área funcional SMIL *Transition Effects*

A área funcional *Transition Effects* de SMIL é dividida em três módulos: *TransitionBase*, derivada das especificações NCL, *BasicTransitions* e *TransitionModifiers.*

NOTA Pode-se consultar o SMIL 2.1 Specification:2005 para outras informações.

O módulo *TransitionBase* é definido pela NCL 3.0 e consiste em um elemento <transitionBase> que especifica um conjunto de efeitos de transição e deve obrigatoriamente ser definido como elemento-filho do elemento <head>.

O elemento <transitionBase>, seus elementos-filhos e seus atributos devem obrigatoriamente estar de acordo com a Tabela 35.

Tabela 35 — Módulo *TransitionBase* estendido

Elementos	Atributos	Conteúdo
transitionBase	*id*	(importBase, transition)+

O módulo *BasicTransitions* é importado a partir da especificação SMIL 2.1. O módulo tem apenas um elemento chamado <transition>.

NOTA Pode-se consultar o SMIL 2.1 Specification, 2005 para outras informações.

No perfil NCL 3.0 DTV Avançado, o elemento <transition> é especificado no elemento <transitionBase> e permite que um padrão (*template*) de transição seja definido. Cada elemento <transition> define um padrão único de transição e deve obrigatoriamente ter um atributo *id* para que possa ser referenciado dentro de um elemento <descriptor>.

Os atributos do elemento <transition> são: *type*; *subtype*; *dur*; *startProgress*; *endProgress*; *direction*; e *fadeColor*.

As transições são classificadas de acordo com uma taxonomia de dois níveis: tipos e subtipos. 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* é obrigatório e utilizado para especificar a transição geral. Se o tipo nomeado não for suportado pelo formatador NCL, a transição deve obrigatoriamente ser ignorada. Isso não é uma condição de erro, pois as implementações são livres para ignorar transições.

O atributo *subtype* fornece o controle específico para a transição. Esse atributo é opcional e, se especificado, deve obrigatoriamente ser um dos subtipos de transição apropriados para o tipo correspondente. Se esse atributo não for especificado, a transição deve obrigatoriamente ser revertida para o subtipo *default* do tipo especificado. Apenas os subtipos *default*, para os cinco tipos de transição obrigatórios, listados na Tabela 36, devem obrigatoriamente ser suportados.

Tabela 36 — Tipos e subtipos de transição exigidos

Tipo de transition	Subtipo default
barWipe	leftToRight
irisWipe	rectangle

clockWipe	clockwiseTwelve
snakeWipe	topLeftHorizontal
fade	crossfade

O atributo *dur* especifica a duração da transição. A duração *default* é de 1 segundo.

O atributo *startProgress* especifica a quantidade de efeito de transição do início da execução. Os valores permitidos são números reais na faixa [0.0,1.0]. Por exemplo, pode-se querer iniciar um *crossfade* com imagem destino já transparente em 50 % inicialmente. Para esse caso, o *startProgress* seria 0.5. O valor *default* é 0.0.

O atributo *endProgress* especifica a quantidade de efeito de transição ao término da execução. Os valores permitidos são números reais na faixa [0.0,1.0] e o valor desse atributo deve obrigatoriamente ser maior ou igual ao valor do atributo *startProgress*. Se *endProgress* for igual a *startProgress*, então a transmissão permanece em um valor fixo pela duração da transmissão. O valor *default* é 1,0.

O atributo *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.

Se o valor do atributo *type* for “fade” e o valor do atributo *subtype* for “fadeToColor” ou “fadeFromColor”, então o atributo *fadeColor* especifica a cor final ou inicial do *fade*. Se o valor do atributo *type* não for “fade”, ou se o valor do atributo *subtype* não for “fadeToColor” ou “fadeFromColor”, então o atributo *fadeColor* deve obrigatoriamente ser ignorado. O valor *default* é “black”.

O módulo *BasicTransition* também define os atributos a serem utilizados nos elementos <descriptor>, para os padrões de transição definidos pelos elementos <transition>: atributos *transIn* e *transOut*. As transições especificadas com um atributo *transIn* iniciarão no começo da duração ativa dos elementos de mídia (quando a apresentação do objeto inicia). As transições especificadas com um atributo *transOut* terminarão no final da duração ativa dos elementos de mídia (quando a apresentação do objeto transita do estado ocorrendo para preparado).

Os atributos *transIn* e *transOut* são adicionados aos elementos <descriptor>. O valor *default* de ambos os atributos é uma *string* vazia, que indica que, obrigatoriamente, nenhuma transição deve ser realizada.

O valor dos atributos *transIn* e *transOut* é uma lista separada por ponto e vírgula dos identificadores de transição. Cada um dos identificadores deve obrigatoriamente corresponder ao valor do identificador XML de um dos elementos de transição anteriormente definidos no elemento <transitionBase>. O objetivo da lista separada por ponto e vírgula é permitir que os autores especifiquem um conjunto de transições alternativas (*fallback*) se a transição preferida não estiver disponível.

A primeira transição na lista deve ser realizada se o agente do usuário implementar esta transição. Se essa transição não estiver disponível, então a segunda transição na lista deve ser realizada, e assim sucessivamente. Se o valor do atributo *transIn* ou *transOut* não corresponder ao valor do identificador XML de nenhum dos elementos de transição previamente definidos, então há um erro. Em caso de ocorrer esse erro, o valor do atributo que deve ser considerado como sendo uma string vazia e, então, obrigatoriamente, nenhuma transição deve ser realizada.

Todas as transições definidas no módulo SMIL *BasicTransitions* aceitam quatro atributos adicionais que podem ser utilizados para controlar a aparência das transições, conforme especificado pelo módulo *TransitionModifiers.* O atributo *horRepeat* especifica quantas vezes será realizado o padrão de transições ao longo do eixo horizontal. O valor *default* é 1 (o padrão ocorre uma vez horizontalmente). O atributo *vertRepeat* especifica quantas vezes será realizado o padrão de transição ao longo do eixo vertical. O valor *default* é 1 (o padrão ocorre uma vez verticalmente). O atributo *borderWidth* especifica a largura de uma borda gerada ao longo de uma área apagada. Os valores permitidos são inteiros maiores ou iguais a 0. Se o valor de *borderWidth* for igual a 0, então convém que nenhuma borda seja gerada ao longo da área apagada. O valor *default* é 0. Se o valor do atributo *type* não for "fade", então o atributo *borderColor* especifica o conteúdo de uma borda gerada ao longo de uma área apagada. Se o valor desse atributo for uma cor, então a borda gerada é preenchida com a cor definida. Se o valor deste atributo for “blend”, então a borda gerada é uma mistura aditiva (ou *blur*) das fontes de mídia. O valor *default* para esse atributo é “black”.

O elemento do módulo *BasicTransition* estendido, seus elementos-filhos e seus atributos devem obrigatoriamente estar de acordo com a Tabela 37.

Tabela 37 — Módulo *BasicTransition* estendido

Elementos	Atributos	Conteúdo
transition	*id, type, subtype, dur, startProgress, endProgress, direction, fadeColor, horRepeat, vertRepeat, borderWidth, borderColor*	vazio

7.2.15 Área funcional SMIL *Metainformation*

Uma metainformação não contém informações de conteúdo utilizadas ou exibidas durante a apresentação de um documento. Ao invés disso, contém informações sobre o conteúdo utilizado ou exibido. A área funcional *Metainformation* de SMIL importada por NCL é composta pelo módulo *metainformation*.

NOTA Pode-se consultar o SMIL 2.1 Specification:2005 para outras informações.

O módulo *metainformation* contém dois elementos que permitem a descrição de documentos NCL. O elemento <meta> especifica um único par de propriedade/valor nos atributos *name* e *content*, respectivamente. O elemento <metadata> contém informações que também se relacionam com a metainformação do documento. Ele atua como o elemento raiz da árvore RDF. O elemento <metadata> pode ter como elementos-filhos: elementos RDF e seus subelementos

NOTA Pode-se consultar a RDF:1999 para outras informações.

Os elementos do módulo *Metainformation*, seus elementos-filhos e seus atributos devem obrigatoriamente estar de acordo com a Tabela 38.

Tabela 38 — Módulo *Meta-Information* estendido

Elementos	Atributos	Conteúdo
meta	*name, content*	vazio
metadata	*empty*	RDF tree

7.3 Perfis da linguagem NCL para o SBTVD

7.3.1 Módulos de perfis

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 (inclusive os discutidos em 7.2) e fornece todas as facilidades para a autoria declarativa de documentos NCL.

Os perfis definidos para o SBTVD são:

a) perfil NCL 3.0 DTV Avançado: o perfil NCL 3.0 DTV Avançado 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* e *TransitionBase* da NCL 3.0, e também os módulos *BasicTransition* e *Meta-Information* da SMIL 2.1. As tabelas apresentadas em 7.2 mostram cada elemento de cada módulo, já estendidos pelos atributos e elementos filhos herdados de outros módulos por esse perfil (ver os esquemas XML em 7.3.2);

b) perfil NCL 3.0 CausalConnector: o perfil *CausalConnector* permite a criação de conectores simples hipermídia. O perfil inclui os módulos *Structure, CausalConnectorFunctionality* e *ConnectorBase*. No perfil, o elemento <body> do módulo *Structure* não é utilizado (ver os esquemas XML em 7.3.3);

c) perfil NCL 3.0 DTV Básico: o perfil NCL 3.0 DTV Básico 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*. As tabelas apresentadas em 7.3.4 mostram cada elemento de cada módulo desse perfil, já estendidos pelos atributos e elementos-filhos herdados de outros módulos (ver os esquemas XML em 7.3.5).

7.3.2 Esquema do perfil NCL 3.0 DTV avançado

NCL30EDTV.xsd

```
<!--
XML Schema for the NCL Language

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/NCL3.0/profiles/NCL30EDTV.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006
-->

<schema xmlns="http://www.w3.org/2001/XMLSchema"
```

```
xmlns:animation="http://www.ncl.org.br/NCL3.0/Animation"
xmlns:compositeInterface="http://www.ncl.org.br/NCL3.0/CompositeNodeInterface"
xmlns:causalConnectorFunctionality="http://www.ncl.org.br/NCL3.0/CausalConnectorFunctionality"
xmlns:connectorBase="http://www.ncl.org.br/NCL3.0/ConnectorBase"
xmlns:connectorCausalExpression="http://www.ncl.org.br/NCL3.0/ConnectorCausalExpression"
xmlns:contentControl="http://www.ncl.org.br/NCL3.0/ContentControl"
xmlns:context="http://www.ncl.org.br/NCL3.0/Context"
xmlns:descriptor="http://www.ncl.org.br/NCL3.0/Descriptor"
xmlns:entityReuse="http://www.ncl.org.br/NCL3.0/EntityReuse"
xmlns:extendedEntityReuse="http://www.ncl.org.br/NCL3.0/ExtendedEntityReuse"
xmlns:descriptorControl="http://www.ncl.org.br/NCL3.0/DescriptorControl"
xmlns:import="http://www.ncl.org.br/NCL3.0/Import"
xmlns:keyNavigation="http://www.ncl.org.br/NCL3.0/KeyNavigation"
xmlns:layout="http://www.ncl.org.br/NCL3.0/Layout"
xmlns:linking="http://www.ncl.org.br/NCL3.0/Linking"
xmlns:media="http://www.ncl.org.br/NCL3.0/Media"
xmlns:mediaAnchor="http://www.ncl.org.br/NCL3.0/MediaContentAnchor"
xmlns:propertyAnchor="http://www.ncl.org.br/NCL3.0/PropertyAnchor"
xmlns:structure="http://www.ncl.org.br/NCL3.0/Structure"
xmlns:switchInterface="http://www.ncl.org.br/NCL3.0/SwitchInterface"
xmlns:testRule="http://www.ncl.org.br/NCL3.0/TestRule"
xmlns:testRuleUse="http://www.ncl.org.br/NCL3.0/TestRuleUse"
xmlns:timing="http://www.ncl.org.br/NCL3.0/Timing"
xmlns:transitionBase="http://www.ncl.org.br/NCL3.0/TransitionBase"
xmlns:metainformation="http://www.w3.org/2001/SMIL20/Metainformation"
xmlns:basicTransition="http://www.w3.org/2001/SMIL20/BasicTransitions"
xmlns:profile="http://www.ncl.org.br/NCL3.0/EDTVProfile"
targetNamespace="http://www.ncl.org.br/NCL3.0/EDTVProfile"
elementFormDefault="qualified" attributeFormDefault="unqualified" >

<!-- import the definitions in the modules namespaces -->
<import namespace="http://www.ncl.org.br/NCL3.0/Animation"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30Animation.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/CompositeNodeInterface"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30CompositeNodeInterface.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/CausalConnectorFunctionality"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30CausalConnectorFunctionality.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/ConnectorBase"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30ConnectorBase.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/ConnectorCausalExpression"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30ConnectorCausalExpression.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/ContentControl"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30ContentControl.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/Context"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30Context.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/Descriptor"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30Descriptor.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/DescriptorControl"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30DescriptorControl.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/EntityReuse"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30EntityReuse.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/ExtendedEntityReuse"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30ExtendedEntityReuse.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/Import"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30Import.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/KeyNavigation"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30KeyNavigation.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/Layout"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30Layout.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/Linking"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30Linking.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/Media"
```

```
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30Media.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/MediaContentAnchor"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30MediaContentAnchor.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/PropertyAnchor"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30PropertyAnchor.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/Structure"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30Structure.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/SwitchInterface"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30SwitchInterface.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/TestRule"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30TestRule.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/TestRuleUse"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30TestRuleUse.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/Timing"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30Timing.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/TransitionBase"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30TransitionBase.xsd"/>
<import namespace="http://www.w3.org/2001/SMIL20/Metainformation"
  schemaLocation="http://www.w3.org/2001/SMIL20/smil20-Metainformation.xsd"/>
<import namespace="http://www.w3.org/2001/SMIL20/BasicTransitions"
  schemaLocation="http://www.w3.org/2001/SMIL20/smil20-BasicTransitions.xsd"/>
```

```
<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- Structure                                           -->
<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- extends ncl element -->

<element name="ncl" substitutionGroup="structure:ncl"/>

<!-- extends head element -->

<complexType name="headType">
  <complexContent>
    <extension base="structure:headPrototype">
      <sequence>
        <element ref="profile:importedDocumentBase" minOccurs="0" maxOccurs="1"/>
        <element ref="profile:ruleBase" minOccurs="0" maxOccurs="1"/>
        <element ref="profile:transitionBase" minOccurs="0" maxOccurs="1"/>
        <element ref="profile:regionBase" minOccurs="0" maxOccurs="unbounded"/>
        <element ref="profile:descriptorBase" minOccurs="0" maxOccurs="1"/>
        <element ref="profile:connectorBase" minOccurs="0" maxOccurs="1"/>
        <element ref="profile:meta" minOccurs="0" maxOccurs="unbounded"/>
        <element ref="profile:metadata" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<element name="head" type="profile:headType" substitutionGroup="structure:head"/>

<!-- extends body element -->

<complexType name="bodyType">
  <complexContent>
    <extension base="structure:bodyPrototype">
      <choice minOccurs="0" maxOccurs="unbounded">
        <group ref="profile:contextInterfaceElementGroup"/>
```

```
      <element ref="profile:media"/>
      <element ref="profile:context"/>
      <element ref="profile:switch"/>
      <element ref="profile:link"/>
    </choice>
  </extension>
 </complexContent>
</complexType>

<element name="body" type="profile:bodyType" substitutionGroup="structure:body"/>

<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- Layout                                                       -->
<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- extends regionBase element -->

<complexType name="regionBaseType">
 <complexContent>
  <extension base="layout:regionBasePrototype">
    <choice minOccurs="1" maxOccurs="unbounded">
      <element ref="profile:importBase"/>
      <element ref="profile:region"/>
    </choice>
  </extension>
 </complexContent>
</complexType>

<complexType name="regionType">
 <complexContent>
  <extension base="layout:regionPrototype">
  </extension>
 </complexContent>
</complexType>

<element name="regionBase" type="profile:regionBaseType" substitutionGroup="layout:regionBase"/>
<element name="region" type="profile:regionType" substitutionGroup="layout:region"/>

<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- Media                                                        -->
<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- extends Media elements -->

<!-- media interface element groups -->
<group name="mediaInterfaceElementGroup">
 <choice>
  <element ref="profile:area"/>
  <element ref="profile:property"/>
 </choice>
</group>

<complexType name="mediaType">
 <complexContent>
  <extension base="media:mediaPrototype">
    <choice minOccurs="0" maxOccurs="unbounded">
      <group ref="profile:mediaInterfaceElementGroup"/>
    </choice>
    <attributeGroup ref="descriptor:descriptorAttrs"/>
    <attributeGroup ref="entityReuse:entityReuseAttrs"/>
    <attributeGroup ref="extendedEntityReuse:extendedEntityReuseAttrs"/>
  </extension>
 </complexContent>
</complexType>
```

```
<element name="media" type="profile:mediaType" substitutionGroup="media:media"/>

<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- Context                                             -->
<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- extends context element -->

<!-- composite node interface element groups -->
<group name="contextInterfaceElementGroup">
  <choice>
    <element ref="profile:port"/>
    <element ref="profile:property"/>
  </choice>
</group>

<complexType name="contextType">
  <complexContent>
    <extension base="context:contextPrototype">
      <choice minOccurs="0" maxOccurs="unbounded">
        <group ref="profile:contextInterfaceElementGroup"/>
        <element ref="profile:media"/>
        <element ref="profile:context"/>
        <element ref="profile:link"/>
        <element ref="profile:switch"/>
      </choice>
      <attributeGroup ref="entityReuse:entityReuseAttrs"/>
    </extension>
  </complexContent>
</complexType>

<element name="context" type="profile:contextType" substitutionGroup="context:context"/>

<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- MediaContentAnchor                                        -->
<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- extends area element -->

<complexType name="componentAnchorType">
  <complexContent>
    <extension base="mediaAnchor:componentAnchorPrototype">
    </extension>
  </complexContent>
</complexType>

<element name="area" type="profile:componentAnchorType" substitutionGroup="mediaAnchor:area"/>

<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- CompositeNodeInterface                                      -->
<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- extends port element -->

<complexType name="compositeNodePortType">
  <complexContent>
    <extension base="compositeInterface:compositeNodePortPrototype">
    </extension>
  </complexContent>
</complexType>
```

```
<element name="port" type="profile:compositeNodePortType" substitutionGroup="compositeInterface:port"/>

<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- PropertyAnchor                                    -->
<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- extends property element -->

<complexType name="propertyAnchorType">
  <complexContent>
    <extension base="propertyAnchor:propertyAnchorPrototype">
    </extension>
  </complexContent>
</complexType>

<element name="property" type="profile:propertyAnchorType" substitutionGroup="propertyAnchor:property"/>

<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- SwitchInterface                                   -->
<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- extends switchPort element -->

<complexType name="switchPortType">
  <complexContent>
    <extension base="switchInterface:switchPortPrototype">
    </extension>
  </complexContent>
</complexType>

<element name="mapping" substitutionGroup="switchInterface:mapping"/>
<element name="switchPort" type="profile:switchPortType" substitutionGroup="switchInterface:switchPort"/>

<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- Descriptor                                        -->
<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- substitutes descriptorParam element -->

<element name="descriptorParam" substitutionGroup="descriptor:descriptorParam"/>


<!-- extends descriptor element -->

<complexType name="descriptorType">
  <complexContent>
    <extension base="descriptor:descriptorPrototype">
      <attributeGroup ref="layout:regionAttrs"/>
      <attributeGroup ref="timing:explicitDurAttrs"/>
      <attributeGroup ref="timing:freezeAttrs"/>
      <attributeGroup ref="keyNavigation:keyNavigationAttrs"/>
      <attributeGroup ref="profile:transitionAttrs"/>
    </extension>
  </complexContent>
</complexType>

<element name="descriptor" type="profile:descriptorType" substitutionGroup="descriptor:descriptor"/>

<!-- extends descriptorBase element -->
<complexType name="descriptorBaseType">
  <complexContent>
    <extension base="descriptor:descriptorBasePrototype">
      <choice minOccurs="1" maxOccurs="unbounded">
        <element ref="profile:importBase"/>
        <element ref="profile:descriptor"/>
```



```
      <element ref="profile:descriptorSwitch"/>
    </choice>
   </extension>
  </complexContent>
</complexType>

<element name="descriptorBase" type="profile:descriptorBaseType" substitutionGroup="descriptor:descriptorBase"/>

<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- Linking                                              -->
<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->

<!-- substitutes linkParam and bindParam elements -->
<element name="linkParam" substitutionGroup="linking:linkParam"/>
<element name="bindParam" substitutionGroup="linking:bindParam"/>

<!-- extends bind element and link element, as a consequence-->

<complexType name="bindType">
  <complexContent>
   <extension base="linking:bindPrototype">
    <attributeGroup ref="descriptor:descriptorAttrs"/>
   </extension>
  </complexContent>
</complexType>

<element name="bind" type="profile:bindType" substitutionGroup="linking:bind"/>

<!-- extends link element -->
<complexType name="linkType">
  <complexContent>
   <extension base="linking:linkPrototype">
   </extension>
  </complexContent>
</complexType>

<element name="link" type="profile:linkType" substitutionGroup="linking:link"/>

<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- Connector                                              -->
<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- extends connectorBase element -->

  <complexType name="connectorBaseType">
    <complexContent>
      <extension base="connectorBase:connectorBasePrototype">
        <choice minOccurs="0" maxOccurs="unbounded">
          <element ref="profile:importBase"/>
                <element ref="profile:causalConnector" />
        </choice>
      </extension>
    </complexContent>
  </complexType>

  <complexType name="simpleActionType">
   <complexContent>
    <extension base="connectorCausalExpression:simpleActionPrototype">
      <attributeGroup ref="animation:animationAttrs"/>
    </extension>
```

```
    </complexContent>
   </complexType>

  <element name="connectorBase" type="profile:connectorBaseType" substitutionGroup="connectorBase:connectorBase"/>

  <element name="causalConnector" substitutionGroup="causalConnectorFunctionality:causalConnector"/>

  <element name="connectorParam" substitutionGroup="causalConnectorFunctionality:connectorParam"/>

  <element name="simpleCondition" substitutionGroup="causalConnectorFunctionality:simpleCondition"/>

  <element name="compoundCondition" substitutionGroup="causalConnectorFunctionality:compoundCondition"/>

  <element name="simpleAction" type="profile:simpleActionType"
substitutionGroup="causalConnectorFunctionality:simpleAction"/>

  <element name="compoundAction" substitutionGroup="causalConnectorFunctionality:compoundAction"/>

  <element name="assessmentStatement" substitutionGroup="causalConnectorFunctionality:assessmentStatement"/>

  <element name="attributeAssessment" substitutionGroup="causalConnectorFunctionality:attributeAssessment"/>

  <element name="valueAssessment" substitutionGroup="causalConnectorFunctionality:valueAssessment"/>

  <element name="compoundStatement" substitutionGroup="causalConnectorFunctionality:compoundStatement"/>

  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <!-- TestRule                                    -->
  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <!-- extends rule element -->
  <complexType name="ruleType">
   <complexContent>
    <extension base="testRule:rulePrototype">
    </extension>
   </complexContent>
  </complexType>

  <element name="rule" type="profile:ruleType" substitutionGroup="testRule:rule"/>

  <!-- extends compositeRule element -->
  <complexType name="compositeRuleType">
   <complexContent>
    <extension base="testRule:compositeRulePrototype">
    </extension>
   </complexContent>
  </complexType>

  <element name="compositeRule" type="profile:compositeRuleType" substitutionGroup="testRule:compositeRule"/>

  <!-- extends ruleBase element -->
  <complexType name="ruleBaseType">
   <complexContent>
    <extension base="testRule:ruleBasePrototype">
     <choice minOccurs="1" maxOccurs="unbounded">
      <element ref="profile:importBase"/>
      <element ref="profile:rule"/>
      <element ref="profile:compositeRule"/>
     </choice>
    </extension>
   </complexContent>
  </complexType>
```

```
  <element name="ruleBase" type="profile:ruleBaseType" substitutionGroup="testRule:ruleBase"/>

   <!-- =================================================================== -->
  <!-- TestRuleUse                                             -->
  <!-- =================================================================== -->
  <!-- extends bindRule element -->
  <complexType name="bindRuleType">
    <complexContent>
      <extension base="testRuleUse:bindRulePrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="bindRule" type="profile:bindRuleType" substitutionGroup="testRuleUse:bindRule"/>

  <!-- =================================================================== -->
  <!-- ContentControl                                          -->
  <!-- =================================================================== -->
  <!-- extends switch element -->

  <!-- switch interface element groups -->
  <group name="switchInterfaceElementGroup">
    <choice>
      <element ref="profile:switchPort"/>
    </choice>
  </group>

  <!-- extends defaultComponent element -->
  <complexType name="defaultComponentType">
    <complexContent>
      <extension base="contentControl:defaultComponentPrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="defaultComponent" type="profile:defaultComponentType"
substitutionGroup="contentControl:defaultComponent"/>

  <complexType name="switchType">
    <complexContent>
      <extension base="contentControl:switchPrototype">
        <choice minOccurs="0" maxOccurs="unbounded">
          <group ref="profile:switchInterfaceElementGroup"/>
          <element ref="profile:bindRule"/>
          <element ref="profile:switch"/>
          <element ref="profile:media"/>
          <element ref="profile:context"/>
        </choice>
       <attributeGroup ref="entityReuse:entityReuseAttrs"/>
      </extension>
    </complexContent>
  </complexType>

  <element name="switch" type="profile:switchType" substitutionGroup="contentControl:switch"/>

  <!-- =================================================================== -->
  <!-- DescriptorControl                                       -->
  <!-- =================================================================== -->
  <!-- extends defaultDescriptor element -->
```

```
  <complexType name="defaultDescriptorType">
    <complexContent>
      <extension base="descriptorControl:defaultDescriptorPrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="defaultDescriptor" type="profile:defaultDescriptorType"
substitutionGroup="descriptorControl:defaultDescriptor"/>

  <!-- extends descriptorSwitch element -->

  <complexType name="descriptorSwitchType">
    <complexContent>
      <extension base="descriptorControl:descriptorSwitchPrototype">
          <choice minOccurs="0" maxOccurs="unbounded">
            <element ref="profile:descriptor"/>
            <element ref="profile:bindRule"/>
          </choice>
      </extension>
    </complexContent>
  </complexType>

  <element name="descriptorSwitch" type="profile:descriptorSwitchType"
substitutionGroup="descriptorControl:descriptorSwitch"/>

  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <!-- Timing                                   -->
  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->

  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <!-- Import                                   -->
  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <complexType name="importBaseType">
    <complexContent>
      <extension base="import:importBasePrototype">
      </extension>
    </complexContent>
  </complexType>

  <complexType name="importNCLType">
    <complexContent>
      <extension base="import:importNCLPrototype">
      </extension>
    </complexContent>
  </complexType>

  <complexType name="importedDocumentBaseType">
    <complexContent>
      <extension base="import:importedDocumentBasePrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="importBase" type="profile:importBaseType" substitutionGroup="import:importBase"/>

  <element name="importNCL" type="profile:importNCLType" substitutionGroup="import:importNCL"/>
  <element name="importedDocumentBase" type="profile:importedDocumentBaseType"
substitutionGroup="import:importedDocumentBase"/>

  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <!-- EntityReuse                                   -->
```

```
  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->

  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <!-- ExtendedEntityReuse                                  -->
  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->

  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <!-- KeyNavigation                                        -->
  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->

  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <!-- TransitionBase                                       -->
  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <!-- extends transitionBase element -->

  <complexType name="transitionBaseType">
    <complexContent>
      <extension base="transitionBase:transitionBasePrototype">
        <choice minOccurs="0" maxOccurs="unbounded">
          <element ref="profile:transition"/>
          <element ref="profile:importBase"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>

  <element name="transitionBase" type="profile:transitionBaseType" substitutionGroup="transitionBase:transitionBase"/>

  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <!-- BasicTransition                                      -->
  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->

  <attributeGroup name="transitionAttrs">
    <attribute ref="basicTransition:transIn"/>
    <attribute ref="basicTransition:transOut"/>
  </attributeGroup>

  <element name="transition" substitutionGroup="basicTransition:transition"/>

  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <!-- Metainformation                                      -->
  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->

  <element name="meta" substitutionGroup="metainformation:meta"/>

  <element name="metadata" substitutionGroup="metainformation:metadata"/>

</schema>
```

7.3.3 Esquema do perfil NCL 3.0 CausalConnector

CausalConnector.xsd

```
<!--
XML Schema for the NCL Language
```

```
This is NCL
Copyright: 2000-2005 LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/profiles/CausalConnector.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:causalConnectorFunctionality="http://www.ncl.org.br/NCL3.0/CausalConnectorFunctionality"
    xmlns:connectorBase="http://www.ncl.org.br/NCL3.0/ConnectorBase"
    xmlns:structure="http://www.ncl.org.br/NCL3.0/Structure"
    xmlns:import="http://www.ncl.org.br/NCL3.0/Import"
    xmlns:profile="http://www.ncl.org.br/NCL3.0/CausalConnectorProfile"
    targetNamespace="http://www.ncl.org.br/NCL3.0/CausalConnectorProfile"
    elementFormDefault="qualified" attributeFormDefault="unqualified">

  <!-- import the definitions in the modules namespaces -->

  <import namespace="http://www.ncl.org.br/NCL3.0/CausalConnectorFunctionality"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30CausalConnectorFunctionality.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/ConnectorBase"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30ConnectorBase.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/Structure"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30Structure.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL30/Import"
  schemaLocation="http://www.ncl.org.br/NCL30/modules/NCL30Import.xsd"/>

 <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
 <!-- Structure                                    -->
 <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
 <!-- extends ncl element -->

  <complexType name="nclType">
   <complexContent>
    <restriction base="structure:nclPrototype">
     <sequence>
      <element ref="structure:head" minOccurs="0" maxOccurs="1"/>
      <element ref="structure:body" minOccurs="0" maxOccurs="0"/>
     </sequence>
    </restriction>
   </complexContent>
  </complexType>

  <element name="ncl" type="profile:nclType" substitutionGroup="structure:ncl"/>

  <!-- extends head element -->

  <complexType name="headType">
   <complexContent>
    <extension base="structure:headPrototype">
     <all>
      <element ref="profile:connectorBase" />
     </all>
    </extension>
   </complexContent>
  </complexType>

  <element name="head" type="profile:headType" substitutionGroup="structure:head"/>

 <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
```



```
  <!-- XConnector                                  -->
  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
    <!-- extends connectorBase element -->

    <complexType name="connectorBaseType">
      <complexContent>
        <extension base="connectorBase:connectorBasePrototype">
          <choice minOccurs="0" maxOccurs="unbounded">
            <element ref="profile:importBase"/>
                <element ref="profile:causalConnector" />
          </choice>
        </extension>
      </complexContent>
    </complexType>

  <element name="connectorBase" type="profile:connectorBaseType" substitutionGroup="connectorBase:connectorBase"/>

  <element name="causalConnector" substitutionGroup="causalConnectorFunctionality:causalConnector"/>

  <element name="connectorParam" substitutionGroup="causalConnectorFunctionality:connectorParam"/>

  <element name="simpleCondition" substitutionGroup="causalConnectorFunctionality:simpleCondition"/>

  <element name="compoundCondition" substitutionGroup="causalConnectorFunctionality:compoundCondition"/>

  <element name="simpleAction" substitutionGroup="causalConnectorFunctionality:simpleAction"/>

  <element name="compoundAction" substitutionGroup="causalConnectorFunctionality:compoundAction"/>

  <element name="assessmentStatement" substitutionGroup="causalConnectorFunctionality:assessmentStatement"/>

  <element name="attributeAssessment" substitutionGroup="causalConnectorFunctionality:attributeAssessment"/>

  <element name="valueAssessment" substitutionGroup="causalConnectorFunctionality:valueAssessment"/>

  <element name="compoundStatement" substitutionGroup="causalConnectorFunctionality:compoundStatement"/>

  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <!-- ImportBase                                  -->
  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->

  <element name="importBase" substitutionGroup="import:importBase">

</schema>
```

7.3.4 Atributos e elementos do perfil NCL 3.0 DTV básico

Os elementos e seus atributos, utilizados no perfil NCL 3.0 DTV Básico, são apresentados nas Tabelas 39 a 55. Salienta-se que os atributos e conteúdos (elementos filhos) de elementos podem ser definidos no módulo em si ou no perfil NCL DTV Básico que agrupa os módulos. Os atributos obrigatórios estão sublinhados. Nas Tabelas 39 a 55, os seguintes símbolos são empregados: (?) opcional (zero ou uma ocorrência), (|) ou (*) zero ou mais ocorrências, (+) uma ou mais ocorrências.

Tabela 39 — Elementos e atributos do módulo *Structure* estendido utilizados no perfil DTV Básico

Elementos	Atributos	Conteúdo

ncl	*id, title, xmlns*	(head?, body?)
head		(importedDocumentBase? ruleBase?, regionBase*, descriptorBase?, connectorBase?),
body	*id*	(port| property| media|context|switch|link)*

Tabela 40 — Elementos e atributos do módulo *Layout* estendido utilizados no perfil DTV Básico

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

Tabela 41 — Elementos e atributos do módulo *Media* estendido utilizados no perfil DTV Básico

Elementos	Atributos	Conteúdo
media	*id, src, refer, newInstance, type, descriptor*	(area|property)*

Tabela 42 — Elementos e atributos do módulo *Context* estendido utilizados no perfil DTV Básico

Elementos	Atributos	Conteúdo
context	*id, refer*	(port|property|media|context|link|switch)*

Tabela 43 — Elementos e atributos do módulo *MediaContentAnchor* estendido utilizados no perfil DTV Básico

Elementos	Atributos	Conteúdo
area	*id, coords, begin, end, text, position, first, last, label*	vazio

Tabela 44 — Elementos e atributos do módulo *CompositeNodeInterface* estendido utilizados no perfil DTV Básico

Elementos	Atributos	Conteúdo
port	*id, component, interface*	vazio

Tabela 45 — Elementos e atributos do módulo *PropertyAnchor* estendido utilizados no perfil DTV Básico

Elementos	Atributos	Conteúdo
property	*name, value*	vazio

Tabela 46 — Elementos e atributos do módulo *SwitchInterface* estendido utilizados no perfil DTV Básico

Elementos	Atributos	Conteúdo
switchPort	*id*	mapping+
mapping	*component, interface*	vazio

Tabela 47 — Elementos e atributos do módulo *Descriptor* estendido utilizados no perfil DTV Básico

Elementos	Atributos	Conteúdo
descriptor	*id, player, explicitDur, region, freeze, moveLeft, moveRight, moveUp; moveDown, focusIndex, focusBorderColor; focusBorderWidth; focusBorderTransparency, focusSrc,focusSelSrc, selBorderColor*	(descriptorParam)*
descriptorParam	*name, value*	vazio
descriptorBase	*id*	(importBase | descriptor | descriptorSwitch)+

Tabela 48 — Elementos e atributos do módulo *Linking* estendido utilizados no perfil DTV Básico

Elementos	Atributos	Conteúdo
bind	*role, component, interface, descriptor*	(bindParam)*
bindParam	*name, value*	vazio
linkParam	*name, value*	vazio
link	*id, xconnector*	(linkParam*, bind+)

Tabela 49 — Elementos e atributos do módulo *CausalConnectorFunctionality* estendido utilizados no perfil DTV Básico

Elementos	Atributos	Conteúdo
causalConnector	*id*	(connectorParam*, (simpleCondition | compoundCondition), (simpleAction |

		compoundAction))
connectorParam	*name, type*	vazio
simpleCondition	*role, delay, eventType, key, transition, min, max, qualifier*	vazio
compoundCondition	*operator, delay*	((simpleCondition | compoundCondition)+, (assessmentStatement | compoundStatement)*)
simpleAction	*role, delay, eventType, actionType, value, min, max, qualifier, repeat, repeatDelay*	vazio
compoundAction	*operator, delay*	(simpleAction | compoundAction)+
assessmentStatement	*comparator*	(attributeAssessment, (attributeAssessment | valueAssessment))
attributeAssessment	*role, eventType, key, attributeType, offset*	vazio
valueAssessment	*value*	vazio
compoundStatement	*operator, isNegated*	(assessmentStatement | compoundStatement)+

Tabela 50 — Elementos e atributos do módulo *ConnectorBase* estendido utilizados no perfil DTV Básico

Elementos	Atributos	Conteúdo
connectorBase	*id*	(importBase|causalConnector)*

Tabela 51 — Elementos e atributos do módulo *TestRule* estendido utilizados no perfil DTV Básico

Elementos	Atributos	Conteúdo
ruleBase	*id*	(importBase|rule|compositeRule)+
rule	*id, var, comparator, value*	vazio
compositeRule	*id, operator*	(rule | compositeRule)+

Tabela 52 — Elementos e atributos do módulo *TestRuleUse* estendido utilizados no perfil DTV Básico

Elementos	Atributos	Conteúdo
bindRule	*constituent, rule*	vazio

Tabela 53 — Elementos e atributos do módulo *ContentControl* estendido utilizados no perfil DTV Básico

Elementos	Atributos	Conteúdo
switch	*id, refer*	(defaultComponent?,(switchPort| bindRule|media| context | switch)*)
defaultComponent	*component*	vazio

Tabela 54 — Elementos e atributos do módulo *DescriptorControl* estendido utilizados no perfil DTV Básico

Elementos	Atributos	Conteúdo
descriptorSwitch	*id*	(defaultDescriptor?, (bindRule | descriptor)*)
defaultDescriptor	*descriptor*	vazio

Tabela 55 — Elementos e atributos do módulo *Import* estendido utilizados no perfil DTV Básico

Elementos	Atributos	Conteúdo
importBase	*alias, documentURI, region*	vazio
importedDocumentBase	*id*	(importNCL)+
importNCL	*alias, documentURI,*	vazio

7.3.5 Esquema do perfil NCL 3.0 DTV Básico

NCL30BDTV.xsd

```
<!--
XML Schema for the NCL Language

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/NCL3.0/profiles/NCL30BDTV.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006
-->

<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:compositeInterface="http://www.ncl.org.br/NCL3.0/CompositeNodeInterface"
  xmlns:causalConnectorFunctionality="http://www.ncl.org.br/NCL3.0/CausalConnectorFunctionality"
  xmlns:connectorBase="http://www.ncl.org.br/NCL3.0/ConnectorBase"
  xmlns:contentControl="http://www.ncl.org.br/NCL3.0/ContentControl"
  xmlns:context="http://www.ncl.org.br/NCL3.0/Context"
  xmlns:descriptor="http://www.ncl.org.br/NCL3.0/Descriptor"
  xmlns:entityReuse="http://www.ncl.org.br/NCL3.0/EntityReuse"
  xmlns:extendedEntityReuse="http://www.ncl.org.br/NCL3.0/ExtendedEntityReuse"
  xmlns:descriptorControl="http://www.ncl.org.br/NCL3.0/DescriptorControl"
  xmlns:import="http://www.ncl.org.br/NCL3.0/Import"
  xmlns:keyNavigation="http://www.ncl.org.br/NCL3.0/KeyNavigation"
  xmlns:layout="http://www.ncl.org.br/NCL3.0/Layout"
  xmlns:linking="http://www.ncl.org.br/NCL3.0/Linking"
  xmlns:media="http://www.ncl.org.br/NCL3.0/Media"
  xmlns:mediaAnchor="http://www.ncl.org.br/NCL3.0/MediaContentAnchor"
  xmlns:propertyAnchor="http://www.ncl.org.br/NCL3.0/PropertyAnchor"
  xmlns:structure="http://www.ncl.org.br/NCL3.0/Structure"
  xmlns:switchInterface="http://www.ncl.org.br/NCL3.0/SwitchInterface"
  xmlns:testRule="http://www.ncl.org.br/NCL3.0/TestRule"
  xmlns:testRuleUse="http://www.ncl.org.br/NCL3.0/TestRuleUse"
  xmlns:timing="http://www.ncl.org.br/NCL3.0/Timing"
```

```
xmlns:profile="http://www.ncl.org.br/NCL3.0/BDTVProfile"
targetNamespace="http://www.ncl.org.br/NCL3.0/BDTVProfile"
elementFormDefault="qualified" attributeFormDefault="unqualified" >

<!-- import the definitions in the modules namespaces -->
<import namespace="http://www.ncl.org.br/NCL3.0/CompositeNodeInterface"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30CompositeNodeInterface.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/CausalConnectorFunctionality"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30CausalConnectorFunctionality.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/ConnectorBase"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30ConnectorBase.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/ContentControl"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30ContentControl.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/Context"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30Context.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/Descriptor"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30Descriptor.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/DescriptorControl"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30DescriptorControl.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/EntityReuse"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30EntityReuse.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/ExtendedEntityReuse"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30ExtendedEntityReuse.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/Import"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30Import.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/KeyNavigation"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30KeyNavigation.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/Layout"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30Layout.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/Linking"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30Linking.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/Media"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30Media.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/MediaContentAnchor"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30MediaContentAnchor.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/PropertyAnchor"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30PropertyAnchor.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/Structure"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30Structure.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/SwitchInterface"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30SwitchInterface.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/TestRule"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30TestRule.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/TestRuleUse"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30TestRuleUse.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/Timing"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30Timing.xsd"/>

<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- Structure                                               -->
<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- extends ncl element -->

<element name="ncl" substitutionGroup="structure:ncl"/>

<!-- extends head element -->

<complexType name="headType">
  <complexContent>
    <extension base="structure:headPrototype">
      <sequence>
        <element ref="profile:importedDocumentBase" minOccurs="0" maxOccurs="1"/>
```

```
      <element ref="profile:ruleBase" minOccurs="0" maxOccurs="1"/>
      <element ref="profile:regionBase" minOccurs="0" maxOccurs="unbounded"/>
      <element ref="profile:descriptorBase" minOccurs="0" maxOccurs="1"/>
      <element ref="profile:connectorBase" minOccurs="0" maxOccurs="1"/>
    </sequence>
  </extension>
 </complexContent>
</complexType>

<element name="head" type="profile:headType" substitutionGroup="structure:head"/>

<!-- extends body element -->

<complexType name="bodyType">
 <complexContent>
  <extension base="structure:bodyPrototype">
    <choice minOccurs="0" maxOccurs="unbounded">
      <group ref="profile:contextInterfaceElementGroup"/>
      <element ref="profile:media"/>
      <element ref="profile:context"/>
      <element ref="profile:switch"/>
      <element ref="profile:link"/>
    </choice>
  </extension>
 </complexContent>
</complexType>

<element name="body" type="profile:bodyType" substitutionGroup="structure:body"/>

<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- Layout                                    -->
<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- extends regionBase element -->

<complexType name="regionBaseType">
 <complexContent>
  <extension base="layout:regionBasePrototype">
    <choice minOccurs="1" maxOccurs="unbounded">
      <element ref="profile:region"/>
      <element ref="profile:importBase"/>
    </choice>
  </extension>
 </complexContent>
</complexType>

<complexType name="regionType">
 <complexContent>
  <extension base="layout:regionPrototype">
  </extension>
 </complexContent>
</complexType>

<element name="regionBase" type="profile:regionBaseType" substitutionGroup="layout:regionBase"/>
<element name="region" type="profile:regionType" substitutionGroup="layout:region"/>

<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- Media                                     -->
<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- extends Media elements -->
```

```
<!-- media interface element groups -->
<group name="mediaInterfaceElementGroup">
  <choice>
    <element ref="profile:area"/>
    <element ref="profile:property"/>
  </choice>
</group>

<complexType name="mediaType">
  <complexContent>
    <extension base="media:mediaPrototype">
      <choice minOccurs="0" maxOccurs="unbounded">
        <group ref="profile:mediaInterfaceElementGroup"/>
      </choice>
      <attributeGroup ref="descriptor:descriptorAttrs"/>
      <attributeGroup ref="entityReuse:entityReuseAttrs"/>
      <attributeGroup ref="extendedEntityReuse:extendedEntityReuseAttrs"/>
    </extension>
  </complexContent>
</complexType>

<element name="media" type="profile:mediaType" substitutionGroup="media:media"/>

<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- Context                                                    -->
<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- extends context element -->

<!-- composite node interface element groups -->
<group name="contextInterfaceElementGroup">
  <choice>
    <element ref="profile:port"/>
    <element ref="profile:property"/>
  </choice>
</group>

<complexType name="contextType">
  <complexContent>
    <extension base="context:contextPrototype">
      <choice minOccurs="0" maxOccurs="unbounded">
        <group ref="profile:contextInterfaceElementGroup"/>
        <element ref="profile:media"/>
        <element ref="profile:context"/>
        <element ref="profile:link"/>
        <element ref="profile:switch"/>
      </choice>
      <attributeGroup ref="entityReuse:entityReuseAttrs"/>
    </extension>
  </complexContent>
</complexType>

<element name="context" type="profile:contextType" substitutionGroup="context:context"/>

<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- MediaContentAnchor                                         -->
<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- extends area element -->

<complexType name="componentAnchorType">
  <complexContent>
    <extension base="mediaAnchor:componentAnchorPrototype">
```



```
    </extension>
  </complexContent>
</complexType>

<element name="area" type="profile:componentAnchorType" substitutionGroup="mediaAnchor:area"/>

<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- CompositeNodeInterface                                    -->
<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- extends port element -->

<complexType name="compositeNodePortType">
  <complexContent>
    <extension base="compositeInterface:compositeNodePortPrototype">
    </extension>
  </complexContent>
</complexType>

<element name="port" type="profile:compositeNodePortType" substitutionGroup="compositeInterface:port"/>

<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- PropertyAnchor                                    -->
<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- extends property element -->

<complexType name="propertyAnchorType">
  <complexContent>
    <extension base="propertyAnchor:propertyAnchorPrototype">
    </extension>
  </complexContent>
</complexType>

<element name="property" type="profile:propertyAnchorType" substitutionGroup="propertyAnchor:property"/>

<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- SwitchInterface                                    -->
<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- extends switchPort element -->

<complexType name="switchPortType">
  <complexContent>
    <extension base="switchInterface:switchPortPrototype">
    </extension>
  </complexContent>
</complexType>

<element name="mapping" substitutionGroup="switchInterface:mapping"/>
<element name="switchPort" type="profile:switchPortType" substitutionGroup="switchInterface:switchPort"/>

<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- Descriptor                                    -->
<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->

<!-- substitutes descriptorParam element -->

<element name="descriptorParam" substitutionGroup="descriptor:descriptorParam"/>


<!-- extends descriptor element -->
```

```
<complexType name="descriptorType">
  <complexContent>
    <extension base="descriptor:descriptorPrototype">
      <attributeGroup ref="layout:regionAttrs"/>
      <attributeGroup ref="timing:explicitDurAttrs"/>
      <attributeGroup ref="timing:freezeAttrs"/>
      <attributeGroup ref="keyNavigation:keyNavigationAttrs"/>
    </extension>
  </complexContent>
</complexType>

<element name="descriptor" type="profile:descriptorType" substitutionGroup="descriptor:descriptor"/>

<!-- extends descriptorBase element -->
<complexType name="descriptorBaseType">
  <complexContent>
    <extension base="descriptor:descriptorBasePrototype">
      <choice minOccurs="1" maxOccurs="unbounded">
        <element ref="profile:importBase"/>
        <element ref="profile:descriptor"/>
        <element ref="profile:descriptorSwitch"/>
      </choice>
    </extension>
  </complexContent>
</complexType>

<element name="descriptorBase" type="profile:descriptorBaseType" substitutionGroup="descriptor:descriptorBase"/>

<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- Linking                                                   -->
<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- substitutes linkParam and bindParam elements -->
<element name="linkParam" substitutionGroup="linking:linkParam"/>
<element name="bindParam" substitutionGroup="linking:bindParam"/>

<!-- extends bind element and link element, as a consequence-->

<complexType name="bindType">
  <complexContent>
    <extension base="linking:bindPrototype">
      <attributeGroup ref="descriptor:descriptorAttrs"/>
    </extension>
  </complexContent>
</complexType>

<element name="bind" type="profile:bindType" substitutionGroup="linking:bind"/>

<!-- extends link element -->
<complexType name="linkType">
  <complexContent>
    <extension base="linking:linkPrototype">
    </extension>
  </complexContent>
</complexType>

<element name="link" type="profile:linkType" substitutionGroup="linking:link"/>

<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- Connector                                                   -->
<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- extends connectorBase element -->
```



```
  <complexType name="connectorBaseType">
    <complexContent>
      <extension base="connectorBase:connectorBasePrototype">
        <choice minOccurs="0" maxOccurs="unbounded">
          <element ref="profile:importBase"/>
              <element ref="profile:causalConnector" />
        </choice>
      </extension>
    </complexContent>
  </complexType>

<element name="connectorBase" type="profile:connectorBaseType" substitutionGroup="connectorBase:connectorBase"/>

<element name="causalConnector" substitutionGroup="causalConnectorFunctionality:causalConnector"/>

<element name="connectorParam" substitutionGroup="causalConnectorFunctionality:connectorParam"/>

<element name="simpleCondition" substitutionGroup="causalConnectorFunctionality:simpleCondition"/>

<element name="compoundCondition" substitutionGroup="causalConnectorFunctionality:compoundCondition"/>

<element name="simpleAction" substitutionGroup="causalConnectorFunctionality:simpleAction"/>

<element name="compoundAction" substitutionGroup="causalConnectorFunctionality:compoundAction"/>

<element name="assessmentStatement" substitutionGroup="causalConnectorFunctionality:assessmentStatement"/>

<element name="attributeAssessment" substitutionGroup="causalConnectorFunctionality:attributeAssessment"/>

<element name="valueAssessment" substitutionGroup="causalConnectorFunctionality:valueAssessment"/>

<element name="compoundStatement" substitutionGroup="causalConnectorFunctionality:compoundStatement"/>

<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- TestRule                                               -->
<!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
<!-- extends rule element -->
<complexType name="ruleType">
  <complexContent>
    <extension base="testRule:rulePrototype">
    </extension>
  </complexContent>
</complexType>

<element name="rule" type="profile:ruleType" substitutionGroup="testRule:rule"/>

<!-- extends compositeRule element -->
<complexType name="compositeRuleType">
  <complexContent>
    <extension base="testRule:compositeRulePrototype">
    </extension>
  </complexContent>
</complexType>

<element name="compositeRule" type="profile:compositeRuleType" substitutionGroup="testRule:compositeRule"/>

<!-- extends ruleBase element -->
<complexType name="ruleBaseType">
```



```
    <complexContent>
      <extension base="testRule:ruleBasePrototype">
        <choice minOccurs="1" maxOccurs="unbounded">
          <element ref="profile:importBase"/>
          <element ref="profile:rule"/>
          <element ref="profile:compositeRule"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>

  <element name="ruleBase" type="profile:ruleBaseType" substitutionGroup="testRule:ruleBase"/>

  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <!-- TestRuleUse                                                   -->
  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <!-- extends bindRule element -->
  <complexType name="bindRuleType">
    <complexContent>
      <extension base="testRuleUse:bindRulePrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="bindRule" type="profile:bindRuleType" substitutionGroup="testRuleUse:bindRule"/>

  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <!-- ContentControl                                                -->
  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <!-- extends switch element -->

  <!-- switch interface element groups -->
  <group name="switchInterfaceElementGroup">
    <choice>
      <element ref="profile:switchPort"/>
    </choice>
  </group>

  <!-- extends defaultComponent element -->
  <complexType name="defaultComponentType">
    <complexContent>
      <extension base="contentControl:defaultComponentPrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="defaultComponent" type="profile:defaultComponentType"
substitutionGroup="contentControl:defaultComponent"/>

  <complexType name="switchType">
    <complexContent>
      <extension base="contentControl:switchPrototype">
          <choice minOccurs="0" maxOccurs="unbounded">
            <group ref="profile:switchInterfaceElementGroup"/>
            <element ref="profile:bindRule"/>
            <element ref="profile:switch"/>
            <element ref="profile:media"/>
            <element ref="profile:context"/>
          </choice>
        <attributeGroup ref="entityReuse:entityReuseAttrs"/>
      </extension>
    </complexContent>
```

```
  </complexType>

  <element name="switch" type="profile:switchType" substitutionGroup="contentControl:switch"/>

  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <!-- DescriptorControl                                                   -->
  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <!-- extends defaultDescriptor element -->
  <complexType name="defaultDescriptorType">
    <complexContent>
      <extension base="descriptorControl:defaultDescriptorPrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="defaultDescriptor" type="profile:defaultDescriptorType"
substitutionGroup="descriptorControl:defaultDescriptor"/>

  <!-- extends descriptorSwitch element -->

  <complexType name="descriptorSwitchType">
    <complexContent>
      <extension base="descriptorControl:descriptorSwitchPrototype">
          <choice minOccurs="0" maxOccurs="unbounded">
            <element ref="profile:descriptor"/>
            <element ref="profile:bindRule"/>
          </choice>
      </extension>
    </complexContent>
  </complexType>

  <element name="descriptorSwitch" type="profile:descriptorSwitchType"
substitutionGroup="descriptorControl:descriptorSwitch"/>

  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <!-- Timing                                                   -->
  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->


  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <!-- Import                                                   -->
  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <complexType name="importBaseType">
    <complexContent>
      <extension base="import:importBasePrototype">
      </extension>
    </complexContent>
  </complexType>

  <complexType name="importNCLType">
    <complexContent>
      <extension base="import:importNCLPrototype">
      </extension>
    </complexContent>
  </complexType>

  <complexType name="importedDocumentBaseType">
    <complexContent>
      <extension base="import:importedDocumentBasePrototype">
```

```
    </extension>
   </complexContent>
  </complexType>

  <element name="importBase" type="profile:importBaseType" substitutionGroup="import:importBase"/>

  <element name="importNCL" type="profile:importNCLType" substitutionGroup="import:importNCL"/>
  <element name="importedDocumentBase" type="profile:importedDocumentBaseType"
substitutionGroup="import:importedDocumentBase"/>

  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <!-- EntityReuse                                    -->
  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->

  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <!-- ExtendedEntityReuse                                    -->
   <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->

  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <!-- KeyNavigation                                    -->
  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->

</schema>
```

8 Objetos de mídia em apresentações NCL

8.1 Implementação modular de Ginga-NCL

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) pode ser carregado para o controle do objeto e de seus eventos NCL. Um exibidor de mídia (ou seu adaptador) deve obrigatoriamente ser capaz de receber os comandos de apresentação, controlar as máquinas de estado dos eventos do objeto de mídia controlado e responder às demandas do formatador.

Para favorecer a incorporação de *players* de terceiros dentro da implementação da arquitetura Ginga, um projeto modular do Ginga-NCL é recomendado, para separar os *players* da máquina de apresentação (formatador NCL).

A Figura 4 sugere uma organização modular para a implementação Ginga-NCL. Os exibidores são módulos *plug-in* da máquina de apresentação. Por ser interessante utilizar os *players* já existentes, que possam ter interfaces proprietárias que não sejam compatíveis com as exigidas pela máquina de apresentação, será necessário desenvolver módulos para fazer as adaptações necessárias. Nesse caso, o exibidor será constituído de um adaptador além do exibidor não-conforme em si.

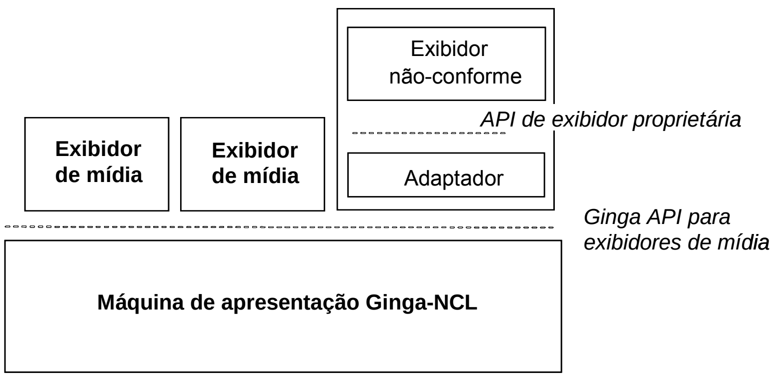

Figura 4 — API para integrar os *players* à implementação da máquina de apresentação NCL

NOTA Como a arquitetura e implementação Ginga-NCL é uma escolha de cada receptor, esta subseção não tem a intenção de padronizar a sintaxe da API da máquina de apresentação. O objetivo desta seção é apenas definir o comportamento esperado do exibidor de mídia quando se está controlando objetos que fazem parte de um documento NCL. As subseções 8.2 a 8.4 dizem respeito aos exibidores de mídia para elemento <media> cujo conteúdo não é código procedural, isto é, elementos <media> cujos tipos são diferentes de application/x-ginga-NCLua" e "application/x-ginga-NCLet". A Seção 8.5 diz respeito aos exibidores de mídia (máquinas Lua e Java) para os elementos <media> cujo conteúdo são códigos procedurais (Lua e Java).

8.2 Comportamento esperado dos exibidores de mídia

8.2.1 Instrução *start*

Antes de enviar a instrução *start*, recomenda-se que o formatador encontre 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* do elemento <descriptor> associado com o objeto de mídia a ser exibido. Se esse atributo não for especificado, o formatador deve obrigatoriamente levar em conta o atributo *type* do elemento <media>. Se esse atributo também não for especificado, o formatador deve obrigatoriamente considerar a extensão do arquivo especificado no atributo *src* do elemento <media>.

A instrução *start* emitida por um formatador deve obrigatoriamente informar ao exibidor de mídia os seguintes parâmetros: o objeto de mídia a ser controlado, seu descritor associado, uma lista de eventos (apresentação, seleção ou atribuição) 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.

Um objeto de mídia deve obrigatoriamente ser derivado de um elemento <media>, cujo atributo *src* deve obrigatoriamente ser usado, pelo exibidor de mídia, para localizar o conteúdo e iniciar a 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 deve obrigatoriamente encerrar a operação de iniciação sem realizar nenhuma ação.

O descritor a ser utilizado deve obrigatoriamente ser escolhido 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 descritor do elemento <bind>), o descritor resultante informado pelo formatador deve obrigatoriamente mesclar 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> deve obrigatoriamente sobrepor os dados do descritor da <media>. Se o elemento <bind> não contiver um descritor explícito, o descritor informado pelo formatador deve obrigatoriamente ser o descritor especificado pelo elemento <media>, se o atributo tiver sido especificado. Caso contrário, um descritor *default* para o tipo de <media> específico deve obrigatoriamente ser escolhido pelo formatador.

Convém que a lista de eventos a serem monitorados por um exibidor de mídia também seja computada pelo formatador, levando em conta a especificação do documento NCL. Ele deve obrigatoriamente checar todos os elos dos quais participa o objeto de mídia e o descritor resultante. Ao computar os eventos a serem monitorados, o formatador deve obrigatoriamente considerar a perspectiva do objeto de mídia, isto é, o caminho dos vários elementos <body> e <context> para alcançar em profundidade o elemento <media> correspondente. Convém que apenas elos contidos nesses elementos <body> e <context> sejam 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 deve ser imediatamente iniciada (isto é, ele comanda o exibidor para pular para o *beginning-time* + *offset-time*). Obviamente, o valor do *offset-time* deve obrigatoriamente ser menor que a duração do evento principal.

Se o *offset-time* for maior que zero, o exibidor de mídia deve obrigatoriamente colocar o evento principal no estado ocorrendo (*occurring*), mas a transição de início (*starts*) do evento obrigatoriamente não deve ser notificada. Se o *offset-time* for zero, o exibidor de mídia deve obrigatoriamente colocar o evento principal no estado ocorrendo e notificar a ocorrência da transição de início. Os eventos que teriam seus tempos de término anteriores ao tempo de início do evento principal e eventos que teriam seus tempos de início após o tempo de término do evento principal não precisam ser monitorados pelo exibidor de mídia (convém que o formatador faça essa verificação quando construir a lista de eventos monitorados).

Os eventos monitorados que têm tempos de início antes do tempo de início do evento principal e tempos de término após o tempo de início do evento principal devem obrigatoriamente ser colocados no estado de ocorrendo, mas suas transições de início obrigatoriamente não devem ser notificadas (elos que dependem dessa transição obrigatoriamente não devem ser disparados). Os eventos monitorados que teriam seus tempos de término após o tempo de início do evento principal, mas antes do tempo de início (*beginning-time* + *offset-time*), devem obrigatoriamente ter seu atributo *occurrences* incrementado, mas as transições de início e término (stops) não devem ser notificadas. Os eventos monitorados que têm seu tempo de início antes do momento de início (*beginning time* + *offset-time*) e tempo de término após o tempo de início devem obrigatoriamente ser colocados no estado de ocorrendo, mas a transição de início correspondente obrigatoriamente não deve ser notificada.

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. Esse parâmetro deve ser considerado apenas se o parâmetro de *offset-time* for igual a “zero”.

Se um exibidor de mídia receber uma instrução de *start* para um objeto já sendo apresentado (pausado ou não), ele deve obrigatoriamente ignorar a instrução e manter o controle da apresentação em andamento. Neste caso, o elemento <simpleAction> que causou a instrução *start* não deve causar qualquer transição na máquina de estados do evento a ele associado.

8.2.2 Instrução *stop*

A instrução *stop* precisa apenas identificar um objeto de mídia que já está sendo controlado. Identificar o objeto de mídia significa identificar o elemento <media>, o descritor correspondente e a perspectiva do objeto de mídia. Assim, se um elemento <simpleAction> com o *actionType* igual a “stop” é ligado por um elo a uma interface de nó, a interface deve ser ignorada quando a ação for executada.

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* deve obrigatoriamente ser 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 devem obrigatoriamente transitar para o estado *sleeping* e suas transições *stops* devem obrigatoriamente ser notificadas.

Os eventos monitorados no estado *occurring* ou *paused* com tempo de término posterior ao tempo de término do evento principal devem obrigatoriamente ser colocados no estado *sleeping*, mas suas transições *stops* não devem ser notificadas e seu atributo *occurrences* não deve ser incrementado. A apresentação do conteúdo do objeto deve obrigatoriamente ser parada. Se o atributo *repetitions* do evento for maior que zero, deve obrigatoriamente ser diminuído em um e a apresentação do evento principal deve obrigatoriamente reiniciar após o tempo entre repetições (o tempo de retardo entre repetições deve obrigatoriamente ter sido 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 deve obrigatoriamente ser removida.

8.2.3 Instrução *abort*

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 deve ser ignorada quando a ação for executada.

Se o objeto não estiver sendo apresentado e não estiver esperando para ser apresentado após uma instrução de *start* atrasada, a instrução *abort* deve obrigatoriamente ser ignorada. Se o objeto estiver sendo apresentado, o evento principal e todos os eventos monitorados no estado *occurring* ou *paused* devem obrigatoriamente transitar para o estado *sleeping*, e suas transições *aborts* devem obrigatoriamente ser notificadas. Qualquer apresentação de conteúdo deve obrigatoriamente parar.

Se o atributo *repetitions* do evento for maior que zero, deve obrigatoriamente ser colocado em zero e a apresentação do objeto de mídia não deve ser 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* deve obrigatoriamente ser removida.

8.2.4 Instrução *pause*

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 deve ser ignorada quando a ação for executada.

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 deve obrigatoriamente ser ignorada. Se o objeto estiver sendo apresentado, o evento principal e todos os eventos monitorados no estado *occurring* devem obrigatoriamente transitar para o estado *paused* e suas transições *pauses* devem obrigatoriamente ser notificadas. A apresentação do objeto deve obrigatoriamente ser pausada e o tempo de pausa decorrido obrigatoriamente não deve ser considerado como parte da duração do objeto. Como exemplo, se um objeto tiver duração explícita de 30 s, e após 25 s for pausado, mesmo se o objeto permanecer pausado por 5 min, após o reinício, o evento principal do objeto deve obrigatoriamente permanecer ocorrendo por 5 s. Se o evento principal ainda não estiver ocorrendo porque o exibidor de mídia está esperando pelo retardo de início, o objeto de mídia deve obrigatoriamente esperar por uma instrução *resume* para continuar aguardando o retardo de início.

8.2.5 Instrução *resume*

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 deve ser ignorada quando a ação for executada.

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 deve obrigatoriamente ser ignorada.

Se o exibidor de mídia estiver pausado aguardando o retardo de início, ele deve obrigatoriamente retomar a 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* devem obrigatoriamente ser colocados no estado *occurring* e suas transições *resumes* devem obrigatoriamente ser notificadas.

8.2.6 Instrução *set*

A instrução *set* pode ser aplicada a um objeto independente do fato de estar sendo ou não apresentado (nesse último caso, embora o objeto não esteja sendo apresentado, seu exibidor de mídia já deve obrigatoriamente estar instanciado). No primeiro caso, a instrução *set* precisa identificar o objeto de mídia sendo controlado, um evento de atribuição monitorado e um valor a ser atribuído ao atributo que definiu o evento. No segundo caso, a instrução também deve obrigatoriamente identificar o elemento <descriptor> que será usado quando da apresentação do objeto (como é feito para a instrução *start*).

Ao imputar um valor ao atributo, o exibidor de mídia deve obrigatoriamente transitar 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*.

Para cada evento de atribuição monitorado, se o exibidor de mídia alterar, por sua própria conta, o valor correspondente de um atributo, deve obrigatoriamente proceder como se tivesse recebido uma instrução de ajuste externa.

8.2.7 Instrução *addEvent*

A instrução *addEvent* é emitida no caso de edição ao vivo de um documento NCL (ver Seção 11). A instrução precisa apenas identificar um objeto de mídia que já esteja sendo controlado e um novo evento que deva obrigatoriamente ser incluído para ser monitorado.

Todas as regras aplicadas à interseção de eventos monitorados com o evento principal devem obrigatoriamente ser 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 deve obrigatoriamente ser colocado no mesmo estado do evento principal (*occurring* ou *paused*), sem notificar a transição correspondente.

8.2.8 Instrução *removeEvent*

A instrução *removeEvent* também é emitida no caso de edição ao vivo de documento NCL. A instrução precisa identificar um objeto de mídia que já esteja sendo controlado e um novo evento que não é mais recomendável que seja controlado. O estado do evento deve obrigatoriamente ser colocado no estado *sleeping* sem gerar nenhuma transição.

8.2.9 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 deve obrigatoriamente transitar o evento para o estado *sleeping* e notificar a transição *stops*. O mesmo deve obrigatoriamente ser feito para 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. Os eventos no estado *occurring* com tempo de término posterior ao tempo de término do evento principal devem obrigatoriamente ser 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 deve obrigatoriamente terminar.

8.3 Comportamento esperado dos exibidores de mídia após instruções aplicadas aos objetos de composição

8.3.1 Elos referindo nós de composição

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 interface seja informada). Como normalmente ocorre, a máquina de estado

do evento de apresentação definido pelo nó de composição deve obrigatoriamente ser controlada como especificado em 7.2.8.

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. Recomenda-se que o valor do atributo derive da máquina de estado do evento de apresentação definido pelo nó de composição.

Se, contudo, 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), a instrução deve obrigatoriamente ser refletida nas máquinas de estado de evento dos nós filhos da composição.

8.3.2 Iniciando a apresentação de um contexto

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 deve obrigatoriamente ser aplicada a todos os eventos de apresentação mapeados pelas portas dos elementos <context> ou <body>.

Se o autor quiser iniciar a apresentação usando uma porta específica, ele também deve obrigatoriamente indicar o *id* de <port> como valor de *interface* <bind>.

8.3.3 Parando a apresentação de um contexto

Se um elemento <context> ou <body> participar em um papel (*role*) *action* cujo tipo de ação é "stop", quando essa ação for acionada, a instrução *stop* também deve obrigatoriamente ser aplicada a todos os eventos de apresentação dos nós filhos da composição.

Se a composição contiver elos sendo avaliados (ou com sua avaliação pausada), as avaliações devem obrigatoriamente ser suspensas e obrigatoriamente nenhuma ação deve ser acionada.

8.3.4 Abortando a apresentação de um contexto

Se um elemento <context> ou <body> participar em um papel (*role*) *action* cujo tipo de ação é "abort", quando essa ação for acionada, a instrução *abort* também deve obrigatoriamente ser aplicada a todos os eventos de apresentação nós filhos da composição.

Se a composição contiver elos sendo avaliados (ou com sua avaliação pausada), as avaliações devem obrigatoriamente ser suspensas e obrigatoriamente nenhuma ação deve ser acionada.

8.3.5 Pausando a apresentação de um contexto

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* também deve obrigatoriamente ser aplicada a todos os eventos de apresentação nós filhos da composição que estejam no estado *occurring*.

Se a composição contiver elos sendo avaliados, todas as avaliações devem obrigatoriamente 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 *paused* na composição for emitida, esses nós devem obrigatoriamente ser identificados, porque, se a composição receber uma instrução *resume*, esses eventos obrigatoriamente não devem ser retomados.

8.3.6 Retomando a apresentação de um contexto

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* também deve obrigatoriamente ser aplicada a todos os eventos de apresentação nós filhos da composição que estejam no estado *paused*, exceto aqueles que já estavam pausados antes da composição ser pausada.

Se a composição contiver elos com avaliações pausadas, elas devem obrigatoriamente ser retomadas.

8.4 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 deve obrigatoriamente entrar 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 deve obrigatoriamente estar no estado *sleeping*.

Os nós de composição não precisam inferir transições *aborts* a partir de seus nós-filhos. Essas transições nos eventos de apresentação de nós de composição devem obrigatoriamente ocorrer apenas quando instruções são aplicadas diretamente ao seu evento de apresentação (ver 8.3).

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* obrigatoriamente não deve entrar no estado *occurring*.

8.5 Comportamento esperado dos exibidores procedurais em aplicativos NCL

Objetos procedurais podem ser inseridos em documentos NCL, trazendo capacidades computacionais adicionais aos documentos declarativos. A forma de adicionar objetos procedurais em documentos NCL é definir um elemento <media> cujo conteúdo (localizado pelo atributo *src*) é o código procedural a ser executado. Os perfis EDTV e BDTV da NCL 3.0 permitem que dois tipos de mídia sejam associados com o elemento <media>: application/x-ginga-NCLua, para códigos procedurais Lua (extensão de arquivo .lua); e application/x-ginga-NCLet, para códigos procedurais Java (Xlet) (extensão de arquivo .class ou .jar).

Autores podem definir elos NCL para iniciar, parar, pausar, retomar ou abortar a execução de um código procedural. Um exibidor procedural (máquina de execução da linguagem) deve obrigatoriamente prover a interface do ambiente de execução procedural com o formatador NCL.

Analogamente ao realizado pelos exibidores de conteúdos de mídia convencional, os exibidores procedurais devem obrigatoriamente controlar as máquinas de estado dos eventos associados com o nó procedural NCL (NCLua ou NCLet). Como exemplo, se um código terminar sua execução, o exibidor deve obrigatoriamente gerar a transição *stops* na máquina de estado de apresentação do evento correspondente à execução procedural.

NCL permite que a execução do código procedural seja sincronizada com outros objetos NCL (procedurais ou não). Um elemento <media> contendo um código procedural também pode definir âncoras (através de elementos <area>) e propriedades (através de elementos <property>).

O código procedural pode ser associado a elementos <area> (através do atributo *label*). Se elos externos iniciarem, pararem, pausarem, retomarem ou abortarem a apresentação da âncora, *callbacks* no código

procedural devem obrigatoriamente ser disparados. A forma como esses *callbacks* são definidos é responsabilidade de cada código procedural associado com o objeto procedural NCL.

Por outro lado, o código procedural pode também comandar o início, parada, pausa, retomada ou aborto dessas âncoras, através de uma API oferecida pela linguagem procedural. Essas transições podem ser utilizadas como condições de elos NCL para disparar ações em outros objetos NCL do mesmo documento. Assim, uma sincronização de duas vias pode ser estabelecida entre o código procedural e o restante do documento NCL.

A outra forma que o código procedural pode ser sincronizado com outros objetos NCL é através de elementos <property>. Um elemento <property> definido como filho de um elemento <media>, representando um código procedural, pode ser mapeado para um código de função (ou método) ou para um atributo do código. Quando é mapeado para um código de função, uma ação de elo "set" aplicada à propriedade deve obrigatoriamente causar a execução da função com os valores atribuídos interpretados como parâmetros de entrada da função. O atributo *name* do elemento <property> deve obrigatoriamente ser utilizado para identificar a função do código procedural. Quando o elemento <property> é mapeado para um atributo do código procedural, a ação "set" deve obrigatoriamente atribuir o valor ao atributo. Como de costume, a máquina de estado de evento associada à propriedade deve ser controlada pelo exibidor procedural.

Um elemento <property> definido como filho de um elemento <media> representando um código procedural, também pode estar associado a um *assessment role* de um elo NCL. Nesse caso, o formatador NCL deve obrigatoriamente questionar o valor da propriedade para avaliar a expressão do elo. Se o elemento <property> for mapeado para um atributo de código, seu valor deve obrigatoriamente ser retornado pelo exibidor procedural ao formatador NCL. Se o elemento <property> for mapeado para uma função de código, a função deve obrigatoriamente ser chamada e o valor de seu resultado deve obrigatoriamente ser retornado pelo exibidor procedural ao formatador NCL.

A instrução *start* emitida por um formatador deve obrigatoriamente informar ao exibidor procedural os seguintes parâmetros: o objeto procedural a ser controlado, seu descritor associado, uma lista de eventos (definidos pelos elementos <area> e <property>, filhos do elemento <media> que define o objeto procedural) que precisam ser monitorados pelo exibidor procedural, o identificador (*id*) do elemento <area> associado ao código procedural a ser executado, e um tempo de retardo, opcional. A partir do atributo *src*, o exibidor procedural deve tentar localizar o código procedural e iniciar sua execução. Se o conteúdo não puder ser localizado, o exibidor procedural deve obrigatoriamente encerrar a operação de inicialização sem realizar nenhuma ação.

A lista de eventos a serem monitorados por um exibidor procedural é aconselhado também que seja computada pelo formatador, levando em conta a especificação do documento NCL. Ele deve obrigatoriamente checar todos os elos dos quais participa o objeto de mídia procedural e o descritor resultante. Ao computar os eventos a serem monitorados, o formatador deve obrigatoriamente considerar a perspectiva do objeto de mídia procedural, isto é, o caminho dos vários elementos <body> e <context> para alcançar em profundidade o elemento <media> correspondente. Apenas elos contidos nesses elementos <body> e <context> convêm ser considerados na computação dos eventos monitorados.

Como com todos os tipos de elemento <media>, o tempo de retardo é um parâmetro opcional, e seu valor *default* é "zero". Se maior que zero, esse parâmetro contém um tempo a ser esperado pelo exibidor procedural antes de iniciar a execução.

Diferente dos procedimentos realizados para outros tipos de elementos <media>, se um exibidor procedural receber uma instrução *start* para um evento associado a um elemento <area> e esse evento estiver no estado *sleeping*, ele deve dar início à execução do código procedural associado ao elemento, mesmo se outra parte do código procedural do objeto de mídia estiver em execução (pausado ou não). Contudo, se o evento associado ao elemento <area> alvo estiver no estado *occurring* ou *paused*, a instrução *start* deve ser ignorada pelo exibidor procedural que continuará controlando a execução anteriormente iniciada. Como conseqüência, diferente do que ocorre para os outros elementos <media>, uma ação <simpleAction> com o atributo *actionType* igual a "stop",

"pause", "resume" ou "abort" deve se ligar, através de um elo, a uma interface do nó procedural, que não deve ser ignorada quando a ação é aplicada.

A instrução *set* emitida por um formatador pode ser aplicada a um objeto procedural independente do fato dele estar sendo executado ou não (nesse último caso, embora o objeto não esteja sendo executado, seu exibidor procedural deve obrigatoriamente já ter sido instanciado). No primeiro caso, a instrução *set* precisa identificar o objeto procedural, um evento de atribuição monitorado e um valor a ser passado ao código procedural associado ao evento. No segundo caso, deve obrigatoriamente também identificar o elemento <descriptor> que será usado quando da execução do objeto (análogo ao que é feito para a instrução *start*).

Para cada evento de atribuição monitorado, se o exibidor procedural trocar por si mesmo o valor do atributo, ele deve proceder obrigatoriamente como se tivesse recebido uma instrução externa de *set*.

Recomenda-se que linguagens procedurais ofereçam uma API que permita que os códigos procedurais questionem quaisquer valores de propriedades predefinidas ou dinâmicas do nó *settings* NCL (elemento <media> do tipo "application/x-ginga-settings"). Contudo, deve-se observar que não é permitido atribuir valores a essas propriedades diretamente. Propriedades dos nós do tipo application/x-ginga-settings podem apenas ser modificadas através do uso de elos NCL.

9 Transmissão de conteúdo e eventos de fluxo NCL

9.1 Bases privadas

O núcleo da máquina de apresentação Ginga-NCL é composto pelo formatador NCL e seu 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 documentos NCL que são coletados dentro de uma estrutura de dados conhecida como base privada. Ginga associa uma base privada a um canal de televisão. Os documentos NCL em uma base privada podem ser iniciados, pausados, retomados, paralisados e podem referir-se uns aos outros.

O Gerenciador de Base Privada é responsável por receber comandos de edição de documentos NCL e pela edição dos documentos NCL ativos (documentos sendo apresentados).

O DSM-CC é adotado pelo Ginga para transportar os comandos de edição em fluxos elementares MPEG-2 TS. Os eventos de fluxo (stream events) DSM-CC e o protocolo de carrossel de objetos DSM-CC (ver ABNT NBR 15606-3), são a base para manipular o documento na máquina de apresentação Ginga.

Como especificado na ISO/IEC 13818-6, os descritores de evento de fluxo DSM-CC têm uma estrutura composta basicamente por um *id* (identificação), uma referência de tempo e um campo de dados privados. A identificação relaciona univocamente cada evento de *fluxo* a um comando de edição.

A referência de tempo indica o exato momento de disparar o evento. Um tempo de referência igual a zero informa que o evento deve obrigatoriamente ser imediatamente disparado após ser recebido (eventos carregando este tipo de referência de tempo são comumente conhecidos como eventos "*do it now*"). O campo de dados privados oferece suporte para parâmetros do evento, como apresentado na Figura 5.

Sintaxe	Número de bits
StreamEventDescriptor ( ) {	
descriptorTag	8
descriptorLenght	8
eventide	8
Reserved	31
eventNPT	33
privateDataLenght	8
commandTag	8
sequenceNumber	7
finalFlag	1
privateDataPayload	8 a 2008
FCS	8
}	

Figura 5 – Descritor de evento de fluxo de comandos de edição

O campo *commandTag* identifica univocamente os comandos de edição, como especificado na Tabela 56. O campo *privateData* deve obrigatoriamente ter no máximo 255 bytes, conforme especificado pela ISO/IEC 13818-6.

Para permitir enviar um comando completo com mais de 255 bytes nos descritores de evento de fluxo, todos os descritores do mesmo comando devem obrigatoriamente ser numerados e enviados em seqüência (isto é, não pode ser multiplexado com outros comandos de edição com o mesmo *commandTag*), com o *finalFlag* igual a 1, exceto para o último descritor, que deve obrigatoriamente ter o campo *finalFlag* igual a 0. O *privateDataPayload* contém os parâmetros de comando de edição. Finalmente, o campo FCS contém um *checksum* de todo o campo *privateData*, inclusive o *payloadDataLength.*

O protocolo de carrossel de objetos DSM-CC permite a transmissão cíclica de objetos de evento e sistemas de arquivo. Os objetos de evento são utilizados para mapear nomes de eventos de fluxo com *ids* de eventos de fluxo. Os objetos de evento são utilizados para informar ao Ginga sobre eventos de fluxo DSM-CC que podem ser recebidos. Os nomes dos eventos permitem especificar tipos de eventos, oferecendo maior nível de abstração às aplicações do *middleware*. Convém que o Gerenciador da Base Privada, bem como os objetos de execução procedural NCL (exemplo, NCLua, NCLet), sejam registrados como observadores dos eventos de fluxo com os quais lidam, utilizando nomes de evento.

Além dos objetos de evento, o protocolo de carrossel de objetos DSM-CC também é utilizado para transportar arquivos organizados em diretórios. O demultiplexador DSM-CC Ginga é responsável por montar o sistema de arquivo no dispositivo receptor (ver Seção 12 e ABNT NBR 15606-3).

Os comandos de edição são codificados como eventos de fluxo DSM-CC. Os arquivos de documento NCL e os conteúdos de objeto de mídia NCL são organizados em estruturas de sistemas de arquivos. Os parâmetros de comando de edição baseados em XML podem ser diretamente transportados no *payload* de um descritor de evento de fluxo ou, alternativamente, organizados em estruturas de sistema de arquivos a serem transportadas, cada uma, em um carrossel de objetos próprio. Um gerador de carrossel DSM-CC é usado para unir os sistemas de arquivos e os objetos de evento de fluxo em um fluxo elementar de dados.

Quando um comando de edição de documento NCL precisa ser enviado, um objeto de evento DSM-CC deve obrigatoriamente ser criado, mapeando a *string* “gingaEditingCommand” em uma *id* de evento de *fluxo* (ver Seção 12), e colocado em um carrossel de objetos DSM-CC. Um ou mais descritores de evento de fluxo DSM-CC com a *id* previamente selecionada são então criados e enviados em outro fluxo elementar MPEG-2 TS. Esses eventos de fluxo normalmente têm sua referência de tempo colocadas em zero, mas podem ser adiados para serem executados em um tempo específico. O Gerenciador da Base Privada deve obrigatoriamente registrar-se como um ouvinte “gingaEditingCommand” e é notificado quando esse evento de fluxo chega.

O *commandTag* recebido é então utilizado pelo Gerenciador da Base Privada para interpretar a semântica da *command string.* Se o *command parameter* baseado em XML for curto o suficiente, ele é transportado diretamente no *payload* dos descritores de evento de fluxo. Se não, o *privateDatePayload* transporta um conjunto de pares de referência. Cada par relaciona um caminho de sistema de arquivos e sua respectiva localização em um carrossel DSM-CC, utilizado para seu transporte. Nesse caso, um dos sistemas de arquivos deve obrigatoriamente conter o parâmetro de comando baseado em XML (um arquivo XML) e, se necessário, conteúdos de objeto de mídia NCL que estejam no mesmo sistema de arquivos. Outros sistemas de arquivos, se necessário, podem conter conteúdos adicionais de objetos de mídia NCL.

A Tabela 56 mostra as *strings* de comando e, cercados por parênteses, os parâmetros transportados como conteúdo *payload* do descritor de evento de *fluxo gingaEditingCommand*. Os comandos são divididos em três grupos: o primeiro para operação da base privada (para abrir, fechar e salvar bases privadas); o segundo para manipulação de documento (para iniciar, pausar, retomar e parar apresentações de documentos); e a última para manipular entidades NCL. 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).

Tabela 56 — 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 baseId
activateBase (baseId)	0x01	Ativa uma base privada aberta
deactivateBase (baseId)	0x02	Desativa uma base privada aberta
saveBase (baseId, location)	0x03	Salva todo o conteúdo da base privada em um dispositivo de armazenamento persistente (se disponível). O atributo location deve obrigatoriamente especificar o dispositivo e caminho para salvar a base
closeBase (baseId)	0x04	Fecha a base privada e descarta todo o conteúdo da base privada
addDocument (baseId, {uri, ior}+)	0x05	Adiciona um documento NCL a uma base privada. O documento NCL é enviado no carrossel de objetos como sistemas de arquivo; o par {uri, ior} relaciona um caminho no sistema de arquivo no provedor de *datacasting* (ver Seção 12) com sua respectiva localização em um carrossel
removeDocument (baseId, documentId)	0x06	Remove um documento NCL de uma base privada
startDocument (baseId, documentId, interfaceId)	0x07	Inicia a reprodução de um documento NCL em uma base privada, iniciando a apresentação a partir de uma interface específica do documento
stopDocument (baseId, documentId)	0x08	Pára a apresentação de um documento NCL em uma base privada. Todos os eventos do documento que estão em andamento devem obrigatoriamente ser paralisados
pauseDocument (baseId, documentId)	0x09	Pausa a apresentação de um documento NCL em uma base privada. Todos os eventos do documento que estão em andamento devem obrigatoriamente ser pausados
resumeDocument (baseId, documentId)	0x0A	Retoma a apresentação de um documento NCL em uma base privada. Todos os eventos do documento que foram previamente pausados pelo o comando de edição pauseDocument devem obrigatoriamente ser retomados
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
removeRegion (baseId, documentId, regionId)	0x0C	Remove um elemento <region> de um <regionBase> de um documento NCL em uma base privada
addRegionBase (baseId, documentId, xmlRegionBase)	0x0D	Adiciona um elemento <regionBase> ao elemento <head> de um documento NCL em uma base privada. A especificação XML do regionBase é enviada no carrossel de objetos como um sistema de arquivo; o parâmetro xmlRegionBase é apenas uma referência para esse conteúdo no carrossel
removeRegionBase (baseId, documentId, regionBaseId)	0x0E	Remove um elemento <regionBase> do elemento <head> de um documento NCL em uma base privada
addRule (baseId, documentId, xmlRule)	0x0F	Adiciona um elemento <rule> ao <ruleBase> de um documento NCL em uma base privada
removeRule (baseId, documentId, ruleId)	0x10	Remove um elemento <rule> do <ruleBase> de um documento NCL em uma base privada
addRuleBase (baseId, documentId, xmlRuleBase)	0x11	Adiciona um elemento <ruleBase> ao elemento <head> de um documento NCL em uma base privada. A especificação XML do ruleBase é enviada no carrossel de objetos como um sistema de arquivo; o parâmetro xmlRuleBase é apenas uma referência

		para esse conteúdo no carrossel

Tabela 56 (continuação)

***String* de comando**	***Tag* de comando**	**Descrição**
removeRuleBase (baseId, documentId, ruleBaseId)	0x12	Remove um elemento <ruleBase> do elemento <head> de um documento NCL em uma base privada
addConnector (baseId, documentId, xmlConnector)	0x13	Adiciona um elemento <connector> ao <connectorBase> de um documento NCL em uma base privada
removeConnector (baseId, documentId, connectorId)	0x14	Remove um elemento <connector> do <connectorBase> de um documento NCL em uma base privada
addConnectorBase (baseId, documentId, xmlConnectorBase)	0x15	Adiciona um elemento <connectorBase> ao elemento <head> de um documento NCL em uma base privada. A especificação XML do connectorBase é enviada no carrossel de objetos como um sistema de arquivo; o parâmetro xmlConnectorBase é apenas uma referência para esse conteúdo no carrossel
removeConnectorBase (baseId, documentId, connectorBaseId)	0x16	Remove um elemento <connectorBase> do elemento <head> de um documento NCL em uma base privada
addDescriptor (baseId, documentId, xmlDescriptor)	0x17	Adiciona um elemento <descriptor> ao <descriptorBase> de um documento NCL em uma base privada
removeDescriptor (baseId, documentId, descriptorId)	0x18	Remove um elemento <descriptor> do <descriptorBase> de um documento NCL em uma base privada
addDescriptorSwitch (baseId, documentId, xmlDescriptorSwitch)	0x19	Adiciona um elemento <descriptorSwitch> ao <descriptorBase> de um documento NCL em uma base privada. A especificação XML do descriptorSwitch é enviada no carrossel de objetos 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
addDescriptorBase (baseId, documentId, xmlDescriptorBase)	0x1B	Adiciona um elemento <descriptorBase> ao elemento <head> de um documento NCL em uma base privada. A especificação XML do descriptorBase é enviada no carrossel de objetos como um sistema de arquivo; o parâmetro xmlDescriptorBase é apenas uma referência para esse conteúdo no carrossel
removeDescriptorBase (baseId, documentId, descriptorBaseId)	0x1C	Remove um elemento <descriptorBase> do elemento <head> de um documento NCL em uma base privada
addTransition (baseId, documentId, xmlTransition)	0x1D	Adiciona um elemento <transition> ao <transitionBase> de um documento NCL em uma base privada
removeTransition (baseId, documentId, transitionId)	0x1E	Remove um elemento <transition> do <transitionBase> de um documento NCL em uma base privada
addTransitionBase (baseId, documentId, xmlTransitionBase)	0x1F	Adiciona um elemento <transitionBase> ao elemento <head> de um documento NCL em uma base privada. A especificação XML do transitionBase é enviada no carrossel de objetos 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
addImportBase (baseId, documentId, docBaseId, xmlImportBase)	0x21	Adiciona um elemento <importBase> à base (elemento <regionBase>, <descriptorBase>, <ruleBase>, <transitionBase>, or <connectorBase>) de um documento NCL em uma base privada
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>, or <connectorBase>) de um documento NCL em uma base privada

Tabela 56 (continuação)

String de comando	*Tag* de comando	Descrição
addImportedDocumentBase (baseId, documentId, xmlImportedDocumentBase)	0x23	Adiciona um elemento <importedDocumentBase> ao elemento <head> de um documento NCL em uma base privada
removeImportedDocumentBase (baseId, documentId, importedDocumentBaseId)	0x24	Remove um elemento <importedDocumentBase> do elemento <head> de um documento NCL em uma base privada
addImportNCL (baseId, documentId, xmlImportNCL)	0x25	Adiciona um elemento <importNCL> ao elemento <importedDocumentBase> de um documento NCL em uma base privada
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
addNode (baseId, documentId, compositeId, {uri, ior}+)	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. A especificação XML do nó e seu conteúdo de mídia são enviados no carrossel de objetos como sistemas de arquivo; o par {uri, ior} relaciona um caminho de sistema de arquivo no provedor de *datacasting* (ver Seção 12) com sua respectiva localização em um carrossel
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
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
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. A interfaceID deve obrigatoriamente 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
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
setPropertyValue(baseId, documentId, nodeId, propertyId, value)	0x2D	Atribui o valor a uma propriedade. A propertyId deve obrigatoriamente identificar um atributo *name* de um elemento <property> ou um atributo *id* de elemento <switchPort>. O <property> ou <switchPort> deve obrigatoriamente pertencer a um nó (elemento <body>, <context>, <switch> ou <media>) de um documento NCL em uma base privada identificada pelos parâmetros

Os receptores que somente implementam o perfil NCL DTV Básico podem não lidar com os seguintes comandos: *pauseDocument*, *resumeDocument*, *addTransition*, *removeTransition*, *addTransitionBase* e *removeTransitionBase*.

Ginga associa uma base privada a um canal de televisão. Por razões de segurança, apenas uma única base privada pode estar ativa por vez. Para simplificar a tarefa do Gerenciador da Base Privada, sugere-se que apenas uma única base privada seja aberta por vez. Assim, se o usuário mudar o canal selecionado, recomenda-se que a base privada atual seja fechada. Nesse caso, o comando *openBase* é sempre seguido pelo comando *activeBase* e o comando *deactiveBase* nunca é usado.

Os comandos *add* têm entidades NCL como seus argumentos (parâmetros de comando baseados em XML). Se a entidade especificada já existe ou não, a consistência do documento deve obrigatoriamente ser mantida pelo formatador NCL, no sentido de que todos os atributos de identidade classificados como necessários devem obrigatoriamente ser definidos. As entidades são definidas utilizando uma notação sintática idêntica àquela usada pelos esquemas NCL, com exceção do comando *addInterface*: o atributo *begin* de um elemento <area> pode receber o valor “now”, especificando o NPT atual do *nodeId*, que deve obrigatoriamente ser o vídeo principal MPEG sendo reproduzido pelo decodificador de *hardware*.

Os identificadores utilizados nos comandos devem obrigatoriamente estar de acordo com a Tabela 57.

Tabela 57 — Identificadores usados nos comandos de edição

Identificadores	Definição
baseId	Identificadores de canal de radiodifusão especificados pelo SBTVD
documentId	Atributo *id* de um elemento <ncl> de um documento NCL
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

9.2 Esquema XML dos parâmetros de comando

As entidades NCL utilizadas nos comandos de edição devem obrigatoriamente ser um documento em conformidade com o perfil de Comando NCL 3.0 definido pelo esquema XML a seguir. Convém que os receptores que apenas implementam o perfil Básico NCL DTV ignorem os elementos e atributos XML relacionados às funcionalidades de Meta-information e Transition Efects.

Observar que, diferentemente dos documentos NCL, diversos elementos NCL podem ter o elemento raiz nos parâmetros de comando XML.

NCL30EdCommand.xsd

```
<!--
XML Schema for the NCL Language

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/NCL3.0/profiles/NCL30EdCommand.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006
-->

<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:animation="http://www.ncl.org.br/NCL3.0/Animation"
  xmlns:compositeInterface="http://www.ncl.org.br/NCL3.0/
CompositeNodeInterface"
  xmlns:causalConnectorFunctionality="http://www.ncl.org.br/NCL3.0/
CausalConnectorFunctionality"
  xmlns:connectorBase="http://www.ncl.org.br/NCL3.0/ConnectorBase"
  xmlns:connectorCausalExpression="http://www.ncl.org.br/NCL3.0/
ConnectorCausalExpression"
  xmlns:contentControl="http://www.ncl.org.br/NCL3.0/ContentControl"
  xmlns:context="http://www.ncl.org.br/NCL3.0/Context"
  xmlns:descriptor="http://www.ncl.org.br/NCL3.0/Descriptor"
  xmlns:entityReuse="http://www.ncl.org.br/NCL3.0/EntityReuse"
  xmlns:extendedEntityReuse="http://www.ncl.org.br/NCL3.0/
ExtendedEntityReuse"
  xmlns:descriptorControl="http://www.ncl.org.br/NCL3.0/
DescriptorControl"
  xmlns:import="http://www.ncl.org.br/NCL3.0/Import"
  xmlns:keyNavigation="http://www.ncl.org.br/NCL3.0/KeyNavigation"
  xmlns:layout="http://www.ncl.org.br/NCL3.0/Layout"
  xmlns:linking="http://www.ncl.org.br/NCL3.0/Linking"
  xmlns:media="http://www.ncl.org.br/NCL3.0/Media"
  xmlns:mediaAnchor="http://www.ncl.org.br/NCL3.0/MediaContentAnchor"
  xmlns:propertyAnchor="http://www.ncl.org.br/NCL3.0/PropertyAnchor"
  xmlns:structure="http://www.ncl.org.br/NCL3.0/Structure"
  xmlns:switchInterface="http://www.ncl.org.br/NCL3.0/SwitchInterface"
  xmlns:testRule="http://www.ncl.org.br/NCL3.0/TestRule"
  xmlns:testRuleUse="http://www.ncl.org.br/NCL3.0/TestRuleUse"
  xmlns:timing="http://www.ncl.org.br/NCL3.0/Timing"
  xmlns:transitionBase="http://www.ncl.org.br/NCL3.0/TransitionBase"
```

```
  xmlns:metainformation="http://www.w3.org/2001/SMIL20/Metainformation"
  xmlns:basicTransition="http://www.w3.org/2001/SMIL20/BasicTransitions"
  xmlns:profile="http://www.ncl.org.br/NCL3.0/EdCommandProfile"
  targetNamespace="http://www.ncl.org.br/NCL3.0/EdCommandProfile"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

  <!-- import the definitions in the modules namespaces -->
  <import namespace="http://www.ncl.org.br/NCL3.0/Animation"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Animation.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/CompositeNodeInterface"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30CompositeNodeInterface.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/CausalConnectorFunctionality"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30CausalConnectorFunctionality.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/ConnectorBase"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30ConnectorBase.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/ConnectorCausalExpression"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30ConnectorCausalExpression.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/ContentControl"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30ContentControl.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/Context"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Context.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/Descriptor"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Descriptor.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/DescriptorControl"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30DescriptorControl.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/EntityReuse"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30EntityReuse.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/ExtendedEntityReuse"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30ExtendedEntityReuse.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/Import"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Import.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/KeyNavigation"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30KeyNavigation.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/Layout"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Layout.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/Linking"
```

```
   schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Linking.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/Media"
   schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30Media.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/MediaContentAnchor"
   schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30MediaContentAnchor.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/PropertyAnchor"
   schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30PropertyAnchor.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/Structure"
   schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Structure.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/SwitchInterface"
   schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30SwitchInterface.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/TestRule"
   schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30TestRule.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/TestRuleUse"
   schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30TestRuleUse.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/Timing"
   schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30Timing.xsd"/>
  <import namespace="http://www.ncl.org.br/NCL3.0/TransitionBase"
   schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30TransitionBase.xsd"/>
  <import namespace="http://www.w3.org/2001/SMIL20/Metainformation"
   schemaLocation="http://www.w3.org/2001/SMIL20/
smil20-Metainformation.xsd"/>
  <import namespace="http://www.w3.org/2001/SMIL20/BasicTransitions"
   schemaLocation="http://www.w3.org/2001/SMIL20/
smil20-BasicTransitions.xsd"/>

  <!-- ========================================================== -->
  <!--EditingCommand                                          -->
  <!-- ========================================================== -->
  <!--defines the command element -->

  <!--This is a pseudo-element, only defined to show the elements that can be used in the root of the command
parameters XML document-->

  <!--
  <complexType name="commandType">
      <choice minOccurs="1" maxOccurs="1">
        <element ref="profile:ncl"/>
        <element ref="profile:region"/>
        <element ref="profile:rule"/>
        <element ref="profile:connector"/>
        <element ref="profile:descriptor"/>
        <element ref="profile:descriptorSwitch"/>
        <element ref="profile:transition"/>
        <element ref="profile:regionBase"/>
```

```
        <element ref="profile:ruleBase"/>
        <element ref="profile:connectorBase"/>
        <element ref="profile:descriptorBase"/>
        <element ref="profile:transitionBase"/>
        <element ref="profile:importBase"/>
        <element ref="profile:importedDocumentBase"/>
        <element ref="profile:importNCL"/>
        <element ref="profile:media"/>
        <element ref="profile:context"/>
        <element ref="profile:switch"/>
        <element ref="profile:port"/>
        <element ref="profile:area"/>
        <element ref="profile:property"/>
        <element ref="profile:switchPort"/>
        <element ref="profile:link"/>
      </choice>
 </complexType>
 <element name="command" type="profile:commandType"/>
-->

 <!-- ============================================================ -->
 <!-- Structure                                                    -->
 <!-- ============================================================ -->
 <!-- extends ncl element -->

 <element name="ncl" substitutionGroup="structure:ncl"/>

 <!-- extends head element -->

 <complexType name="headType">
  <complexContent>
   <extension base="structure:headPrototype">
    <sequence>
     <element ref="profile:importedDocumentBase" minOccurs="0" maxOccurs="1"/>
     <element ref="profile:ruleBase" minOccurs="0" maxOccurs="1"/>
     <element ref="profile:transitionBase" minOccurs="0" maxOccurs="1"/>
     <element ref="profile:regionBase" minOccurs="0" maxOccurs="unbounded"/>
     <element ref="profile:descriptorBase" minOccurs="0" maxOccurs="1"/>
     <element ref="profile:connectorBase" minOccurs="0" maxOccurs="1"/>
     <element ref="profile:meta" minOccurs="0" maxOccurs="unbounded"/>
     <element ref="profile:metadata" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
   </extension>
  </complexContent>
 </complexType>

 <element name="head" type="profile:headType" substitutionGroup="structure:head"/>

 <!-- extends body element -->

 <complexType name="bodyType">
```

```
  <complexContent>
    <extension base="structure:bodyPrototype">
      <choice minOccurs="0" maxOccurs="unbounded">
        <group ref="profile:contextInterfaceElementGroup"/>
        <element ref="profile:media"/>
        <element ref="profile:context"/>
        <element ref="profile:switch"/>
        <element ref="profile:link"/>
      </choice>
    </extension>
  </complexContent>
</complexType>

<element name="body" type="profile:bodyType" substitutionGroup="structure:body"/>

<!-- ========================================================== -->
<!-- Layout                                                     -->
<!-- ========================================================== -->
<!-- extends regionBase element -->

<complexType name="regionBaseType">
  <complexContent>
    <extension base="layout:regionBasePrototype">
      <choice minOccurs="1" maxOccurs="unbounded">
        <element ref="profile:region"/>
        <element ref="profile:importBase"/>
      </choice>
    </extension>
  </complexContent>
</complexType>

<complexType name="regionType">
  <complexContent>
    <extension base="layout:regionPrototype">
    </extension>
  </complexContent>
</complexType>

<element name="regionBase" type="profile:regionBaseType" substitutionGroup="layout:regionBase"/>
<element name="region" type="profile:regionType" substitutionGroup="layout:region"/>

<!-- ========================================================== -->
<!-- Media                                                      -->
<!-- ========================================================== -->
<!-- extends Media elements -->

<!-- media interface element groups -->
<group name="mediaInterfaceElementGroup">
  <choice>
    <element ref="profile:area"/>
    <element ref="profile:property"/>
  </choice>
</group>
```

```
<complexType name="mediaType">
  <complexContent>
    <extension base="media:mediaPrototype">
      <choice minOccurs="0" maxOccurs="unbounded">
        <group ref="profile:mediaInterfaceElementGroup"/>
      </choice>
      <attributeGroup ref="descriptor:descriptorAttrs"/>
      <attributeGroup ref="entityReuse:entityReuseAttrs"/>
      <attributeGroup ref="extendedEntityReuse:extendedEntityReuseAttrs"/>
    </extension>
  </complexContent>
</complexType>

<element name="media" type="profile:mediaType" substitutionGroup="media:media"/>

<!-- ============================================================ -->
<!-- Context                                                      -->
<!-- ============================================================ -->
<!-- extends context element -->

<!-- composite node interface element groups -->
<group name="contextInterfaceElementGroup">
  <choice>
    <element ref="profile:port"/>
    <element ref="profile:property"/>
  </choice>
</group>

<complexType name="contextType">
  <complexContent>
    <extension base="context:contextPrototype">
      <choice minOccurs="0" maxOccurs="unbounded">
        <group ref="profile:contextInterfaceElementGroup"/>
        <element ref="profile:media"/>
        <element ref="profile:context"/>
        <element ref="profile:link"/>
        <element ref="profile:switch"/>
      </choice>
      <attributeGroup ref="entityReuse:entityReuseAttrs"/>
    </extension>
  </complexContent>
</complexType>

<element name="context" type="profile:contextType" substitutionGroup="context:context"/>

<!-- ============================================================ -->
<!-- MediaContentAnchor                                           -->
<!-- ============================================================ -->
<!-- extends area element -->

<complexType name="componentAnchorType">
```

```
  <complexContent>
    <extension base="mediaAnchor:componentAnchorPrototype">
       <attribute name="now" type="string" use="optional"/>
    </extension>
  </complexContent>
</complexType>

<element name="area" type="profile:componentAnchorType" substitutionGroup="mediaAnchor:area"/>

<!-- ========================================================= -->
<!-- CompositeNodeInterface                                    -->
<!-- ========================================================= -->
<!-- extends port element -->

<complexType name="compositeNodePortType">
  <complexContent>
    <extension base="compositeInterface:compositeNodePortPrototype">
    </extension>
  </complexContent>
</complexType>

<element name="port" type="profile:compositeNodePortType" substitutionGroup="compositeInterface:port"/>

<!-- ========================================================= -->
<!-- PropertyAnchor                                            -->
<!-- ========================================================= -->
<!-- extends property element -->

<complexType name="propertyAnchorType">
  <complexContent>
    <extension base="propertyAnchor:propertyAnchorPrototype">
    </extension>
  </complexContent>
</complexType>

<element name="property" type="profile:propertyAnchorType" substitutionGroup="propertyAnchor:property"/>

<!-- ========================================================= -->
<!-- SwitchInterface                                           -->
<!-- ========================================================= -->
<!-- extends switchPort element -->

<complexType name="switchPortType">
  <complexContent>
    <extension base="switchInterface:switchPortPrototype">
    </extension>
  </complexContent>
</complexType>

<element name="mapping" substitutionGroup="switchInterface:mapping"/>
<element name="switchPort" type="profile:switchPortType" substitutionGroup="switchInterface:switchPort"/>

<!-- ========================================================= -->
<!-- Descriptor                                                -->
```

```
 <!-- ============================================================ -->

 <!-- substitutes descriptorParam element -->

 <element name="descriptorParam" substitutionGroup="descriptor:descriptorParam"/>


 <!-- extends descriptor element -->

 <complexType name="descriptorType">
  <complexContent>
   <extension base="descriptor:descriptorPrototype">
    <attributeGroup ref="layout:regionAttrs"/>
    <attributeGroup ref="timing:explicitDurAttrs"/>
    <attributeGroup ref="timing:freezeAttrs"/>
    <attributeGroup ref="keyNavigation:keyNavigationAttrs"/>
    <attributeGroup ref="profile:transitionAttrs"/>
   </extension>
  </complexContent>
 </complexType>

 <element name="descriptor" type="profile:descriptorType" substitutionGroup="descriptor:descriptor"/>

 <!-- extends descriptorBase element -->
 <complexType name="descriptorBaseType">
  <complexContent>
   <extension base="descriptor:descriptorBasePrototype">
    <choice minOccurs="1" maxOccurs="unbounded">
     <element ref="profile:importBase"/>
     <element ref="profile:descriptor"/>
     <element ref="profile:descriptorSwitch"/>
    </choice>
   </extension>
  </complexContent>
 </complexType>

 <element name="descriptorBase" type="profile:descriptorBaseType"
substitutionGroup="descriptor:descriptorBase"/>

 <!-- ============================================================ -->
 <!-- Linking                                                      -->
 <!-- ============================================================ -->

 <!-- substitutes linkParam and bindParam elements -->
 <element name="linkParam" substitutionGroup="linking:linkParam"/>
 <element name="bindParam" substitutionGroup="linking:bindParam"/>

 <!-- extends bind element and link element, as a consequence-->

 <complexType name="bindType">
  <complexContent>
```

```
    <extension base="linking:bindPrototype">
      <attributeGroup ref="descriptor:descriptorAttrs"/>
    </extension>
  </complexContent>
 </complexType>

 <element name="bind" type="profile:bindType" substitutionGroup="linking:bind"/>

 <!-- extends link element -->
 <complexType name="linkType">
  <complexContent>
    <extension base="linking:linkPrototype">
    </extension>
  </complexContent>
 </complexType>

 <element name="link" type="profile:linkType" substitutionGroup="linking:link"/>

 <!-- ============================================================ -->
 <!-- Connector                                    -->
 <!-- ============================================================ -->
 <!-- extends connectorBase element -->

  <complexType name="connectorBaseType">
     <complexContent>
        <extension base="connectorBase:connectorBasePrototype">
          <choice minOccurs="0" maxOccurs="unbounded">
            <element ref="profile:importBase"/>
        <element ref="profile:causalConnector" />
          </choice>
        </extension>
     </complexContent>
  </complexType>

  <complexType name="simpleActionType">
    <complexContent>
      <extension base="connectorCausalExpression:simpleActionPrototype">
        <attributeGroup ref="animation:animationAttrs"/>
      </extension>
    </complexContent>
  </complexType>

 <element name="connectorBase" type="profile:connectorBaseType"
substitutionGroup="connectorBase:connectorBase"/>

 <element name="causalConnector" substitutionGroup="causalConnectorFunctionality:causalConnector"/>

 <element name="connectorParam" substitutionGroup="causalConnectorFunctionality:connectorParam"/>

 <element name="simpleCondition" substitutionGroup="causalConnectorFunctionality:simpleCondition"/>

 <element name="compoundCondition" substitutionGroup="causalConnectorFunctionality:compoundCondition"/>
```

```
  <element name="simpleAction" type="profile:simpleActionType"
substitutionGroup="causalConnectorFunctionality:simpleAction"/>

  <element name="compoundAction" substitutionGroup="causalConnectorFunctionality:compoundAction"/>

  <element name="assessmentStatement"
substitutionGroup="causalConnectorFunctionality:assessmentStatement"/>

  <element name="attributeAssessment" substitutionGroup="causalConnectorFunctionality:attributeAssessment"/>

  <element name="valueAssessment" substitutionGroup="causalConnectorFunctionality:valueAssessment"/>

  <element name="compoundStatement" substitutionGroup="causalConnectorFunctionality:compoundStatement"/>

  <!-- ========================================================= -->
  <!-- TestRule                                                  -->
  <!-- ========================================================= -->
  <!-- extends rule element -->
  <complexType name="ruleType">
    <complexContent>
      <extension base="testRule:rulePrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="rule" type="profile:ruleType" substitutionGroup="testRule:rule"/>

  <!-- extends compositeRule element -->
  <complexType name="compositeRuleType">
    <complexContent>
      <extension base="testRule:compositeRulePrototype">
      </extension>
    </complexContent>
  </complexType>

  <element name="compositeRule" type="profile:compositeRuleType"
substitutionGroup="testRule:compositeRule"/>

  <!-- extends ruleBase element -->
  <complexType name="ruleBaseType">
    <complexContent>
      <extension base="testRule:ruleBasePrototype">
        <choice minOccurs="1" maxOccurs="unbounded">
          <element ref="profile:importBase"/>
          <element ref="profile:rule"/>
          <element ref="profile:compositeRule"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>
```

```
 <element name="ruleBase" type="profile:ruleBaseType" substitutionGroup="testRule:ruleBase"/>

 <!-- ========================================================== -->
 <!-- TestRuleUse                                                -->
 <!-- ========================================================== -->
 <!-- extends bindRule element -->
 <complexType name="bindRuleType">
  <complexContent>
   <extension base="testRuleUse:bindRulePrototype">
   </extension>
  </complexContent>
 </complexType>

 <element name="bindRule" type="profile:bindRuleType" substitutionGroup="testRuleUse:bindRule"/>

 <!-- ========================================================== -->
 <!-- ContentControl                                             -->
 <!-- ========================================================== -->
 <!-- extends switch element -->

 <!-- switch interface element groups -->
 <group name="switchInterfaceElementGroup">
  <choice>
   <element ref="profile:switchPort"/>
  </choice>
 </group>

 <!-- extends defaultComponent element -->
 <complexType name="defaultComponentType">
  <complexContent>
   <extension base="contentControl:defaultComponentPrototype">
   </extension>
  </complexContent>
 </complexType>

 <element name="defaultComponent" type="profile:defaultComponentType"
substitutionGroup="contentControl:defaultComponent"/>

 <complexType name="switchType">
  <complexContent>
   <extension base="contentControl:switchPrototype">
     <choice minOccurs="0" maxOccurs="unbounded">
      <group ref="profile:switchInterfaceElementGroup"/>
      <element ref="profile:bindRule"/>
      <element ref="profile:switch"/>
      <element ref="profile:media"/>
      <element ref="profile:context"/>
     </choice>
    <attributeGroup ref="entityReuse:entityReuseAttrs"/>
   </extension>
  </complexContent>
 </complexType>

 <element name="switch" type="profile:switchType" substitutionGroup="contentControl:switch"/>
```

```
 <!-- ========================================================== -->
 <!-- DescriptorControl                                          -->
 <!-- ========================================================== -->
 <!-- extends defaultDescriptor element -->
 <complexType name="defaultDescriptorType">
  <complexContent>
   <extension base="descriptorControl:defaultDescriptorPrototype">
   </extension>
  </complexContent>
 </complexType>

 <element name="defaultDescriptor" type="profile:defaultDescriptorType"
substitutionGroup="descriptorControl:defaultDescriptor"/>

 <!-- extends descriptorSwitch element -->

 <complexType name="descriptorSwitchType">
  <complexContent>
   <extension base="descriptorControl:descriptorSwitchPrototype">
      <choice minOccurs="0" maxOccurs="unbounded">
        <element ref="profile:descriptor"/>
        <element ref="profile:bindRule"/>
      </choice>
   </extension>
  </complexContent>
 </complexType>

 <element name="descriptorSwitch" type="profile:descriptorSwitchType"
substitutionGroup="descriptorControl:descriptorSwitch"/>

 <!-- ========================================================== -->
 <!-- Timing                                                     -->
 <!-- ========================================================== -->


 <!-- ========================================================== -->
 <!-- Import                                                     -->
 <!--========================================================== -->
 <complexType name="importBaseType">
  <complexContent>
   <extension base="import:importBasePrototype">
   </extension>
  </complexContent>
 </complexType>

 <complexType name="importNCLType">
  <complexContent>
   <extension base="import:importNCLPrototype">
   </extension>
  </complexContent>
```

```
 </complexType>

 <complexType name="importedDocumentBaseType">
  <complexContent>
   <extension base="import:importedDocumentBasePrototype">
   </extension>
  </complexContent>
 </complexType>

 <element name="importBase" type="profile:importBaseType" substitutionGroup="import:importBase"/>

 <element name="importNCL" type="profile:importNCLType" substitutionGroup="import:importNCL"/>
 <element name="importedDocumentBase" type="profile:importedDocumentBaseType"
substitutionGroup="import:importedDocumentBase"/>

 <!-- ================================================================ -->
 <!-- EntityReuse                                                      -->
 <!-- ================================================================ -->

 <!-- ================================================================ -->
 <!-- ExtendedEntityReuse                                              -->
 <!-- ================================================================ -->

 <!-- ================================================================ -->
 <!-- KeyNavigation                                                    -->
 <!-- ================================================================ -->

 <!-- ================================================================ -->
 <!-- TransitionBase                                                   -->
 <!-- ================================================================ -->
 <!-- extends transitionBase element -->

 <complexType name="transitionBaseType">
  <complexContent>
   <extension base="transitionBase:transitionBasePrototype">
    <choice minOccurs="0" maxOccurs="unbounded">
     <element ref="profile:transition"/>
     <element ref="profile:importBase"/>
    </choice>
   </extension>
  </complexContent>
 </complexType>

 <element name="transitionBase" type="profile:transitionBaseType"
substitutionGroup="transitionBase:transitionBase"/>

 <!-- ================================================================ -->
 <!-- BasicTransition                                                  -->
 <!-- ================================================================ -->

 <attributeGroup name="transitionAttrs">
  <attribute ref="basicTransition:transIn"/>
  <attribute ref="basicTransition:transOut"/>
 </attributeGroup>
```

```
 <element name="transition" substitutionGroup="basicTransition:transition"/>

 <!-- ============================================================ -->
 <!-- Metainformation                                              -->
 <!-- ============================================================ -->

 <element name="meta" substitutionGroup="metainformation:meta"/>

 <element name="metadata" substitutionGroup="metainformation:metadata"/>

</schema>
```

10 Objetos procedurais Lua em apresentações NCL

10.1 Linguagem Lua - Funções opcionais da biblioteca de Lua

A linguagem de *script* adotada pelo Ginga-NCL é Lua (elementos <media> do tipo application/x-ginga-NCLua). A definição completa de Lua é apresentada no Anexo B.

As funções a seguir são dependentes de plataforma e não são de implementação obrigatória:

a) no módulo *package*: *loadlib* é opcional;

b) no módulo *io:* todas as funções são opcionais;

c) no módulo os: *clock, execute, exit, getenv, remove, rename, tmpname* e *setlocale* são opcionais;

d) no módulo *debug*: todas as funções são opcionais, a API C de *debug* também é opcional.

10.2 Modelo de execução

O ciclo de vida de um objeto NCLua é controlado pelo formatador NCL. O formatador é responsável por iniciar a execução de um objeto NCLua e por mediar a comunicação entre esse objeto e outros objetos em um documento NCL, como definido em 8.5.

Como com todos os exibidores de objetos de mídia, um exibidor Lua, uma vez instanciado, deve executar os procedimentos de iniciação do objeto NCL que controlará. Porém, diferente dos outros exibidores de mídia, o código de iniciação deve também ser especificado pelo autor do objeto NCLua. Os procedimentos de iniciação são executados apenas uma vez, para cada instância, e cria funções e objetos que podem ser usados durante a execução do objeto NCLua e, em particular, registra um ou mais tratadores de eventos para a comunicação com o formatador NCL.

Depois da iniciação, a execução do objeto NCLua torna-se orientada a evento, em ambas as direções. Isto é, qualquer ação comandada pelo formatador NCL é dirigida aos tratadores de evento registrados, e qualquer notificação de mudança de estado de eventos NCL é enviada como um evento ao formatador NCL (como por exemplo, o fim da execução do código procedural). O exibidor Lua estará então pronto para executar qualquer intrução de *start* ou *set* (ver 8.5).

10.3 Módulos adicionais

10.3.1 Módulos obrigatórios

Além da biblioteca padrão de Lua, os seguintes módulos devem ser obrigatoriamente oferecidos e automaticamente carregados:

a) módulo *ncledit*: permite que *scripts* NCLua alterem a apresentação de um documento NCL;

b) módulo *canvas*: oferece uma API para desenhar primitivas gráficas e manipular imagens;

c) módulo *event*: permite que aplicações NCLua comuniquem-se com o *middleware* através de eventos (eventos NCL e de teclas);

d) módulo *settings*: exporta uma tabela com variáveis definidas pelo autor do documento NCL e variáveis de ambiente reservadas em um nó "application/x-ginga-settings";

e) módulo *persistent*: exporta uma tabela com variáveis persistentes, que estão disponíveis para manipulação apenas por objetos procedurais.

A definição de cada função nos módulos mencionados respeita a seguinte nomenclatura:

funcname (parnameI: partypeI [; optnameI: opttypeI]) -> retname: rettype

10.3.2 Módulo *ncledit*

Há uma importante diferença com relação aos comandos de edição provenientes dos eventos de fluxo DSM-CC (ver Seção 9) e os comandos de edição realizados pelos *scripts* Lua (objetos NCLua). O primeiro altera não somente a apresentação de um documento NCL, mas também a especificação de um documento NCL. Ou seja, no final do processo um novo documento NCL é gerado, incorporando todos os resultados da edição. Por outro lado, os comandos de edição provenientes dos objetos de mídia NCLua alteram somente a apresentação do documento NCL. O documento original é preservado durante todo o processo de edição.

Todas as funções descritas podem receber uma referência de tempo como parâmetro opcional (parâmetros opcionais são indicados nas assinaturas da função como parâmetros entre colchetes). Esse parâmetro opcional pode ser usado para especificar o exato momento em que o comando de edição deve ser executado. Se este parâmetro não for fornecido na chamada de função, o comando de edição deve ser executado imediatamente. Quando fornecido, o parâmetro pode ter dois tipos diferentes de valores, com diferentes significados. Se for um valor numérico, define a quantidade de tempo, em segundos, que a execução do comando deve ser postergada. Contudo, esse parâmetro também pode especificar o exato momento para execução do comando, em termos de valores absolutos. Para isso, o parâmetro deve obrigatoriamente ter um valor de tabela com os seguintes campos: *year* (quatro dígitos), *month* (1 a 12), *day* (1 a 31), *hour* (0 a 23), *minute* (0 a 59), *second* (0 a 61) e *isdst* (sinalizador de horário de verão, um booleano).

ncledit.openBase (baseId, location: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
location	Identifica o local a procurar pela base. Se for nulo, uma nova base deve obrigatoriamente ser criada

Valor de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Abre uma base privada existente localizada pelo atributo location. O atributo location deve obrigatoriamente especificar o dispositivo e o caminho para salvar a base. Se a base privada não existir ou o atributo location não for informado, uma nova base é criada com baseId.

ncledit.activateBase (baseId: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Ativa uma base privada aberta.

ncledit.deactivateBase (baseId: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Desativa uma base privada aberta.

ncledit.saveBase (baseId, location: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
location	Identifica o dispositivo e o caminho onde a base privada deve obrigatoriamente ser salva

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Salva todo o conteúdo da base privada em um dispositivo de armazenamento persistente (se disponível). O atributo location deve obrigatoriamente especificar o dispositivo e o caminho para salvar a base.

ncledit.closeBase (baseId: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Fecha a base privada e descarta todo seu conteúdo.

ncledit.addDocument (baseId, documentId: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
document	Definição de documento NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Adiciona um documento NCL a uma base privada.

ncledit.removeDocument (baseId, documentId: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Remove um documento NCL de uma base privada.

ncledit.startDocument (baseId, documentId, interfaceId: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
interfaceId	Identifica uma interface de documento NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Começa a reprodução de um documento NCL em uma base privada, começando a apresentação a partir da interface de documento especificada.

ncledit.stopDocument (baseId, documentId: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Pára a apresentação de um documento NCL em uma base privada. Todos os eventos do documento que estão ocorrendo devem obrigatoriamente ser parados.

ncledit.pauseDocument (baseId, documentId: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Pausa a apresentação de um documento NCL em uma base privada. Todos os eventos do documento que estão ocorrendo devem obrigatoriamente ser pausados.

ncledit.resumeDocument (baseId, documentId: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Retoma a apresentação de um documento NCL em uma base privada. Todos os eventos de documentos que foram previamente pausados pelo comando de edição pauseDocument devem obrigatoriamente ser retomados.

ncledit.addRegion (baseId, documentId, regionBaseId, regionId, region: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
regionBaseId	Identifica um elemento <regionBase>
regionId	Identifica um elemento <region> da NCL
region	Definição do elemento <region> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Adiciona um elemento <region> como filho de outra <region> na <regionBase> de um documento NCL.

ncledit.removeRegion (baseId, documentId, regionId: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
regionId	Identifica um elemento <region> da NCL

Valores de retorno

true	Comando bem sucedido
false	Comando falhou

Descrição

Remove um elemento <region> da <regionBase> de um documento NCL em uma base privada.

ncledit.addRegionBase (baseId, documentId, regionBase: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
regionBase	Definição do elemento <regionBase> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Adiciona um elemento <regionBase> ao elemento <head> de um documento NCL em uma base privada.

ncledit.removeRegionBase (baseId, documentId, regionBaseId: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
regionBaseId	Identifica um elemento <regionBase> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Remove um elemento <regionBase> do elemento <head> de um documento NCL em uma base privada.

ncledit.addRule (baseId, documentId, rule: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
rule	Definição do elemento <rule> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Adiciona um elemento <rule> à <ruleBase> de um documento NCL em uma base privada.

ncledit.removeRule (baseId, documentId, ruleId: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
ruleId	Identifica um elemento <rule> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Remove um elemento <rule> da <ruleBase> de um documento NCL em uma base privada.

ncledit.addRuleBase (baseId, documentId, ruleBase [, timeReference: number or table])

Argumentos

baseId	String que identifica uma base privada
documentId	String que identifica um documento NCL
ruleBase	String com uma definição do elemento <ruleBase> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Adiciona um elemento <ruleBase> ao elemento <head> de um documento NCL em uma base privada.

ncledit.removeRuleBase (baseId, documentId, ruleBaseId: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
ruleBaseId	Identifica um elemento <ruleBase> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Remove um elemento <ruleBase> do elemento <head> de um documento NCL em uma base privada.

ncledit.addConnector (baseId, documentId, connector: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
connector	Definição do elemento <connector> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Adiciona um elemento <connector> à <connectorBase> de um documento NCL em uma base privada.

ncledit.removeConnector (baseId, documentId, connectorId: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
connectorId	Identifica um elemento <connector> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Remove um elemento <connector> da <connectorBase> de um documento NCL em uma base privada.

ncledit.addConnectorBase (baseId, documentId, connectorBase: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
connectorBase	Definição do elemento <connectorBase> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Adiciona um elemento <connectorBase> ao elemento <head> de um documento NCL em uma base privada.

ncledit.removeConnectorBase (baseId, documentId, connectorBaseId: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
connectorBaseId	Identifica um elemento <connectorBase> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Remove um elemento <connectorBase> do elemento <head> de um documento NCL em uma base privada.

ncledit.addDescriptor (baseId, documentId, descriptor: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
descriptor	Definição do elemento <descriptor> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Adiciona um elemento <descriptor> à <descriptorBase> de um documento NCL em uma base privada.

ncledit.removeDescriptor (baseId, documentId, descriptorId: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
descriptorId	Identifica um elemento <descriptor> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Remove um elemento <descriptor> da <descriptorBase> de um documento NCL em uma base privada.

ncledit.addDescriptorSwitch (baseId, documentId, descrSwitch: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
descrSwitch	Definição do elemento <descriptorSwitch> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Adiciona um elemento <descriptorSwitch> à <descriptorBase> de um documento NCL em uma base privada.

ncledit.removeDescriptorSwitch (baseId, documentId, descrSwitchId: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
descrSwitchId	Identifica um elemento <descriptorSwitch> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Remove um elemento <descriptorSwitch> da <descriptorBase> de um documento NCL em uma base privada.

ncledit.addDescriptorBase (baseId, documentId, descriptorBase: string [, timeReference number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
descriptorBase	Definição do elemento <descriptorBase> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Adiciona um elemento <descriptorBase> ao elemento <head> de um documento NCL em uma base privada.

ncledit.removeDescriptorBase (baseId, documentId, descrBaseId: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
descrBaseId	Identifica um elemento <descriptorBase> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Remove um elemento <descriptorBase> do elemento <head> de um documento NCL em uma base privada.

ncledit.addTransition (baseId, documentId, transition: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
transition	Definição do elemento <transition> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Adiciona um elemento <transition> à <transitionBase> de um documento NCL em uma base privada.

ncledit.removeTransition (baseId, documentId, transitioned: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
transitionId	Identifica um elemento <transition> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Remove um elemento <transition> da <transitionBase> de um documento NCL em uma base privada.

ncledit.addTransitionBase (baseId, documentId, transitionBase: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
transitionBase	Definição do elemento <transitionBase> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Adiciona um elemento <transitionBase> ao elemento <head> de um documento NCL em uma base privada.

ncledit.removeTransitionBase (baseId, documentId, transBaseId: string [, timeReference: number or table]) -> boolean: ret

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
transBaseId	Identifica um elemento <transitionBase> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Remove um elemento <transitionBase> do elemento <head> de um documento NCL em uma base privada.

ncledit.addImportBase (baseId, documentId, docBaseId, importBase: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
docBaseId	Identifica um elemento base do NCL (<regionBase>, <descriptorBase>, <ruleBase>, <transitionBase> ou <connectorBase>)
importBase	Definição do elemento <importBase> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Adiciona um elemento <importBase> a um elemento base (<regionBase>, <descriptorBase>, <ruleBase>, <transitionBase> ou <connectorBase>) de um documento NCL em uma base privada.

ncledit.removeImportBase (baseId, documentId, docBaseId, documentURI: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
docBaseId	Identifica um elemento base do NCL (<regionBase>, <descriptorBase>, <ruleBase>, <transitionBase> ou <connectorBase>)
documentURI	Identifica um elemento <importBase> da NCL através de seu atributo documentURI

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Remove um elemento <importBase> de um elemento base, identificado pelo parâmetro docBaseId de um documento NCL em uma base privada.

ncledit.addImportedDocumentBase (baseId, documentId, impDocBase: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
impDocBase	Definição do elemento <importedDocumentBase> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Adiciona um elemento <importedDocumentBase> ao elemento <head> de um documento NCL em uma base privada.

ncledit.removeImportedDocumentBase (baseId, documentId, impDocBaseId: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
impDocBaseId	Identifica um elemento <importedDocumentBase> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Remove um elemento <importedDocumentBase> do elemento <head> de um documento NCL em uma base privada.

ncledit.addImportNCL (baseId, documentId, importNCL: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
importNCL	Definição do elemento <importNCL> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Adiciona um elemento <importNCL> ao elemento <importedDocumentBase> de um documento NCL em uma base privada.

---

ncledit.removeImportNCL (baseId, documentId, documentURI: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
documentURI	Identifica um elemento <importNCL> da NCL através de seu atributo documentURI

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Remove um elemento <importNCL> do elemento <importedDocumentBase> de um documento NCL em uma base privada.

---

ncledit.addNode (baseId, documentId, compositeId, node: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
compositeId	Identifica um nó *de composição* NCL (<body>, <context> ou <switch>)
node	Identifica uma definição do elemento nó do NCL (<media>, <context> ou <switch>)

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Adiciona um nó NCL (elemento <media>, <context> ou <switch>) a um nó de composição (elemento <body>, <context> ou <switch>) de um documento NCL em uma base privada.

---

ncledit.removeNode (baseId, documentId, compositeId, nodeId: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
compositeId	Identifica um nó *de composição* NCL (<body>, <context> ou <switch>)
nodeId	Identifica um nó do NCL (<media>, <context> ou <switch>)

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Remove um nó NCL (elemento <media>, <context> ou <switch>) de um nó *de composição* (elemento <body>, <context> ou <switch>) de um documento NCL em uma base privada.

---

ncledit.addInterface (baseId, documentId, nodeId, interface: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
nodeId	Identifica um nó do NCL (<media>, <body>, <context> ou <switch>)
interface	Identifica uma definição de interface do NCL (<port>, <area>, <property> ou <switchPort>)

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Adiciona uma interface do NCL (elemento <port>, <area>, <property> ou <switchPort>) a um nó (elemento <media>, <body>, <context> ou <switch>) de um documento NCL em uma base privada.

---

ncledit.removeInterface (baseId, documentId, nodeId, interfaceId: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
nodeId	Identifica um nó do NCL (<media>, <body>, <context> ou <switch>)
interfaceId	Identifica uma interface do NCL (<port>, <area>, <property> ou <switchPort>)

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Remove uma interface do NCL (elemento <port>, <area>, <property> ou <switchPort>) de um nó (elemento <media>, <body>, <context> ou <switch>) de um documento NCL em uma base privada.

ncledit.addLink (baseId, documentId, compositeId, link: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
compositeId	Identifica um nó *de composição* NCL (<body>, <context> ou <switch>)
link	Definição do elemento <link> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Adiciona um elemento <link> da NCL a um nó *de composição* (elemento <body>, <context> ou <switch>) de um documento NCL em uma base privada.

ncledit.removeLink (baseId, documentId, compositeId, linked: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
compositeId	Identifica um nó de composição NCL (<body>, <context> ou <switch>)
linkId	Identifica um elemento <link> da NCL

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Remove um elemento <link> da NCL de um nó de composição (elemento <body>, <context> ou <switch>) de um documento NCL em uma base privada.

ncledit.setPropertyValue (baseId, documentId, nodeId, propertyId, value: string [, timeReference: number or table]) -> ret: boolean

Argumentos

baseId	Identifica uma base privada
documentId	Identifica um documento NCL
nodeId	Identifica um nó NCL (<body>, <context>, <switch>, <media>) contendo a propriedade a ser modificada
propertyId	Identifica um elemento <property> da NCL
value	Valor a ser ajustado

Valores de retorno

true	Comando bem-sucedido
false	Comando falhou

Descrição

Define o valor para uma propriedade. O propertyId deve obrigatoriamente identificar um atributo *name* do elemento <property> ou um atributo *id* do elemento <switchPort>. A <property> ou <switchPort> deve obrigatoriamente pertencer a um nó (elemento <body>, <context>, <switch> ou <media>) de um documento NCL em uma base privada identificado pelos parâmetros.

10.3.3 Módulo *canvas*

10.3.3.1 Objeto *canvas*

Quando um objeto de mídia NCLua é iniciado, a região do elemento <media> correspondente (do tipo application/x-ginga-NCLua) fica disponível como a variável global *canvas* para o *script* Lua. Se o elemento de <media> não tiver nenhuma região especificada (propriedades *left, right, top and bottom*), então o valor para *canvas* deve ser estabelido como “nil”.

Exemplificando: para um região definida em um documento NCL como:

```
<region id="luaRegion" width="300" height="100" top="200" left="20"/>
```

A variável `'canvas'` em um objeto de mídia associado à região “luaRegion” é ligada a um objeto canvas de tamanho 300x100, associada com a região (20,200).

Um canvas oferece uma API gráfica para ser usada por aplicações NCLua. Através dela é possível desenhar linhas, retângulos, fontes, imagens etc.

Um canvas guarda em seu estado atributos sob os quais as primitivas de desenho operam, por exemplo, se seu atributo de cor for azul, uma chamada a `canvas:drawLine()` desenha uma linha azul no canvas.

As coordenadas passadas são sempre relativas ao ponto mais à esquerda e ao topo do canvas (0,0).

10.3.3.2 Construtores

Através de qualquer canvas é possível criar novos canvas e combiná-los através de operações de composição.

---

canvas:new (image_path: string) -> canvas: object

Argumentos

image_path	Caminho da imagem

Valor de retorno

canvas	Canvas representando a imagem

Descrição

Retorna um novo canvas cujo conteúdo é a imagem recebida como parâmetro.

O novo canvas deve obrigatoriamente manter os aspectos de transparência da imagem original.

---

canvas:new (width, height: number) -> canvas: object

Argumentos

width	Largura do canvas
height	Altura do canvas

Valores de retorno

canvas	Novo canvas

Descrição

Retorna um novo canvas com o tamanho recebido.

Inicialmente todos os pixels devem ser transparentes, obrigatoriamente.

10.3.3.3 Atributos

Todos os métodos de atributos possuem o prefixo attr e servem tanto para ler quanto para alterar um atributo (com algumas exceções).

Quando o método é chamado sem parâmetros de entrada o valor corrente do atributo é retornado, em contra-partida, quando é chamado com parâmetros, esses devem ser os novos valores do atributo.

---

canvas:attrSize () -> width, height: number

Argumentos

Valores de retorno

width	Largura do canvas
height	Altura do canvas

Descrição

Retorna as dimensões do canvas.

É importante observar que não é permitido alterar as dimensões de um canvas.

---

canvas:attrColor (R, G, B, A: number)

Argumentos

R	Componente vermelha da cor
G	Componente verde da cor
B	Componente azul da cor
A	Componente alpha da cor

Descrição

Altera a cor do canvas.

As cores são passadas em RGBA, onde A varia de 0 (totalmente transparente) a 255 (totalmente opaco).

As primitivas (ver 10.2.3.4) são desenhadas com a cor desse atributo do canvas.

O valor inicial é ‘0,0,0,255’ (preto).

---

canvas:attrColor (clr_name: string)

Argumentos

clr_name	Nome da cor

Altera a cor do canvas.

As cores são passadas através de uma string correspondendo a uma das 16 cores NCL pré-definidas:

'white', 'aqua', 'lime', 'yellow', 'red', 'fuchsia', 'purple', 'maroon',

'blue', 'navy', 'teal', 'green', 'olive', 'silver', 'gray', 'black'

Para valores em string, o alpha é opaco (correspondendo a “A = 255”).

As primitivas (ver 10.2.3.4) são desenhadas com a cor desse atributo do canvas.

O valor inicial é 'black'.

---

canvas:attrColor () -> R, G, B, A: number

Valores de retorno

R	Componente vermelha da cor
G	Componente verde da cor
B	Componente azul da cor
A	Componente alpha da cor

Descrição

Retorna a cor do canvas.

canvas:attrFont (face: string; size: number; style: string)

Argumentos

face	Nome da fonte
size	Tamanho da fonte
style	Estilo da fonte

Descrição

Altera a fonte do canvas.

As seguintes fontes devem obrigatoriamente estar disponíveis: 'Times', 'Courier' e 'Helvetica'.

O tamanho é em pixels e representa a altura máxima de uma linha escrita com a fonte escolhida.

Os estilos possíveis são: 'bold', 'italic' ou 'bold-italic'. O valor `nil` assume que nenhum dos estilos será usado.

Qualquer valor passado não suportado deve obrigatoriamente gerar um erro.

O valor inicial da fonte é indeterminado.

canvas:attrFont () -> face: string; size: number; style: string

Valores de retorno

face	Nome da fonte
size	Tamanho da fonte
style	Estilo da fonte

Descrição

Retorna a fonte do canvas.

canvas:attrClip (x, y, width, height: number)

Argumentos

x	Coordenada da área de *clipping*
y	Coordenada da área de *clipping*
width	Largura da área de *clipping*
height	Altura da área de *clipping*

Descrição

Altera a área de *clipping* do canvas.

As primitivas de desenho (ver 10.2.3.4) e o método `canvas:compose()` só operam dentro desta região de clipping.

O valor inicial é o canvas inteiro.

canvas:attrClip () -> x, y, width, height: number

Valores de retorno

x	Coordenada da área de *clipping*
y	Coordenada da área de *clipping*
width	Largura da área de *clipping*
height	Altura da área de *clipping*

Descrição

Retorna a área de *clipping* do canvas.

10.3.3.4 Primitivas

Todos os métodos a seguir levam em consideração os atributos do canvas.

canvas:drawLine (x1, y1, x2, y2: number)

Argumentos

x1	Extremidade 1 da linha
y1	Extremidade 1 da linha
x2	Extremidade 2 da linha
y2	Extremidade 2 da linha

Descrição

Desenha uma linha com suas extremidades em (x1,y1) e (x2,y2).

canvas:drawRect (mode: string; x, y, width, height: number)

Argumentos

mode	Modo de desenho
x	Coordenada do retângulo
y	Coordenada do retângulo
width	Largura do retângulo
height	Altura do retângulo

Descrição

Função para desenho e preenchimento de retângulos.

O parâmetro mode pode receber 'frame' para desenhar apenas a moldura do retângulo ou 'fill' para preenchê-lo.

canvas:drawPolygon (mode: string) -> drawer: function

Argumentos

mode	Modo de desenho

Valores de retorno

f	Função de desenho

Descrição

Método para desenho e preenchimento de polígonos.

O parâmetro mode recebe o valor 'open', para desenhar o polígono sem ligar o último ponto ao primeiro; 'close', para desenhar o polígono ligando o último ponto ao primeiro; or 'fill', para desenhar o polígono ligando o último ponto ao primeiro e colorir a região interior.

A função canvas:drawPolygon retorna uma função anônima "drawer" com a assinatura:

```
function (x, y) end
```

A função retornada recebe as coordenadas do próximo vértice do polígono e retorna a si mesmo com resultado. Esse procedimento recorrente facilita a composição:

```
canvas:drawPolygon('fill')(1,1)(10,1)(10,10)(1,10)()
```

Ao receber nil a função "drawer" efetua a operação encadeada. Qualquer chamada subsequente deve obrigatoriamente gerar um erro.

---

canvas:drawEllipse (mode: string; xc, yc, width, height, ang_start, ang_end: number)

Argumentos

mode	Modo de desenho
xc	Centro da elipse
yc	Centro da elipse
width	Largura da elipse
height	Altura da elipse
ang_start	Ângulo de início
ang_end	Ângulo de fim

Descrição

Desenha elipses e outras primitivas similares tais como círculos, arcos e setores.

O parâmetro mode pode receber 'arc' para desenhar apenas a circunferência ou 'fill' para preenchimento interno.

---

canvas:drawText (text: string; x, y: number)

Argumentos

text	Texto a ser desenhado
x	Coordenada do texto
y	Coordenada do texto

Descrição

Desenha o texto passado na posição (x,y) do canvas utilizando a fonte configurada em `canvas:attrFont()`.

10.3.3.5 Miscelânea

---

canvas:flush ()

Descrição

Atualiza o canvas após operações de desenho e de composição.

É suficiente chamá-lo apenas uma vez após uma sequência de operações.

canvas:compose (x, y: number; src: canvas; [src_x, src_y, src_width, src_height: number])

Argumentos

x	Posição da composição
y	Posição da composição
src	Canvas a ser composto
src_x	Posição da composição no canvas src
src_y	Posição da composição no canvas src
src_width	Largura da composição no canvas src
src_height	Altura da composição no canvas src

Descrição

Faz sobre o canvas (canvas de destino), em sua posição (x,y), a composição pixel a pixel com src (canvas de origem).

Os outros parâmetros são opcionais e indicam que parte do canvas src compor. Quando ausentes, o canvas inteiro é composto.

Essa operação chama src:flush() automaticamente antes da composição.

A composição satisfaz a equação:

Cd = Cs*As + Cd*(255 - As)/255

Ad = As*As + Ad*(255 - As)/255

onde:

Cd = cor do canvas de destino (canvas)

Ad = alfa do canvas de destino (canvas)

Cs = cor do canvas de origem (src)

As = alfa do canvas de origem (src)

Após a operação o canvas de destino possui o resultado da composição e src não sofre qualquer alteração.

canvas:pixel (x, y, R, G, B, A: number)

Argumentos

x	Posição do *pixel*
y	Posição do *pixel*
R	Componente vermelha da cor
G	Componente verde da cor
B	Componente azul da cor
A	Componente alpha da cor

Descrição

Altera a cor de um *pixel* do canvas.

canvas:pixel (x, y: number) -> R, G, B, A: number

Argumentos

x	Posição do *pixel*
y	Posição do *pixel*

Valores de retorno

R	Componente vermelha da cor
G	Componente verde da cor
B	Componente azul da cor
A	Componente alpha da cor

Descrição

Retorna a cor de um *pixel* do canvas.

canvas:measureText (x, y: number; text: string) -> x1, x2, y1, y2: number

Argumentos

x	Coordenada do texto
y	Coordenada do texto
texto	Texto a ser medido

Valores de retorno

x1	Extremidade mais à esquerda do texto
y1	Extremidade mais à esquerda do texto
x2	Extremidade mais à direita do texto
y2	Extremidade mais à direita do texto

Descrição

Retorna as coordenadas limitrofes para o texto passado, caso ele fosse desenhado na posição (x,y) do canvas com a fonte configurada em canvas:attrFont().

10.3.4 Módulo *event*

10.3.4.1 Visão geral

Este módulo oferece uma API para tratamento de eventos. Através dele o formatador NCL e uma aplicação NCLua podem se comunicar de maneira assíncrona.

Uma aplicação também pode usufruir desse mecanismo internamente, através de eventos da classe "user".

Provavelmente o uso mais comum de aplicações NCLua será tratando eventos: sejam eles eventos NCL (ver 7.2.8) ou de interação com o usuário (pelo controle remoto, por exemplo).

Durante sua iniciação, antes de se tornar orientado a eventos, um *script* Lua deve obrigatoriamente registrar uma função de tratamento de eventos. Após a iniciação, qualquer ação tomada pela aplicação é somente em resposta a um evento enviado pelo formatador NCL à função "handler".

=== example.lua ===

```
...                -- código de iniciação

function handler (evt)

  ...              -- código tratador

end

event.register(handler)   -- registro do tratador no middleware
```

=== fim ===

Dentre os tipos de eventos que podem chegar ao handler estão todos os eventos gerados pelo formatador NCL. Um *script* Lua também é capaz de gerar eventos, ditos "espontâneos", com uma chamada à função `event.post(evt)`.

10.3.4.2 Funções

---

event.post (dst: string; evt: event) -> sent: boolean; err_msg: string

Argumentos

dst	Destinatário do evento
evt	Evento a ser postado

Return values

sent	If the event was successfully sent
err_msg	Error message in case of errors

Descrição

Posta o evento passado.

O parâmetro "dst" é o destinatário do evento e pode assumir os valores "in" (envio para a própria aplicação) e "out" (envio para o formatador NCL).

---

event.timer (time: number, f: function)

Argumentos

time	Tempo em milisegundos
f	Função de *callback*

Descrição

Cria um timer que expira após time (em milisegundos) e então chama a função f.

A assinatura de f é simples, sem recebimento de parâmetros:

```
function f () end
```

Uma vez criado, não há como cancelar o timer.

O valor de 0 milisegundos é válido. Nesse caso, event.timer() deve, obrigatoriamente, retornar imediatamente e `f` deve ser chamada assim que possível.

---

event.register (f: function)

Argumentos

f	Função de *callback*.

Descrição

Registra a função passada como um *listener* de eventos, isto é, sempre que ocorrer um evento, `f` será chamada. A função `f` é, assim, a função de tratamento de eventos (function "handler").

A assinatura de `f` é:

```
function f (evt) end -> handled: boolean
```

Onde `evt` é o evento que, ao ocorrer, ativa a função.

A função pode retornar "true", para sinalizar que o evento foi tratado e, portanto, não deve ser enviado a outros tratadores.

É recomendado que a função, definida pela aplicação, retorne rapidamente, já que, enquanto ela estiver executando, nenhum outro evento será processado.

O formatador NCL deve obrigatoriamente garantir que as funções recebam os eventos na ordem em que foram registradas, e se nenhuma delas retornar o valor "true", o formatador NCL deve notificar os outros tratadores registrados.

---

event.unregister (f: function)

Argumentos

f	Função de *callback*

Descrição

Tira do registro a função passada como um *listener*, isto é, novos eventos não serão mais passados a `f`.

---

event.uptime () -> ms: number

Valores de retorno

ms	Tempo em milisegundos

Descrição

Retorna o número de milisegundos decorridos desde o início da aplicação.

10.3.4.3 Classes de eventos

A função `event.post()` e o "handler" registrado em `event.register()` recebem eventos como parâmetros.

Um evento é descrito por uma tabela Lua normal, onde o campo `class` é obrigatório e identifica a classe do evento.

As seguintes classes de eventos são definidas:

Classe `key`:

evt = { class='key', type: string, key: string}

* type pode ser 'press' ou 'release'.

* key é o valor da tecla em questão; a tabela "event.keys" possui todas as teclas disponíveis na NCL.

EXEMPLO evt = { class='key', type='press', key=”0”}

Classe `ncl`:

As relações entre os nós de mídia NCL são baseadas em eventos. Lua tem acesso a esses eventos através da `Classe ncl`.

Os eventos podem agir nas duas direções, isto é, o formatador pode enviar eventos de ação para mudar o estado do exibidor Lua, e este, por sua vez, pode disparar eventos de transição para indicar mudanças de estado.

Nos eventos, o campo type deve obrigatoriamente assumir um dos três valores:
'presentation', 'selection' ou 'attribution'

Eventos podem ser direcionados a âncoras específicas ou ao nó como um todo, isto é identificado no campo area, que assume o nó inteiro, quando ausente.

No caso de um evento gerado pelo formatador, o campo action deve obrigatoriamente ter um dos valores:

'start', 'stop', 'abort', 'pause', 'resume' e 'set'

Tem-se então:

Tipo ‘presentation’:

evt = { class='ncl', type='presentation', area='?', action='start'/'stop'/'abort'/'pause'/'resume' }

Tipo ‘attribution’:

evt = { class='ncl', type='attribution', area='?', action='set' }

Para eventos gerados pelo exibidor Lua, o campo "transition" deverá obrigatoriamente assumir um dos valores

'starts', 'stops', 'aborts', 'pauses', 'resumes' e 'stops'

Tem-se então:

Tipo ‘presentation’:

evt = { class='ncl', type='presentation', area='?', transition='starts'/'stops'/'aborts'/'pauses'/'resumes' }

Tipo ‘selection’:

evt = { class='ncl', type='selection', area='?', transition='stops' }

Tipo ‘attribution’:

evt = { class='ncl', type='attribution', area='?', transition='stops' }

tcp class:

O uso do canal de interatividade é realizado por meio desta classe de eventos. A classe é opcional no caso da implementação Ginga para receptores *full*-seg.

Uma aplicação NCLua envia dados por um canal de interatividade, postando eventos na forma:

evt = { class='tcp', to='addr:port', value=string }

De forma similar, uma aplicação NCLua recebe dados transportados por um canal de interatividade fazendo uso de eventos na forma:

evt = { class='tcp', from='addr:port', value=string }

NOTA Questões tais como autenticação, temporização/reconexão de uma conexão, se uma conexão deve ser mantida aberta ou não etc., recomenda-se serem tratadas em uma implementação específica do *middleware*.

sms class:

O compartamento de envio e recebimento por meio de SMS é muito semelhante ao definido na classe *tcp*. Também como aquela classe, a classe *sms* é opcional no caso da implementação Ginga para receptores *full*-seg.

Uma aplicação NCLua envia dados por SMS, postando eventos na forma:

evt = { class='sms', to='phone number', value=string }

De forma similar, uma aplicação NCLua recebe dados transportados por SMS fazendo uso de eventos na forma:

evt = { class='sms', from='phone number', value=string }

NOTA Questões como autenticação etc., recomenda-se serem tratadas em uma implementação específica do *middleware*.

Classe user:

Utilizando Classe user, aplicações podem estender suas funcionalidades, criando seus próprios eventos.

Nessa classe, nenhum campo está definido (além, obviamente, do campo class).

10.3.5 Módulo *settings*

Exporta a tabela *settings* com variáveis definidas pelo autor do documento NCL e variáveis de ambiente reservadas, contidas no nó application/x-ginga-settings.

Não é permitido atribuir valores aos campos representando variáveis no nós *settings*. Um erro deve ser gerado caso uma tentativa de atribuição seja feita. Propriedades de um nós *settings* só podem ser modificadas por meio de elos NCL.

A tabela *settings* é particiona seus grupos em várias subtabelas, correspondendo a cada grupo do nó application/x-ginga-settings. Por exemplo, em um objeto NCLua, a variável do nó *settings* "system.CPU" é referida como settings.system.CPU.

Exemplos de uso:

lang = settings.system.language

age = settings.user.age

val = settings.default.selBorderColor

settings.service.myVar = 10

settings.user.age = 18 --> ERRO!

10.3.6 Módulo *persistent*

Aplicações NCLua podem salvar dados em uma área restrita do middleware e recuperá-los entre execuções. O exibidor Lua permite a uma aplicação NCLua persistir um valor para ser posteroirmente usado por ela ou por um outro objeto procedural. Para tanto, o exibidor define uma área reservada, inaccessível a objetos NCL não procedurais. Esta área é dividida entre os grupos "*service*", "*channel*" e "*shared*", com a mesma semântica dos grupos homônimos do nó NCL *settings*. Não existe nenhuma variável pré-definida ou reservada nesses grupos, e objetos procedurais podem atribuir valores a essas variáveis diretamente. Recomenda-se que outras linguagens procedurais, Java em particular para os objetos NCLets (<media> elements of type application/x-ginga-NCLet), ofereçam uma API dando acesso à mesma área.

Neste modulo *persistent,* Lua oferece uma API para exportar a tabela *persistent* com as variáveis definidas na área reservada.

O uso da tabela *persistent* é semelhante ao uso da tabela *settings* exceto pelo fato que, neste caso, o código procedural pode mudar os valores dos campos.

Exemplos de uso:

persistent.service.total = 10

color = persistent.shared.color

10.4 Lua-API para Ginga-J

10.4.1 Mapeamento

Dependendo da configuração do *middleware*, é possível ter acesso em Lua à mesma API fornecida pelo Ginga-J, a fim de ter acesso a alguns recursos do decodificador e facilidades do Ginga. A API para Ginga-J fornecida em Lua é opcional, mas quando oferecida deve obrigatoriamente seguir a mesma especificação apresentada na ABNT NBR 15606-4.

10.4.2 Pacotes

As hierarquias dos pacotes Java que compõem a API do Ginga-J são mapeadas para hierarquias equivalentes dos pacotes Lua que têm um pacote raiz em comum, chamado ginga. Mais especificamente, um pacote "x" na API do Ginga-J é mapeada para um pacote Lua ginga.x equivalente. Nesse contexto, um pacote Lua *equivalente* significa um pacote que contenha classes e subpacotes equivalentes aos definidos no pacote Java.

O conjunto de pacotes Ginga-J que estarão disponíveis no ambiente de execução de um *script* Lua pode ser restringido por políticas de segurança. Se um pacote "x" da API do Ginga-J estiver disponível no ambiente Lua, ginga.x manterá uma referência para uma tabela Lua com todas as definições relacionadas a "x" (classes e subpacotes). Caso contrário, ginga.x é uma referência nil. Alguns exemplos de mapeamentos de nome dos pacotes do Ginga-J para pacotes Lua são apresentados na Tabela 58.

Tabela 58 — Exemplos de mapeamentos de nome entre os pacotes Ginga-J e pacotes Lua

Pacote Ginga-J	Pacote Lua
org.sbtvd.net.tuning	ginga.org.sbtvd.net.tuning

org.sbtvd.media	ginga.org.sbtvd.media
javax.media	ginga.javax.media
org.dvb	ginga.org.dvb
org.havi	ginga.org.havi
org.davic	ginga.org.davic

10.4.3 Tipos básicos

Os tipos de dados básicos do Java, usados na API Ginga-J, são mapeados para os tipos de dados básicos de Lua. Esses mapeamentos são apresentados na Tabela 59. Além dos tipos primitivos de Java, a tabela também especifica o mapeamento de *strings* e matrizes.

Tabela 59 — Mapeamento de tipos de dados básicos

Tipo Java	Tipo Lua
short	number
int	number
long	number
float	number
double	number
byte	number
char	string (with only one character)
boolean	boolean
Array objects	table
String objects	string

10.4.4 Classes

Toda classe Java da API Ginga-J é representada em Lua como uma tabela, definida em seu respectivo pacote. Por exemplo, a classe

org.sbtvd.net.tuning.ChannelManager

é representada em Lua como uma entrada ChannelManager no pacote ginga.org.sbtvd.net.tuning, ou seja, essa classe é acessada através de

ginga.org.sbtvd.net.tuning.ChannelManager.

Todos os membros estáticos de uma classe Java são mapeados para campos da tabela Lua equivalente. Cada classe representada em Lua também tem uma operação *newInstance*, que desempenha o papel de um *construtor*.

10.4.5 Objetos

Toda vez que o método *newInstance* fornecido por uma classe representada em Lua é chamado, ele retorna uma nova instância (*objeto*) dessa classe. O objeto retornado é uma tabela Lua que tem todos os membros de instância especificados por sua classe (campos e métodos públicos).

10.4.6 Objetos de *callback* (observadores)

Muitos métodos definidos na API Ginga-J esperam receber um objeto observador (*listener*) como parâmetro. Esses objetos observadores podem ser implementados em Lua como tabelas que têm todos os métodos especificados na interface do observador.

10.4.7 Exceções

As exceções do Java também são mapeadas para tabelas Lua, seguindo as mesmas regras para mapear objetos do Java para Lua. Para gerar uma exceção, é recomendado que um objeto observador implementado em Lua use a função error fornecida por Lua (ver Anexo B). Para capturar uma exceção gerada por um método da API, é recomendado que o *script* Lua use a função pcall (ver Anexo B).

11 Ponte

11.1 Revisão

A ponte de mão-dupla entre o Ginga-NCL e o Ginga-J é feita:

— em um sentido, através dos relacionamentos do NCL, definidos nos elementos <link> que se referem aos elementos <media> que representam os códigos Xlet (tipo application/x-ginga-NCLet) suportados pelo Ginga-J; e através dos *scripts* Lua (elementos <media> do tipo application/x-ginga-NCLua) que referenciam os métodos do Ginga-J;

— no caminho inverso, através das funções do Ginga-J que podem monitorar qualquer evento NCL e também podem comandar alterações em elementos e propriedades NCL, através de relacionamentos definidos em elementos <link> ou através de comandos de edição do NCL.

11.2 Ponte através dos elementos NCL <link> e <media>

O Ginga-NCL pode atuar sobre o Ginga-J através de elementos <link> e através de elementos <media> do tipo application/x-ginga-NCLet.

De modo análogo ao conteúdo de mídia convencional, o NCL permite que o código Xlet seja sincronizado com outros objetos NCL (procedurais ou não). Os autores do NCL podem definir elos NCL para iniciar, parar, pausar, retomar ou abortar a execução de um código procedural Xlet (representado por um elemento <media> do tipo application/x-ginga-NCLet) como fazem para conteúdos de apresentação usual (ver 8.5). Um *player* (exibidor) NCLet (baseado na máquina Java) deve obrigatoriamente fazer a interface do ambiente de execução procedural com o formatador NCL (ver 8.5).

Um elemento <media> contendo um código Java pode definir âncoras (através de elementos <area> e atributos (através de elementos <property>). O *player* deve obrigatoriamente controlar a máquina de estado dos eventos associados com esses elementos de interface.

O código Xlet pode ser associado a elementos <area>. Se os elos externos iniciarem, pararem, pausarem ou retomarem a apresentação da âncora, as *callBacks* no código Xlet devem obrigatoriamente ser disparadas. Por outro lado, o código Xlet pode comandar o início, parada, pausa, retomada ou aborto dessas âncoras através de uma API oferecida pela linguagem procedural. As transições causadas por esses comandos podem ser usadas como condições dos elos NCL para disparar ações em outros objetos NCL do mesmo documento. Assim, uma sincronização de duas vias pode ser estabelecida entre o código Xlet e o restante do documento NCL.

Um elemento <property> definido como filho do elemento <media> do tipo application/x-ginga-NCLet pode ser mapeado para um método do código Xlet ou para um atributo do código Xlet. Quando é mapeado para um método do código, uma ação “set” do elo aplicada ao atributo deve obrigatoriamente causar a execução do método, com os valores definidos pela ação do elo sendo interpretados como parâmetros de entrada do método. O atributo *name* do elemento <property> deve obrigatoriamente ser usado para identificar o método do código procedural. Quando o elemento <property> é mapeado para um atributo do código Xlet, a ação “set” deve obrigatoriamente atribuir um valor ao atributo.

O elemento <property> pode também ser associado a um *assessment role* de um elo NCL. Nesse caso, o formatador NCL deve obrigatoriamente consultar o valor do atributo, a fim de avaliar a expressão do elo. Se o elemento <property> for mapeado para um atributo do código, o valor do atributo deve obrigatoriamente ser retornado pelo *player* Xlet para o formatador NCL. Se o elemento <property> for mapeado para um método do código, o método deve obrigatoriamente ser chamado e seu valor deve obrigatoriamente ser retornado pelo *player* Xlet para o formatador NCL.

11.3 Ponte através das funções Lua e métodos do Ginga-J

Dependendo da configuração do *middleware*, é possível ter acesso em Lua à mesma API fornecida pelo Ginga-J, a fim de ter acesso a alguns recursos do decodificador e facilidades do Ginga. A API fornecida em Lua deve obrigatoriamente seguir a mesma especificação apresentada na ABNT NBR 15606-4.

O Ginga-J também oferece uma API que permite que o código Xlet consulte quaisquer valores de propriedade predefinidos ou dinâmico do nó de configurações do NCL (elemento <media> do tipo "application/x-ginga-settings").

Além do mais, o Ginga-J oferece API que fornecem um conjunto de métodos para dar suporte aos comandos de edição do NCL e comandos do Gerenciador da Base Privada.

Para obter uma completa descrição das sintaxes e semântica da API NCL do Ginga-J, ver ABNT NBR 15606-4.

12 Requisitos de codificação de mídias e métodos de transmissão referenciados em documentos NCL

12.1 Uso do canal de interatividade

Um formatador NCL deve obrigatoriamente ignorar com êxito qualquer método de codificação ou transmissão que não seja suportado pelo navegador. A fim de adquirir conteúdo de dados que seja referenciado pelos elementos <media> através de um protocolo de canal interativo específico, os mecanismos especificados para o canal de interatividade do SBTVD devem obrigatoriamente ser utilizados.

12.2 Métodos de codificação e transmissão de vídeo – Dados de vídeo referenciados em elementos <media>

12.2.1 Transmissão de vídeo MPEG-1

12.2.1.1 Transmissão como fluxo elementar de vídeo

Para transmitir conteúdo de vídeo MPEG-1 como um fluxo elementar de vídeo, os dados do vídeo devem obrigatoriamente ser transmitidos como *MPEG-2 packetized elementary stream* (vídeo PES), com o tipo de fluxo especificado em conformidade com a atribuição de tipos de fluxos ISO/IEC 13818-1 (valor 0x01 para vídeo ISO/IEC 11172-2).

12.2.1.2 Transmissão em carrossel de objetos

Para transmitir dados de vídeo MPEG-1 por meio de um carrossel de objetos (valor de tipo de fluxo igual a 0x0B, de acordo com a atribuição de tipos de fluxos DSM-CC em ISO/IEC 13818-6), um dos métodos de transmissão a seguir deve obrigatoriamente ser usado:

a) como um arquivo de fluxo multiplexado em sistemas MPEG-1 (de acordo com a ISO/IEC 11172-1);

b) como um arquivo de fluxo elementar de vídeo MPEG-1;

c) como um arquivo de fluxo multiplexado no formato TS especificado em 12.4.

12.2.2 Transmissão de vídeo MPEG-2

12.2.2.1 Transmissão como fluxo elementar de vídeo

Para transmitir conteúdo de vídeo MPEG-2 como um fluxo elementar de vídeo, os dados do vídeo devem obrigatoriamente ser transmitidos como *MPEG-2 packetized elementary stream* (vídeo PES), com o tipo de fluxo especificado de acordo com a atribuição de tipos de fluxos ISO/IEC 13818-1 (valor 0x02 para vídeo ISO/IEC 13818-2).

12.2.2.2 Transmissão em carrossel de objetos

Para transmitir dados de vídeo MPEG-2 por meio de um carrossel de objetos (valor de tipo de fluxo igual a 0x0B, consultar a atribuição de tipos de fluxos DSM-CC em ISO/IEC 13818-6), um dos métodos de transmissão a seguir deve obrigatoriamente ser usado:

a) como um arquivo de fluxo elementar de vídeo MPEG-2;

b) como um arquivo de fluxo multiplexado no formato TS especificado em 12.4.

12.2.3 Transmissão de vídeo MPEG-4 e H.264|MPEG-4 AVC

12.2.3.1 Transmissão como fluxo elementar de vídeo

Para transmitir conteúdo de vídeo MPEG-4 como um fluxo elementar de vídeo, os dados do vídeo devem obrigatoriamente ser transmitidos como *MPEG-2 packetized elementary stream* (vídeo PES), com o tipo de fluxo especificado de acordo com a atribuição de tipos de fluxos ISO/IEC 13818-1 (valor 0x10 para vídeo ISO/IEC 14496 e H.264|MPEG-4 AVC).

12.2.3.2 Transmissão em carrossel de objetos

Para transmitir dados de vídeo MPEG-4 ou H.264|MPEG-4 AVC por meio de um carrossel de objetos (valor de tipo de fluxo igual a 0x0B, consulte a atribuição de tipos de fluxos DSM-CC em ISO/IEC 13818-6), um dos métodos de transmissão a seguir deve obrigatoriamente ser usado:

a) como um arquivo de fluxo elementar de vídeo MPEG-4 (ou H.264|MPEG-4 AVC);

b) como um arquivo de fluxo multiplexado no formato TS especificado em 12.4.

12.3 Métodos de codificação e transmissão de áudio – dados de áudio referenciados em elementos <media>

12.3.1 Transmissão de áudio MPEG-1

12.3.1.1 Transmissão como fluxo elementar de áudio

Para transmitir conteúdo de áudio MPEG-1 como um fluxo elementar de áudio, os dados do áudio devem obrigatoriamente ser transmitidos como *MPEG-2 packetized elementary stream* (áudio PES), com o tipo de fluxo especificado de acordo com a atribuição de tipos de fluxos ISO/IEC 13818-1 (valor 0x03 para áudio ISO/IEC 11172-3).

12.3.1.2 Transmissão em carrossel de objetos

Para transmitir dados de áudio MPEG-1 por meio de um carrossel de objetos (valor de tipo de fluxo igual a 0x0B, consultar a atribuição de tipos de fluxos DSM-CC em ISO/IEC 13818-6), um dos métodos de transmissão a seguir deve obrigatoriamente ser usado:

a) como um arquivo de fluxo multiplexado em sistemas MPEG-1 (de acordo com a ISO/IEC 11172-1);

b) como um arquivo de fluxo elementar de áudio MPEG-1;

c) como um arquivo de fluxo multiplexado no formato TS especificado em 12.4.

12.3.2 Transmissão de áudio MPEG-2

12.3.2.1 Transmissão como fluxo elementar de áudio

Para transmitir conteúdo de áudio MPEG-2 AAC como um fluxo elementar de áudio, os dados do áudio devem obrigatoriamente ser transmitidos como *MPEG-2 packetized elementary stream* (áudio PES), com o tipo de fluxo especificado de acordo com a atribuição de tipos de fluxos ISO/IEC 13818-1 (valor 0x0F para áudio ISO/IEC 13818-7).

Para transmitir conteúdo de áudio MPEG-2 BC como um fluxo elementar de áudio, os dados do áudio devem obrigatoriamente ser transmitidos como *MPEG-2 packetized elementary stream* (PES), com o tipo de fluxo especificado de acordo com a atribuição de tipos de fluxos ISO/IEC 13818-1 (valor 0x04 para áudio ISO/IEC 13818-3).

12.3.2.2 Transmissão em carrossel de objetos

Para transmitir dados de áudio MPEG-2 por meio de um carrossel de objetos (valor de tipo de fluxo igual a 0x0B, consulte a atribuição de tipos de fluxos DSM-CC em ISO/IEC 13818-6), um dos métodos de transmissão a seguir deve obrigatoriamente ser usado:

a) como um arquivo de fluxo elementar de áudio MPEG-2;

b) como um arquivo de fluxo multiplexado no formato TS especificado em 12.4.

12.3.3 Transmissão de áudio MPEG-4

12.3.3.1 Transmissão como fluxo elementar de áudio

Para transmitir conteúdo de áudio MPEG-4 como um fluxo elementar de áudio, os dados do áudio devem obrigatoriamente ser transmitidos como MPEG-2 *packetized elementary stream* (áudio PES), com o tipo de fluxo especificado em comformidade com a atribuição de tipos de fluxos ISO/IEC 13818-1 (valor 0x11 para áudio ISO/IEC 14496-3).

12.3.3.2 Transmissão em carrossel de objetos

Para transmitir dados de áudio MPEG-4 por meio de um carrossel de objetos (valor de tipo de fluxo igual a 0x0B, consultar atribuição de tipos de fluxos DSM-CC em ISO/IEC 13818-6), um dos métodos de transmissão a seguir deve obrigatoriamente ser usado:

a) como um arquivo de fluxo elementar de áudio MPEG-4;

b) como um arquivo de fluxo multiplexado no formato TS especificado em 12.4.

12.3.4 Transmissão de áudio AC3

12.3.4.1 Transmissão como fluxo elementar de áudio

Para transmitir conteúdo de áudio AC3 como um fluxo elementar de áudio, os dados de áudio devem obrigatoriamente ser transmitidos como MPEG-2 *packetized elementary stream* (áudio PES) com o tipo de áudio especificado como 0x81.

12.3.4.2 Transmissão em carrossel de objetos

Para transmitir dados de áudio AC3 por meio de um carrossel de objetos (valor de tipo de fluxo igual a 0x0B, consulte a atribuição de tipos de fluxos DSM-CC em ISO/IEC 13818-6), um dos métodos de transmissão a seguir deve obrigatoriamente ser usado:

a) como um arquivo de fluxo elementar de áudio AC3;

b) como um arquivo de fluxo multiplexado no formato TS especificado em 12.4.

12.3.5 Transmissão de áudio PCM (AIFF-C)

Recomenda-se que o áudio AIFF-C PCM seja transmitido como um arquivo através de um carrossel de objetos (tipo de fluxo com valor 0x0B, consulte a atribuição de tipos de fluxos DSM-CC em ISO/IEC 13818-6).

12.4 Formato TS para transmissão de vídeo/áudio MPEG – Especificação da codificação de dados

12.4.1 Transmissão de video e audio multiplexados

Para transmitir dados de vídeo MPEG-1/2/4 ou H.264|MPEG-4 AVC junto com dados de áudio MPEG-1/2/4 ou AC3 em arquivos multiplexados em um carrossel de objetos, cada arquivo de vídeo/áudio multiplexado é codificado em um formato TS, conforme definido na ISO/IEC 13818-1.

12.4.2 PSI requerido

Uma tabela PAT deve obrigatoriamente ser descrita. Qualquer PAT deve obrigatoriamente ser descrita com o *program_number* cujo valor é diferente de 0 e este valor deve obrigatoriamente representar um PID da PMT. Os valores disponíveis de *program_number* serão definidos em um regulamento de padrão operacional.

Uma tabela PMT deve obrigatoriamente ser descrita. Qualquer descritor de identificação de fluxo que indique um segundo *loop* deve obrigatoriamente conter um descritor PMT. Caso contrário, um descritor pode ser inserido conforme necessário.

É recomendado que os valores disponíveis para *component_tag* e suas regras de ocorrência em descritores ES e PMT defaults em um segundo *loop* sejam equivalentes a um regulamento operacional padrão dedicado ao fluxo principal do tipo de mídia responsável pela transmissão do fluxo em questão.

Em uma implementação na qual um fluxo de transporte é decodificado a partir de um arquivo que foi transmitido com base na especificação da codificação de dados definida nesta seção e é entregue em uma interface digital de alta velocidade, uma tabela SIT deve obrigatoriamente ser descrita (ver ABNT NBR 15606-1). Em outros casos, as SIT não são necessárias, salvo especificação explícita em contrário.

Qualquer tabela diferente de PAT, PMT e SIT (por exemplo: CAT, NIT, SDT, BAT, EIT, RST, TDT, TOT, PCAT, SDTT e ST – ver ABNT NBR 15601) obrigatoriamente não deve ser descrita.

Uma tabela PAT deve obrigatoriamente ocorrer em um fluxo a uma freqüência não menor que uma vez a cada 100 ms. Uma tabela PMT deve obrigatoriamente ocorrer em um fluxo a uma freqüência não menor que uma vez a cada 100 ms.

Por toda a duração de um arquivo de formato TS, as tabelas PAT e PMT obrigatoriamente não devem ser modificadas ou atualizadas.

12.4.3 Transmissão em carrossel de objetos

Para transmitir um arquivo codificado com a especificação de codificação de dados de acordo com 12.6 em um carrossel de objetos, a transmissão deve obrigatoriamente estar de acordo com a ABNT NBR 15606-1.

12.4.4 Restrições na reprodução

Para ao mesmo tempo receber um serviço por difusão e reproduzir um arquivo TS recebido por um carrossel de objetos, dois sistemas de processamento de fluxo de transporte separados são necessários. As restrições na integração e coordenação de um conteúdo/evento recebido por um serviço de difusão com um arquivo TS não são descritas nesta Norma.

12.5 Esquema de codificação e transmissão de imagens estáticas e gráficos de *bitmap* referenciados por elementos <media>

12.5.1 Transmissão de MPEG-2 I-frame, MPEG-4 I-VOP e H.264|MPEG-4 AVC I-picture

12.5.1.1 Transmissão em video PES para reprodução linear

Para transmitir uma imagem estática em quadros MPEG-2 I por meio de um componente vídeo PES, o esquema de codificação deve obrigatoriamente estar conforme às convenções definidas na ABNT NBR 15606-1. O componente PES deve obrigatoriamente ser transmitido como um fluxo cujo valor de tipo é igual a 0x02.

Para transmitir uma imagem estática em MPEG-4 I-VOP por meio de um componente vídeo PES, o esquema de codificação deve obrigatoriamente estar conforme às convenções definidas na ABNT NBR 15606-1. O componente PES deve obrigatoriamente ser transmitido como um fluxo cujo valor de tipo é igual a 0x10.

Para transmitir uma imagem estática em H.264|MPEG-4 AVC I-picture por meio de um componente vídeo PES, o esquema de codificação deve obrigatoriamente estar conforme às convenções definidas na ABNT NBR 15606-1. O componente PES deve obrigatoriamente ser transmitido como um fluxo cujo valor de tipo é igual a 0x1B.

12.5.1.2 Transmissão em módulo de carrossel para reprodução interativa

Para transmitir uma imagem estática em quadros MPEG-2 I por meio de um módulo de carrossel, o esquema de codificação deve obrigatoriamente estar conforme a ABNT NBR 15606-1. A imagem estática deve obrigatoriamente ser transmitida como um arquivo no módulo.

Para transmitir uma imagem estática em MPEG4-I-VOP por meio de um módulo de carrossel, o esquema de codificação deve obrigatoriamente estar conforme a ABNT NBR 15606-1. A imagem estática deve obrigatoriamente ser transmitida como um arquivo no módulo.

Para transmitir uma imagem estática em H.264|MPEG-4 AVC I-picture por meio de um módulo de carrossel, o esquema de codificação deve obrigatoriamente estar conforme às convenções na ABNT NBR 15606-1. A imagem estática deve obrigatoriamente ser transmitida como um arquivo no módulo.

Nesses casos, o valor do tipo de fluxo do componente carrossel de objetos deve obrigatoriamente ser 0x0B.

12.5.2 Transmissão de imagem estática JPEG

As imagens estáticas JPEG devem obrigatoriamente ser transmitidas através de um carrossel de objetos com o tipo de fluxo com valor de 0x0B.

12.5.3 Esquema de codificação e transmissão do bitmap PNG

Para os dados de bitmap PNG que são exibidos somente sob o controle de dados CLUT especificados separadamente desta Norma, os dados da paleta, dentro dos dados PNG, podem ser abreviados.

O gráfico de bitmap PNG deve obrigatoriamente ser transmitido através de um carrossel de objetos com o tipo de fluxo com valor de 0X0B.

12.5.4 Esquema de codificação e transmissão da animação MNG

Para que os dados de de bitmap PNG no formato de animação MNG que são exibidos somente sob o controle de dados CLUT especificados separadamente desta Norma, os dados da paleta dentro dos dados PNG podem ser omitidos. O gráfico de animação de bitmap MNG deve obrigatoriamente ser transmitido através de um carrossel de objetos com o tipo de fluxo com valor de 0X0B.

12.5.5 Esquema de codificação e transmissão de dados e animação de gráficos GIF

Os dados de gráficos e animações GIF devem obrigatoriamente ser transmitidos através de um carrossel de objetos com o tipo de fluxo com valor de 0X0B.

12.6 Codificação e transmissão de caracteres - arquivos de texto externos referenciados por elementos <media>

Um arquivo de texto codificado de acordo com a ISO 8859-1 deve obrigatoriamente ser transmitido por meio de um carrossel de objetos com o valor do tipo de fluxo de 0x0B ou 0x0D.

12.7 Transmissão de documentos XML

12.7.1 Transmissão de documentos NCL e outros documentos XML usados nos comandos de edição

Para transmitir um documento NCL ou outro arquivo de documento XML usado em parâmetros de Comandos de Edição NCL, um dos métodos de transmissão a seguir deve obrigatoriamente ser usado:

a) por meio de um protocolo de canal de interatividade;

b) por meio de um carrossel de objetos.

Se um protocolo de canal interativo for usado para baixar um documento NCL ou outro arquivo de Documento XML mencionado em um parâmetro do comando de edição addNode, o parâmetro *uri* do comando de edição addDocument ou addNode (ver Seção 9) não pode ter seu esquema igual a “x-sbtvd”, e seu correspondente par de parâmetros do *ior* (ver ABNT NBR 15606-3) deve obrigatoriamente ser definido como NULL. O parâmetro *uri* deve obrigatoriamente especificar a localização do documento e o esquema de protocolo usado para transmitir o documento.

Além disso, pelo menos um programa relacionado ao carrossel de objetos (com tipo de fluxo com valor de 0x0B) deve obrigatoriamente ser transmitido, carregando um objeto de evento mapeando o nome “gingaEditingCommand” para a *eventId* do evento de fluxo DSM-CC, que deve obrigatoriamente carregar o comando de edição addDocument.

A fim de transmitir os arquivos de Documentos NCL ou outros arquivos de Documento XML, usados nos parâmetros do comando de edição addDocument ou addNode, por meio de um objeto carrossel, o tipo de fluxo com valor de 0x0B deve obrigatoriamente ser usado. No mesmo carrossel de objetos que carrega a especificação XML, um objeto de evento deve obrigatoriamente ser transmitido, a fim de mapear o nome "*gingaEditingCommand*" para o *eventId* do descritor de eventos do fluxo DSM-CC, que deve obrigatoriamente carregar o comando de edição *addDocument* ou *addNode* (ver Seção 9).

O campo *privateDataPayload* do descritor de eventos do fluxo deve obrigatoriamente carregar um conjunto de pares de referência *{uri, ior}*. O parâmetro uri do primeiro par deve obrigatoriamente ter o esquema "x-sbtvd" e o caminho local absoluto do documento NCL ou especificação do nó NCL (o caminho no servidor de dados). O parâmetro *ior* correspondente no par deve obrigatoriamente fazer referência ao IOR de especificação do documento NCL ou nó NCL (*carouselId*, *moduleId*, *objectKey*; de acordo com a ABNT NBR 15606-3 e ISO/IEC 13818-6) no carrossel de objetos.

Se outros sistemas de arquivos precisarem ser transmitidos usando outros carrosséis de objeto, a fim de completar o comando addDocument ou addNode com conteúdo de mídia, outros pares {*uri*, *ior*} devem obrigatoriamente estar presentes no comando. Nesse caso, o parâmetro *uri* deve obrigatoriamente ter o esquema "x-sbtvd" e o caminho local absoluto da raiz do sistema de arquivos (o caminho no servidor de transmissão de dados) e o repectivo parâmetro *ior* no par deve obrigatoriamente fazer referência ao IOR (*carouselId*, *moduleId*, *objectKey*; de acordo com a ABNT NBR 15606-3 e ISO/IEC 13818-6) de qualquer arquivo filho da raiz ou diretório no carrossel de objetos (o IOR do *gateway* de serviços do carrossel).

A Figura 6 ilustra um exemplo de transmissão de documento NCL por meio de um carrossel de objetos. Nesse exemplo, um provedor de conteúdo quer transmitir um programa interativo chamado "interactiveProgram.ncl" armazenado em um de seus servidores de dados (sistema local de arquivos, de acordo com a Figura 6).

Um carrossel de objetos deve então ser gerado (domínio de serviço = 1, de acordo com a Figura 6) carregando todo o conteúdo do programa interativo (arquivo .ncl e todos os arquivos de mídia) e também um objeto de evento (*moduleId* = 2 e *objectKey* = 2, de acordo com a Figura 6), mapeando o nome "gingaEditingCommand" para o valor de *eventId* (valor "3" de acordo com a Figura 6).

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 Seção 9). O parâmetro *uri* deve conter o esquema "x-sbtvd" e o caminho absoluto do documento NCL ("C:\nclRepository\weather" de acordo com a Figura 5). Finalmente, o IOR do documento NCL no carrossel de objetos é transportado no parâmetro *xmlDocument* (*carouselId* = 1, *moduleId* = 1, *objectKey* = 2 de acordo com a Figura 6).

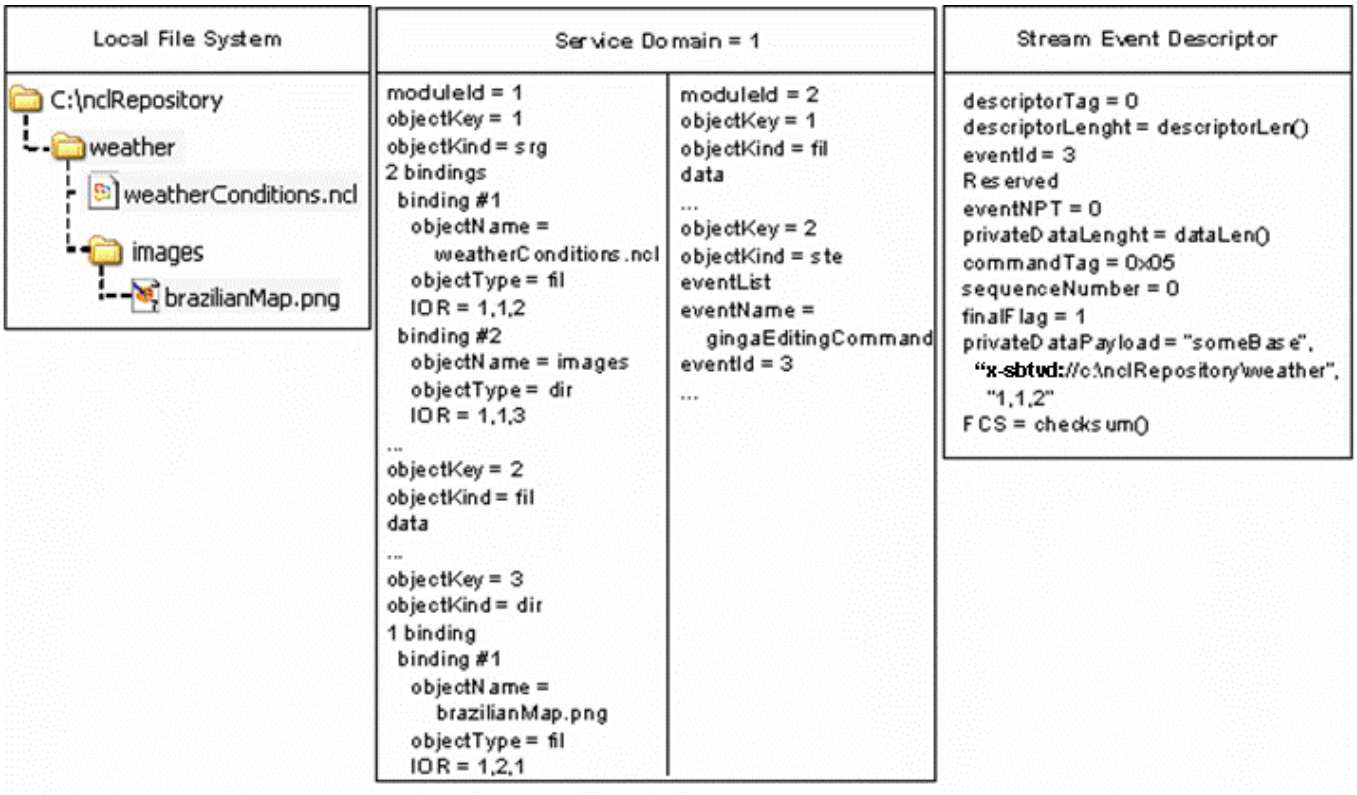

Figura 6 — Exemplo de uma transmissão de documento NCL

12.7.2 Transmissão de documentos XML externos

Os documentos XML externos referenciados pelos elementos <media>, como por exemplo um objeto de mídia XHTML, deve obrigatoriamente ser transmitido através de um carrossel de objetos com o tipo de fluxo com valor de 0x0B.

13 Segurança

O modelo de segurança Ginga é totalmente compatível com o modelo de segurança GEM, como tratado na ABNT NBR 15606-4. Ele lida com as mesmas áreas de segurança; ou seja, autenticação de aplicativos de difusão, políticas de segurança para aplicativos, segurança sobre o canal de interação e gerenciamento de certificados.

A autenticação de aplicativos Ginga-NCL deve obrigatoriamente ser realizada do mesmo modo para aplicativos Ginga-J. Se estiver assinado, o aplicativo deve obrigatoriamente seguir a estrutura de assinatura GEM conforme especificado na ABNT NBR 15606-4. Aplicativos Ginga-NCL não-autenticados irão operar dentro de um ambiente de caixa de areia (sand box). Os aplicativos Ginga-NCL autenticados associados a um arquivo de solicitação de permissão podem ter permissões outorgadas fora da caixa de areia.

Anexo A
(normativo)

Esquemas dos módulos NCL 3.0 usados nos perfis TVD Básico e TVD Avançado

A.1 Módulo *Structure*: NCL30Structure.xsd

```
<!--
XML Schema for the NCL modules

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/NCL3.0/modules/NCL30Structure.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the Structure module namespace,
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:structure="http://www.ncl.org.br/NCL3.0/Structure"
  targetNamespace="http://www.ncl.org.br/NCL3.0/Structure"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <!-- define the top-down structure of an NCL language document.         -->
  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->

  <complexType name="nclPrototype">
    <sequence>
      <element ref="structure:head" minOccurs="0" maxOccurs="1"/>
      <element ref="structure:body" minOccurs="0" maxOccurs="1"/>
    </sequence>
    <attribute name="id" type="ID" use="required"/>
    <attribute name="title" type="string" use="optional"/>
  </complexType>

  <complexType name="headPrototype">
  </complexType>

  <complexType name="bodyPrototype">
    <attribute name="id" type="ID" use="optional"/>
  </complexType>

  <!-- declare global elements in this module -->
  <element name="ncl" type="structure:nclPrototype"/>
  <element name="head" type="structure:headPrototype"/>
  <element name="body" type="structure:bodyPrototype"/>

</schema>
```

A.2 Módulo *Layout*: NCL30Layout.xsd

```
<!--
XML Schema for the NCL modules

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/NCL3.0/modules/NCL30Layout.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Layout module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:layout="http://www.ncl.org.br/NCL3.0/Layout"
  targetNamespace="http://www.ncl.org.br/NCL3.0/Layout"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <complexType name="regionBasePrototype">
    <attribute name="id" type="ID" use="optional"/>
    <attribute name="type" type="string" use="optional"/>
    <attribute name="device" type="string" use="optional"/>
  </complexType>

  <complexType name="regionPrototype">
    <sequence minOccurs="0" maxOccurs="unbounded">
      <element ref="layout:region" />
    </sequence>
    <attribute name="id" type="ID" use="required"/>
    <attribute name="title" type="string" use="optional"/>
    <attribute name="height" type="string" use="optional"/>
    <attribute name="left" type="string" use="optional"/>
    <attribute name="right" type="string" use="optional"/>
    <attribute name="top" type="string" use="optional"/>
    <attribute name="bottom" type="string" use="optional"/>
    <attribute name="width" type="string" use="optional"/>
    <attribute name="zIndex" type="integer" use="optional"/>
  </complexType>


  <!-- declare global attributes in this module -->

  <!-- define the region attributeGroup -->
  <attributeGroup name="regionAttrs">
    <attribute name="region" type="string" use="optional"/>
  </attributeGroup>

  <!-- declare global elements in this module -->
  <element name="regionBase" type="layout:regionBasePrototype"/>
  <element name="region" type="layout:regionPrototype"/>

</schema>
```

A.3 Módulo *Media*: NCL30Media.xsd

```
<!--
XML Schema for the NCL modules

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/NCL3.0/modules/NCL30Media.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Media module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:media="http://www.ncl.org.br/NCL3.0/Media"
  targetNamespace="http://www.ncl.org.br/NCL3.0/Media"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <complexType name="mediaPrototype">
    <attribute name="id" type="ID" use="required"/>
    <attribute name="type" type="string" use="optional"/>
    <attribute name="src" type="anyURI" use="optional"/>
  </complexType>


  <!-- declare global elements in this module -->
  <element name="media" type="media:mediaPrototype"/>

</schema>
```

A.4 Módulo *Context*: NCL30Context.xsd

```
<!--
XML Schema for the NCL modules

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/NCL3.0/modules/NCL30Context.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Context module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:context="http://www.ncl.org.br/NCL3.0/Context"
  targetNamespace="http://www.ncl.org.br/NCL3.0/Context"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <!-- define the compositeNode element prototype -->
  <complexType name="contextPrototype">
    <attribute name="id" type="ID" use="required"/>
  </complexType>


  <!-- declare global elements in this module -->
  <element name="context" type="context:contextPrototype"/>

</schema>
```

A.5 Módulo *MediaContentAnchor*: NCL30MediaContentAnchor.xsd

```
<!--
XML Schema for the NCL modules

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/NCL3.0/modules/NCL30MediaContentAnchor.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Media Content Anchor module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:mediaAnchor="http://www.ncl.org.br/NCL3.0/MediaContentAnchor"
  targetNamespace="http://www.ncl.org.br/NCL3.0/MediaContentAnchor"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <!-- define the temporalAnchorAttrs attribute group -->
  <attributeGroup name="temporalAnchorAttrs">
    <attribute name="begin" type="string" use="optional"/>
    <attribute name="end" type="string" use="optional"/>
  </attributeGroup>

  <!-- define the textAnchorAttrs attribute group -->
  <attributeGroup name="textAnchorAttrs">
    <attribute name="text" type="string" use="optional"/>
    <attribute name="position" type="unsignedLong" use="optional"/>
  </attributeGroup>

  <!-- define the sampleAnchorAttrs attribute group -->
  <attributeGroup name="sampleAnchorAttrs">
    <attribute name="first" type="unsignedLong" use="optional"/>
    <attribute name="last" type="unsignedLong" use="optional"/>
  </attributeGroup>

  <!-- define the coordsAnchorAttrs attribute group -->
  <attributeGroup name="coordsAnchorAttrs">
    <attribute name="coords" type="string" use="optional"/>
  </attributeGroup>

  <!-- define the labelAttrs attribute group -->
  <attributeGroup name="labelAttrs">
    <attribute name="label" type="string" use="optional"/>
  </attributeGroup>
```

```
 <complexType name="componentAnchorPrototype">
  <attribute name="id" type="ID" use="required"/>
  <attributeGroup ref="mediaAnchor:coordsAnchorAttrs" />
  <attributeGroup ref="mediaAnchor:temporalAnchorAttrs" />
  <attributeGroup ref="mediaAnchor:textAnchorAttrs" />
  <attributeGroup ref="mediaAnchor:sampleAnchorAttrs" />
  <attributeGroup ref="mediaAnchor:labelAttrs" />
 </complexType>

 <!-- declare global elements in this module -->
 <element name="area" type="mediaAnchor:componentAnchorPrototype"/>

</schema>
```

A.6 Módulo *CompositeNodeInterface*: NC30CompositeNodeInterface.xsd

```
<!--
XML Schema for the NCL modules

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/NCL3.0/modules/NCL30CompositeNodeInterface.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Composite Node Interface module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:compositeInterface="http://www.ncl.org.br/NCL3.0/CompositeNodeInterface"
  targetNamespace="http://www.ncl.org.br/NCL3.0/CompositeNodeInterface"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

  <complexType name="compositeNodePortPrototype">
    <attribute name="id" type="ID" use="required" />
    <attribute name="component" type="IDREF" use="required"/>
    <attribute name="interface" type="string" use="optional" />
  </complexType>

  <!-- declare global elements in this module -->
  <element name="port" type="compositeInterface:compositeNodePortPrototype" />

</schema>
```

A.7 Módulo *PropertyAnchor*: NCL30PropertyAnchor.xsd

```
<!--
XML Schema for the NCL modules

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/NCL3.0/modules/NCL30PropertyAnchor.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Property Anchor module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:propertyAnchor="http://www.ncl.org.br/NCL3.0/PropertyAnchor"
  targetNamespace="http://www.ncl.org.br/NCL3.0/PropertyAnchor"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <complexType name="propertyAnchorPrototype">
    <attribute name="name" type="string" use="required" />
    <attribute name="value" type="string" use="optional" />
  </complexType>


  <!-- declare global elements in this module -->
  <element name="property" type="propertyAnchor:propertyAnchorPrototype"/>

</schema>
```

A.8 Módulo *SwitchInterface*: NCL30SwitchInterface.xsd

```
<!--
XML Schema for the NCL modules

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/NCL3.0/modules/NCL30SwitchInterface.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Switch Interface module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:switchInterface="http://www.ncl.org.br/NCL3.0/SwitchInterface"
  targetNamespace="http://www.ncl.org.br/NCL3.0/SwitchInterface"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

  <complexType name="mappingPrototype">
    <attribute name="component" type="IDREF" use="required"/>
    <attribute name="interface" type="string" use="optional"/>
  </complexType>

  <complexType name="switchPortPrototype">
    <sequence>
      <element ref="switchInterface:mapping" minOccurs="1" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="id" type="ID" use="required"/>
  </complexType>

  <!-- declare global elements in this module -->
  <element name="mapping" type="switchInterface:mappingPrototype"/>
  <element name="switchPort" type="switchInterface:switchPortPrototype" />

</schema>
```

A.9 Módulo *Descriptor*: NCL30Descriptor.xsd

```
<!--
XML Schema for the NCL modules

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/NCL3.0/modules/NCL30Descriptor.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Descriptor module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:descriptor="http://www.ncl.org.br/NCL3.0/Descriptor"
  targetNamespace="http://www.ncl.org.br/NCL3.0/Descriptor"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

  <complexType name="descriptorParamPrototype">
    <attribute name="name" type="string" use="required" />
    <attribute name="value" type="string" use="required"/>
  </complexType>

  <complexType name="descriptorPrototype">
    <sequence minOccurs="0" maxOccurs="unbounded">
      <element ref="descriptor:descriptorParam"/>
    </sequence>
    <attribute name="id" type="ID" use="required"/>
    <attribute name="player" type="string" use="optional"/>
  </complexType>

<!--
Formatters should support the following descriptorParam names.
* For audio players: soundLevel; balanceLevel; trebleLevel; bassLevel.
* For text players: style, which refers to a style sheet with information for text presentation.
* For visual media players: background, specifying the background color used to fill the area of a region displaying
media; scroll, which allows the specification of how an author would like to configure the scroll in a region; fit,
indicating how an object will be presented (hidden, fill, meet, meetBest, slice); transparency, indicating the degree
of transparency of an object presentation (the value must be between 0 and 1, or a real number in the range
[0,100] ending by the caracter "%" (ex. 30%)); visible, indicating if the presentation is to be seen or hidden; and the
object positioning parameters: top, left, bottom, right, width, height, sie and bounds.
* For players in general: reusePlayer, which determines if a new player must be instantiated or if a player already
instantiated must be used; and playerLife, which specifies what will happen to the player instance at the end of the
presentation.
-->

  <complexType name="descriptorBasePrototype">
    <attribute name="id" type="ID" use="optional"/>
  </complexType>
```

```
  <!-- declare global elements in this module -->
  <element name="descriptorParam" type="descriptor:descriptorParamPrototype"/>
  <element name="descriptor" type="descriptor:descriptorPrototype"/>
  <element name="descriptorBase" type="descriptor:descriptorBasePrototype"/>

  <!-- declare global attributes in this module -->
  <attributeGroup name="descriptorAttrs">
    <attribute name="descriptor" type="string" use="optional"/>
  </attributeGroup>

</schema>
```

A.10 Módulo *Linking*: NCL30Linking.xsd

```
<!--
XML Schema for the NCL modules

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/NCL3.0/modules/NCL30Linking.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Linking module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
 xmlns:linking="http://www.ncl.org.br/NCL3.0/Linking"
 targetNamespace="http://www.ncl.org.br/NCL3.0/Linking"
 elementFormDefault="qualified" attributeFormDefault="unqualified" >


 <complexType name="paramPrototype">
  <attribute name="name" type="string" use="required"/>
  <attribute name="value" type="anySimpleType" use="required"/>
 </complexType>

 <complexType name="bindPrototype">
  <sequence minOccurs="0" maxOccurs="unbounded">
   <element ref="linking:bindParam"/>
  </sequence>
  <attribute name="role" type="string" use="required"/>
  <attribute name="component" type="IDREF" use="required"/>
  <attribute name="interface" type="string" use="optional"/>
 </complexType>

 <complexType name="linkPrototype">
  <sequence>
   <element ref="linking:linkParam" minOccurs="0" maxOccurs="unbounded"/>
   <element ref="linking:bind" minOccurs="2" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="id" type="ID" use="optional"/>
  <attribute name="xconnector" type="string" use="required"/>
 </complexType>


 <!-- declare global elements in this module -->
 <element name="linkParam" type="linking:paramPrototype"/>
 <element name="bindParam" type="linking:paramPrototype"/>
 <element name="bind" type="linking:bindPrototype" />
 <element name="link" type="linking:linkPrototype" />

</schema>
```

A.11 Módulo *ConnectorCommonPart*: NCL30ConnectorCommonPart.xsd

```
<!--
XML Schema for the NCL modules

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/NCL3.0/modules/NCL30ConnectorCommonPart.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Connector Common Part module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:connectorCommonPart="http://www.ncl.org.br/NCL3.0/ConnectorCommonPart"
  targetNamespace="http://www.ncl.org.br/NCL3.0/ConnectorCommonPart"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


<complexType name="parameterPrototype">
  <attribute name="name" type="string" use="required"/>
  <attribute name="type" type="string" use="optional"/>
</complexType>

<simpleType name="eventPrototype">
  <restriction base="string">
    <enumeration value="presentation" />
    <enumeration value="selection" />
    <enumeration value="attribution" />
    <enumeration value="composition" />
  </restriction>
</simpleType>

<simpleType name="logicalOperatorPrototype">
  <restriction base="string">
    <enumeration value="and" />
    <enumeration value="or" />
  </restriction>
</simpleType>

<simpleType name="transitionPrototype">
  <restriction base="string">
    <enumeration value="starts" />
    <enumeration value="stops" />
    <enumeration value="pauses" />
    <enumeration value="resumes" />
    <enumeration value="aborts" />
```

```
  </restriction>
</simpleType>

</schema>
```

A.12 Módulo *ConnectorAssessmentExpression*: NCL30ConnectorAssessmentExpression.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2006 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30ConnectorAssessmentExpression.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Connector Assessment Expression module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:connectorAssessmentExpression="http://www.ncl.org.br/NCL3.0/ConnectorAssessmentExpression"
  xmlns:connectorCommonPart="http://www.ncl.org.br/NCL3.0/ConnectorCommonPart"
  targetNamespace="http://www.ncl.org.br/NCL3.0/ConnectorAssessmentExpression"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

<!-- import the definitions in the modules namespaces -->
<import namespace="http://www.ncl.org.br/NCL3.0/ConnectorCommonPart"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30ConnectorCommonPart.xsd"/>

<simpleType name="comparatorPrototype">
  <restriction base="string">
    <enumeration value="eq" />
    <enumeration value="ne" />
    <enumeration value="gt" />
    <enumeration value="lt" />
    <enumeration value="gte" />
    <enumeration value="lte" />
  </restriction>
</simpleType>

<simpleType name="attributePrototype">
  <restriction base="string">
    <enumeration value="repeat" />
    <enumeration value="occurrences" />
    <enumeration value="state" />
    <enumeration value="nodeProperty" />
  </restriction>
</simpleType>

<simpleType name="statePrototype">
  <restriction base="string">
    <enumeration value="sleeping" />
```

```
    <enumeration value="occurring" />
    <enumeration value="paused" />
  </restriction>
</simpleType>

<simpleType name="valueUnion">
  <union memberTypes="string" connectorAssessmentExpression:statePrototype"/>
</simpleType>

<complexType name="assessmentStatementPrototype" >
  <sequence>
    <element ref="connectorAssessmentExpression:attributeAssessment"/>
    <choice>
      <element ref="connectorAssessmentExpression:attributeAssessment"/>
      <element ref="connectorAssessmentExpression:valueAssessment"/>
    </choice>
  </sequence>
  <attribute name="comparator" type="connectorAssessmentExpression:comparatorPrototype" use="required"/>
</complexType>

<complexType name="attributeAssessmentPrototype">
  <attribute name="role" type="string" use="required"/>
  <attribute name="eventType" type="connectorCommonPart:eventPrototype" use="required"/>
  <attribute name="key" type="string" use="optional"/>
  <attribute name="attributeType" type="connectorAssessmentExpression:attributePrototype" use="optional"/>
  <attribute name="offset" type="string" use="optional"/>
</complexType>

<complexType name="valueAssessmentPrototype">
  <attribute name="value" type="connectorAssessmentExpression:valueUnion" use="required"/>
</complexType>

<complexType name="compoundStatementPrototype">
  <choice minOccurs="1" maxOccurs="unbounded">
    <element ref="connectorAssessmentExpression:assessmentStatement" />
    <element ref="connectorAssessmentExpression:compoundStatement" />
  </choice>
  <attribute name="operator" type="connectorCommonPart:logicalOperatorPrototype" use="required"/>
  <attribute name="isNegated" type="boolean" use="optional"/>
</complexType>

  <!-- declare global elements in this module -->
<element name="assessmentStatement" type="connectorAssessmentExpression:assessmentStatementPrototype"
/>
<element name="attributeAssessment" type="connectorAssessmentExpression:attributeAssessmentPrototype" />
<element name="valueAssessment" type="connectorAssessmentExpression:valueAssessmentPrototype" />
<element name="compoundStatement" type="connectorAssessmentExpression:compoundStatementPrototype" />

</schema>
```

A.13 Módulo *ConnectorCausalExpression*: NCL30ConnectorCausalExpression.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2006 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30ConnectorCausalExpression.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Connector Causal Expression module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:connectorCausalExpression="http://www.ncl.org.br/NCL3.0/ConnectorCausalExpression"
    xmlns:connectorCommonPart="http://www.ncl.org.br/NCL3.0/ConnectorCommonPart"
  targetNamespace="http://www.ncl.org.br/NCL3.0/ConnectorCausalExpression"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

<!-- import the definitions in the modules namespaces -->
<import namespace="http://www.ncl.org.br/NCL3.0/ConnectorCommonPart"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30ConnectorCommonPart.xsd"/>

<simpleType name="conditionRoleUnion">
  <union memberTypes="string connectorCausalExpression:conditionRolePrototype"/>
</simpleType>

<simpleType name="conditionRolePrototype">
  <restriction base="string">
    <enumeration value="onBegin" />
    <enumeration value="onEnd" />
    <enumeration value="onPause" />
    <enumeration value="onResume" />
    <enumeration value="onAbort" />
  </restriction>
</simpleType>

<simpleType name="maxUnion">
  <union memberTypes="positiveInteger connectorCausalExpression:unboundedString"/>
</simpleType>

<simpleType name="unboundedString">
  <restriction base="string">
    <pattern value="unbounded"/>
  </restriction>
</simpleType>
```

```
<complexType name="simpleConditionPrototype">
  <attribute name="role" type="connectorCausalExpression:conditionRoleUnion" use="required"/>
  <attribute name="eventType" type="connectorCommonPart:eventPrototype" use="optional"/>
  <attribute name="key" type="string" use="optional"/>
  <attribute name="transition" type="connectorCommonPart:transitionPrototype" use="optional"/>
  <attribute name="delay" type="string" use="optional"/>
  <attribute name="min" type="positiveInteger" use="optional"/>
  <attribute name="max" type="connectorCausalExpression:maxUnion" use="optional"/>
  <attribute name="qualifier" type="connectorCommonPart:logicalOperatorPrototype" use="optional"/>
</complexType>

<complexType name="compoundConditionPrototype">
  <attribute name="operator" type="connectorCommonPart:logicalOperatorPrototype" use="required"/>
  <attribute name="delay" type="string" use="optional"/>
</complexType>

<simpleType name="actionRoleUnion">
  <union memberTypes="string connectorCausalExpression:actionNamePrototype"/>
</simpleType>

<simpleType name="actionNamePrototype">
  <restriction base="string">
    <enumeration value="start" />
    <enumeration value="stop" />
    <enumeration value="pause" />
    <enumeration value="resume" />
    <enumeration value="abort" />
    <enumeration value="set" />
  </restriction>
</simpleType>

<simpleType name="actionOperatorPrototype">
  <restriction base="string">
    <enumeration value="par" />
    <enumeration value="seq" />
  </restriction>
</simpleType>

<complexType name="simpleActionPrototype">
  <attribute name="role" type="connectorCausalExpression:actionRoleUnion" use="required"/>
  <attribute name="eventType" type="connectorCommonPart:eventPrototype" use="optional"/>
  <attribute name="actionType" type="connectorCausalExpression:actionNamePrototype" use="optional"/>
  <attribute name="delay" type="string" use="optional"/>
  <attribute name="value" type="string" use="optional"/>
  <attribute name="repeat" type="positiveInteger" use="optional"/>
  <attribute name="repeatDelay" type="string" use="optional"/>
  <attribute name="min" type="positiveInteger" use="optional"/>
  <attribute name="max" type="connectorCausalExpression:maxUnion" use="optional"/>
  <attribute name="qualifier" type="connectorCausalExpression:actionOperatorPrototype" use="optional"/>
</complexType>

<complexType name="compoundActionPrototype">
  <choice minOccurs="2" maxOccurs="unbounded">
    <element ref="connectorCausalExpression:simpleAction" />
```

```
    <element ref="connectorCausalExpression:compoundAction" />
  </choice>
  <attribute name="operator" type="connectorCausalExpression:actionOperatorPrototype" use="required"/>
  <attribute name="delay" type="string" use="optional"/>
</complexType>

  <!-- declare global elements in this module -->
<element name="simpleCondition" type="connectorCausalExpression:simpleConditionPrototype" />
<element name="compoundCondition" type="connectorCausalExpression:compoundConditionPrototype" />
<element name="simpleAction" type="connectorCausalExpression:simpleActionPrototype" />
<element name="compoundAction" type="connectorCausalExpression:compoundActionPrototype" />

</schema>
```

A.14 Módulo *CausalConnector*: NCL30CausalConnector.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2006 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30CausalConnector.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Causal Connector module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:causalConnector="http://www.ncl.org.br/NCL3.0/CausalConnector"
  targetNamespace="http://www.ncl.org.br/NCL3.0/CausalConnector"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

<complexType name="causalConnectorPrototype">
  <attribute name="id" type="ID" use="required"/>
</complexType>

  <!-- declare global elements in this module -->
<element name="causalConnector" type="causalConnector:causalConnectorPrototype"/>
</schema>
```

A.15 Módulo *ConnectorBase*: NCL30ConnectorBase.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2006 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30ConnectorBase.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Connector Base module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:connectorBase="http://www.ncl.org.br/NCL3.0/ConnectorBase"
  targetNamespace="http://www.ncl.org.br/NCL3.0/ConnectorBase"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

<complexType name="connectorBasePrototype">
  <attribute name="id" type="ID" use="optional"/>
</complexType>

  <!-- declare global elements in this module -->
<element name="connectorBase" type="connectorBase:connectorBasePrototype"/>
</schema>
```

A.16 NCL30CausalConnectorFunctionality.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/
NCL30CausalConnectorFunctionality.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL CausalConnectorFunctionality module namespace.
-->

<schema xmlns="http://www.w3.org/2001/XMLSchema"
      xmlns:connectorCommonPart="http://www.ncl.org.br/NCL3.0/
ConnectorCommonPart"
      xmlns:connectorAssessmentExpression="http://www.ncl.org.br/NCL3.0/
ConnectorAssessmentExpression"
      xmlns:connectorCausalExpression="http://www.ncl.org.br/NCL3.0/
ConnectorCausalExpression"
      xmlns:causalConnector="http://www.ncl.org.br/NCL3.0/
CausalConnector"
      xmlns:causalConnectorFunctionality="http://www.ncl.org.br/NCL3.0/
CausalConnectorFunctionality"
      targetNamespace="http://www.ncl.org.br/NCL3.0/
CausalConnectorFunctionality"
      elementFormDefault="qualified" attributeFormDefault="unqualified">

   <!-- import the definitions in the modules namespaces -->

   <import namespace="http://www.ncl.org.br/NCL3.0/ConnectorCommonPart"
   schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30ConnectorCommonPart.xsd"/>
   <import namespace="http://www.ncl.org.br/NCL3.0/ConnectorAssessmentExpression"
   schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30ConnectorAssessmentExpression.xsd"/>
   <import namespace="http://www.ncl.org.br/NCL3.0/ConnectorCausalExpression"
   schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30ConnectorCausalExpression.xsd"/>
   <import namespace="http://www.ncl.org.br/NCL3.0/CausalConnector"
   schemaLocation="http://www.ncl.org.br/NCL3.0/modules/
NCL30CausalConnector.xsd"/>

 <!-- ========================================================== -->
 <!-- CausalConnectorFunctionality                               -->
 <!-- ========================================================== -->
  <element name="connectorParam" type="connectorCommonPart:parameterPrototype"/>
```

```
  <!-- extends causalConnector element -->

  <complexType name="causalConnectorType">
   <complexContent>
    <extension base="causalConnector:causalConnectorPrototype">
     <sequence>
      <element ref="causalConnectorFunctionality:connectorParam" minOccurs="0" maxOccurs="unbounded"/>
      <choice>
       <element ref="causalConnectorFunctionality:simpleCondition" />
       <element ref="causalConnectorFunctionality:compoundCondition" />
      </choice>
      <choice>
       <element ref="causalConnectorFunctionality:simpleAction" />
       <element ref="causalConnectorFunctionality:compoundAction" />
      </choice>
     </sequence>
    </extension>
   </complexContent>
  </complexType>

  <!-- extends compoundCondition element -->

  <complexType name="compoundConditionType">
   <complexContent>
    <extension base="connectorCausalExpression:compoundConditionPrototype">
     <sequence>
      <choice>
       <element ref="causalConnectorFunctionality:simpleCondition" />
       <element ref="causalConnectorFunctionality:compoundCondition" />
      </choice>
      <choice minOccurs="1" maxOccurs="unbounded">
       <element ref="causalConnectorFunctionality:simpleCondition" />
       <element ref="causalConnectorFunctionality:compoundCondition" />
       <element ref="causalConnectorFunctionality:assessmentStatement" />
       <element ref="causalConnectorFunctionality:compoundStatement" />
      </choice>
     </sequence>
    </extension>
   </complexContent>
  </complexType>

 <element name="causalConnector" type="causalConnectorFunctionality:causalConnectorType"
substitutionGroup="causalConnector:causalConnector"/>

 <element name="simpleCondition" substitutionGroup="connectorCausalExpression:simpleCondition"/>

 <element name="compoundCondition" type="causalConnectorFunctionality:compoundConditionType"
substitutionGroup="connectorCausalExpression:compoundCondition"/>

 <element name="simpleAction" substitutionGroup="connectorCausalExpression:simpleAction"/>
```

```
  <element name="compoundAction" substitutionGroup="connectorCausalExpression:compoundAction"/>

  <element name="assessmentStatement"
substitutionGroup="connectorAssessmentExpression:assessmentStatement"/>

  <element name="attributeAssessment"
substitutionGroup="connectorAssessmentExpression:attributeAssessment"/>

  <element name="valueAssessment" substitutionGroup="connectorAssessmentExpression:valueAssessment"/>

  <element name="compoundStatement"
substitutionGroup="connectorAssessmentExpression:compoundStatement"/>

</schema>
```

A.17 Módulo *TestRule*: NCL30TestRule.xsd

```
<!--
XML Schema for the NCL modules

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/NCL3.0/modules/NCL30TestRule.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL TestRule module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:testRule="http://www.ncl.org.br/NCL3.0/TestRule"
  targetNamespace="http://www.ncl.org.br/NCL3.0/TestRule"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <complexType name="rulePrototype">
    <attribute name="id" type="ID" use="required"/>
    <attribute name="var" type="string" use="required"/>
    <attribute name="value" type="string" use="required"/>
    <attribute name="comparator" use="required">
      <simpleType>
        <restriction base="string">
          <enumeration value="eq"/>
          <enumeration value="ne"/>
          <enumeration value="gt"/>
          <enumeration value="gte"/>
          <enumeration value="lt"/>
          <enumeration value="lte"/>
        </restriction>
      </simpleType>
    </attribute>
  </complexType>

  <complexType name="compositeRulePrototype">
    <choice minOccurs="2" maxOccurs="unbounded">
      <element ref="testRule:rule"/>
      <element ref="testRule:compositeRule"/>
    </choice>
    <attribute name="id" type="ID" use="required"/>
    <attribute name="operator" use="required">
      <simpleType>
        <restriction base="string">
          <enumeration value="and"/>
```

```
        <enumeration value="or"/>
       </restriction>
     </simpleType>
   </attribute>
 </complexType>

 <complexType name="ruleBasePrototype">
   <attribute name="id" type="ID" use="optional"/>
 </complexType>

 <!-- declare global elements in this module -->
 <element name="rule" type="testRule:ruleProtototype"/>
 <element name="compositeRule" type="testRule:compositeRulePrototype"/>
 <element name="ruleBase" type="testRule:ruleBasePrototype"/>

</schema>
```

A.18 Módulo *TestRuleUse*: NCL30TestRuleUse.xsd

```
<!--
XML Schema for the NCL modules

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/NCL3.0/modules/NCL30TestRuleUse.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL TestRuleUse module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:testRule="http://www.ncl.org.br/NCL3.0/TestRuleUse"
  targetNamespace="http://www.ncl.org.br/NCL3.0/TestRuleUse"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <complexType name="bindRulePrototype">
    <attribute name="constituent" type="IDREF" use="required" />
    <attribute name="rule" type="string" use="required" />
  </complexType>


  <!-- declare global elements in this module -->
  <element name="bindRule" type="testRule:bindRulePrototype"/>

</schema>
```

A.19 Módulo *ContentControl*: NCL30ContentControl.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30ContentControl.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL ContentControl module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:contentControl="http://www.ncl.org.br/NCL3.0/ContentControl"
  targetNamespace="http://www.ncl.org.br/NCL3.0/ContentControl"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

  <complexType name="defaultComponentPrototype">
    <attribute name="component" type="IDREF" use="required" />
  </complexType>

  <!-- define the switch element prototype -->

  <complexType name="switchPrototype">
    <choice>
    <element ref="contentControl:defaultComponent" minOccurs="0" maxOccurs="1"/>
    </choice>
    <attribute name="id" type="ID" use="required"/>
  </complexType>

  <!-- declare global elements in this module -->
  <element name="defaultComponent" type="contentControl:defaultComponentPrototype"/>
  <element name="switch" type="contentControl:switchPrototype"/>

</schema>
```

A.20 Módulo *DescriptorControl*: NCL30DescriptorControl.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2005 LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30DescriptorControl.xsd
Author: TeleMidia Laboratory
Revision: 19/06/2006

Schema for the NCL DescriptorControl module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:descriptorControl="http://www.ncl.org.br/NCL3.0/DescriptorControl"
  targetNamespace="http://www.ncl.org.br/NCL3.0/DescriptorControl"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

  <complexType name="defaultDescriptorPrototype">
    <attribute name="descriptor" type="IDREF" use="required" />
  </complexType>

  <!-- define the descriptor switch element prototype -->
  <complexType name="descriptorSwitchPrototype">
    <choice>
    <element ref="descriptorControl:defaultDescriptor" minOccurs="0" maxOccurs="1"/>
    </choice>
    <attribute name="id" type="ID" use="required”/>
  </complexType>

  <!-- declare global elements in this module -->
  <element name="defaultDescriptor" type="descriptorControl:defaultDescriptorPrototype"/>
  <element name="descriptorSwitch" type="descriptorControl:descriptorSwitchPrototype"/>

</schema>
```

A.21 Módulo *Timing*: NCL30Timing.xsd

```
<!--
XML Schema for the NCL modules

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/NCL3.0/modules/NCL30Timing.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Timing module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:timing="http://www.ncl.org.br/NCL3.0/Timing"
  targetNamespace="http://www.ncl.org.br/NCL3.0/Timing"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

  <!-- declare global attributes in this module -->

  <!-- define the explicitDur attribute group -->
  <attributeGroup name="explicitDurAttrs">
    <attribute name="explicitDur" type="string" use="optional"/>
  </attributeGroup>

  <!-- define the freeze attribute group -->
  <attributeGroup name="freezeAttrs">
    <attribute name="freeze" type="boolean" use="optional"/>
  </attributeGroup>

</schema>
```

A.22 Módulo *Import*: NCL30Import.xsd

```
<!--
XML Schema for the NCL modules

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/NCL3.0/modules/NCL30Import.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Import module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:import="http://www.ncl.org.br/NCL3.0/Import"
  targetNamespace="http://www.ncl.org.br/NCL3.0/Import"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

  <complexType name="importBasePrototype">
     <attribute name="alias" type="ID" use="required"/>
     <attribute name="region" type="IDREF" use="optional"/>
     <attribute name="documentURI" type="anyURI" use="required"/>
  </complexType>

  <complexType name="importNCLPrototype">
     <attribute name="alias" type="ID" use="required"/>
     <attribute name="documentURI" type="anyURI" use="required"/>
  </complexType>

  <complexType name="importedDocumentBasePrototype">
    <sequence minOccurs="1" maxOccurs="unbounded">
      <element ref="import:importNCL" />
    </sequence>
    <attribute name="id" type="ID" use="optional" />
  </complexType>

  <!-- declare global elements in this module -->
  <element name="importBase" type="import:importBasePrototype"/>
  <element name="importNCL" type="import:importNCLPrototype"/>
  <element name="importedDocumentBase" type="import:importedDocumentBasePrototype"/>

</schema>
```

A.23 Módulo *EntityReuse*: NCL30EntityReuse.xsd

```
<!--
XML Schema for the NCL modules

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/NCL3.0/modules/NCL30EntityReuse.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL EntityReuse module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:entityReuse="http://www.ncl.org.br/NCL3.0/EntityReuse"
  targetNamespace="http://www.ncl.org.br/NCL3.0/EntityReuse"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

  <attributeGroup name="entityReuseAttrs">
    <attribute name="refer" type="string" use="optional"/>
  </attributeGroup>

</schema>
```

A.24 Módulo *ExtendedEntityReuse*: NCL30ExtendedEntityReuse.xsd

```
<!--
XML Schema for the NCL modules

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/NCL3.0/modules/NCL30ExtendedEntityReuse.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL ExtendedEntityReuse module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:extendedEntityReuse="http://www.ncl.org.br/NCL3.0/ExtendedEntityReuse"
  targetNamespace="http://www.ncl.org.br/NCL3.0/ExtendedEntityReuse"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

  <attributeGroup name="extendedEntityReuseAttrs">
    <attribute name="newInstance" type="boolean" use="optional"/>
  </attributeGroup>

</schema>
```

A.25 Módulo *KeyNavigation*: NCL30KeyNavigation.xsd

```
<!--
XML Schema for the NCL modules

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/NCL3.0/modules/NCL30KeyNavigation.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL KeyNavigation module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
 xmlns:keyNavigation="http://www.ncl.org.br/NCL3.0/KeyNavigation"
 targetNamespace="http://www.ncl.org.br/NCL3.0/KeyNavigation"
 elementFormDefault="qualified" attributeFormDefault="unqualified" >

<simpleType name="colorPrototype">
 <restriction base="string">
  <enumeration value="white" />
  <enumeration value="black" />
  <enumeration value="silver" />
  <enumeration value="gray" />
  <enumeration value="red" />
  <enumeration value="maroon" />
  <enumeration value="fuchsia" />
  <enumeration value="purple" />
  <enumeration value="lime" />
  <enumeration value="green" />
  <enumeration value="yellow" />
  <enumeration value="olive" />
  <enumeration value="blue" />
  <enumeration value="navy" />
  <enumeration value="aqua" />
  <enumeration value="teal" />
 </restriction>
</simpleType>

 <!-- declare global attributes in this module -->

 <!-- define the keyNavigation attribute group -->
 <attributeGroup name="keyNavigationAttrs">
```

```
   <attribute name="moveLeft" type="IDREF" use="optional"/>
   <attribute name="moveRight" type="IDREF" use="optional"/>
   <attribute name="moveUp" type="IDREF" use="optional"/>
   <attribute name="moveDown" type="IDREF" use="optional"/>
   <attribute name="focusIndex" type="IDREF" use="optional"/>
   <attribute name="focusBorderColor" type="keyNavigation:colorPrototype" use="optional"/>
   <attribute name="focusBorderWidth" type="string" use="optional"/>
   <attribute name="focusBorderTransparency" type="string" use="optional"/>
   <attribute name="focusScr" type="string" use="optional"/>
   <attribute name="focusSelScr" type="string" use="optional"/>
   <attribute name="selBorderColor" type="keyNavigation:colorPrototype" use="optional"/>
  </attributeGroup>

</schema>
```

A.26 Módulo *TransitionBase*: NCL30TransitionBase.xsd

```
<!--
XML Schema for the NCL modules

This is NCL
Copyright: 2000-2006 PUC-RIO/LABORATORIO TELEMIDIA, All Rights Reserved.
See http://www.telemidia.puc-rio.br

Public URI: http://www.ncl.org.br/NCL3.0/modules/NCL30TransitionBase.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Transition Base module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:transitionBase="http://www.ncl.org.br/NCL3.0/TransitionBase"
  targetNamespace="http://www.ncl.org.br/NCL3.0/TransitionBase"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

<complexType name="transitionBasePrototype">
  <attribute name="id" type="ID" use="optional"/>
</complexType>

  <!-- declare global elements in this module -->
<element name="transitionBase" type="transitionBase:transitionBasePrototype"/>
</schema>
```

A.27 Módulo *Animation*: NCL30Animation.xsd

```
<!--
XML Schema for the NCL modules

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/NCL3.0/modules/NCL30Animation.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Timing module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:animation="http://www.ncl.org.br/NCL3.0/Animation"
  targetNamespace="http://www.ncl.org.br/NCL3.0/Animation"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <!-- declare global attributes in this module -->

  <!-- define the animation attribute group -->
  <attributeGroup name="animationAttrs">
    <attribute name="duration" type="string" use="optional"/>
    <attribute name="by" type="string" use="optional"/>
  </attributeGroup>

</schema>
```

Anexo B
(informativo)

Manual de referência de Lua 5.1

B.1 Introdução

NOTA Este Anexo apresenta a especificação da linguagem de programação Lua, versão 5.1. Este conteúdo é uma tradução do livro de Roberto Ierusalimschy, Luiz Henrique de Figueiredo e Waldemar Celes, Lua 5.1 *Reference Manual* (Lua.org, agosto de 2006. ISBN 85-903798-3-3) e é impresso novamente aqui com aprovação dos autores. O livro ambém está disponível online, em inglês, em <http://www.lua.org/manual/5.1/>.

Lua é uma linguagem de programação de extensão projetada para dar suporte à programação procedimental em geral e que oferece facilidades para a descrição de dados. A linguagem também oferece um bom suporte para programação orientada a objetos, programação funcional e programação orientada a dados. Lua foi planejada para ser utilizada por qualquer aplicação que necessite de uma linguagem de script leve e poderosa. Lua é implementada como uma biblioteca, escrita em C limpo (isto é, no subconjunto comum de ANSI C e C++).

Por ser uma linguagem de extensão, Lua não possui a noção de um programa principal: ela somente funciona embarcada em um programa cliente anfitrião, chamado de programa hospedeiro ou simplesmente de hospedeiro. Esse programa hospedeiro pode invocar funções para executar um pedaço de código Lua, pode escrever e ler variáveis Lua e pode registrar funções C para serem chamadas pelo código Lua. Através do uso de funções C, Lua pode ser estendida para lidar de maneira apropriada com uma ampla variedade de domínios, permitindo assim a criação de linguagems de programação personalizadas que compartilham um arcabouço sintático. A distribuição Lua inclui um exemplo de um programa hospedeiro chamado lua, o qual usa a biblioteca de Lua para oferecer um interpretador de linha de comando Lua completo.

Lua é um software livre e, como de praxe, é fornecido sem garantias, conforme dito na sua lincença. A implementação descrita nesta Norma, bem como artigos técnicos sobre Lua, estão disponíveis disponível no sítio web oficial de Lua, www.lua.org.

B.2 A Linguagem

B.2.1 Notação utilizada

As construções da linguagem serão explicadas usando a notação BNF estendida usual, na qual {*a*} significa 0 ou mais *a*'s e [*a*] significa um *a* opcional. Não-terminais são mostrados como non-terminal, palavras-chave são mostradas como **kword** e outros símbolos terminais são mostrados como `=´. A sintaxe completa de Lua pode ser encontrada em B.8.

B.2.2 Convenções Léxicas

Em Lua, *Nomes* (também chamados de identificadores) podem ser qualquer cadeia de letras, dígitos e sublinhados que não começam com um dígito. Esta definição está de acordo com a definição de nomes na maioria das linguagens (a definição de letras depende de qual é o idioma (*locale*): qualquer caractere considerado alfabético pelo idioma corrente pode ser usado como um identificador). Identificadores são usados para nomear variáveis e campos de tabelas.

As seguintes palavras-chave são reservadas e não podem ser utilizadas como nomes:

```
and       break     do        else      elseif
end       false     for       function  if
```

```
in        local     nil       not       or
repeat    return    then      true      until     while
```

Lua é uma linguagem que diferencia minúsculas de maiúsculas: and é uma palavra reservada, mas And e AND são dois nomes válidos diferentes. Como convenção, nomes que começam com um sublinhado seguido por letras maiúsculas (tais como _VERSION) são reservados para variáveis globais internas usadas por Lua.

As seguintes cadeias denotam outros itens léxicos:

```
+     -     *     /     %     ^     #
==    ~=    <=    >=    <     >     =
(     )     {     }     [     ]
;     :     ,     .     ..    ...
```

Cadeias de caracteres literais podem ser delimitadas através do uso de aspas simples ou aspas duplas, e podem conter as seguintes seqüências de escape no estilo de C: '\a' (campainha), '\b' (backspace), '\f' (alimentação de formulário), '\n' (quebra de linha), '\r' (retorno de carro), '\t' (tabulação horizontal), '\v' (tabulação vertical), '\\' (barra invertida), '\"' (citação [aspa dupla]) e '\'' (apóstrofo [aspa simples]). Além disso, uma barra invertida seguida por uma quebra de linha real resulta em uma quebra de linha na cadeia de caracteres. Um caractere em uma cadeia de caracteres também pode ser especificado pelo seu valor numérico usando a seqüência de escape *ddd*, onde *ddd* é uma seqüência de até três dígitos decimais. Se um caractere numérico representado como um seqüência de escape for seguido por um dígito, a seqüência de escape deve possuir exatamente três dígitos. Cadeias de caracteres em Lua podem conter qualquer valor de 8 bits, incluindo zeros dentro delas, os quais podem ser especificados como '\0'.

Para colocar uma aspa dupla (simples), uma quebra de linha, uma barra invertida ou inserir um zero dentro de uma cadeia de caracteres literal delimitada por aspas duplas (simples) deve-se usar uma seqüência de escape. Qualquer outro caractere pode ser inserido diretamente dentro da cadeia literal (alguns caracteres de controle podem causar problemas para o sistema de arquivos, mas Lua não tem nenhum problema em relação a eles).

Cadeias literais longas também podem ser definidas usando um formato longo delimitado por colchetes longos. Definimos uma abertura de colchete longo de nível n como um abre colchete seguido por n sinais de igual seguido por outro abre colchete. Dessa forma, uma abertura de colchete longo de nível 0 é escrita como [[, uma abertura de colchete longo de nível 1 é escrita como [=[e assim por diante. Um fechamento de colchete longo é definido de maneira similar; por exemplo, um fechamento de colchete longo de nível 4 é escrito como]====]. Uma cadeia de caracteres longa começa com uma abertura de colchete longo de qualquer nível e termina no primeiro fechamento de colchete longo do mesmo nível. Literais expressos desta forma podem se estender por várias linhas, não interpretam nenhuma seqüência de escape e ignoram colchetes longos de qualquer outro nível. Estes literais podem conter qualquer coisa, exceto um fechamento de colchete longo de nível igual ao da abertura.

Por conveniência, quando uma abertura de colchete longo é imediatamente seguida por uma quebra de linha, a quebra de linha não é incluída na cadeia de caracteres. Como exemplo, em um sistema usando ASCII (no qual 'a' é codificado como 97, quebra de linha é codificado como 10 e '1' é codificado como 49), os cinco literais abaixo denotam a mesma cadeia:

```
a = 'alo\n123"'
a = "alo\n123\""
a = '\97lo\10\04923"'
a = [[alo
123"]]
a = [==[
alo
123"]==]
```

Uma constante numérica pode ser escrita com uma parte decimal opcional e com um expoente decimal opcional. Lua também aceita constantes hexadecimais inteiras, através do uso do prefixo 0x. Exemplos de constantes numéricas válidas são:

```
3   3.0   3.1416   314.16e-2   0.31416E1   0xff   0x56
```

Um comentário começa com um hífen duplo (--) em qualquer lugar, desde que fora de uma cadeia de caracteres. Se o texto imediatamente depois de -- não é uma abertura de colchete longo, o comentário é um comentário curto, o qual se estende até o fim da linha. Caso contrário, ele é um comentário longo, que se estende até o fechamento de colchete longo correspondente. Comentários longos são freqüentemente usados para desabilitar código temporariamente.

B.2.3 Valores e tipos

B.2.3.1 Tipos básicos

Lua é uma linguagem dinamicamente tipada. Isto significa que variáveis não possuem tipos; somente valores possuem tipos. Não existe definição de tipos na linguagem. Todos os valores carregam o seu próprio tipo.

Todos os valores em Lua são valores de primeira classe. Isto significa que todos os valores podem ser armazenados em variáveis, passados como argumentos para outras funções e retornados como resultados.

Existem oito tipos básicos em Lua: *nil*, *boolean*, *number*, *string*, *function*, *userdata*, *thread* e *table*. *Nil* é o tipo do valor **nil**, cuja propriedade principal é ser diferente de qualquer outro valor; ele geralmente representa a ausência de um valor útil. *Boolean* é o tipo dos valores **false** e **true**. Tanto **nil** como **false** tornam uma condição falsa; qualquer outro valor torna a condição verdadeira. *Number* representa números reais (ponto flutuante de precisão dupla). É fácil construir interpretadores Lua que usem outra representação interna para números, tais como precisão simples de ponto flutuante ou inteiros longos; ver o arquivo luaconf.h. O tipo *string* representa cadeias de caracteres. Em Lua, cadeias de caracteres podem conter qualquer caractere de 8 bits, incluindo zeros ('\0') dentro dela (ver B.2.2).

Lua pode chamar (e manipular) funções escritas em Lua e funções escritas em C (ver B.2.6.9).

O tipo *userdata* permite que dados C arbitrários possam ser armazenados em variáveis Lua. Este tipo corresponde a um bloco de memória e não tem operações pré-definidas em Lua, exceto atribuição e teste de identidade. Contudo, através do uso de *metatables*, o programador pode definir operações para valores userdata (ver B.2.9). Valores userdata não podem ser criados ou modificados em Lua, somente através da API C. Isto garante a integridade dos dados que pertencem ao programa hospedeiro.

O tipo *thread* representa fluxos de execução independentes e é usado para implementar co-rotinas (ver B.2.12). Não confunda o tipo thread de Lua com processos leves do sistema operacional. Lua dá suporte a co-rotinas em todos os sistemas, até mesmo naqueles que não dão suporte a processos leves.

O tipo *table* implementa arrays associativos, isto é, arrays que podem ser indexados não apenas por números, mas por qualquer valor (exceto **nil**). Tabelas podem ser heterogêneas; isto é, elas podem conter valores de todos os tipos (exceto **nil**). Tabelas são o único mecanismo de estruturação de dados em Lua; elas podem ser usadas para representar arrays comuns, tabelas de símbolos, conjuntos, registros, grafos, árvores, etc. Para representar registros, Lua usa o nome do campo como um índice. A linguagem dá suporte a esta representação oferecendo a.name como um açúcar sintático para a["name"]. Existem várias maneiras convenientes de se criar tabelas em Lua (ver B.2.6.8).

Da mesma forma que os índices, o valor de um campo da tabela pode possuir qualquer tipo (exceto **nil**). Em particular, dado que funções são valores de primeira classe, campos de tabela podem conter funções. Portanto, tabelas podem também possuir metódos (ver B.2.6.10).

Valores do tipo table, function, thread e userdata (completo) são objetos: variáveis não contêm realmente estes valores, somente *referências* para eles. Atribuição, passagem de parâmetro, e retorno de funções sempre lidam com referências para tais valores; estas operações não implicam em qualquer espécie de cópia.

A função *type* retorna uma cadeia de caracteres descrevendo o tipo de um dado valor.

B.2.3.2 Coerção

Lua provê conversão automática entre valores do tipo string e do tipo number em tempo de execução. Qualquer operação aritmética aplicada a uma cadeia de caracteres tenta converter esta cadeia para um número, seguindo as regras de conversão usuais. De forma análoga, sempre que um número é usado onde uma cadeia de caracteres é esperada, o número é convertido para uma cadeia, em um formato razoável. Para um controle completo sobre como números são convertidos para cadeias, use a função format da biblioteca string (ver *string.format*).

B.2.4 Variáveis

Variáveis são lugares usados para armazenar valores.

Existem três tipos de variáveis em Lua: variáveis globais, variáveis locais e campos de tabelas.

Um nome simples pode denotar uma variável global ou uma variávei local (ou um parâmetro formal de uma função, que é um caso particular de variável local):

```
var ::= Nome
```

Nome denota identificadores, como definido em B.2.2.

Assume-se que toda variável é uma variável global a menos que ela seja explicitamente declarada como uma variável local (ver B.2.5.8). Variáveis locais possuem escopo léxico: variáveis locais podem ser livremente acessadas por funções definidas dentro do seu escopo (ver B.2.7).

Antes da variável receber a sua primeira atribuição, o seu valor é **nil**.

Colchetes são usados para indexar uma tabela:

```
var ::= expprefixo `[´ exp `]´
```

A semântica de acessos a variáveis globais e a campos de tabelas pode ser mudada através do uso de meta-tabelas. Um acesso a uma variável indexada t[i] é equivalente a uma chamada gettable_event(t,i) (ver B.2.9 para uma descrição completa da função gettable_event. Esta função não é definida nem pode ser chamada em Lua. Ela é usada aqui somente para fins didáticos).

A sintaxe var.Nome é apenas um açúcar sintático para var["Nome"]:

```
var ::= expprefixo `.´ Nome
```

Todas as variáveis globais são mantidas como campos em tabelas Lua comuns, chamadas de tabelas de ambiente ou simplesmente de ambientes (ver B.2.10). Cada função tem sua própria referência para um ambiente, de forma que todas as variáveis globais dentro de uma função irão se referir para esta tabela de ambiente. Quando uma função é criada, ela herda o ambiente da função que a criou. Para obter a tabela de ambiente de uma função Lua, deve-se chamar *getfenv*. Para trocar a tabela de ambiente, deve-se chamar *setfenv* (a única maneira de tratar o ambiente de funções C é através da a biblioteca de depuração; ver B.5.11).

Um acesso a uma variável global x é equivalente a _env.x, que por sua vez é equivalente a

```
gettable_event(_env, "x")
```

onde _env é o ambiente da função corrente (ver B.2.9 para uma descrição completa da função gettable_event. Esta função não é definida nem pode ser chamada em Lua. De modo análogo, a variável _env não é definida em Lua. Elas foram usadas aqui somente para fins didáticos).

B.2.5 Comandos

B.2.5.1 Conceitos básicos

Lua oferece um conjunto quase convencional de comandos, similar ao conjunto de comandos disponíveis em Pascal ou C. Este conjunto inclui atribuição, estruturas de controle, chamadas de funções e declarações de variáveis.

B.2.5.2 Trechos

A unidade de execução de Lua é denominada de trecho. Um trecho é simplesmente uma seqüência de comandos, os quais são executados sequencialmente. Cada comando pode opcionalmente ser seguido por um ponto-e-vírgula:

```
trecho ::= {comando [`;´]}
```

Não existem comandos vazios e portanto a construção ';;' não é válida.

Lua trata um trecho como o corpo de uma função anônima com um número variável de argumentos (ver B.2.6.10). Desta forma, trechos podem definir variáveis locais, receber argumentos e retornar valores.

Um trecho pode ser armazenado em um arquivo ou em uma cadeia de caracteres dentro do programa hospedeiro. Quando um trecho é executado, ele é primeiro pré-compilado em instruções para uma máquina virtual e depois o código compilado é executado por um interpretador para a máquina virtual.

Trechos também podem ser pré-compilados em uma forma binária; ver o programa luac para mais detalhes. Programas na forma de código fonte e na forma de um arquivo fonte já compilado são intercambiáveis; Lua automaticamente determina qual é o tipo do arquivo e age em conformidade com ele.

B.2.5.3 Blocos

Um bloco é uma lista de comandos; sintaticamente, um bloco é a mesma coisa que um trecho:

```
bloco ::= trecho
```

Um bloco pode ser explicitamente delimitado para produzir um único comando:

```
comando ::= do bloco end
```

Blocos explícitos são úteis para controlar o escopo de declarações de variáveis. Blocos explícitos são também usados às vezes para adicionar um comando **return** ou **break** no meio de outro bloco (ver B.2.5.5).

B.2.5.4 Atribuição

Lua permite atribuição múltipla. Em virtude disto, a sintaxe para atribuição define uma lista de variáveis no lado esquerdo e uma lista de expressões no lado direito. Os elementos em ambos os lados são separados por vírgulas:

```
comando ::= listavar `=´ listaexp
```

```
listavar ::= var {`,´ var}
listaexp ::= exp {`,´ exp}
```

Expressões são discutidas em B.2.6.

Antes da atribuição ser realizada, a lista de valores é ajustada para o comprimento da lista de variáveis. Se há mais valores do que o necessário, os valores em excesso são descartados. Se há menos valores do que o necessário, a lista é estendida com tantos **nil**'s quantos sejam necessários. Se a lista de expressões termina com uma chamada de função, então todos os valores retornados por esta chamada entram na lista de valores, antes do ajuste ser realizado (exceto quando a chamada é delimitada por parênteses; ver B.2.6).

Um comando de atribuição primeiro avalia todas as suas expressões e somente depois é que a atribuição é realizada. Desta forma, o código

```
i = 3
i, a[i] = i+1, 20
```

atribui 20 a a[3], sem afetar a[4] porque o i em a[i] é avaliado (para 3) antes de receber o valor 4. De modo similar, a linha

```
x, y = y, x
```

troca os valores de x e y.

A semântica de atribuições para variáveis globais e campos de tabelas pode ser mudada através do uso de meta-tabelas. Uma atribuição para uma variável indexada t[i] = val é equivalente a settable_event(t,i,val) (ver B.2.9 para uma descrição completa da função settable_event. Esta função não é definida nem pode ser chamada em Lua. Ela foi usada aqui somente para fins didáticos).

Uma atribuição a uma variável global x = val é equivalente à atribuição _env.x = val, que por sua vez é equivalente a

```
settable_event(_env, "x", val)
```

onde _env é o ambiente da função sendo executada (a variável _env não é definida em Lua. Ela foi usada aqui somente para fins didáticos).

B.2.5.5 Estruturas de controle

As estruturas de controle **if**, **while** e **repeat** possuem o significado usual e a sintaxe familiar:

```
comando ::= while exp do bloco end
comando ::= repeat bloco until exp
comando ::= if exp then bloco {elseif exp then bloco} [else bloco] end
```

Lua também possui um comando **for**, o qual possui duas variações (ver B.2.5.6).

A expressão da condição de uma estrutura de controle pode retornar qualquer valor. Tanto **false** como **nil** são considerados um valor falso. Todos os valores diferentes de **nil** e **false** são considerados como verdadeiros (em particular, o número 0 e a cadeia de caracteres vazia também são considerados valores verdadeiros).

No laço **repeat**–**until**, o bloco mais interno não termina na palavra-chave **until**, mas somente depois da condição. Desta forma, a condição pode referenciar variáveis locais declaradas dentro do bloco do laço.

O comando **return** é usado para retornar valores de uma função ou de um trecho (que nada mais é do que uma função). Funções e trechos podem retornar mais de um valor, de modo que a sintaxe para o comando **return** é

```
comando ::= return [listaexp]
```

O comando **break** é usado para terminar a execução de um laço **while**, **repeat** ou **for**, pulando para o próximo comando depois do laço:

```
comando ::= break
```

Um **break** termina a execução do laço mais interno.

Os comandos **return** e **break** somente podem ser escritos como o *último* comando de um bloco. Se é realmente necessário ter um **return** ou **break** no meio de um bloco, então um bloco interno explícito pode ser usado, como nas expressões idiomáticas do return end e do break end, pois agora tanto o **return** como o **break** são os últimos comandos em seus respectivos blocos (internos).

B.2.5.6 Comando for

O comando **for** possui duas variações: uma numérica e outra genérica.

O laço **for** numérico repete um bloco de código enquanto uma variável de controle varia de acordo com uma progressão aritmética. Ele possui a seguinte sintaxe:

```
comando ::= for nome `=´ exp `,´ exp [`,´ exp] do bloco end
```

O *bloco* é repetido para *nome* começando com o valor da primeira *exp*, até que ele passe o valor da segunda *exp* através de seguidos passos, sendo que a cada passo o valor da terceira *exp* é somado a *nome*. De forma mais precisa, um comando **for** como

```
for v = e1, e2, e3 do block end
```

é equivalente ao código:

```
do
  local var, limit, step = tonumber(e1), tonumber(e2), tonumber(e3)
  if not (var and limit and step) then error() end
  while (step > 0 and var <= limit) or (step <= 0 and var >= limit) do
    local v = var
    block
    var = var + step
  end
end
```

Observar o seguinte:

— todas as três expressões de controle são avaliadas um única vez, antes do laço começar. Elas devem obrigatoriamente produzir números;

— *var*, *limit* e *step* são variáveis invisíveis. Os nomes foram utilizados aqui somente para fins didáticos;

— se a terceira expressão (o passo) está ausente, então um passo de tamanho 1 é usado;

— é possível usar **break** para sair de um laço **for**;

— a variável de laço v é local ao laço; não é possível usar o valor desta variável após o fim do **for** ou depois do **for** ter sido interrompido pelo uso de um **break**. Se for preciso o valor desta variável, dese atribui-lo a outra variável antes de interromper ou sair do laço.

O comando **for** genérico funciona utilizando funções, chamadas de *iteradoras*. A cada iteração, a função iteradora é chamada para produzir um novo valor, parando quando este novo valor é **nil**. O laço **for** genérico possui a seguinte sintaxe:

```
comando ::= for listadenomes in listaexp do bloco end
listadenomes ::= Nome {`,´ Nome}
```

Um comando **for** como

```
for var_1, ···, var_n in explist do block end
```

é equivalente ao código:

```
do
  local f, s, var = explist
  while true do
    local var_1, ···, var_n = f(s, var)
    var = var_1
    if var == nil then break end
    block
  end
end
```

Observar o seguinte:

— *explist* é avaliada somente uma vez. Os seus resultados são uma função *iteradora*, um *estado* e um valor inicial para a primeira *variável iteradora*;

— *f*, *s* e *var* são variáveis invisíveis. Os nomes foram utilizados aqui somente para fins didáticos;

— é possível usar **break** para sair de um laço **for**;

— as variáveis de laço *var_i* são locais ao laço; não é possível usar os valores delas após o término do **for**. Se precisa-se destes valores, deve-se atribuí-los a outras variáveis antes de interromper o laço ou sair do mesmo.

B.2.5.7 Chamadas de função como comandos

Para permitir possíveis efeitos colaterais, funções podem ser executadas como comandos:

```
comando ::= chamadadefuncao
```

Neste caso, todos os valores retornados pela função são descartados. Chamadas de função são explicadas em B.2.6.9.

B.2.5.8 Declarações locais

Variáveis locais podem ser declaradas em qualquer lugar dentro de um bloco. A declaração pode incluir uma atribuição inicial:

```
comando ::= local listadenomes [`=´ listaexp]
```

Caso ocorra uma atribuição inicial, a sua semântica é a mesma de uma atribuição múltipla (ver B.2.5.4). Caso contrário, todas as variáveis são inicializadas com **nil**.

Um trecho também é um bloco (ver B.2.5.2) e portanto variáveis locais podem ser declaradas em um trecho fora de qualquer bloco explícito. O escopo de uma variável declarada desta forma se estende até o fim do trecho.

As regras de visibilidade para variáveis locais são explicadas em B.2.7.

B.2.6 Expressões

B.2.6.1 Expressões básicas

As expressões básicas em Lua são as seguintes:

```
exp ::= expprefixo
exp ::= nil | false | true
exp ::= Numero
exp ::= Cadeia
exp ::= funcao
exp ::= construtortabela
exp ::= `...´
exp ::= exp opbin exp
exp ::= opunaria exp
expprefixo ::= var | chamadadefuncao | `(´ exp `)´
```

Números e cadeias literais são explicados em B.2.2; variáveis são explicadas em B.2.4; definições de funções são explicadas em B.6.10; chamadas de funções são explicadas em B.2.6.9; construtores de tabelas são explicados em B.2.6.8. Expressões *vararg*, denotadas por três pontos ('...'), somente podem ser usadas quando estão imediatamente dentro de uma função que possui um número variável de argumentos; elas são explicadas em B.2.6.10.

Operadores binários compreendem operadores aritméticos (ver B.2.6.2), operadores relacionais (ver B.2.6.3), operadores lógicos (ver B.2.6.4) e o operador de concatenação (ver B.2.6.5). Operadores unários compreendem o menos unário (ver B.2.6.2), o **not** unário (ver B.2.6.4) e o operador de tamanho unário (ver B.2.6.6).

Tanto chamadas de funções como expressões vararg podem resultar em múltiplos valores. Se a expressão é usada como um comando (ver B.2.5.7) (o que somente é possível para chamadas de funções), então a sua lista de retorno é ajustada para zero elementos, descartando portanto todos os valores retornados. Se a expressão é usada como o último (ou o único) elemento de uma lista de expressões, então nenhum ajuste é feito (a menos que a chamada seja delimitada por parênteses). Em todos os demais contextos, Lua ajusta a lista de resultados para um elemento, descartando todos os valores exceto o primeiro.

Seguem alguns exemplos:

```
f()                -- ajusta para 0 resultados
g(f(), x)          -- f() é ajustado para 1 resultado
g(x, f())          -- g recebe x mais todos os resultados de f()
a,b,c = f(), x     -- f() é ajustado para 1 resultado (c recebe nil)
a,b = ...          -- a recebe o primeiro parâmetro da lista vararg,
                   -- b recebe o segundo (tanto a como b podem receber nil caso
não
                   -- exista um parâmetro correspondente na lista)

a,b,c = x, f()     -- f() é ajustado para 2 resultados
a,b,c = f()        -- f() é ajustado para 3 resultados
```

```
return f()         -- retorna todos os resultados de f()
return ...         -- retorna todos os resultados recebidos da lista vararg
return x,y,f()     -- retorna x, y e todos os resultados de f()
{f()}              -- cria uma lista com todos os resultados de f()
{...}              -- cria uma lista com todos os parâmetros da lista vararg
{f(), nil}         -- f() é ajustado para 1 resultado
```

Uma expressão delimitada por parênteses sempre resulta em um único valor. Dessa forma, (f(x,y,z)) é sempre um único valor, mesmo que f retorne múltiplos valores (o valor de (f(x,y,z)) é o primeiro valor retornado por f, ou **nil** se f não retorna nenhum valor).

B.2.6.2 Operadores aritméticos

Lua provê os operadores aritméticos usuais: os operadores binários + (adição), - (subtração), * (multiplicação), / (divisão), % (módulo) e ^ (exponenciação); e o operador unário - (negação). Se os operandos são números ou cadeias de caracteres que podem ser convertidas para números (ver B.2.3.2), então todas as operações possuem o seu significado usual. A exponenciação funciona para qualquer expoente. Por exemplo, x^(-0.5) calcula o inverso da raiz quadrada de x. Módulo é definido como

```
a % b == a - math.floor(a/b)*b
```

Ou seja, é o resto de uma divisão arredondada em direção a menos infinito.

B.2.6.3 Operadores relacionais

Os operadores relacionais em Lua são:

```
==    ~=    <    >    <=    >=
```

Estes operadores sempre possuem como resultado **false** ou **true**.

A igualdade (==) primeiro compara o tipo de seus operandos. Se os tipos são diferentes, então o resultado é **false**. Caso contrário, os valores dos operandos são comparados. Números e cadeias de caracteres são comparados de maneira usual. Objetos (valores do tipo table, userdata, thread e function) são comparados por referência: dois objetos são considerados iguais somente se eles são o mesmo objeto. Toda vez que um novo objeto é criado (um valor com tipo table, userdata, thread ou function) este novo objeto é diferente de qualquer outro objeto que existia anteriormente.

É possível mudar a maneira como Lua compara os tipos table e userdata através do uso do meta-método "eq" (ver B.2.9).

As regras de conversão em B.2.3.2 não se aplicam a comparações de igualdade. Portanto, "0"==0 é avaliado como **false** e t[0] e t["0"] denotam posições diferentes em uma tabela.

O operador ~= é exatamente a negação da igualdade (==).

Os operadores de ordem trabalham da seguinte forma. Se ambos os argumentos são números, então eles são comparados como tais. Caso contrário, se ambos os argumentos são cadeias de caracteres, então seus valores são comparados de acordo com a escolha de idioma atual. Caso contrário, Lua tenta chamar o meta-método "lt" ou o meta-método "le" (ver B.2.9).

B.2.6.4 Operadores lógicos

Os operadores lógicos em Lua são **and**, **or** e **not**. Assim como as estruturas de controle (ver B.2.5.5), todos os operadores lógicos consideram **false** e **nil** como falso e qualquer coisa diferente como verdadeiro.

O operador de negação **not** sempre retorna **false** ou **true**. O operador de conjunção **and** retorna seu primeiro argumento se este valor é **false** ou **nil**; caso contrário, **and** retorna seu segundo argumento. O operador de disjunção **or** retorna seu primeiro argumento se o valor deste é diferente de **nil** e de **false**; caso contrário, **or** retorna o seu segundo argumento. Tanto **and** como **or** usam avaliação de curto-circuito; isto é, o segundo operando é avaliado somente quando é necessário. Seguem alguns exemplos:

```
10 or 20            --> 10
10 or error()       --> 10
nil or "a"          --> "a"
nil and 10          --> nil
false and error()   --> false
false and nil       --> false
false or nil        --> nil
10 and 20           --> 20
```

Neste Norma, --> indica o resultado da expressão precedente.

B.2.6.5 Concatenação

O operador de concatenação de cadeias de caracteres em Lua é denotado por dois pontos ('..'). Se ambos os operandos são cadeias de caracteres ou números, então eles são convertidos para cadeias de caracteres de acordo com as regras mencionadas em B.2.3.2. Caso contrário, o meta-método "concat" é chamado (ver B.2.9).

B.2.6.6 O operador de tamanho

O operador de tamanho é denotado pelo operador unário **#**. O tamanho de uma cadeia de caracteres é o seu número de bytes (isto é, o significado usual de tamanho de uma cadeia quando cada caractere ocupa um byte).

O tamanho de uma tabela t é definido como qualquer índice inteiro n tal que t[n] não é **nil** e t[n+1] é **nil**; além disso, se t[1] é **nil**, n pode ser zero. Para um array comum, com todos os valores diferentes de **nil** indo de 1 até um dado n, o seu tamanho é exatamente aquele n, o índice do seu último valor. Se o array possui "buracos" (isto é, valores **nil** entre dois outros valores diferentes de **nil**), então **#**t pode ser qualquer um dos índices que imediatamente precedem um valor **nil** (isto é, ele pode considerar qualquer valor **nil** como o fim do array).

B.2.6.7 Precedência

A precedência de operadores em Lua segue a tabela abaixo, da menor prioridade para a maior:

```
or
and
<     >     <=    >=    ~=    ==
..
+     -
*     /     %
not   #     - (unary)
^
```

Como usual, pode-se usar parênteses para mudar as precedências de uma expressão. Os operadores de concatenação ('..') e de exponenciação ('^') são associativos à direita. Todos os demais operadores binários são associativos à esquerda.

B.2.6.8 Construtores de tabelas

Construtores de tabelas são expressões que criam tabelas. Toda vez que um construtor é avaliado, uma nova tabela é criada. Construtores podem ser usados para criar tabelas vazias ou para criar uma tabela e inicializar alguns dos seus campos. A sintaxe geral de construtores é

```
construtortabela ::= `{´ [listadecampos] `}´
listadecampos ::= campo {separadordecampos campo} [separadordecampos]
campo ::= `[´ exp `]´ `=´ exp | Nome `=´ exp | exp
separadordecampos ::= `,´ | `;´
```

Cada campo da forma [exp1] = exp2 adiciona à nova tabela uma entrada cuja chave é exp1 e cujo valor é exp2. Um campo da forma Nome = exp é equivalente a ["Nome"] = exp. Finalmente, campos da forma exp são equivalentes a [i] = exp, onde i representa números inteiros consecutivos, iniciando com 1. Campos nos outros formatos não afetam esta contagem. Por exemplo,

```
a = { [f(1)] = g; "x", "y"; x = 1, f(x), [30] = 23; 45 }
```

é equivalente a

```
do
  local t = {}
  t[f(1)] = g
  t[1] = "x"         -- primeira exp
  t[2] = "y"         -- segunda exp
  t.x = 1            -- t["x"] = 1
  t[3] = f(x)        -- terceira exp
  t[30] = 23
  t[4] = 45          -- quarta exp
  a = t
end
```

Se o último campo na lista possui a forma exp e a expressão é uma chamada de função ou uma expressão com um número variável de argumentos, então todos os valores retornados pela expressão entram na lista consecutivamente (ver B.2.6.9). Para evitar isto, coloque parênteses ao redor da chamada de função (ou da expressão com número variável de argumentos) (ver B.2.6).

A lista de campos pode ter um separador a mais no fim, como uma conveniência para código gerado automaticamente.

B.2.6.9 Chamadas de função

Uma chamada de função em Lua tem a seguinte sintaxe:

```
chamadadefuncao ::= expprefixo args
```

Em uma chamada de função, primeiro expprefixo e args são avaliados. Se o valor de expprefixo possui tipo *function*, então esta função é chamada com os argumentos fornecidos. Caso contrário, o meta-método "call" de expprefixo é chamado, tendo como primeiro parâmetro o valor de expprefixo, seguido pelos argumentos originais da chamada (ver B.2.9).

A forma

```
chamadadefuncao ::= expprefixo `:´ Nome args
```

pode ser usada para chamar "métodos". Uma chamada v:nome(*args*) é um açúcar sintático para v.nome(v,*args*), com a diferença de que v é avaliado somente uma vez.

Argumentos possuem a seguinte sintaxe:

```
args ::= `(´ [listaexp] `)´
args ::= construtordetabela
args ::= Cadeia
```

Todas as expressões fornecidas como argumento são avaliadas antes da chamada. Uma chamada da forma f{*campos*} é uma açúcar sintático para f({*campos*}); ou seja, a lista de argumentos consiste somente em uma tabela nova. Uma chamada da forma f'*cadeia*' (ou f"*cadeia*" ou f[[*cadeia*]]) é um açúcar sintático para f('*cadeia*'); ou seja, a lista de argumentos consiste somente em uma cadeia de caracteres literal.

Uma exceção em relação à sintaxe de formato livre de Lua é que não é possível colocar uma quebra de linha antes do '(' em uma chamada de função. Esta restrição evita algumas ambigüidades na linguagem. Se fosse escrito

```
a = f
(g).x(a)
```

Lua poderia ver isto como um comando único, a = f(g).x(a). Portanto, se forem desejados dois comandos, deve-se obrigatoriamente colocar um ponto-e-vírgula entre eles. Se realmente for desejado chamar f, deve-se remover a quebra de linha antes de (g).

Uma chamada da forma return *chamadadefuncao* é denominada de chamada final. Lua implementa chamadas finais próprias (ou recursões finais próprias): em uma chamada final, a função chamada reusa a entrada na pilha da função que a chamou. Portanto, não há limite no número de chamadas finais aninhadas que um programa pode executar. Contudo, uma chamada final apaga qualquer informação de depuração sobre a função chamadora. Uma chamada final somente acontece com uma sintaxe particular, onde o **return** possui uma única chamada de função como argumento; esta sintaxe faz com que a chamada de função retorne exatamente os valores de retorno da função chamada. Dessa forma, nenhum dos exemplos a seguir são chamadas finais:

```
return (f(x))        -- o número de resultados é ajustado para 1
return 2 * f(x)
return x, f(x)       -- resultados adicionais
f(x); return         -- resultados descartados
return x or f(x)     -- o número de resultados é ajustado para 1
```

B.2.6.10 Definições de funções

A sintaxe para a definição de uma função é

```
funcao ::= function corpodafuncao
funcao ::= `(´ [listapar] `)´ bloco end
```

O seguinte açúcar sintático simplifica definições de funções:

```
comando ::= function nomedafuncao corpodafuncao
comando ::= local function Nome corpodafuncao
nomedafuncao ::= Nome {`.´ Nome} [`:´ Nome]
```

O comando

```
function f () body end
```

é traduzido para

```
f = function () body end
```

O comando

```
function t.a.b.c.f () body end
```

é traduzido para

```
t.a.b.c.f = function () body end
```

O comando

```
local function f () body end
```

é traduzido para

```
local f; f = function () body end
```

e não para

```
local f = function () body end
```

Isto somente faz diferença quando o corpo da função contém uma referência para f.

Uma definição de função é uma expressão executável, cujo valor tem tipo *function*. Quando Lua pré-compila um trecho, todas os corpos das funções do trecho são pré-compilados também. Então, sempre que Lua executa a definição de uma função, a função é instanciada (ou fechada). Esta instância da função (ou fecho) é o valor final da expressão. Instâncias diferentes da mesma função podem se referir a diferentes variáveis locais externas e podem ter diferentes tabelas de ambiente.

Parâmetros comportam-se como variáveis locais que são inicializadas com os valores dos argumentos:

```
listapar ::= listadenomes [`,´ `...´] | `...´
```

Quando uma função é chamada, a lista de argumentos é ajustada para o tamanho da lista de parâmetros, a não ser que a função seja de aridade variável ou *vararg*, o que é indicado por três pontos ('...') no final da sua lista de parâmetros. Uma função vararg não ajusta sua lista de argumentos; ao invés disso, ela coleta todos os argumentos extras e os fornece para a função através de uma expressão *vararg*, a qual também é representada como três pontos. O valor desta expressão é uma lista de todos os argumentos extras correntes, similar a uma função com múltiplos valores de retorno. Se uma expressão vararg é usada dentro de outra expressão ou no meio de uma lista de expressões, então a sua lista de valores de retorno é ajustada para um elemento. Se a expressão é usada como o último elemento de uma lista de expressões, então nenhum ajuste é feito (a menos que a chamada seja delimitada por parênteses).

Como um exemplo, considere as seguintes definições:

```
function f(a, b) end
function g(a, b, ...) end
function r() return 1,2,3 end
```

Neste caso, tem-se o seguinte mapeamento de argumentos para parâmetros e para as expressões vararg:

```
CHAMADA          PARÂMETROS

f(3)             a=3, b=nil
f(3, 4)          a=3, b=4
f(3, 4, 5)       a=3, b=4
f(r(), 10)       a=1, b=10
f(r())           a=1, b=2

g(3)             a=3, b=nil, ... -->  (nada)
g(3, 4)          a=3, b=4,   ... -->  (nada)
g(3, 4, 5, 8)    a=3, b=4,   ... -->  5  8
g(5, r())        a=5, b=1,   ... -->  2  3
```

Resultados são retornados usando o comando **return** (ver B.2.5.5). Se o controle alcança o fim de uma função sem encontrar um comando **return**, então a função retorna sem nenhum resultado.

A sintaxe de dois pontos é usada para definir métodos, isto é, funções que possuem um parâmetro extra implícito self. Desta forma, o comando

```
function t.a.b.c:f (params) body end
```

é uma açúcar sintático para

```
t.a.b.c.f = function (self, params) body end
```

B.2.7 Regras de visibilidade

Lua é uma linguagem com escopo léxico. O escopo das variáveis começa no primeiro comando depois da sua declaração e vai até o fim do bloco mais interno que inclui a declaração. Considerar o seguinte exemplo:

```
x = 10                -- variável global
do                    -- bloco novo
  local x = x         -- novo 'x', com valor 10
  print(x)            --> 10
  x = x+1
  do                  -- outro bloco
    local x = x+1     -- outro 'x'
    print(x)          --> 12
  end
  print(x)            --> 11
end
print(x)              --> 10  (o x global)
```

Em uma declaração como local x = x, o novo x sendo declarado não está no escopo ainda e portanto o segundo x se refere a uma variável externa.

Por causa das regras de escopo léxico, variáveis locais podem ser livremente acessadas por funções definidas dentro do seu escopo. Uma variável local usada por uma função mais interna é chamada de *upvalue* ou variável local externa, dentro da função mais interna.

Cada execução de um comando **local** define novas variáveis locais. Considerar o exemplo a seguir:

```
a = {}
local x = 20
for i=1,10 do
  local y = 0
```

```
  a[i] = function () y=y+1; return x+y end
end
```

O laço cria dez fechos (isto é, dez instâncias da função anônima). Cada um destes fechos usa uma variável y diferente, enquanto todos eles compartilham a mesma variável x.

B.2.8 Tratamento de erros

Dado que Lua é uma linguagem embarcada de extensão, todas as ações de Lua começam a partir de código C no programa hospedeiro que chama uma função da biblioteca de Lua (ver *lua_pcall*). Sempre que um erro ocorre durante a compilação ou execução, o controle retorna para C, que pode tomar as medidas apropriadas (tais como imprimir uma mensagem de erro).

O código Lua pode explicitamente gerar um erro através de uma chamada à função *error*. Se for preciso capturar erros em Lua, pode-se usar a função *pcall*.

B.2.9 Metatabelas

Todo valor em Lua pode ter uma *meta-tabela*. Esta *meta-tabela* é uma tabela Lua comum que define o comportamento do valor original com relação a certas operações especiais. É possível mudar vários aspectos do comportamento de operações sobre um valor especificando campos específicos na meta-tabela do valor. Por exemplo, quando um valor não numérico é o operando de uma adição, Lua verifica se existe uma função associada com o campo "__add" na meta-tabela do valor. Se a função existe, Lua chama esta função para realizar a adição.

Chamamos as chaves em uma meta-tabela de eventos e os valores de *meta-métodos*. No exemplo anterior, o evento é "add" e o meta-método é a função que realiza a adição.

É possível obter a meta-tabela de qualquer valor usando a função *getmetatable*.

Pode-se mudar a meta-tabela de tabelas através da função *setmetatable*. Não se pode mudar a meta-tabela de outros tipos de Lua (a menos que a biblioteca de depuração for utilizada); deve-se obrigatoriamente usar a API C para fazer isto.

Tabelas e objetos do tipo userdata completos possuem meta-tabelas individuais (embora múltiplas tabelas e objetos userdata possam compartilhar suas meta-tabelas); valores de todos os outros tipos compartilham um única meta-tabela por tipo. Sendo assim, há somente uma meta-tabela para todos os números, uma para todas as cadeias de caracteres, etc.

Uma meta-tabela pode controlar como um objeto se comporta em operações aritméticas, comparações com relação à ordem, concatenação, operação de tamanho e indexação. Uma meta-tabela também pode definir uma função a ser chamada quando um objeto userdata é coletado pelo coletor de lixo. Para cada uma destas operações Lua associa uma chave específica chamada um evento. Quando Lua realiza uma destas operações sobre um valor, Lua verifica se este valor possui uma meta-tabela com o evento correspondente. Se este é o caso, o valor associado àquela chave (o meta-método) controla como Lua irá realizar a operação.

Meta-tabelas controlam as operações listadas a seguir. Cada operação é identificada por seu nome correspondente. A chave para cada operação é uma cadeia de caracteres começando com o nome da operação sendo precedido por dois sublinhados, '__'; por exemplo, a chave para a operação "add" é a cadeia "__add". A semântica destas operações é melhor explicada por meio de uma função Lua que descreve como o interpretador executa a operação.

O código mostrado aqui é meramente ilustrativo; o comportamento real está codificado no interpretador e é muito mais eficiente do que esta simulação. Todas as funções usadas nestes descrições (*rawget*, *tonumber* etc.) são descritas em B.5.2. Em particular, para recuperar o meta-método de um dado objeto, usa-se a seguinte expressão:

```
metatable(obj)[event]
```

Isto deve ser lido como

```
rawget(getmetatable(obj) or {}, event)
```

Isto é, o acesso a um meta-método não invoca outros meta-métodos e o acesso a objetos que não possuem meta-tabelas não falha (ele simplesmente resulta em **nil**).

- **"add":** a operação +.

A função *getbinhandler* abaixo define como Lua escolhe um tratador para uma operação binária. Primeiro, Lua tenta o primeiro operando. Se este tipo não definir um tratador para a operação, então Lua tenta o segundo operando.

```
function getbinhandler (op1, op2, event)
  return metatable(op1)[event] or metatable(op2)[event]
end
```

Usando essa função, o comportamento do op1 + op2 é

```
function add_event (op1, op2)
  local o1, o2 = tonumber(op1), tonumber(op2)
  if o1 and o2 then      -- ambos operandos são numéricos?
    return o1 + o2       -- `+' aqui é o  `add' primitivo
  else                   -- ao menos um dos operandos não é numérico
    local h = getbinhandler(op1, op2, "__add")
    if h then
      -- chama o tratador passando ambos os operandos
      return h(op1, op2)
    else                 -- sem tratador disponível: comportamento padrão
      error("...")
    end
  end
end
```

- **"sub":** a operação -. Comportamento similar ao da operação “add”.
- **"mul":** a operação *. Comportamento similar ao da operação “add”.
- **"div":** a operação /. Comportamento similar ao da operação “add”.
- **"mod":** a operação %. Comportamento similar à operação “add”, com a operação o1 - floor(o1/o2)*o2 como operação primitiva.
- **"pow":** a operação ^ (exponenciação). Comportamento similar ao da operação “add”, com a função pow (proveniente da biblioteca matemática do C) como operação primitiva.
- **"unm":** a operação unária -.

```
function unm_event (op)
  local o = tonumber(op)
  if o then         -- operando é numérico?
    return -o      -- `-' aqui é o `unm' primitivo
  else              -- o operando não é numérico
    -- tenta encontrar um tratador para o operando
    local h = metatable(op).__unm
```

```
    if h then
      -- chama o tratador passando o operando
      return h(op)
    else             -- sem tratador disponível: comportamento padrão
      error("...")
    end
  end
end
```

- **"concat":** a operação .. (concatenação).

```
function concat_event (op1, op2)
  if (type(op1) == "string" or type(op1) == "number") and
     (type(op2) == "string" or type(op2) == "number") then
    return op1 .. op2  -- concatenação de strings primitiva
  else
    local h = getbinhandler(op1, op2, "__concat")
    if h then
      return h(op1, op2)
    else
      error("...")
    end
  end
end
```

- **"len":** a operação #.

```
function len_event (op)
  if type(op) == "string" then
    return strlen(op)         -- tamanho de string primitivo
  elseif type(op) == "table" then
    return #op                  -- tamanho de tabela primitivo
  else
    local h = metatable(op).__len
    if h then
      -- chama o tratador passando o operando
      return h(op)
    else                            -- sem tratador disponível: comportamento
padrão
      error("...")
    end
  end
end
```

Ver B.2.6.6 para obter uma descrição do comprimento de uma tabela.

- **"eq":** a operação **==**. A função *getcomphandler* define como Lua escolhe um metamétodo para comparação de operadores. Um metamétodo só é selecionado quando ambos os objetos em comparação têm o mesmo tipo e o mesmo metamétodo para a operação selecionada.

```
function getcomphandler (op1, op2, event)
  if type(op1) ~= type(op2) then return nil end
  local mm1 = metatable(op1)[event]
  local mm2 = metatable(op2)[event]
```

```
  if mm1 == mm2 then return mm1 else return nil end
end
```

O evento “eq” é definido como:

```
function eq_event (op1, op2)
  if type(op1) ~= type(op2) then  -- different types?
    return false                            -- different objects
  end
  if op1 == op2 then                    -- primitive equal?
    return true                              -- objects are equal
  end
  -- try metamethod
  local h = getcomphandler(op1, op2, "__eq")
  if h then
    return h(op1, op2)
  else
    return false
  end
end
```

a ~= b é equivalente a not (a = = b).

- **"lt":** a operação <.

```
function lt_event (op1, op2)
  if type(op1) == "number" and type(op2) == "number" then
    return op1 < op2   -- comparação numérica
  elseif type(op1)=="string" and type(op2)=="string" then
    return op1 < op2   -- comparação lexicográfica
  else
    local h = getcomphandler(op1, op2, "__lt")
    if h then
      return h(op1, op2)
    else
      error("...");
    end
  end
end
```

a > b é equivalente a b < a.

- **"le":** a operação <=.

```
function le_event (op1, op2)
  if type(op1) == "number" and type(op2) == "number" then
    return op1 <= op2   -- comparação numérica
  elseif type(op1)=="string" and type(op2)=="string" then
    return op1 <= op2   -- comparação lexicográfica
  else
    local h = getcomphandler(op1, op2, "__le")
    if h then
      return h(op1, op2)
    else
      h = getcomphandler(op1, op2, "__lt")
      if h then
```

```
        return not h(op2, op1)
      else
        error("...");
      end
    end
  end
end
```

a >= b é equivalente a b <= a. Na ausência de um metamétodo “le”, Lua tenta o “lt”, assumindo que a <= b é equivalente a not (b < a).

- **"index":** acesso de indexação *table[key]*.

```
function gettable_event (table, key)
  local h
  if type(table) == "table" then
    local v = rawget(table, key)
    if v ~= nil then return v end
    h = metatable(table).__index
    if h == nil then return nil end
  else
    h = metatable(table).__index
    if h == nil then
      error("...");
    end
  end
  if type(h) == "function" then
    return h(table, key)      -- chama o tratador
  else return h[key]           -- ou repita a operação
  end
end
```

- **"newindex":** atribuição indexada *table[key] = value*.

```
function settable_event (table, key, value)
  local h
  if type(table) == "table" then
    local v = rawget(table, key)
    if v ~= nil then rawset(table, key, value); return end
    h = metatable(table).__newindex
    if h == nil then rawset(table, key, value); return end
  else
    h = metatable(table).__newindex
    if h == nil then
      error("...");
    end
  end
  if type(h) == "function" then
    return h(table, key,value)      -- chama o tratador
  else h[key] = value                  -- ou repita a operação
  end
end
```

- **"call":** chamado quando Lua chama um valor.

```
function function_event (func, ...)
  if type(func) == "function" then
    return func(...)       -- chamada primitiva
  else
    local h = metatable(func).__call
    if h then
      return h(func, ...)
    else
      error("...")
    end
  end
end
```

B.2.10 Ambientes

Além de meta-tabelas, objetos do tipo thread, function e userdata possuem outra tabela associada com eles, chamada de seu ambiente. Assim como meta-tabelas, ambientes são tabelas normais e vários objetos podem compartilhar o mesmo ambiente.

Ambientes associados com objetos do tipo userdata não possuem significado para Lua. É apenas uma conveniência para programadores associarem uma tabela a um objeto userdata.

Ambientes associados com fluxos de execução (threads) são chamados de ambientes globais. Eles são usados como o ambiente padrão pelos seus fluxos de execução e funções não aninhadas criadas pelo fluxo de execução (através de *loadfile*, *loadstring* ou *load*) e podem ser diretamente acessados pelo código C (ver B.3.4).

Ambientes associados com funções C podem ser diretamente acessados pelo código C (ver B.3.4). Eles são usados como o ambiente padrão para outras funções C criadas pela função.

Ambientes associados com funções Lua são usados para resolver todos os acessos a variáveis globais dentro da função (ver B.2.4). Eles são usados como o ambiente padrão para outras funções Lua criadas pela função.

É possível mudar o ambiente de uma função Lua ou do fluxo de execução que está sendo executado atualmente chamando *setfenv*. É possível obter o ambiente de uma função Lua ou do fluxo de execução sendo executado atualmente chamando *getfenv*. Para tratar o ambiente de outros objetos (userdata, funções C, outros fluxos de execução) deve-se obrigatoriamente usar a API C.

B.2.11 Coleta de lixo

B.2.11.1 Conceitos básicos

Lua realiza gerenciamento automático da memória. Isto significa que se não for preciso se preocupar com a alocação de memória para novos objetos nem com a liberação de memória quando os objetos não são mais necessários. Lua gerencia a memória automaticamente executando um coletor de lixo de tempos em tempos para coletar todos os objetos mortos (ou seja, aqueles objetos que não são mais acessíveis a partir de Lua). Todos os objetos em Lua estão sujeitos ao gerenciamento automático de memória: tabelas, userdata, funções, fluxos de execução e cadeias de caracteres.

Lua implementa um coletor de lixo marca-e-limpa (*mark-and-sweep*) incremental. O coletor usa dois números para controlar o seu ciclo de coleta de lixo: a *pausa do coletor de lixo* e o multiplicador de passo do coletor de lixo.

A pausa do coletor de lixo controla quanto tempo o coletor espera antes de iniciar um novo ciclo. Valores maiores fazem o coletor ser menos agressivo. Valores menores do que 1 significam que o coletor não irá esperar para iniciar um novo ciclo. Um valor de 2 significa que o coletor irá esperar até que a memória total em uso dobre antes de iniciar um novo ciclo.

O multiplicador de passo controla a velocidade relativa do coletor em relação à alocação de memória. Valores maiores fazem o coletor ser mais agressivo mas também aumentam o tamanho de cada passo incremental. Valores menores do que 1 fazem com que o coletor seja muito lento e pode ocorrer que o coletor nunca termine um ciclo. O valor padrão, 2, significa que o coletor é executado a uma velocidade que é "duas vezes" a velocidade de alocação de memória.

É possível mudar estes números através de chamadas às funções *lua_gc* em C ou *collectgarbage* em Lua. Ambas recebem valores em pontos percentuais como argumentos (de modo que um argumento cujo valor é 100 significa um valor real de 1). Com estas funções também se pode controlar o coletor diretamente (e.g., pará-lo e reiniciá-lo).

B.2.11.2 Metamétodos de coleta de lixo

Usando a API C, pode-se configurar os meta-métodos do coletor de lixo para objetos userdata (ver B.2.9). Estes meta-métodos também são chamados de finalizadores. Finalizadores permitem que se coordene a coleta de lixo de Lua com o gerenciamento de recursos externos (tais como o fechamento de arquivos, conexões de rede ou de bancos de dados ou a liberação de sua própria memória).

Objetos userdata com um campo __gc em suas meta-tabelas não são recolhidos imediatamente pelo coletor de lixo. Ao invés disso, Lua os coloca naquela lista. Depois que a coleta é realizada, Lua faz o equivalente da seguinte função para cada objeto userdata em uma lista:

```
function gc_event (userdata)
  local h = metatable(userdata).__gc
  if h then
    h(userdata)
  end
end
```

Ao final do ciclo de coleta de lixo, os finalizadores para os objetos userdata são chamados na ordem reversa ao de sua criação, entre aqueles coletados naquele ciclo. Isto é, o primeiro finalizador a ser chamado é aquele associado com o objeto userdata que foi criado por último no programa. O userdata só é efetivamente liberado no próximo ciclo de coleta de lixo.

B.2.11.3 Tabelas fracas

Uma tabela fraca é uma tabela cujos elementos são referências fracas. Uma referência fraca é ignorada pelo coletor de lixo. Em outras palavras, se as únicas referências para um objeto são referências fracas, então o coletor de lixo irá coletar este objeto.

Uma tabela fraca pode ter chaves fracas, valores fracos ou ambos. Uma tabela com chaves fracas permite a coleta de suas chaves mas impede a coleta de seus valores. Uma tabela com chaves fracas e valores fracos permite a coleta tanto das chaves como dos valores. Em qualquer caso, se a chave é coletada ou o valor é coletado, o par inteiro é removido da tabela. A fragilidade de uma tabela é controlada pelo campo __mode de sua meta-tabela. Se o campo __mode é uma cadeia de caracteres contendo o caractere 'k', as chaves da tabela são fracas. Se __mode contém 'v', os valores na tabela são fracos.

Depois de usar uma tabela como uma meta-tabela, não se deve mudar o valor de seu campo __mode. Caso contrário, o comportamento fraco das tabelas controladas por esta meta-tabela é indefinido.

B.2.12 Co-rotinas

Lua oferece suporte a co-rotinas, também conhecidas como fluxos de execução (*threads*) colaborativos. Uma co-rotina em Lua representa um fluxo de execução independente. Ao contrário de processos leves em sistemas que dão suporte a múltiplos fluxos de execução, uma co-rotina somente suspende sua execução através de uma chamada explícita a uma função de cessão.

É possível criar uma co-rotina com uma chamada à *coroutine.create*. O seu único argumento é uma função que é a função principal da co-rotina. A função create somente cria uma nova co-rotina e retorna uma referência para ela (um objeto do tipo *thread*); ela não inicia a execução da co-rotina.

Quando a função *coroutine.resume* é chamada pela primeira vez, recebendo como seu primeiro argumento o objeto do tipo thread retornado por *coroutine.create*, a co-rotina inicia a sua execução, na primeira linha de sua função principal. Depois que a co-rotina começa a ser executada, ela continua executando até terminar ou ceder.

Uma função pode terminar sua execução de duas maneiras: normalmente, quando sua função principal retorna (explicitamente ou implicitamente, depois da última instrução); e de maneira anormal, se ocorre um erro não protegido No primeiro caso, *coroutine.resume* retorna **true** mais quaisquer valores retornados pela função principal da co-rotina. No caso de acontecerem erros, *coroutine.resume* retorna **false** mais uma mensagem de erro.

Uma co-rotina cede a execução através de uma chamada à função *coroutine.yield*. Quando uma co-rotina cede, a *coroutine.resume* correspondente retorna imediatamente, mesmo se a cessão aconteceu dentro de uma chamada de função aninhada (isto é, não ocorreu dentro da função principal, mas em uma função chamada direta ou indiretamente pela função principal). No caso de uma cessão, *coroutine.resume* também retorna **true**, mais quaisquer valores passados para *coroutine.yield*. Na próxima vez que for recomeçada a execução da mesma co-rotina, ela continua sua execução do ponto onde ela cedeu, com a chamada para *coroutine.yield* retornando quaisquer argumentos extras passados para *coroutine.resume*.

Como *coroutine.create*, a função *coroutine.wrap* também cria uma co-rotina, mas ao invés de retornar a própria co-rotina, ela retorna uma função que, quando chamada, retoma a execução da co-rotina. Quaisquer argumentos passados para esta função vão como argumentos extras para *coroutine.resume*. *coroutine.wrap* retorna todos os valores retornados por *coroutine.resume*, exceto o primeiro (o código booleano de erro). Diferentemente de *coroutine.resume*, *coroutine.wrap* não captura erros; qualquer erro é propagado para o chamador.

Como um exemplo, considerar o seguinde código:

```
function foo (a)
  print("foo", a)
  return coroutine.yield(2*a)
end

co = coroutine.create(function (a,b)
      print("co-body", a, b)
      local r = foo(a+1)
      print("co-body", r)
      local r, s = coroutine.yield(a+b, a-b)
      print("co-body", r, s)
      return b, "end"
end)

print("main", coroutine.resume(co, 1, 10))
print("main", coroutine.resume(co, "r"))
print("main", coroutine.resume(co, "x", "y"))
print("main", coroutine.resume(co, "x", "y"))
```

Quando se executa esse código, ele produz o seguinte resultado:

```
co-body 1       10
foo     2
main    true    4
co-body r
main    true    11      -9
co-body x       y
main    true    10      end
main    false   cannot resume dead coroutine
```

B.3 Interface de programação da aplicação (API)

B.3.1 Conceitos básicos

Todas as funções da API, bem como os tipos e constantes relacionados, estão declarados no arquivo de cabeçalho lua.h.

Mesmo quando usamos o termo "função", qualquer operação na API pode, de forma alternativa, ser provida como uma macro. Tais macros usam cada um dos seus argumentos exatamente uma vez (com exceção do primeiro argumento, que é sempre um estado Lua) e portanto não geram qualquer efeito colateral oculto.

Como na maioria das bibliotecas C, as funções da API Lua não verificam a validade ou a consistência dos seus argumentos. Contudo, é possível mudar este comportamento compilando Lua com uma definição apropriada para a macro luai_apicheck, no arquivo luaconf.h.

B.3.2 Pilha

Lua usa uma *pilha virtual* para passar e receber valores de C. Cada elemento nesta pilha representa um valor Lua (**nil**, um número, uma cadeia de caracteres etc.).

Sempre que Lua chama C, a função chamada recebe uma nova pilha, que é independente de pilhas anteriores e de pilhas de funções C que ainda estejam ativas. Esta pilha contém inicialmente quaisquer argumentos para a função C e é onde a função C empilha os seus resultados para serem retornados ao chamador (ver *lua_CFunction*).

Por conveniência, a maioria das operações de consulta na API não segue uma disciplina estrita de pilha. Ao invés disso, elas podem se referir a qualquer elemento na pilha usando um índice: Um índice positivo representa uma posição *absoluta* na pilha (começando em 1); um índice negativo representa uma posição relativa ao topo da pilha. De maneira mais específica, se a pilha possui *n* elementos, então o índice 1 representa o primeiro elemento (isto é, o elemento que foi empilhado na pilha primeiro) e o índice *n* representa o último elemento; o índice -1 também representa o último elemento (isto é, o elemento no topo) e o índice *-n* representa o primeiro elemento. Diz-se que um índice é *válido* se ele está entre 1 e o topo da pilha (isto é, se 1 ≤ abs(índice) ≤ topo).

B.3.3 Tamanho da pilha

Quando se interage com a API de Lua, se é responsável por assegurar consistência. Em particular, se é responsável por controlar estouro da pilha. Pode-se usar a função *lua_checkstack* para aumentar o tamanho da pilha.

Sempre que Lua chama C, ela assegura que pelo menos LUA_MINSTACK posições na pilha estão disponíveis. LUA_MINSTACK é definida como 20, então geralmente não é preciso se preocupar com o espaço da pilha a menos que o seu código possua laços empilhando elementos na pilha.

A maioria das funções de consulta aceita como índices qualquer valor dentro do espaço da pilha disponível, isto é, índices até o tamanho máximo da pilha que se configurou através da função *lua_checkstack*. Tais índices são chamados índices aceitáveis. Mais formalmente, definimos um índice aceitável como a seguir:

```
(índice < 0 && abs(índice) <= topo) ||
(índice > 0 && índice <= espaçodapilha)
```

Observar que 0 nunca é um índice aceitável.

B.3.4 Pseudo-índices

A menos que seja dito o contrário, qualquer função que aceita índices válidos pode também ser chamada com *pseudo-índices*, que representam alguns valores Lua que são acessíveis para o código C mas que não estão na pilha. Pseudo-índices são usados para acessar o ambiente do fluxo de execução, o ambiente da função, o registro e os upvalues da função C (ver B.3.5).

O ambiente do fluxo de execução (onde as variáveis globais existem) está sempre no pseudo-índice LUA_GLOBALSINDEX. O ambiente da função C rodando está sempre no pseudo-índice LUA_ENVIRONINDEX.

Para acessar e mudar o valor de variáveis globais, pode-se usar operações de tabelas usuais sobre uma tabela de ambiente. Por exemplo, para acessar o valor de uma variável global, faça

```
lua_getfield(L, LUA_GLOBALSINDEX, varname);
```

B.3.5 Fechos C

Quando uma função C é criada, é possível associar alguns valores a ela, criando então um fecho C; estes valores são chamados de *upvalues* e são acessíveis para a função sempre que ela é chamada (ver *lua_pushcclosure*).

Sempre que uma função C é chamada, seus upvalues são posicionados em pseudo-índices específicos. Estes pseudo-índices são gerados pela macro lua_upvalueindex. O primeiro valor associado com uma função está na posição lua_upvalueindex(1), e assim por diante. Qualquer acesso a lua_upvalueindex(*n*), onde *n* é maior do que o número de upvalues da função atual, produz um índice aceitável (embora inválido).

B.3.6 Registro

Lua provê um registro, uma tabela pré-definida que pode ser usada por qualquer código C para armazenar qualquer valor Lua que o código C precise armazenar. Esta tabela está sempre localizada no pseudo-índice LUA_REGISTRYINDEX. Qualquer biblioteca de C pode armazenar dados nesta tabela, mas ela deve tomar cuidado para escolher chaves diferentes daquelas usadas por outras bibliotecas, para evitar colisões. Tipicamente, deve-se usar como chave uma cadeia de caracteres contendo o nome da sua biblioteca ou um objeto do tipo userdata leve com o endereço de um objeto C em seu código.

As chaves inteiras no registro são usadas pelo mecanismo de referência, implementado pela biblioteca auxiliar, e portanto não devem ser usadas para outros propósitos.

B.3.7 Tratamento de erros em C

Internamente, Lua usa o mecanismo de longjmp de C para tratar erros (pode-se também utilizar exceções se for utilizado; ver o arquivo luaconf.h.) Quando Lua se depara com qualquer erro (tais como erros de alocação de memória, erros de tipo, erros de sintaxe e erros de tempo de execução) ela dispara um erro; isto é, ela faz um desvio longo. Um ambiente protegido usa setjmp para estabelecer um ponto de recuperação; qualquer erro desvia o fluxo de execução para o ponto de recuperação ativado mais recentemente.

A maioria das funções na API pode disparar um erro, por exemplo devido a um erro de alocação de memória. A documentação para cada função indica se ela pode disparar erros.

Dentro de uma função C pode-se disparar um erro chamando *lua_error*.

B.3.8 Funções e tipos

Todas as funções e tipos da API C esão listadas a seguir em ordem alfabética. Cada função tem um indicador como este: [-o, +p, *x*]

O primeiro campo, o, representa quantos elementos a função desempilha da pilha. O segundo campo, p, indica quantos elementos a função empilha na pilha. (Qualquer função sempre empilha seus resultados depois de desempilhar seus argumentos.) Um campo na forma x|y significa que a função pode empilhar (ou desempilhar) x ou y elementos, dependendo da situação; uma marca de interrogação '?' significa que não podemos saber quantos elementos a função desempilha/empilha olhando somente os seus argumentos (e.g., o número de elementos pode depender do que está na pilha). O terceiro campo, x, diz se a funçao pode disparar erros: '-' significa que a função nunca dispara qualquer erro; 'm' significa que a função pode disparar um erro somente devido à falta de memória; 'e' significa que a função pode disparar outros tipos de erro; 'v' significa que a função pode disparar um erro de maneira proposital.

---

lua_Alloc

```
typedef void * (*lua_Alloc) (void *ud, void *ptr, size_t osize, size_t nsize);
```

O tipo da função de alocação de memória usada pelos estados Lua. A função de alocação deve prover uma funcionalidade similar à de realloc, mas não exatamente a mesma. Seus argumentos são ud, um ponteiro opaco passado para *lua_newstate*; ptr, um ponteiro para o bloco sendo alocado/realocado/liberado; osize, o tamanho original do bloco; e nsize, o novo tamanho do bloco. ptr é NULL se e somente se osize é zero. Quando nsize é zero, a função de alocação deve retornar NULL; se osize é diferente de zero, o bloco de memória apontado por ptr deve ser liberado. Quando nsize não é zero, a função de alocação retorna NULL se e somente se ela não pode alocar o tamanho do bloco requisitado. Quando nsize não é zero e osize é zero, a função de alocação deve comportar-se como malloc. Quando nsize e osize não são zero, a função de alocação comporta-se como realloc. Lua assume que a função de alocação nunca falha quando osize >= nsize.

Segue uma implementação simples para a função de alocação. Ela é usada na biblioteca auxiliar por *luaL_newstate*.

```
static void *l_alloc (void *ud, void *ptr, size_t osize, size_t nsize) {
  (void)ud;       /* not used */
  (void)osize;   /* not used */
  if (nsize == 0) {
    free(ptr);  /* ANSI requires that free(NULL) has no effect */
    return NULL;
  }
  else
    /* ANSI requires that realloc(NULL, size) == malloc(size) */
    return realloc(ptr, nsize);
}
```

Este código assume que free(NULL) não possui nenhum efeito e que realloc(NULL, size) é equivalente a malloc(size). ANSI C garante esses dois comportamentos.

lua_atpanic

```
lua_CFunction lua_atpanic (lua_State *L, lua_CFunction panicf);
```

Estabelece uma nova função de pânico e retorna a função de pânico antiga.

Se um erro ocorre fora de qualquer ambiente protegido, Lua chama uma função de pânico e então chama exit(EXIT_FAILURE), terminando então a aplicação hospedeira. A sua função de pânico pode evitar esta saída caso ela nunca retorne (e.g., fazendo uma desvio longo).

A função de pânico pode acessar a mensagem de erro no topo da pilha.

lua_call

```
void lua_call (lua_State *L, int nargs, int nresults);
```

Chama uma função.

Para chamar uma função deve-se usar o seguinte protocolo: primeiro, a função a ser chamada é empilhada na pilha; em seguida, os argumentos da função são empilhados em ordem direta; isto é, o primeiro argumento é empilhado primeiro. Por último chama-se *lua_call*; nargs é o número de argumentos que se empilhou na pilha. Todos os argumentos e o valor da função são desempilhados da pilha quando a função é chamada. Os resultados da função são empilhados na pilha quando a função retorna. O número de resultados é ajustado para nresults, a menos que nresults seja LUA_MULTRET. Neste caso, *todos* os resultados da função são empilhados. Lua cuida para que os valores retornados caibam dentro do espaço da pilha. Os resultados da função são empilhados na pilha em ordem direta (o primeiro resultado é empilhado primeiro), de modo que depois da chamada o último resultado está no topo da pilha.

Qualquer erro dentro da função chamada é propagado para cima (com um longjmp).

O seguinte exemplo mostra como o programa hospedeiro pode fazer o equivalente a este código Lua:

```
a = f("how", t.x, 14)
```

Segue o mesmo código em C:

```
lua_getfield(L, LUA_GLOBALSINDEX, "f");  /* função a ser chamada */
lua_pushstring(L, "how");                  /* primeiro argumento */
lua_getfield(L, LUA_GLOBALSINDEX, "t"); /* tabela a ser indexada */
lua_getfield(L, -1, "x"); /* empilha o resultado de t.x (2º arg) */
lua_remove(L, -2);                        /* remove 't' da pilha */
lua_pushinteger(L, 14);                          /* 3º argumento */
lua_call(L, 3, 1);   /* chama 'f' com 3 argumentos e 1 resultado */
lua_setfield(L, LUA_GLOBALSINDEX, "a"); /* estabelece 'a' global */
```

O código acima é "balanceado": ao seu final, a pilha está de volta à sua configuração original. Isto é considerado uma boa prática de programação.

lua_CFunction

```
typedef int (*lua_CFunction) (lua_State *L);
```

O tipo para funções C.

A fim de se comunicar apropriadamente com Lua, uma função C deve usar o seguinte protocolo, o qual define o modo como parâmetros e resultados são passados: uma função C recebe seus argumentos de Lua na sua pilha em ordem direta (o primeiro argumento é empilhado primeiro). Portanto, quando a função inicia, lua_gettop(L) retorna o número de argumentos recebidos pela função. O primeiro argumento (se houver) está no índice 1 e seu último argumento está no índice lua_gettop(L). Para retornar valores para Lua, uma função C apenas os empilha na pilha, em ordem direta (o primeiro resultado é empilhado primeiro) e retorna o número de resultados. Qualquer outro valor na pilha abaixo dos resultados será devidamente descartado por Lua. Como uma função Lua, uma função C chamada por Lua também pode retornar muitos resultados.

Como um exemplo, a seguinte função recebe um número variável de argumentos numéricos e retorna a média e a soma deles:

```
static int foo (lua_State *L) {
  int n = lua_gettop(L);                    /* number of arguments */
  lua_Number sum = 0;
  int i;
  for (i = 1; i <= n; i++) {
    if (!lua_isnumber(L, i)) {
      lua_pushstring(L,"incorrect argument to function `average'");
      lua_error(L);
   }
   sum += lua_tonumber(L, i);
  }
  lua_pushnumber(L, sum/n);                 /* first result */
  lua_pushnumber(L, sum);                   /* second result */
  return 2;                                 /* number of results */
}
```

---

lua_checkstack

```
int lua_checkstack (lua_State *L, int extra);
```

Garante que existem pelo menos extra posições disponíveis na pilha. A função retorna falso se ela não puder aumentar o tamanho da pilha para o tamanho desejado. Esta função nunca comprime a pilha; se a pilha já é maior do que o novo tamanho, ela não terá o seu tamanho modificado.

---

lua_close

```
void lua_close (lua_State *L);
```

Destrói todos os objetos no estado Lua fornecido (chamando os meta-métodos de coleta de lixo correspondentes, se houver) e libera toda a memória dinâmica usada por aquele estado. Em várias plataformas, pode não ser necessário chamar esta função, porque todos os recursos são naturalmente liberados quando o programa hospedeiro morre. Por outro lado, programas que ficam rodando por muito tempo, como um *daemon* ou um servidor web, podem precisar liberar estados tão logo eles não sejam mais necessários, para evitar um crescimento demasiado do uso da memória.

---

lua_concat

```
void lua_concat (lua_State *L, int n);
```

Concatena os n valores no topo da pilha, desempilha-os e deixa o resultado no topo da pilha. Se n é 1, o resultado é o único valor na pilha (isto é, a função não faz nada); se n é 0, o resultado é a cadeia de caracteres vazia. A concatenação é realizada de acordo com a semântica usual de Lua (ver B.2.6.5).

---

lua_cpcall

```
int lua_cpcall (lua_State *L, lua_CFunction func, void *ud);
```

Chama a função C `func` em modo protegido. `func` inicia somente com um único elemento na sua pilha, o objeto userdata leve contendo ud. Em caso de erros, *`lua_cpcall`* retorna o mesmo código de erro de *`lua_pcall`*, mais o objeto de erro no topo da pilha; caso contrário, ela retorna zero e não muda a pilha. Todos os valores retornados por `func` são descartados.

---

lua_createtable

```
void lua_createtable (lua_State *L, int narr, int nrec);
```

Cria uma nova tabela vazia e a empilha no topo da pilha. A nova tabela possui espaço pré-alocado para `narr` elementos array e `nrec` elementos não-array. Esta pré-alocação é útil quando se sabe exatamente quantos elementos a tabela irá ter. Caso contrário pode-se usar a função *`lua_newtable`*.

---

lua_dump

```
int lua_dump (lua_State *L, lua_Writer writer, void *data);
```

Descarrega uma função como um trecho de código binário. Recebe um função Lua no topo da pilha e produz um trecho de código binário que, se carregado novamente, resulta em uma função equivalente àquela que foi descarregada. Para produzir partes do trecho de código, *lua_dump* chama a função writer (ver *lua_Writer*) com o argumento data fornecido para escrevê-los.

O valor retornado é o código de erro retornado pela última chamada à função writer; 0 significa que não ocorreram erros.

Esta função não desempilha a função Lua da pilha.

---

lua_equal

```
int lua_equal (lua_State *L, int index1, int index2);
```

Retorna 1 se os dois valores nos índices aceitáveis index1 e index2 são iguais, seguindo a semântica do operador == de Lua (ou seja, pode chamar meta-métodos). Caso contrário retorna 0. Também retorna 0 se qualquer um dos índices não é válido.

---

lua_error

```
int lua_error (lua_State *L);
```

Gera um erro Lua. A mensagem de erro (que pode ser de fato um valor Lua de qualquer tipo) deve estar no topo da pilha. Esta função faz um desvio longo e portanto nunca retorna. (ver *luaL_error*).

lua_gc

```
int lua_gc (lua_State *L, int what, int data);
```

Controla o coletor de lixo.

Essa função executa várias tarefas, de acordo com o valor do parâmetro *what*:

- **LUA_GCSTOP:** pára o coletor de lixo.
- **LUA_GCRESTART:** reinicia o coletor de lixo.
- **LUA_GCCOLLECT:** realiza um ciclo completo de coleta de lixo.
- **LUA_GCCOUNT:** retorna a quantidade de memória (em Kbytes) que está sendo usada correntemente por Lua.
- **LUA_GCCOUNTB:** retorna o resto da divisão da quantidade de bytes de memória usada correntemente por Lua por 1024.
- **LUA_GCSTEP:** realiza um passo incremental de coleta de lixo. O "tamanho" do passo é controlado por `data` (valores maiores significam mais passos) de maneira não especificada. Se for desejado controlar o tamanho do passo, deve-se ajustar de maneira experimental o valor de `data`. A função retorna 1 se o passo finalizou um ciclo de coleta de lixo
- **LUA_GCSETPAUSE:** estabelece `data`/100 como o novo valor para a *pausa* do coletor (ver B.2.11). A função retorna o valor anterior da pausa.
- **LUA_GCSETSTEPMUL:** estabelece `data`/100 como o novo valor para o *multiplicador de passo* do coletor (ver B.2.11). A função retorna o valor anterior do multiplicador de passo.

lua_getallocf

```
lua_Alloc lua_getallocf (lua_State *L, void **ud);
```

Retorna a função de alocação de memória de um dado estado. Se ud não é NULL, Lua armazena em *ud o ponteiro opaco passado para *lua_newstate*.

lua_getfenv

```
void lua_getfenv (lua_State *L, int index);
```

Coloca na pilha a tabela de ambiente do valor no índice fornecido.

lua_getfield

```
void lua_getfield (lua_State *L, int index, const char *k);
```

Coloca na pilha o valor `t[k]`, onde `t` é o valor no índice válido fornecido. Como em Lua, esta função pode disparar um meta-método para o evento "index" (ver B.2.9).

lua_getglobal

```
void lua_getglobal (lua_State *L, const char *name);
```

Coloca na pilha o valor da global name. Esta função é definida como uma macro:

```
#define lua_getglobal(L,s)  lua_getfield(L, LUA_GLOBALSINDEX, s)
```

lua_getmetatable

```
int lua_getmetatable (lua_State *L, int index);
```

Coloca na pilha a meta-tabela do valor no índice aceitável fornecido. Se o índice não é válido ou se o valor não possui uma meta-tabela, a função retorna 0 e não coloca nada na pilha.

lua_gettable

```
void lua_gettable (lua_State *L, int index);
```

Coloca na pilha o valor t[k], onde t é o valor no índice válido fornecido e k é o valor no topo da pilha.

Esta função desempilha a chave 'k' (colocando o resultado no seu lugar). Como em Lua, esta função pode disparar um meta-método para o evento "index" (ver B.2.9).

lua_gettop

```
int lua_gettop (lua_State *L);
```

Retorna o índice do elemento no topo da pilha. Visto que os índices começam em 1, este resultado é igual ao número de elementos na pilha (e portanto 0 significa uma pilha vazia).

lua_insert

```
void lua_insert (lua_State *L, int index);
```

Move o elemento no topo para o índice válido fornecido, deslocando os elementos acima deste índice para abrir espaço. Esta função não pode ser chamada com um pseudo-índice, porque um pseudo-índice não é uma posição real da pilha.

lua_Integer

```
typedef ptrdiff_t lua_Integer;
```

O tipo usado pela API Lua para representar valores inteiros.

O tipo padrão é um ptrdiff_t, que é usualmente o maior tipo inteiro com sinal que a máquina manipula "confortavelmente".

lua_isboolean

```
int lua_isboolean (lua_State *L, int index);
```

Retorna 1 se o valor no índice aceitável fornecido possui tipo booleano e 0 caso contrário.

lua_iscfunction

```
int lua_iscfunction (lua_State *L, int index);
```

Retorna 1 se o valor no índice aceitável fornecido é uma função C e 0 caso contrário.

lua_isfunction

```
int lua_isfunction (lua_State *L, int index);
```

Retorna 1 se o valor no índice aceitável fornecido é uma função (C ou Lua) e 0 caso contrário.

lua_islightuserdata

```
int lua_islightuserdata (lua_State *L, int index);
```

Retorna 1 se o valor no índice aceitável fornecido é um objeto userdata leve e 0 caso contrário.

lua_isnil

```
int lua_isnil (lua_State *L, int index);
```

Retorna 1 se o valor no índice aceitável fornecido é **nil** e 0 caso contrário.

lua_isnone

```
int lua_isnone (lua_State *L, int index);
```

Retorna 1 se o índice aceitável fornecido não é válido (isto é, se ele se refere a um elemento fora do espaço da pilha corrente) e 0 caso contrário.

lua_isnoneornil

```
int lua_isnoneornil (lua_State *L, int index);
```

Retorna 1 se o índice aceitável fornecido não é válido (isto é, se ele se refere a um elemento fora do espaço da pilha corrente) ou se o valor neste índice é **nil** e 0 caso contrário.

lua_isnumber

```
int lua_isnumber (lua_State *L, int index);
```

Retorna 1 se o valor no índice aceitável fornecido é um número ou uma cadeia de caracteres que pode ser convertida para um número e 0 caso contrário.

lua_isstring

```
int lua_isstring (lua_State *L, int index);
```

Retorna 1 se o valor no índice aceitável fornecido é uma cadeia de caracteres ou um número (o qual sempre pode ser convertido para uma cadeia) e 0 caso contrário.

---

lua_istable

```
int lua_istable (lua_State *L, int index);
```

Retorna 1 se o valor no índice aceitável fornecido é uma tabela e 0 caso contrário.

---

lua_isthread

```
int lua_isthread (lua_State *L, int index);
```

Retorna 1 se o valor no índice aceitável fornecido é do tipo thread e 0 caso contrário.

---

lua_isuserdata

```
int lua_isuserdata (lua_State *L, int index);
```

Retorna 1 se o valor no índice aceitável fornecido é um objeto userdata (completo ou leve) e 0 caso contrário.

---

lua_lessthan

```
int lua_lessthan (lua_State *L, int index1, int index2);
```

Retorna 1 se o valor no índice aceitável index1 é menor do que o valor no índice aceitável index2, seguindo a semântica do operador < de Lua (ou seja, pode chamar meta-métodos). Caso contrário retorna 0. Também retorna 0 se qualquer um dos índices não for válido.

---

lua_load

```
int lua_load (lua_State *L, lua_Reader reader, void *data, const char *chunkname);
```

Carrega um trecho de código Lua. Se não ocorrer nenhum erro, *lua_load* empilha o trecho compilado como uma função Lua no topo da pilha. Caso contrário, empilha uma mensagem de erro. Os valores de retorno de *lua_load* são:

- 0 --- sem erros;
- LUA_ERRSYNTAX --- erro de sintaxe durante a pré-compilação.
- LUA_ERRMEM --- erro de alocação de memória.

Esta função somente carrega um trecho; ela não o executa.

lua_load automaticamente detecta se o trecho está na forma de texto ou na forma binária e o carrega de maneira correta (ver o programa luac).

A função *lua_load* usa uma função reader fornecida pelo usuário para ler o trecho de código (ver *lua_Reader*). O argumento data é um valor opaco passado para a função de leitura.

O argumento chunkname dá um nome ao trecho, o qual é usado para mensagens de erro e em informações de depuração (ver B.3.9).

lua_newstate

```
lua_State *lua_newstate (lua_Alloc f, void *ud);
```

Cria um estado novo independente. Retorna NULL se não puder criar o estado (devido à falta de memória). O argumento f é a função de alocação; Lua faz toda a alocação de memória para este estado através desta função. O segundo argumento, ud, é um ponteiro opaco que Lua simplesmente passa para a função de alocação a cada chamada.

lua_newtable

```
void lua_newtable (lua_State *L);
```

Cria uma nova tabela vazia e a coloca na pilha. É equivalente a lua_createtable(L, 0, 0).

lua_newthread

```
lua_State *lua_newthread (lua_State *L);
```

Cria um novo objeto do tipo thread, coloca-o na pilha e retorna um ponteiro para um *lua_State* que representa este novo fluxo de execução. O novo fluxo de execução retornado por esta função compartilha todos os objetos globais (tais como tabelas) com o estado original, mas possui uma pilha de execução independente.

Não há uma função explícita para terminar ou destruir um fluxo de execução. Objetos do tipo thread estão sujeitos à coleta de lixo, assim como qualquer outro objeto de Lua.

lua_newuserdata

```
void *lua_newuserdata (lua_State *L, size_t size);
```

Esta função aloca um novo bloco de memória com o tamanho fornecido, coloca na pilha um novo objeto userdata completo com o endereço do bloco e retorna este endereço.

Objetos userdata representam valores C em Lua. Um *userdata completo* representa um bloco de memória. Ele é um objeto (assim como uma tabela): deve-se criá-lo, ele pode ter sua própria meta-tabela e pode-se detectar quando ele está sendo coletado. Um objeto userdata completo somente é igual a ele mesmo (usando a igualdade primitiva, sem o uso de meta-métodos).

Quando Lua coleta um userdata completo com um meta-método gc, Lua chama o meta-método e marca o userdata como finalizado. Quando este userdata é coletado novamente então Lua libera sua memória correspondente.

lua_next

```
int lua_next (lua_State *L, int index);
```

Desempilha uma chave da pilha e empilha um par chave-valor da tabela no índice fornecido (o "próximo" par depois da chave fornecida). Se não há mais elementos na tabela, então *lua_next* retorna 0 (e não empilha nada).

Um percorrimento típico parece com este:

```
/* table is in the stack at index `t' */
lua_pushnil(L);  /* first key */
while (lua_next(L, t) != 0) {
  /* `key' is at index -2 and `value' at index -1 */
  printf("%s - %s\n",
    lua_typename(L, lua_type(L, -2)), lua_typename(L, lua_type(L, -1)));
  lua_pop(L, 1);  /* removes `value'; keeps `key' for next iteration */
}
```

Durante o percorrimento de uma tabela, não chamar *lua_tolstring* diretamente sobre uma chave, a menos que se saiba que a chave é realmente uma cadeia de carecteres. Lembrar que *lua_tolstring* altera o valor no índice fornecido; isto confunde a próxima chamada para *lua_next*.

---

lua_Number

```
typedef double lua_Number;
```

O tipo de números em Lua. Por padrão, ele é double, mas pode ser mudado em luaconf.h.

Através do arquivo de configuração é possível mudar Lua para operar com outro tipo para números (por exemplo, float ou long).

---

lua_objlen

```
size_t lua_objlen (lua_State *L, int index);
```

Retorna o "tamanho" do valor no índice aceitável fornecido: para cadeias de caracteres, isto é o tamanho da cadeia; para tabelas, isto é o resultado do operador de tamanho ('#'); para objetos do tipo userdata, isto é o tamanho do bloco de memória alocado para o userdata; para outros valores, o tamanho é 0.

---

lua_pcall

```
lua_pcall (lua_State *L, int nargs, int nresults, int errfunc);
```

Chama uma função em modo protegido.

Tanto nargs quanto nresults possuem o mesmo significado que possuíam em *lua_call*. Se não há erros durante a chamada, *lua_pcall* comporta-se exatamente como *lua_call*. Contudo, se há qualquer erro, *lua_pcall* o captura, coloca um único valor na pilha (a mensagem de erro) e retorna um código de erro. Como *lua_call*, *lua_pcall* sempre remove a função e seus argumentos da pilha.

Se errfunc é 0, então a mensagem de erro retornada na pilha é exatamente a mensagem de erro original. Caso contrário, errfunc é o índice na pilha de um função de tratamento de erros. (Na implementação atual, este índice não pode ser um pseudo-índice.) No caso de erros de tempo de execução, esta função será chamada com a mensagem de erro e seu valor de retorno será a mensagem retornada na pilha por *lua_pcall*.

Tipicamente, a função de tratamento de erros é usada para adicionar mais informação de depuração à mensagem de erro, como um traço da pilha. Tal informação não pode ser obtida após o retorno de *lua_pcall*, pois neste ponto a pilha já foi desfeita.

A função *lua_pcall* retorna 0 em caso de sucesso ou um dos seguintes códigos de erro (definidos em lua.h):

- LUA_ERRRUN: um erro em tempo de execução.
- LUA_ERRMEM: erro de alocação de memória. Para tais erros, Lua não chama a função de tratamento erros.
- LUA_ERRERR: erro durante a execução da função de tratamento de erros.

---

lua_pop

```
void lua_pop (lua_State *L, int n);
```

Desempilha n elementos da pilha.

---

lua_pushboolean

```
void lua_pushboolean (lua_State *L, int b);
```

Empilha um valor booleano com valor b na pilha.

---

lua_pushcclosure

```
void lua_pushcclosure (lua_State *L, lua_CFunction fn, int n);
```

Empilha um novo fecho C na pilha.

Quando uma função C é criada, é possível associar alguns valores a ela, criando então um fecho C (ver *§3.4*); estes valores são então acessíveis para a função sempre que ela é chamada. Para associar valores com uma função C, primeiro estes valores devem ser colocados na pilha (quando há múltiplos valores, o primeiro valor é empilhado primeiro). Então *lua_pushcclosure* é chamada para criar e colocar a função C na pilha, com o argumento n informando quantos valores devem ser associados com a função. *lua_pushcclosure* também desempilha estes valores da pilha.

---

lua_pushcfunction

```
void lua_pushcfunction (lua_State *L, lua_CFunction f);
```

Empilha uma função C na pilha. Esta função recebe um ponteiro para uma função C e coloca na pilha um valor Lua do tipo function que, quando chamado, invoca a função C correspondente.

Qualquer função para ser registrada em Lua deve seguir o protocolo correto para receber seus parâmetros e retornar seus resultados (ver *lua_CFunction*).

`lua_pushcfunction` é definida como uma macro:

```
#define lua_pushcfunction(L,f)  lua_pushcclosure(L,f,0)
```

---

lua_pushfstring

```
const char *lua_pushfstring (lua_State *L, const char *fmt, ...);
```

Coloca na pilha uma cadeia de caracteres formatada e retorna um ponteiro para esta cadeia. Ela é similar à função C sprintf, mas possui algumas diferenças importantes:

- não é preciso alocar espaço para o resultado: o resultado é uma cadeia de caracteres e Lua cuida da alocação de memória (e da desalocação, através da coleta de lixo);
- os especificadores de conversão são bastante restritos. Não há *flags*, tamanhos ou precisões. Os especificadores de conversão podem ser somente '%%' (insere um '%' na cadeia), '%s' (insere uma cadeia terminada por zero, sem restrições de tamanho), '%f' (insere um *lua_Number*), '%p' (insere um ponteiro como um número hexadecimal), '%d' (insere um int) e '%c' (insere um int como um caractere).

---

lua_pushinteger

```
void lua_pushinteger (lua_State *L, lua_Integer n);
```

Coloca um número com valor n na pilha.

---

lua_pushlightuserdata

```
void lua_pushlightuserdata (lua_State *L, void *p);
```

Coloca um objeto do tipo userdata leve na pilha.

Um userdata representa valores de C em Lua. Um *userdata* leve representa um ponteiro. Ele é um valor (como um número): não se pode criá-lo, ele não possui uma meta-tabela individual e ele não é coletado (uma vez que ele nunca foi criado). Um userdata leve é igual a "qualquer" userdata leve com o mesmo endereço C.

---

lua_pushlstring

```
void lua_pushlstring (lua_State *L, const char *s, size_t len);
```

Empilha a cadeia de caracteres apontada por s com tamanho len na pilha. Lua cria (ou reusa) uma cópia interna da cadeia fornecida, de forma que a memória apontada por s pode ser liberada ou reutilizada imediatamente após o retorno da função. A cadeia pode conter zeros dentro dela.

---

lua_pushnil

```
void lua_pushnil (lua_State *L);
```

Coloca um valor nil na pilha.

---

lua_pushnumber

```
void lua_pushnumber (lua_State *L, lua_Number n);
```

Coloca um número com valor n na pilha.

---

lua_pushstring

```
void lua_pushstring (lua_State *L, const char *s);
```

Empilha a cadeia terminada por zero apontada por s na pilha. Lua cria (ou reusa) uma cópia interna da cadeia fornecida, de forma que a memória apontada por s pode ser liberada ou reutilizada imediatamente após o retorno da função. A cadeia não pode conter zeros dentro dela; assume-se que a cadeia termina no primeiro zero.

---

lua_pushthread

```
void lua_pushthread (lua_State *L);
```

Empilha o fluxo de execução representado por L na pilha. Retorna 1 se este fluxo de execução é o fluxo de execução principal do seu estado.

---

lua_pushvalue

```
void lua_pushvalue (lua_State *L, int index);
```

Empilha uma cópia do elemento no índice válido fornecido na pilha.

---

lua_pushvfstring

```
const char *lua_pushvfstring (lua_State *L, const char *fmt,
                                va_list argp);
```

Equivalente a *lua_pushfstring*, exceto que esta função recebe uma va_list ao invés de um número variável de argumentos.

---

lua_rawequal

```
int lua_rawequal (lua_State *L, int index1, int index2);
```

Retorna 1 se os dois valores nos índices aceitáveis index1 e index2 são iguais primitivamente (isto é, sem fazer chamadas a meta-métodos). Caso contrário retorna 0. Também retorna 0 se qualquer um dos índices não for válido.

---

lua_rawget

```
void lua_rawget (lua_State *L, int index);
```

Similar a *lua_gettable*, mas faz um acesso primitivo (i.e., sem usar meta-métodos).

---

lua_rawgeti

```
void lua_rawgeti (lua_State *L, int index, int n);
```

Coloca na pilha o valor t[n], onde t é o valor no índice válido fornecido. O acesso é primitivo; isto é, ele não invoca meta-métodos.

lua_rawset

```
void lua_rawset (lua_State *L, int index);
```

Similar a *lua_settable*, mas faz uma atribuição primitiva (i.e., sem usar meta-métodos).

lua_rawseti

```
void lua_rawseti (lua_State *L, int index, int n);
```

Faz o equivalente a t[n] = v, onde t é o valor no índice válido fornecido e v é o valor no topo da pilha.

Esta função desempilha o valor da pilha. A atribuição é primitiva; isto é, ela não invoca meta-métodos.

lua_Reader

```
typedef const char * (*lua_Reader)
                     (lua_State *L, void *data, size_t *size);
```

A função de leitura usada por *lua_load*. Toda vez que ela precisa de outro pedaço do trecho, *lua_load* chama a função de leitura, passando junto o seu parâmetro data. A função de leitura deve retornar um ponteiro para um bloco de memória com um novo pedaço do trecho e atribuir a *size o tamanho do bloco. O bloco deve existir até que a função de leitura seja chamada novamente. Para sinalizar o fim do trecho, a função de leitura deve retornar NULL. A função de leitura pode retornar pedaços de qualquer tamanho maior do que zero.

lua_register

```
void lua_register (lua_State *L, const char *name, lua_CFunction f);
```

Estabelece a função C f como o novo valor da global name. Esta função é definida como uma macro:

```
#define lua_register(L,n,f)  (lua_pushcfunction(L, f), lua_setglobal(L, n))
```

lua_remove

```
void lua_remove (lua_State *L, int index);
```

Remove o elemento no índice válido fornecido, deslocando para baixo os elementos acima deste índice para preencher o buraco. Esta função não pode ser chamada com um pseudo-índice, visto que o pseudo-índice não é uma posição real da pilha.

lua_replace

```
void lua_replace (lua_State *L, int index);
```

Move o elemento do topo para a posição fornecida (e desempilha-o), sem deslocar qualquer elemento (substituindo portanto o valor na posição fornecida).

lua_resume

```
int lua_resume (lua_State *L, int narg);
```

Inicia e recomeça uma co-rotina em um fluxo de execução.

Para iniciar uma co-rotina, deve-se primeiro criar um novo fluxo de execução (ver *lua_newthread*); em seguida deve-se colocar na sua pilha a função principal mais quaisquer argumentos; por último chama-se *lua_resume*, com narg sendo o número de argumentos. Esta chamada retorna quando a co-rotina suspende ou finaliza sua execução. Quando ela retorna, a pilha contém todos os valores passados para *lua_yield* ou todos os valores retornados pelo corpo da função. *lua_resume* retorna *LUA_YIELD* se a co-rotina cede, 0 se a co-rotina termina sua execução sem erros ou um código de erro no caso de acontecerem erros (ver *lua_pcall*). No caso de erros, a pilha não é desfeita, de forma que pode-se usar a API de depuração sobre ela. A mensagem de erro está no topo da pilha. Para reiniciar uma co-rotina, deve-se colocar na pilha dela somente os valores a serem passados como resultados de yield e então chamar *lua_resume*.

---

lua_setallocf

```
void lua_setallocf (lua_State *L, lua_Alloc f, void *ud);
```

Muda a função de alocação de um dado estado para f com objeto userdata ud.

---

lua_setfenv

```
int lua_setfenv (lua_State *L, int index);
```

Desempilha uma tabela da pilha e estabelece esta tabela como sendo o novo ambiente para o valor no índice fornecido. Se o valor no índice fornecido não é nem uma função, nem um fluxo de execução e nem um objeto userdata, *lua_setfenv* retorna 0. Caso contrário a função retorna 1.

---

lua_setfield

```
void lua_setfield (lua_State *L, int index, const char *k);
```

Faz o equivalente a t[k] = v, onde t é o valor no índice válido fornecido e v é o valor no topo da pilha.

Esta função desempilha o valor da pilha. Como em Lua, esta função pode disparar um meta-método para o evento "newindex" (ver B.2.9).

---

lua_setglobal

```
void lua_setglobal (lua_State *L, const char *name);
```

Desempilha um valor da pilha e o estabelece como o novo valor da global name. Esta função é definida como uma macro:

```
#define lua_setglobal(L,s)   lua_setfield(L, LUA_GLOBALSINDEX, s)
```

---

lua_setmetatable

```
int lua_setmetatable (lua_State *L, int index);
```

Desempilha uma tabela da pilha e estabelece esta tabela como a nova meta-tabela para o valor no índice aceitável fornecido.

---

lua_settable

```
void lua_settable (lua_State *L, int index);
```

Faz o equivalente a t[k] = v, onde t é o valor no índice válido fornecido, v é o valor no topo da pilha e k é o valor logo abaixo do topo.

Esta função desempilha tanto a chave como o valor da pilha. Da mesma forma que em Lua, esta função pode disparar um meta-método para o evento "newindex" (ver B.2.9).

---

lua_settop

```
void lua_settop (lua_State *L, int index);
```

Aceita qualquer índice aceitável, ou 0, e estabelece este índice como o topo da pilha. Se o novo topo é maior do que o antigo, então os novos elementos são preenchidos com **nil**. Se index é 0, então todos os elementos da pilha são removidos.

---

lua_State

```
typedef struct lua_State lua_State;
```

Estrutura opaca que guarda o estado completo de um interpretador Lua. A biblioteca de Lua é totalmente reentrante: não existem variáveis globais. Toda a informação sobre um estado é mantida nesta estrutura.

Um ponteiro para este estado deve ser passado como o primeiro argumento para toda função na biblioteca, exceto para *lua_newstate*, que cria um novo estado Lua a partir do zero.

---

lua_status

```
int lua_status (lua_State *L);
```

Retorna o status do fluxo de execução L.

O status pode ser 0 para um fluxo de execução normal, um código de erro se o fluxo de execução terminou sua execução com um erro ou LUA_YIELD se o fluxo de execução está suspenso.

---

lua_toboolean

```
int lua_toboolean (lua_State *L, int index);
```

Converte um valor Lua no índice aceitável fornecido para um valor booleano C (0 ou 1). Como todos os testes em Lua, *lua_toboolean* retorna 1 para qualquer valor Lua diferente de **false** e de **nil**; caso contrário a função retorna 0. A função também retorna 0 quando chamada com um índice não válido (se for desejado aceitar somente valores booleanos de fato, usar *lua_isboolean* para testar o tipo do valor).

lua_tocfunction

```
lua_CFunction lua_tocfunction (lua_State *L, int index);
```

Converte um valor no índice aceitável fornecido para uma função C. Tal valor deve ser uma função C; caso contrário, retorna NULL.

lua_tointeger

```
lua_Integer lua_tointeger (lua_State *L, int idx);
```

Converte o valor Lua no índice aceitável fornecido para o tipo inteiro com sinal *lua_Integer*. O valor Lua deve ser um número ou uma cadeia que pode ser convertida para um número (ver B.2.3.2); caso contrário, *lua_tointeger* retorna 0.

Se o número não é um inteiro, ele é truncado de alguma maneira não especificada.

lua_tolstring

```
const char *lua_tolstring (lua_State *L, int index, size_t *len);
```

Converte o valor Lua no índice aceitável fornecido para uma cadeia C. Se len não é NULL, a função também estabelece *len como o tamanho da cadeia. O valor Lua deve ser uma cadeia de caracteres ou um número; caso contrário, a função retorna NULL. Se o valor é um número, então *lua_tolstring* também *muda o valor real na pilha para uma cadeia*. (Esta mudança confunde *lua_next* quando *lua_tolstring* é aplicada a chaves durante um percorrimento de tabela.)

lua_tolstring retorna um ponteiro totalmente alinhado para uma cadeia de caracteres dentro do estado Lua. Esta cadeia sempre tem um zero ('\0') após o seu último caractere (como em C), mas pode conter outros zeros no seu corpo. Visto que Lua possui coleta de lixo, não há garantia de que o ponteiro retornado por *lua_tolstring* será válido após o valor correspondente ser removido da pilha.

lua_tonumber

```
lua_Number lua_tonumber (lua_State *L, int index);
```

Converte o valor Lua no índice aceitável fornecido para o tipo C *lua_Number* (ver *lua_Number*). O valor Lua deve ser um número ou uma cadeia que pode ser convertida para um número (ver B.2.3.2); caso contrário, *lua_tonumber* retorna 0.

lua_topointer

```
const void *lua_topointer (lua_State *L, int index);
```

Converte o valor no índice aceitável fornecido para um ponteiro C genérico (void*). O valor pode ser um objeto userdata, uma tabela, um fluxo de execução ou uma função; objetos diferentes irão fornecer ponteiros diferentes. Não há maneira de converter o ponteiro de volta ao seu valor original.

Tipicamente esta função é usada somente para informações de depuração.

lua_tostring

```
const char *lua_tostring (lua_State *L, int index);
```

Equivalente a *lua_tolstring* com len sendo igual a NULL.

lua_tothread

```
lua_State *lua_tothread (lua_State *L, int index);
```

Converte o valor no índice aceitável fornecido para um fluxo de execução (representado como lua_State*). Este valor deve ser um fluxo de execução; caso contrário, a função retorna NULL.

lua_touserdata

```
void *lua_touserdata (lua_State *L, int index);
```

Se o valor no índice aceitável fornecido é um objeto userdata completo, a função retorna o endereço do seu bloco. Se o valor é um userdata leve, a função retorna seu ponteiro. Caso contrário, retorna NULL.

lua_type

```
int lua_type (lua_State *L, int index);
```

Retorna o tipo do valor no índice aceitável fornecido ou LUA_TNONE para um índice não válido (isto é, um índice para uma posição da pilha "vazia"). Os tipos retornados por *lua_type* são codificados pelas seguintes constantes definidas em lua.h: LUA_TNIL, LUA_TNUMBER, LUA_TBOOLEAN, LUA_TSTRING, LUA_TTABLE, LUA_TFUNCTION, LUA_TUSERDATA, LUA_TTHREAD e LUA_TLIGHTUSERDATA.

lua_typename

```
const char *lua_typename  (lua_State *L, int tp);
```

Retorna o nome do tipo codificado pelo valor tp, que deve ser um dos valores retornados por *lua_type*.

lua_Writer

```
typedef int (*lua_Writer) (lua_State *L, const void* p, size_t sz, void* ud);
```

O tipo da função de escrita usada por *lua_dump*. Toda vez que ela produz outro pedaço de trecho, *lua_dump* chama a função de escrita, passando junto o buffer a ser escrito (p), seu tamanho (sz) e o parâmetro data fornecido para *lua_dump*.

A função de escrita retorna um código de erro: 0 significa nenhum erro; qualquer outro valor significa um erro e faz *lua_dump* parar de chamar a função de escrita.

lua_xmove

```
void lua_xmove (lua_State *from, lua_State *to, int n);
```

Troca valores entre diferentes fluxos de execução do *mesmo* estado global.

Esta função desempilha n valores da pilha from e os empilha na pilha to.

---

lua_yield

```
int lua_yield  (lua_State *L, int nresults);
```

Cede uma co-rotina.

Esta função somente deve ser chamada como a expressão de retorno de uma função C, como a seguir:

```
return lua_yield (L, nresults);
```

Quando uma função C chama *lua_yield* desta maneira, a co-rotina sendo executada suspende a sua execução e a chamada a *lua_resume* que iniciou esta co-rotina retorna. O parâmetro results é o número de valores da pilha que são passados como resultados para *lua_resume*.

B.3.9 Interface de depuração

Lua não possui mecanismos de depuração pré-definidos. Ao invés disto, ela oferece uma interface especial por meio de funções e *ganchos*. Esta interface permite a construção de diferentes tipos de depuradores, medidores e outras ferramentas que necessitam de "informação interna" do interpretador.

---

lua_Debug

```
typedef struct lua_Debug {
  int event;
  const char *name;               /* (n) */
  const char *namewhat;           /* (n) */
  const char *what;               /* (S) */
  const char *source;             /* (S) */
  int currentline;                /* (l) */
  int nups;                       /* (u) number of upvalues */
  int linedefined;                /* (S) */
  int lastlinedefined;            /* (S) */
  char short_src[LUA_IDSIZE];     /* (S) */
  /* private part */
  ...
} lua_Debug;
```

Uma estrutura usada para guardar diferentes pedaços de informação sobre uma função ativa. *lua_getstack* preenche somente a parte privada desta estrutura, para uso posterior. Para preencher os outros campos de *lua_Debug* com informação útil, chame *lua_getinfo*.

Os campos de *lua_Debug* possuem o seguinte significado:

- **source:** se a função foi definida em uma cadeia de caracters, então source é essa cadeia. Se a função foi definida em um arquivo, então source inicia com um '@' seguido pelo nome do arquivo;
- **short_src:** uma versão "adequada" para impressão de source, para ser usada em mensagens de erro;
- **linedefined** o número da linha onde a definição da função começa;

- **lastlinedefined:** o número da linha onde a definição da função termina;
- **what:** a cadeia "Lua" se a função é uma função Lua, "C" se ela é uma função C, "main" se ela é a parte principal de um trecho e "tail" se ela foi uma função que fez uma recursão final. No último caso, Lua não possui nenhuma outra informação sobre a função;
- **currentline:** a linha corrente onde a função fornecida está executando. Quando nenhuma informação sobre a linha está disponível, atribui-se -1 a currentline;
- **name:** um nome razoável para a função fornecida. Dado que funções em Lua são valores de primeira classe, elas não possuem um nome fixo: algumas funções podem ser o valor de múltiplas variáveis globais, enquanto outras podem estar armazenadas somente em um campo de uma tabela. A função lua_getinfo verifica como a função foi chamada para encontrar um nome adequado. Se não é possível encontrar um nome, então atribui-se NULL a name;
- **namewhat:** explica o campo name. O valor de namewhat pode ser "global", "local", "method", "field", "upvalue" ou "" (a cadeia vazia), de acordo com como a função foi chamada (Lua usa a cadeia vazia quando nenhuma outra opção parece se aplicar);
- **nups:** o número de upvalues da função.

---

lua_gethook

```
lua_Hook lua_gethook (lua_State *L);
```

Retorna a função gancho atual.

---

lua_gethookcount

```
int lua_gethookcount (lua_State *L);
```

Retorna a contagem de gancho atual.

---

lua_gethookmask

```
int lua_gethookmask (lua_State *L);
```

Retorna a máscara de gancho atual.

---

lua_getinfo

```
int lua_getinfo (lua_State *L, const char *what, lua_Debug *ar);
```

Retorna informação sobre uma função específica ou uma invocação de função específica.

Para obter informação sobre uma invocação de função, o parâmetro ar deve ser um registro de ativação válido que foi preenchido por uma chamada anterior a *lua_getstack* ou foi fornecido como argumento para um gancho (ver *lua_Hook*).

Para obter informação sobre uma função deve-se colocá-la na pilha e iniciar a cadeia what com o caractere '>'. (Neste caso, lua_getinfo desempilha a função no topo da pilha.) Por exemplo, para saber em qual linha uma função f foi definida, pode-se escrever o seguinte código:

```
lua_Debug ar;
lua_getfield(L, LUA_GLOBALSINDEX, "f");  /* get global `f' */
lua_getinfo(L, ">S", &ar);
printf("%d\n", ar.linedefined);
```

Cada caractere na cadeia what seleciona alguns campos da estrutura ar para serem preenchidos ou um valor a ser empilhado na pilha:

- **'n':** preenche os campos name e namewhat;
- **'S':** preenche os campos source, short_src, linedefined, lastlinedefined e what;
- **'l':** preenche o campo currentline;
- **'u':** preenche o campo nups;
- **'f':** coloca na pilha a função que está executando no nível fornecido;
- **'L':** coloca na pilha uma tabela cujos índices são o número das linhas que são válidas na função (uma linha válida é uma linha com algum código associado, isto é, uma linha onde pode-se colocar um ponto de parada. Linhas não válidas incluem linhas vazias e comentários).

Esta função retorna 0 em caso de erro (por exemplo, no caso de uma opção inválida em what).

---

lua_getlocal

```
const char *lua_getlocal (lua_State *L, const lua_Debug *ar, int n);
```

Obtém informação sobre uma variável local de um registro de ativação fornecido. O parâmetro ar deve ser um registro de ativação válido que foi preenchido por uma chamada anterior a *lua_getstack* ou foi fornecido como um argumento para um gancho (ver *lua_Hook*). O índice n seleciona qual variável local inspecionar (1 é o primeiro parâmetro ou variável local ativa e assim por diante, até a última variável local ativa). *lua_getlocal* coloca o valor da variável na pilha e retorna o nome dela.

Nomes de variáveis começando com '(' (abre parênteses) representam variáveis internas (variáveis de controle de laços, temporários e funções C locais.).

Retorna NULL (e não empilha nada) quando o índice é maior do que o número de variáveis locais ativas.

---

lua_getstack

```
int lua_getstack (lua_State *L, int level, lua_Debug *ar);
```

Obtém informação sobre a pilha de tempo de execução do interpretador.

Esta função preenche partes de uma estrutura *lua_Debug* com uma identificação do registro de ativação da função executando em um dado nível. O nível 0 é a função executando atualmente, ao passo que o nível *n+1* é a função que chamou o nível *n*. Quando não há erros, *lua_getstack* retorna 1; quando chamada com um nível maior do que a profundidade da pilha, a função retorna 0.

---

lua_getupvalue

```
const char *lua_getupvalue (lua_State *L, int funcindex, int n);
```

Obtém informação sobre um upvalue de um fecho (para funções Lua, upvalues são variáveis locais externas que a função usa e que são conceitualmente incluídas no fecho dela). *lua_getupvalue* obtém o índice n de um upvalue, coloca o valor do upvalue na pilha e retorna o nome dele. funcindex aponta para o fecho na pilha. (Upvalues não possuem uma ordem específica, uma vez que eles são ativos ao longo de toda a função. Então, eles são numerados em uma ordem arbitrária.)

Retorna NULL (e não empilha nada) quando o índice é maior do que o número de upvalues. Para funções C, esta função usa a cadeia vazia "" como um nome para todos os upvalues.

---

lua_Hook

```
typedef void (*lua_Hook) (lua_State *L, lua_Debug *ar);
```

O tipo para funções de gancho de depuração.

Sempre que um gancho é chamado, atribui-se ao campo event de seu argumento ar o evento específico que disparou o gancho. Lua identifica estes eventos com a seguintes constantes: LUA_HOOKCALL, LUA_HOOKRET, LUA_HOOKTAILRET, LUA_HOOKLINE e LUA_HOOKCOUNT. Além disso, para eventos de linha, o campo currentline também é atribuído. Para obter o valor de qualquer campo em ar, o gancho deve chamar *lua_getinfo*. Para eventos de retorno, event pode ser LUA_HOOKRET, o valor normal, ou LUA_HOOKTAILRET. No último caso, Lua está simulando um retorno de uma função que fez uma recursão final; neste caso, é inútil chamar *lua_getinfo*.

Enquanto Lua está executando um gancho, ela desabilita outras chamadas a ganchos. Portanto, se um gancho chama Lua de volta para executar uma função ou um trecho, esta execução ocorre sem quaisquer chamadas a ganchos.

---

lua_sethook

```
int lua_sethook (lua_State *L, lua_Hook func, int mask,  int count);
```

Estabelece a função de gancho de depuração.

O argumento f é uma função de gancho. mask especifica sobre quais eventos o gancho será chamado: ele é formado por uma conjunção bit-a-bit das constantes LUA_MASKCALL, LUA_MASKRET, LUA_MASKLINE e LUA_MASKCOUNT. O argumento count somente possui significado quando a máscara inclui LUA_MASKCOUNT. Para cada evento, o gancho é chamado como explicado abaixo:

- **o gancho de chamada (CALL):** é chamado quando o interpretador chama uma função. O gancho é chamado logo após Lua entrar na nova função, antes da função receber seus argumentos;
- **o gancho de retorno (RET):** é chamado quando o interpretador retorna de uma função. O gancho é chamado logo após Lua sair da função. Não se tem acesso aos valores a serem retornados pela função;
- **o gancho de linha (LINE):** é chamado quando o interpretador está para iniciar a execução de uma nova linha de código ou quando ele volta atrás no código (mesmo que para a mesma linha) (este evento somente acontece quando Lua está executando uma função Lua;
- **o gancho do lua (COUNT):** é chamado após o interpretador executar cada uma das instruções count (este evento somente ocorre quando Lua está executando uma função Lua).

Um gancho é desabilitado atribuindo-se zero a mask.

lua_setlocal

```
const char *lua_setlocal (lua_State *L, const lua_Debug *ar, int n);
```

Estabelece o valor de uma variável local de um registro de ativação fornecido. Os parâmetros ar e n são como em *lua_getlocal* (ver *lua_getlocal*). *lua_setlocal* atribui o valor no topo da pilha à variável e retorna o nome dela. A função também desempilha o valor da pilha.

Retorna NULL (e não desempilha nada) quando o índice é maior do que o número de variáveis locais ativas.

lua_setupvalue

```
const char *lua_setupvalue (lua_State *L, int funcindex, int n);
```

Estabelece o valor de um upvalue de um fecho. A função atribui o valor no topo da pilha ao upvalue e retorna o nome dele. Ela também desempilha o valor da pilha. Os parâmetros funcindex e n são como na função *lua_getupvalue* (ver *lua_getupvalue*).

Retorna NULL (e não desempilha nada) quando o índice é maior do que o número de upvalues.

B.4 Biblioteca auxiliar

B.4.1 Conceitos básicos

A biblioteca auxiliar fornece várias funções convenientes para a interface de C com Lua. Enquanto a API básica fornece as funções primitivas para todas as interações entre C e Lua, a biblioteca auxiliar fornece funções de mais alto nível para algumas tarefas comuns.

Todas as funções da biblioteca auxiliar são definidas no arquivo de cabeçalho lauxlib.h e possuem um prefixo luaL_.

Todas as funções na biblioteca auxiliar são construídas sobre a API básica e portanto elas não oferecem nada que não possa ser feito com a API básica.

Várias funções na biblioteca auxiliar são usadas para verificar argumentos de funções C. O nome delas sempre é luaL_check* ou luaL_opt*. Todas essas funções disparam um erro se a verificação não é satisfeita. Visto que a mensagem de erro é formatada para argumentos (e.g., "bad argument #1"), não se deve usar estas funções para outros valores da pilha.

B.4.2 Funções e tipos

Todas as funções e tipos da biblioteca auxiliar são listadas a seguir em ordem alfabética.

luaL_addchar

```
void luaL_addchar (luaL_Buffer B, char c);
```

Adiciona o caractere c ao buffer B (ver *luaL_Buffer*).

luaL_addlstring

```
void luaL_addlstring (luaL_Buffer *B, const char *s, size_t l);
```

Adiciona a cadeia de caracteres apontada por s com tamanho l ao buffer B (ver *luaL_Buffer*). A cadeia pode conter zeros dentro dela.

luaL_addsize

```
void luaL_addsize (luaL_Buffer B, size_t n);
```

Adiciona ao buffer B (ver *luaL_Buffer*) uma cadeia de comprimento n copiada anteriormente para a área de buffer (ver *luaL_prepbuffer*).

luaL_addstring

```
void luaL_addstring (luaL_Buffer *B, const char *s);
```

Adiciona a cadeia terminada por 0 apontada por s ao buffer B (ver *luaL_Buffer*). A cadeia não pode conter zeros dentro dela.

luaL_addvalue

```
void luaL_addvalue (luaL_Buffer *B);
```

Adiciona o valor no topo da pilha ao buffer B (ver *luaL_Buffer*). Desempilha o valor.

Esta é a única função sobre buffers de cadeias que pode (e deve) ser chamada com um elemento extra na pilha, que é o valor a ser adicionado ao buffer.

luaL_argcheck

```
void luaL_argcheck (lua_State *L, int cond, int numarg, const char *extramsg);
```

Verifica se cond é verdadeira. Se não, dispara um erro com a seguinte mensagem, onde func é recuperada a partir da pilha de chamada:

```
bad argument #<narg> to <func> (<extramsg>)
```

luaL_argerror

```
int luaL_argerror (lua_State *L, int numarg, const char *extramsg);
```

Dispara um erro com a seguinte mensagem, onde func é recuperada a partir da pilha de chamada:

```
bad argument #<narg> to <func> (<extramsg>)
```

Esta função nunca retorna, mas é idiomático usá-la em funções C como return luaL_argerror(*args*).

luaL_Buffer

```
typedef struct luaL_Buffer luaL_Buffer;
```

O tipo para um *buffer* de cadeia de caracteres.

Um buffer de cadeia permite código C construir cadeias Lua pouco a pouco. O seu padrão de uso é o seguinte:

- primeiro declara-se uma variável b do tipo *luaL_Buffer*;
- em seguida inicializa-se a variável com uma chamada luaL_buffinit(L, &b);
- depois adiciona-se pedaços da cadeia ao buffer chamando qualquer uma das funções luaL_add*;
- termina-se fazendo uma chamada luaL_pushresult(&b). Esta chamada deixa a cadeia final no topo da pilha.

Durante essa operação normal, um buffer de cadeia usa um número variável de posições da pilha. Então, quando se está usando um buffer, não se deve assumir que sabe onde o topo da pilha está. Pode-se usar a pilha entre chamadas sucessivas às operações de buffer desde que este uso seja balanceado; isto é, quando se chama uma operação de buffer, a pilha está no mesmo nível em que ela estava imediatamente após a operação de buffer

anterior (a única exceção a esta regra é *luaL_addvalue*). Após chamar *luaL_pushresult* a pilha está de volta ao seu nível quando o buffer foi inicializado, mais a cadeia final no seu topo.

---

luaL_buffinit

```
void luaL_buffinit (lua_State *L, luaL_Buffer *B);
```

Inicializa um buffer B. Esta função não aloca qualquer espaço; o buffer deve ser declarado como uma variável (ver *luaL_Buffer*).

---

luaL_callmeta

```
int luaL_callmeta (lua_State *L, int obj, const char *met);
```

Chama um meta-método.

Se o objeto no índice obj possui uma meta-tabela e esta meta-tabela possui um campo e, esta função chama esse campo e passa o objeto como seu único argumento. Neste caso esta função retorna 1 e coloca na pilha o valor retornado pela chamada. Se não há meta-tabela ou meta-método, esta função retorna 0 (sem empilhar qualquer valor na pilha).

---

luaL_checkany

```
void luaL_checkany (lua_State *L, int narg);
```

Verifica se a função tem um argumento de qualquer tipo (incluindo **nil**) na posição narg.

---

luaL_checkint

```
int luaL_checkint (lua_State *L, int narg);
```

Verifica se o argumento narg da função é um número e retorna este número convertido para um int.

---

luaL_checkinteger

```
lua_Integer luaL_checkinteger (lua_State *L, int narg);
```

Verifica se o argumento narg da função é um número e retorna este número convertido para um *lua_Integer*.

---

luaL_checklong

```
long luaL_checklong (lua_State *L, int narg);
```

Verifica se o argumento narg da função é um número e retorna este número convertido para um long.

---

luaL_checklstring

```
const char *luaL_checklstring (lua_State *L, int narg, size_t *l);
```

Verifica se o argumento narg da função é uma cadeia e retorna esta cadeia; se l não é NULL preenche *l com o tamanho da cadeia.

luaL_checknumber

```
lua_Number luaL_checknumber (lua_State *L, int narg);
```

Verifica se o argumento narg da função é um número e retorna este número.

luaL_checkoption

```
int luaL_checkoption (lua_State *L, int narg, const char *def, const char *const
lst[]);
```

Verifica se o argumento narg da função é uma cadeia e procura por esta cadeia no array lst (o qual deve ser terminado por NULL). Retorna o índice no array onde a cadeia foi encontrada. Dispara um erro se o argumento não é uma cadeia ou se a cadeia não pôde ser encontrada.

Se def não é NULL, a função usa def como um valor padrão quando não há argumento narg ou se este argumento é **nil**.

Esta é uma função útil para mapear cadeias para enumerações de C. (A convenção usual em bibliotecas Lua é usar cadeias ao invés de números para selecionar opções.)

luaL_checkstack

```
void luaL_checkstack (lua_State *L, int sz, const char *msg);
```

Aumenta o tamanho da pilha para top + sz elementos, disparando um erro se a pilha não pode ser aumentada para aquele tamanho. msg é um texto adicional a ser colocado na mensagem de erro.

luaL_checkstring

```
const char *luaL_checkstring (lua_State *L, int narg);
```

Verifica se o argumento narg da função é uma cadeia e retorna esta cadeia.

luaL_checktype

```
void luaL_checktype (lua_State *L, int narg, int t);
```

Verifica se o argumento narg da função tem tipo t. Ver *lua_type* para a codificação de tipos para t.

luaL_checkudata

```
void *luaL_checkudata (lua_State *L, int narg, const char *tname);
```

Verifica se o argumento narg da função é um objeto userdata do tipo tname (ver *luaL_newmetatable*).

luaL_dofile

```
int luaL_dofile (lua_State *L, const char *filename);
```

Carrega e executa o arquivo fornecido. É definida como a seguinte macro:

```
(luaL_loadfile(L, filename) || lua_pcall(L, 0, LUA_MULTRET, 0))
```

A função retorna 0 se não há erros ou 1 em caso de erros.

luaL_dostring

```
void *luaL_checkudata (lua_State *L, int narg, const char *tname);
```

Carrega e executa a cadeia fornecida. É definida como a seguinte macro:

```
(luaL_loadstring(L, str) || lua_pcall(L, 0, LUA_MULTRET, 0))
```

A função retorna 0 se não há erros ou 1 em caso de erros.

luaL_error

```
int luaL_error (lua_State *L, const char *fmt, ...);
```

Dispara um erro. O formato da mensagem de erro é dado por fmt mais quaisquer argumentos extras, seguindo as mesmas regras de *lua_pushfstring*. Também adiciona no início da mensagem o nome do arquivo e o número da linha onde o erro ocorreu, caso esta informação esteja disponível.

Esta função nunca retorna, mas é idiomático usá-la em funções C como return luaL_error(*args*).

luaL_getmetafield

```
int luaL_getmetafield (lua_State *L, int obj, const char *met);
```

Coloca na pilha o campo e da meta-tabela do objeto no índice obj. Se o objeto não possui uma meta-tabela ou se a meta-tabela não possui este campo, retorna 0 e não empilha nada.

luaL_getmetatable

```
void luaL_getmetatable (lua_State *L, const char *tname);
```

Coloca na pilha a meta-tabela associada com o nome tname no registro (ver *luaL_newmetatable*).

luaL_gsub

```
const char *luaL_gsub (lua_State *L, const char *s, const char *p, const char *r);
```

Cria uma cópia da cadeia s substituindo qualquer ocorrência da cadeia p pela cadeia r. Coloca a cadeia resultante na pilha e a retorna.

luaL_loadbuffer

```
int luaL_loadbuffer (lua_State *L, const char *buff, size_t sz, const char *name);
```

Carrega um buffer como um trecho de código Lua. Esta função usa *lua_load* para carregar o trecho no buffer apontado por buff com tamanho sz.

Esta função retorna os mesmos resultados de *lua_load*. name é o nome do trecho, usado para informações de depuração e mensagens de erro.

luaL_loadfile

```
int luaL_loadfile (lua_State *L, const char *filename);
```

Carrega um arquivo como um trecho de código Lua. Esta função usa *lua_load* para carregar o trecho no arquivo chamado filename. Se filename é NULL, então ela carrega a partir da entrada padrão. A primeira linha no arquivo é ignorada se ela começa com #.

Esta função retorna os mesmos resultados de *lua_load*, mas ela possui um código de erro extra LUA_ERRFILE se ela não pode abrir/ler o arquivo.

Da mesma forma que *lua_load*, esta função somente carrega o trecho; ela não o executa.

luaL_loadstring

```
int luaL_loadstring (lua_State *L, const char *s);
```

Carrega uma cadeia como um trecho de código Lua. Esta função usa *lua_load* para carregar o trecho na cadeia (terminada por zero) s.

Esta função retorna os mesmos resultados de *lua_load*.

Assim como *lua_load*, esta função somente carrega o trecho; ela não o executa.

luaL_newmetatable

```
int luaL_newmetatable (lua_State *L, const char *tname);
```

Se o registro já possui a chava tname, retorna 0. Caso contrário, cria uma nova tabela para ser usada como uma meta-tabela para o objeto userdata, adiciona esta tabela ao registro com chave tname e retorna 1.

Em ambos os casos coloca na pilha o valor final associado com tname no registro.

luaL_newstate

```
lua_State *luaL_newstate (void);
```

Cria um novo estado Lua. Chama *lua_newstate* com uma função de alocação baseada na função padrão de C realloc e então estabelece uma função de pânico (ver *lua_atpanic*) que imprime uma mensagem de erro para a saída de erro padrão em caso de erros fatais.

Retorna o novo estado ou NULL se ocorreu um erro de alocação de memória.

---

luaL_openlibs

```
void luaL_openlibs (lua_State *L);
```

Abre todas as bibliotecas padrões no estado fornecido.

---

luaL_optint

```
int luaL_optint (lua_State *L, int narg, int d);
```

Se o argumento narg da função é um número, retorna este número convertido para um int. Se este argumento está ausente ou se ele é **nil**, retorna d. Caso contrário, dispara um erro.

---

luaL_optinteger

```
lua_Integer luaL_optinteger (lua_State *L, int narg, lua_Integer d);
```

Se o argumento narg da função é um número, retorna este número convertido para um *lua_Integer*. Se este argumento está ausente ou se ele é **nil**, retorna d. Caso contrário, dispara um erro.

---

luaL_optlong

```
long luaL_optlong (lua_State *L, int narg, long d);
```

Se o argumento narg da função é um número, retorna este número convertido para um long. Se este argumento está ausente ou se ele é **nil**, retorna d. Caso contrário, dispara um erro.

---

luaL_optlstring

```
const char *luaL_optlstring (lua_State *L, int narg, const char *d, size_t *l);
```

Se o argumento narg da função é uma cadeia, retorna esta cadeia. Se este argumento está ausente ou se ele **nil**, retorna d. Caso contrário, dispara um erro.

Se l não é NULL, preenche a posição *l com o tamanho do resultado.

---

luaL_optnumber

```
lua_Number luaL_optnumber (lua_State *L, int narg, lua_Number d);
```

Se o argumento narg da função é um número, retorna este número. Se este argumento está ausente ou se ele é **nil**, retorna d. Caso contrário, dispara um erro.

luaL_optstring

```
const char *luaL_optstring (lua_State *L, int narg, const char *d);
```

Se o argumento narg da função é uma cadeia, retorna esta cadeia. Se este argumento está ausente ou se ele é **nil**, retorna d. Caso contrário, dispara um erro.

luaL_prepbuffer

```
char *luaL_prepbuffer (luaL_Buffer *B);
```

Retorna um endereço para um espaço de tamanho LUAL_BUFFERSIZE onde se pode copiar uma cadeia para ser adicionada ao buffer B (ver *luaL_Buffer*). Após copiar a cadeia para este espaço deve-se chamar *luaL_addsize* com o tamanho da cadeia para adicioná-la realmente ao buffer.

luaL_pushresult

```
void luaL_pushresult (luaL_Buffer *B);
```

Finaliza o uso do buffer B deixando a cadeia final no topo da pilha.

luaL_ref

```
int luaL_ref (lua_State *L, int t);
```

Cria e retorna uma *referência*, na tabela no índice t, para o objeto no topo da pilha (e desempilha o objeto).

Uma referência é uma chave inteira única. Desde que não se adicione manualmente chaves inteiras na tabela t, *luaL_ref* garante a unicidade da chave que ela retorna. Pode-se recuperar um objeto referenciado pelo referência r chamando lua_rawgeti(L, t, r). A função *luaL_unref* libera uma referência e o objeto associado a ela.

Se o objeto no topo da pilha é **nil**, *luaL_ref* retorna a constante LUA_REFNIL. A constante LUA_NOREF é garantidamente diferente de qualquer referência retornada por *luaL_ref*.

luaL_Reg

```
typedef struct luaL_Reg {
          const char *name;
          lua_CFunction func;
        } luaL_Reg;
```

O tipo para arrays de funções a serem registrados por *luaL_register*. name é o nome da função e func é um ponteiro para a função. Qualquer array de *luaL_Reg* deve terminar com uma entrada sentinela na qual tanto name como func são NULL.

luaL_register

```
void luaL_register (lua_State *L, const char *libname, const luaL_Reg *l);
```

Abre uma biblioteca.

Quando chamada com libname igual a NULL, simplesmente registra todas as funções na lista l (ver *luaL_Reg*) na tabela no topo da pilha.

Quando chamada com um valor de libname diferente de NULL, luaL_register cria uma nova tabela t, estabelece ela como o valor da variável global libname, estabelece ela como o valor de package.loaded[libname] e registra nela todas as funções na lista l. Se existe uma tabela em package.loaded[libname] ou na variável libname, a função reusa esta tabela ao invés de criar uma nova.

Em qualquer caso a função deixa a tabela no topo da pilha.

---

luaL_typename

```
const char *luaL_typename (lua_State *L, int idx);
```

Retorna o nome do tipo do valor no índice fornecido.

---

luaL_typerror

```
int luaL_typerror (lua_State *L, int narg, const char *tname);
```

Gera um erro com uma mensagem como a seguinte:

```
location: bad argument narg to 'func' (tname expected, got rt)
```

onde *location* é produzida por *luaL_where*, *func* é o nome da função corrente e *rt* é o nome do tipo do argumento.

---

luaL_unref

```
void luaL_unref (lua_State *L, int t, int ref);
```

Libera a referência ref da tabela no índice t (ver luaL_ref). A entrada é removida da tabela, de modo que o objeto mencionado pode ser coletado. A referência ref também é liberada para ser usada novamente.

Se ref for LUA_NOREF ou LUA_REFNIL, luaL_unref não faz nada.

---

luaL_where

```
void luaL_where (lua_State *L, int lvl);
```

Coloca na pilha uma cadeia identificando a posição atual do controle no nível lvl na pilha de chamada. Tipicamente esta cadeia possui o seguinte formato:

```
chunkname:currentline:
```

Nível 0 é a função executando correntemente, nível 1 é a função que chamou a função que está executando atualmente, etc.

Esta função é usada para construir um prefixo para mensagens de erro.

B.5 Bibliotecas padrão

B.5.1 Visão geral

As bibliotecas padrão de Lua oferecem funções úteis que são implementadas diretamente através da API C. Algumas dessas funções oferecem serviços essenciais para a linguagem (e.g., *type* e *getmetatable*); outras oferecem acesso a serviços "externos" (e.g., E/S); e outras poderiam ser implementadas em Lua mesmo, mas são bastante úteis ou possuem requisitos de desemepenho críticos que merecem uma implementação em C (e.g., *table.sort*).

Todas as bibliotecas são implementadas atráves da API C oficial e são fornecidas como módulos C separados. Correntemente, Lua possui as seguintes bibliotecas padrão:

- biblioteca básica;
- biblioteca de pacotes;
- manipulação de cadeias de caracteres;
- manipulação de tabelas;
- funções matemáticas (sen, log, etc.);
- entrada e saída;
- facilidades do sistema operacional;
- facilidades de depuração.

Excetuando-se a biblioteca básica e a biblioteca de pacotes, cada biblioteca provê todas as suas funções como campos de uma tabela global ou como métodos de seus objetos.

Para ter acesso a essas bibliotecas, o programa hospedeiro C deve chamar a função *luaL_openlibs*, que abre todas as bibliotecas padrão. De modo alternativo, é possível abri-las individualmente chamando luaopen_base (para a biblioteca básica), luaopen_package (para a biblioteca de pacotes), luaopen_string (para a biblioteca de cadeias de caracteres), luaopen_table (para a biblioteca de tabelas), luaopen_math (para a biblioteca matemática), luaopen_io (para a biblioteca de E/S), luaopen_os (para a biblioteca do Sistema Operacional), e luaopen_debug (para a biblioteca de depuração). Essas funções estão declaradas em lualib.h e não devem ser chamadas diretamente: deve-se chamá-las como qualquer outra função C de Lua, e.g., usando *lua_call*.

B.5.2 Funções básicas

A biblioteca de funções básicas oferece algumas funções essenciais a Lua. Se não se inclui esta biblioteca em sua aplicação, deve-se verificar cuidadosamente se necessita fornecer implementações para algumas de suas facilidades.

---

assert (v [, message])

Produz um erro quando o valor de seu argumento v é falso (i.e., **nil** ou **false**); caso contrário, retorna todos os seus argumentos. message é uma mensagem de erro; quando ausente, a mensagem padrão é "assertion failed!"

collectgarbage (opt [, arg])

Esta função é uma interface genérica para o coletor de lixo. Ela realiza diferentes funções de acordo com o seu primeiro argumento, opt:

- **"stop" :** pára o coletor de lixo.
- **"restart" :** reinicia o coletor de lixo.
- **"collect" :** realiza um ciclo de coleta de lixo completo.
- **"count" :** retorna a memória total que está sendo usada por Lua (em Kbytes).
- **"step" :** realiza um passo de coleta de lixo. O "tamanho" do passo é controlado por `arg` (valores maiores significam mais passos) de maneira não especificada. Se for desejado controlar o tamanho do passo, deve-se ajustar de maneira experimental o valor de `arg`. Retorna **true** se o passo terminou um ciclo de coleta de lixo.
- **"steppause" :** estabelece `arg`/100 como o novo valor para a *pausa* do coletor (ver B.2.11).
- **"setstepmul" :** estabelece `arg`/100 como o novo valor para o *multiplicador de passo* do coletor (ver B.2.11).

dofile (filename)

Abre o arquivo indicado e executa o seu conteúdo como um trecho de código Lua. Quando chamada sem argumentos, dofile executa o conteúdo da entrada padrão (stdin). Retorna todos os valores retornados pelo trecho. Em caso de erros, dofile propaga o erro para o seu chamador (isto é, dofile não executa em modo protegido).

error (message [, level])

Termina a última função protegida chamada e retorna message como a mensagem de erro. A função error nunca retorna.

Geralmente, error adiciona alguma informação sobre a posição do erro no início da mensagem. O argumento level especifica como obter a posição do erro. Quando ele é igual a 1 (o padrão), a posição do erro é onde a função error foi chamada. Quando ele é 2, a posição do erro é onde a função que chamou error foi chamada; e assim por diante. Passando um valor 0 para level evita a adição de informação da posição do erro à mensagem.

_G

Uma variável global (não uma função) que armazena o ambiente global (isto é, _G._G = _G). Lua por si só não usa esta variável; uma modificação do seu valor não afeta qualquer ambiente e vice-versa. (Use *setfenv* para mudar ambientes.)

getfenv (f)

Retorna o ambiente que está sendo usado correntemente pela função. f pode ser uma função Lua ou um número que especifica a função naquele nível de pilha: a função que chamou getfenv possui nível 1. Se a função fornecida não é uma função Lua ou se f é 0, getfenv retorna o ambiente global. O valor padrão para f é 1.

getmetatable (object)

Se object não possui uma meta-tabela, retorna **nil**. Caso contrário, se a meta-tabela do objeto possui um campo "__metatable", retorna o valor associado. Caso contrário, retorna a meta-tabela do objeto fornecido.

ipairs (t)

Retorna três valores: uma função iteradora, a tabela t e 0, de modo que a construção

```
for i,v in ipairs(t) do body end
```

irá iterar sobre os pares (1,t[1]), (2,t[2]), ···, até a primeira chave inteira ausente da tabela.

load (func [, chunkname])

Carrega um trecho usando a função func para obter seus pedaços. Cada chamada a func deve retornar uma cadeia de caracteres que concatena com resultados anteriores. Quando func retorna **nil** (ou quando não retorna nenhum valor), isso indica o fim do trecho.

Se não ocorrerem erros, retorna o trecho compilado como uma função; caso contrário, retorna **nil** mais a mensagem de erro. O ambiente da função retornada é o ambiente global.

chunkname é usado como o nome do trecho para mensagens de erro e informação de depuração. Quando ausente, o valor padrão é "=(load)".

loadfile ([filename])

Similar a *load*, mas obtém o trecho do arquivo filename ou da entrada padrão, se nenhum nome de arquivo é fornecido.

loadstring (string [, chunkname])

Similar a *load*, mas obtém o trecho da cadeia fornecida.

Para carregar e rodar uma dada cadeia, use a expressão idiomática

```
assert(loadstring(s))()
```

Quando ausente, o valor padrão para chunkname é a cadeia fornecida.

next (table [, index])

Permite a um programa pecorrer todos os campos de uma tabela. Seu primeiro argumento é uma tabela e seu segundo argumento é um índice nesta tabela. next retorna o próximo índice da tabela e seu valor associado. Quando chamada com **nil** como seu segundo argumento, next retorna um índice inicial e seu valor associado. Quando chamada com o último índice ou com **nil** em uma tabela vazia, next retorna **nil**. Se o segundo argumento está ausente, então ele é interpretado como **nil**. Em particular, pode-se usar next(t) para verificar se a tabela está vazia.

A ordem na qual os índices são enumerados não é especificada, *até mesmo para índices numéricos*. (Para percorrer uma tabela em ordem numérica, use o **for** numérico ou a função *ipairs*.).

O comportamento de next é *indefinido* se, durante o percorrimento, atribui-se qualquer valor a um campo não existente na tabela. Pode-se contudo modificar campos existentes. Em particular, pode-se limpar campos existentes.

---

pairs (t)

Retorna três valores: a função *next*, a tabela t e **nil**, de modo que a construção

```
for k,v in pairs(t) do body end
```

irá iterar sobre todos os pares chave–valor da tabela t.

Ver a função *next* para os cuidados que se deve ter ao modificar a tabela durante o seu percorrimento.

---

pcall (f, arg1, arg2, ...)

Chama a função f com os argumentos fornecidos em *modo protegido*. Isto significa que qualquer erro dentro de f não é propagado; ao invés disso, pcall captura o erro e retorna um código indicando o status. Seu primeiro resultado é o código de status (um booleano), que é verdadeiro se a chamada aconteceu sem erros. Neste caso, pcall também retorna todos os resultados da chamada, depois deste primeiro resultado. No caso de acontecer um erro, pcall retorna **false** mais a mensagem de erro.

---

print (e1, e2, ...)

Recebe qualquer número de argumentos e imprime os seus valores para stdout, usando a função *tostring* para convertê-los para cadeias de caracteres. print não é projetada para saída formatada, mas somente como uma maneira rápida de mostrar um valor, tipicamente para depuração. Para saída formatada, use *string.format*.

---

rawequal (v1, v2)

Verifica se v1 é igual a v2, sem invocar nenhum meta-método. Retorna um booleano.

---

rawget (table, index)

Obtém o valor real de table[index], sem invocar nenhum meta-método. table deve ser uma tabela; index pode ser qualquer valor.

---

rawset (table, index, value)

Atribui value como o valor real de table[index], sem invocar nenhum meta-método. table deve ser uma tabela, index pode ser qualquer valor diferente de **nil** e value pode ser qualquer valor Lua.

Essa função retorna table.

---

select (index, ...)

Se index é um número, retorna todos os argumentos após o argumento número index. Caso contrário, index deve ser a cadeia "#" e select retorna o número total de argumentos extras recebidos.

setfenv (f, table)

Estabelece o ambiente a ser usado pela função fornecida. f pode ser uma função Lua ou um número que especifica a função naquele nível de pilha: a função chamando setfenv possui nível 1. setfenv retorna a função fornecida.

Como um caso especial, quando f é 0 setfenv muda o ambiente do fluxo de execução corrente. Neste caso, setfenv não retorna nenhum valor.

setmetatable (table, metatable)

Estabelece a meta-tabela para a tabela fornecida (não se pode mudar a meta-tabela de outros tipos a partir de Lua, somente a partir de C.) Se metatable é **nil**, remove a meta-tabela da tabela fornecida. Se a meta-tabela original tem um campo "__metatable", dispara um erro.

Essa função retorna table.

tonumber (obj [, base])

Tenta converter seu argumento para um número. Se o argumento já é um número ou uma cadeia de caracteres que pode ser convertida para um número, então tonumber retorna este número; caso contrário, retorna **nil**.

Um argumento opcional especifica a base para interpretar o numeral. A base pode ser qualquer inteiro entre 2 e 36, inclusive. Em bases acima de 10, a letra 'A' (maiúscula ou minúscula) representa 10, 'B' representa 11 e assim por diante, com 'Z' representando 35. Na base 10 (o padrão), o número pode ter uma parte decimal, bem como uma parte expoente opcional (ver B.2.2). Em outras bases, somente inteiros sem sinal são aceitos.

tostring (obj)

Recebe um argumento de qualquer tipo e o converte para uma cadeia de caracteres em um formato razoável. Para um controle completo de como números são convertidos, use *string.format*.

Se a meta-tabela de e possui um campo "__tostring", então tostring chama o valor correspondente com e como argumento e usa o resultado da chamada como o seu resultado.

type (v)

Retorna o tipo de seu único argumento, codificado como uma cadeia de caracteres. Os resultados possíveis desta função são "nil" (uma cadeia de caracteres, não o valor **nil**), "number", "string", "boolean", "table", "function", "thread" e "userdata".

unpack (list [, i [, j]])

Retorna os elementos da tabela fornecida. Esta função é equivalente a

```
return list[i], list[i+1], ···, list[j]
```

exceto que o código acima pode ser escrito somente para um número fixo de elementos. Por padrão, i é 1 e j é o tamanho da lista, como definido pelo operador de tamanho (ver B.2.6.6).

_VERSION

Uma variável global (não uma função) que armazena uma cadeia contendo a versão corrente do interpretador. O conteúdo corrente desta variável é "Lua 5.1".

---

xpcall (f, err)

Esta função é similar a *pcall*, exceto que se pode estabelecer um novo tratador de erros.

xpcall chama a função f em modo protegido, usando err como um tratador de erros. Qualquer erro dentro de f não é propagado; ao invés disso, xpcall captura o erro, chama a função err com o objeto de erro original e retorna um código indicando um status. Seu primeiro resultado é o código de status (um booleano), que é verdadeiro se a chamada ocorreu sem erros. Neste caso, xpcall também retorna todos os resultados da chamada, depois deste primeiro resultado. Em caso de erro, xpcall retorna **false** mais o resultado de err.

B.5.3 Manipulação de co-rotinas

As operações relacionadas a co-rotinas constituem uma sub-biblioteca da biblioteca básica e estão dentro da tabela coroutine. Ver B.2.12 para uma descrição geral de co-rotinas.

---

coroutine.create (f)

Cria uma nova co-rotina, com corpo f. f deve ser uma função Lua. Retorna esta nova co-rotina, um objeto com tipo "thread".

---

coroutine.resume (co [, val1, ..., valn])

Inicia ou continua a execução da co-rotina co. Na primeira vez que se "continua" uma co-rotina, ela começa executando o seu corpo. Os valores val1, ··· são passados como os argumentos para o corpo da função. Se a co-rotina já cedeu a execução antes, resume a continua; os valores val1, ··· são passados como os resultados da cessão.

Se a co-rotina executa sem nenhum erro, resume retorna **true** mais quaisquer valores passados para yield (se a co-rotina cede) ou quaisquer valores retornados pelo corpo da função (se a co-rotina termina). Se há qualquer erro, resume retorna **false** mais a mensagem de erro.

---

coroutine.running ()

Retorna a co-rotina sendo executada ou **nil** quando chamada pelo fluxo de execução principal.

---

coroutine.status (co)

Retorna o status da co-rotina co, como uma cadeia de caracteres: "running", se a co-rotina está executando (isto é, ela chamou status); "suspended", se a co-rotina está suspensa em uma chamada a yield ou se ela não começou a sua execução ainda; "normal" se a co-rotina está ativa mas não está executando (isto é, ela continuou outra co-rotina); e "dead" se a co-rotina terminou sua função principal ou se ela parou com um erro.

coroutine.wrap (f)

Cria uma nova co-rotina, com corpo f. f deve ser uma função Lua. Retorna uma função que recomeça a co-rotina cada vez que é chamada. Quaisquer argumentos passados para a função comportam-se como os argumentos extras para resume. Retorna os mesmos valores retornados por resume, exceto o primeiro booleano. Em caso de erro, propaga o erro.

coroutine.yield ([val1, ..., valn])

Suspende a execução da co-rotina chamadora. A co-rotina não pode estar executando uma função C, um meta-método ou um iterador. Quaisquer argumentos para yield são passados como resultados extras para resume.

B.5.4 Módulos

A biblioteca de pacotes provê facilidades básicas para carregar e construir módulos em Lua. Ela exporta duas de suas funções diretamente no ambiente global: *require* e *module*. Todas as outras funções são exportadas em uma tabela package.

module (name [, ...])

Cria um módulo. Se há uma tabela em package.loaded[name], esta tabela é o módulo. Caso contrário, se existe uma tabela global t com o nome fornecido, esta tabela é o módulo. Caso contrário cria uma nova tabela t e a estabelece como o valor da global name e o valor de package.loaded[name]. Esta função também inicializa t._NAME com o nome fornecido, t._M com o módulo (o próprio t) e t._PACKAGE com o nome do pacote (o nome do módulo completo menos o último componente). Finalmente, module estabelece t como o novo ambiente da função corrente e o novo valor de package.loaded[name], de modo que *require* retorna t.

Se name é um nome composto (isto é, um nome com componentes separados por pontos), module cria (ou reusa, se elas já existem) tabelas para cada componente. Por exemplo, se name é a.b.c, então module armazena a tabela do módulo no campo c do campo b da global a.

Esta função pode receber algumas *opções* depois do nome do módulo, onde cada opção é uma função a ser aplicada sobre o módulo.

require (modname)

Carrega o módulo fornecido. Esta função começa procurando na tabela *package.loaded* para determinar se modname já foi carregado. Em caso afirmativo, require retorna o valor armazenado em package.loaded[modname]. Caso contrário, ela tenta achar um *carregador* para o módulo.

Para encontrar um carregador, require é guiada pelo array *package.loaders*. Modificando este array, podemos mudar como require procura por um módulo. A seguinte explicação é baseada na configuração padrão para *package.loaders*.

Primeiro require consulta package.preload[modname]. Se existe um valor nesse campo, este valor (que deve ser uma função) é o carregador. Caso contrário require busca por um carregador Lua usando o caminho armazenado em *package.path*. Se isso também falha, ela busca por um carregador C usando o caminho armazenado em *package.cpath*. Se isso também falha, ela tenta um carregador *tudo-em-um* (ver *package.loaders*).

Uma vez que um carregador é encontrado, require chama o carregador com um único argumento, modname. Se o carregador retorna qualquer valor, require atribui o valor retornado a package.loaded[modname]. Se o carregador não retorna nenhum valor e não foi atribuído nenhum valor a package.loaded[modname], então require atribui **true** a esta posição. Em qualquer caso, require retorna o valor final de package.loaded[modname].

Se ocorre um erro durante o carregamento ou a execução do módulo ou se não é possível encontrar um carregador para o módulo, então require sinaliza um erro.

---

package.cpath

O caminho usado por *require* para procurar por um carregador C.

Lua inicializa o caminho C *package.cpath* da mesma forma que inicializa o caminho Lua *package.path*, usando a variável de ambiente LUA_CPATH ou um caminho padrão definido em luaconf.h.

---

package.loaded

Uma tabela usada por *require* para controlar quais módulos já foram carregados. Quando se requisita um módulo modname e package.loaded[modname] não é falso, *require* simplesmente retorna o valor armazenado lá.

---

package.loaders

Uma tabela usada por *require* para controlar como carregar módulos.

Cada posição nesta tabela é uma função buscadora. Quando está procurando um módulo, *require* chama cada uma destas funções buscadoras em ordem crescente, com o nome do módulo (o argumento fornecido a *require*) como seu único parâmetro. A função pode retornar outra função (o *carregador* do módulo) ou uma cadeia de caracteres explicando porque ela não achou aquele módulo (ou **nil** se ela não tem nada a dizer). Lua inicializa esta tabela com quatro funções.

A primeira função buscadora simplesmente procurar um carregador no tabela *package.preload*.

A segunda função buscadora procura um carregador como uma biblioteca Lua, usando o caminho armazenado em *package.path*. Um caminho é uma seqüência de *padrões* separados por ponto-e-vírgulas. Para cada padrão, a função buscadora irá mudar cada ponto de interrogação no padrão para filename, que é o nome do módulo com cada ponto substituído por um "separador de diretório" (como "/" no Unix); então ela tentará abrir o nome do arquivo resultante. Por exemplo, se o caminho é a cadeia de caracteres

```
"./?.lua;./?.lc;/usr/local/?/init.lua"
```

a busca por um arquivo Lua para o módulo foo tentará abrir os arquivos ./foo.lua, ./foo.lc e /usr/local/foo/init.lua, nessa ordem.

A terceira função buscadora procura um carregador como uma biblioteca C, usando o caminho fornecido pela variável *package.cpath*. Por exemplo, se o caminho C é a cadeia

```
"./?.so;./?.dll;/usr/local/?/init.so"
```

a função buscadora para o módulo foo tentará abrir os arquivos ./foo.so, ./foo.dll e /usr/local/foo/init.so, nessa ordem. Uma vez que ela encontra uma biblioteca C, esta função buscadora primeiro usa uma facilidade de ligação dinâmica para ligar a aplicação com a biblioteca. Então ela tenta encontrar uma função C dentro da biblioteca para ser usada como carregador. O nome desta função C é a cadeia "luaopen_" concatenada com uma cópia do nome do módulo onde cada ponto é substituído por um sublinhado. Além disso, se o nome do módulo possui um hífen,

seu prefixo até (e incluindo) o primeiro hífen é removido. Por exemplo, se o nome do módulo é a.v1-b.c, o nome da função será luaopen_b_c.

A quarta função buscadora tenta um carregador tudo-em-um. Ela procura no caminho C uma biblioteca para a raiz do nome do módulo fornecido. Por exemplo, quando requisitando a.b.c, ela buscará por uma biblioteca C para a. Se encontrar, ela busca nessa biblioteca por uma função de abertura para o submódulo; no nosso exemplo, seria luaopen_a_b_c. Com esta facilidade, um pacote pode empacotar vários submódulos C dentro de uma única biblioteca, com cada submódulo guardando a sua função de abertura original.

---

package.loadlib (libname, funcname)

Liga dinamicamente o programa hospedeiro com a biblioteca C libname. Dentro desta biblioteca, procura por uma função funcname e retorna essa função como uma função C. (Desse modo, funcname deve seguir o protocolo (ver *lua_CFunction*)).

Esta é uma função de baixo nível. Ela contorna completamente o sistema de pacotes e de módulos. Diferentemente de *require*, ela não realiza qualquer busca de caminho e não adiciona extensões automaticamente. libname deve ser o nome do arquivo completo da biblioteca C, incluindo se necessário um caminho e uma extensão. funcname deve ser o nome exato exportado pela biblioteca C (que pode depender de como o compilador e o ligador C são usados).

Esta função não é provida por ANSI C. Dessa forma, ela está disponível somente em algumas plataformas (Windows, Linux, Mac OS X, Solaris, BSD, mais outros sistemas Unix que dão suporte ao padrão dlfcn).

---

package.path

O caminho usado por *require* para buscar um carregador Lua.

Ao iniciar, Lua inicializa esta variável com o valor da variável de ambiente LUA_PATH ou com um caminho padrão definido em luaconf.h, se a variável de ambiente não está definida. Qualquer ";;" no valor da variável de ambiente será substituído pelo caminho padrão.

---

package.preload

Uma tabela para armazenar carregadores para módulos específicos (ver *require*).

---

package.seeall (module)

Estabelece uma meta-tabela para module com seu campo __index se referindo ao ambiente global, de modo que esse módulo herda valores do ambiente global. Para ser usada como uma opção à função *module*.

B.5.5 Manipulação de cadeias de caracteres

Esta biblioteca provê funções genéricas para a manipulação de cadeias de caracteres, tais como encontrar e extrair subcadeias e casamento de padrões. Ao indexar uma cadeia em Lua, o primeiro caractere está na posição 1 (não na posição 0, como em C). Índices podem ter valores negativos e são interpretados como uma indexação de trás para frente, a partir do final da cadeia. Portanto, o último caractere está na posição -1 e assim por diante.

A biblioteca de cadeias provê todas as suas funções dentro da tabela string. Ela também estabelece uma meta-tabela para cadeias onde o campo __index aponta para a tabela string. Em conseqüência disso, pode-se usar as funções de cadeias em um estilo orientado a objetos. Por exemplo, string.byte(s, i) pode ser escrito como s:byte(i).

---

string.byte (s [, i [, j]])

Retorna o código numérico interno dos caracteres s[i], s[i+1], ···, s[j]. O valor padrão para i é 1; o valor padrão para j é i.

Códigos numéricos não são necessariamente portáveis entre plataformas.

---

string.char (i1, i2, ...)

Recebe zero ou mais inteiros. Retorna uma cadeia com tamanho igual ao número de argumentos, na qual cada caractere possui um código numérico interno igual ao seu argumento correspondente.

Códigos numéricos não são necessariamente portáveis entre plataformas.

---

string.dump (function)

Retorna uma cadeia contendo a representação binária da função fornecida, de modo que um *loadstring* posterior nesta cadeia retorna uma cópia da função. function deve ser uma função Lua sem upvalues.

---

string.find (s, pattern [, init [, plain]])

Procura o primeiro casamento do padrão pattern na cadeia s. Se a função acha um casamento, então find retorna os índices de s onde esta ocorrência começou e terminou; caso contrário, retorna **nil**. O terceiro argumento, init, é um valor numérico opcional e especifica onde iniciar a busca; seu valor padrão é 1 e pode ser negativo. Um valor **true** para o quarto argumento, plain, que é opcional, desabilita as facilidades de casamento de padrões, de modo que a função faz uma operação "encontra subcadeia" simples, sem considerar nenhum caractere em pattern como "mágico". Se plain é fornecido, então init deve ser fornecido também.

Se o padrão possui capturas, então em um casamento bem-sucedido os valores capturados são também retornados, após os dois índices.

---

string.format (formatstring, e1, e2, ...)

Retorna a versão formatada de seu número variável de argumentos seguindo a descrição dada no seu primeiro argumento (que deve ser uma cadeia). O formato da cadeia segue as mesmas regras da família printf de funções C padrão. As únicas diferenças são que as opções/modificadores *, l, L, n, p e h não são oferecidas e que há uma opção extra, q. A opção q formata uma cadeia em uma forma adequada para ser lida de volta de forma segura pelo interpretador Lua; a cadeia é escrita entre aspas duplas e todas as aspas duplas, quebras de linha, barras invertidas e zeros dentro da cadeia são corretamente escapados quando escritos. Por exemplo, a chamada

```
string.format('%q', 'a string with "quotes" and \n new line')
```

produzirá a cadeia:

```
"a string with \"quotes\" and \
 new line"
```

As opções c, d, E, e, f, g, G, i, o, u, X e x esperam um número como argumento, enquanto que q e s esperam uma cadeia.

Esta função não aceita valores de cadeias contendo zeros dentro delas, exceto quando esses valores são argumentos para a opção q.

---

string.gmatch (s, pattern)

Retorna uma função iteradora que, cada vez que é chamada, retorna a próxima captura de `pattern` na cadeia `s`. Se `pattern` não especifica nenhuma captura, então o casamento inteiro é produzido a cada chamada. Como um exemplo, o seguinte laço

```
s = "hello world from Lua"
for w in string.gmatch(s, "%a+") do
  print(w)
end
```

irá iterar sobre todas as palavras da cadeia `s`, imprimindo uma por linha. O próximo exemplo coleta todos os pares `key=value` da cadeia fornecida e os coloca em uma tabela:

```
t = {}
s = "from=world, to=Lua"
for k, v in string.gmatch(s, "(%w+)=(%w+)") do
  t[k] = v
end
```

Para essa função, um '^' no início de um padrão não funciona como uma âncora, visto que isso iria impedir a iteração.

---

string.gsub (s, pattern, repl [, n])

Retorna uma cópia de s na qual todas as (ou as primeiras n, se fornecido) ocorrências de pattern são substituídas por uma cadeia de substituição especificada por repl, que pode ser uma cadeia, uma tabela ou uma função. gsub também retorna, como seu segundo valor, o número total de substituições que ocorreram.

Se repl é uma cadeia, então seu valor é usado para a substituição. O caractere % funciona como um caractere de escape: qualquer seqüência em repl da forma %*n*, com *n* entre 1 e 9, representa o valor da *n*-ésima subcadeia capturada. A seqüência %0 representa o casamento inteiro. A seqüência %% representa um % simples.

Se repl é uma tabela, então a tabela é consultada a cada casamento, usando a primeira captura como a chave; se o padrão não especifica nenhuma captura, então o casamento inteiro é usado como a chave.

Se repl é uma função, então esta função é chamada toda vez que o casamento ocorre, com todas as subcadeias capturadas sendo passadas como argumentos, na ordem em que foram capturadas; se o padrão não especifica nenhuma captura, então o casamento inteiro é passado como um único argumento.

Se o valor retornado pela consulta à tabela ou pela chamada de função é uma cadeia ou um número, então esse valor é usado como a cadeia de substituição; caso contrário, se ele é **false** ou **nil**, então não há substituição (isto é, o casamento original é mantido na cadeia).

Aqui estão alguns exemplos:

```
x = string.gsub("hello world", "(%w+)", "%1 %1")
--> x="hello hello world world"

x = string.gsub("hello world", "%w+", "%0 %0", 1)
--> x="hello hello world"

x = string.gsub("hello world from Lua", "(%w+)%s*(%w+)", "%2 %1")
--> x="world hello Lua from"

x = string.gsub("home = $HOME, user = $USER", "%$(%w+)", os.getenv)
--> x="home = /home/roberto, user = roberto"

x = string.gsub("4+5 = $return 4+5$", "%$(.-)%$", function (s)
      return loadstring(s)()
    end)
--> x="4+5 = 9"

local t = {name="lua", version="5.1"}
x = string.gsub("$name%-$version.tar.gz", "%$(%w+)", t)
--> x="lua-5.1.tar.gz"
```

string.len (s)

Recebe uma cadeia e retorna seu tamanho. A cadeia vazia "" tem tamanho 0. Zeros dentro da cadeia são contados, então "a\000bc\000" possui tamanho 5.

string.lower (s)

Recebe uma cadeia e retorna uma cópia desta cadeia com todas as letras maiúsculas convertidas para minúsculas. Todos os demais caracteres permanecem iguais. A definição de o que é uma letra maiúscula depende do idioma (*locale*) corrente.

string.match (s, pattern [, init])

Procura o primeiro *casamento* de pattern na cadeia s. Se encontra um, então match retorna as capturas do padrão; caso contrário retorna **nil**. Se pattern não especifica nenhuma captura, então o casamento inteiro é retornado. Um terceiro argumento numérico opcional, init, especifica onde iniciar a busca; seu valor padrão é 1 e pode ser negativo.

string.rep (s, n)

Retorna uma cadeia que é a concatenação de n cópias da cadeia s.

string.reverse (s)

Retorna uma cadeia que é a cadeia s invertida.

string.sub (s, i [, j])

Retorna uma subcadeia de s que inicia em i e continua até j; i e j podem ser negativos. Se j está ausente, então assume-se que ele é igual a -1 (que é o mesmo que o tamanho da cadeia). Em particular, a chamada string.sub(s,1,j) retorna um prefixo de s com tamanho j e string.sub(s, -i) retorna um sufixo de s com tamanho i.

string.upper (s)

Recebe uma cadeia e retorna uma cópia desta cadeia com todas as letras minúsculas convertidas para maiúsculas. Todos os demais caracteres permanecem iguais. A definição de o que é uma letra minúscula depende do idioma (*locale*) corrente.

B.5.6 Padrões

Classes de caracteres:

Uma classe de caracteres é usada para representar um conjunto de caracteres. As seguintes combinações são permitidas em descrições de uma classe de caracteres:

- ***x*:** (onde *x* não é um dos caracteres mágicos `^$()%.[]*+-?`) representa o próprio caractere *x*.
- **. :** (um ponto) representa todos os caracteres.
- **%a:** representa todas as letras.
- **%c:** representa todos os caracteres de controle.
- **%d:** representa todos os dígitos.
- **%l:** representa todas as letras minúsculas.
- **%p:** representa todos os caracteres de pontuação.
- **%s:** representa todos os caracteres de espaço.
- **%u:** representa todas as letras maiúsculas.
- **%w:** representa todos os caracteres alfanuméricos.
- **%x:** representa todos os dígitos hexadecimais.
- **%z:** representa o caractere com representação 0.
- %***x*:** (onde *x* é qualquer caractere não-alfanumérico) representa o caractere *x*. Esta é a maneira padrão de escapar os caracteres mágicos. Qualquer caractere de pontuação (até mesmo os não mágicos) pode ser precedido por um '%' quando usado para representar a si mesmo em um padrão.
- **[*set*]:** representa a classe que é a união de todos os caracteres em *set*. Uma faixa de caracteres pode ser especificada separando os caracteres finais da faixa com um '-'. Todas as classes %*x* descritas acima também podem ser usadas como componentes em *set*. Todos os outros caracteres em *set* representam eles mesmos. Por exemplo, `[%w_]` (ou `[_%w]`) representa todos os caracteres alfanuméricos mais o sublinhado, `[0-7]` representa os dígitos octais e `[0-7%l%-]` representa os dígitos octais mais as letras minúsculas mais o caractere '-'.

  A interação entre faixas e classes não é definida. Portanto, padrões como [%a-z] ou [a-%%] não possuem significado.

- **[^*set*] :** representa o complemento de *set*, onde *set* é interpretado como acima.

Para todas as classes representadas por uma única letra (%a, %c, etc.), a letra maiúscula correspondente representa o complemento da classe. Por exemplo, %S representa todos os caracteres que não são de espaço.

As definições de letra, espaço e outros grupos de caracteres dependem do idioma (*locale*) corrente. Em particular, a classe [a-z] pode não ser equivalente a %l.

Um item de padrão pode ser:

- uma classe de um único caractere, que casa qualquer caractere simples que pentença à classe;
- uma classe de um único caractere seguida por '*', que casa 0 ou mais repetições de caracteres da classe. Estes itens de repetição sempre casarão a maior seqüência possível;
- uma classe de um único caractere seguida por '+', que casa 1 ou mais repetições de caracteres da classe. Estes itens de repetição sempre casarão a maior seqüência possível;
- uma classe de um único caractere seguida por '-', que também casa 0 ou mais repetições de caracteres da classe. Diferentemente de '*', estes itens de repetição sempre casarão a *menor* seqüência possível;
- uma classe de um único caractere seguida por '?', que casa 0 ou 1 ocorrência de um caractere da classe;
- %*n*, para *n* entre 1 e 9; tal item casa uma subcadeia igual à *n*-ésima cadeia capturada;
- %b*xy*, onde *x* e *y* são dois caracteres distintos; tal item casa cadeias que começam com *x*, terminam com *y* e onde o número de *x*s e de *y*s é *balanceado*. Isto significa que, se alguém ler a cadeia da esquerda para a direita, contando *+1* para um *x* e *-1* para um *y*, o *y* final é o primeiro *y* onde o contador alcança 0. Por exemplo, o item %b() casa expressões com parênteses balanceados.

Padrão:

Um padrão é uma seqüência de itens de padrão. Um '^' no início de um padrão ancora o casamento no início da cadeia sendo usada. Um '$' no fim de um padrã ancora o casamento no fim da cadeia sendo usada. Em outras posições, '^' e '$' não possuem significado especial e representam a si mesmos.

Um padrão pode conter subpadrões delimitados por parênteses; eles descrevem capturas. Quando um casamento ocorre, as subcadeias da cadeia sendo usada que casaram com as capturas são armazenadas (*capturadas*) para uso futuro. Capturas são numeradas de acordo com os seus parênteses esquerdos. Por exemplo, no padrão "(a*(.)%w(%s*))", a parte da cadeia casando "a*(.)%w(%s*)" é armazenada como a primeira captura (e portanto tem número 1); o caractere casando "." é capturado com o número 2 e a parte casando "%s*" possui número 3.

Como um caso especial, a captura vazia () captura a posição da cadeia corrente (um número). Por exemplo, se aplicarmos o padrão "()aa()" na cadeia "flaaap", haverá duas capturas: 3 e 5.

Um padrão não pode conter zeros dentro dele. Use %z como alternativa.

B.5.7 Manipulação de tabelas

Esta biblioteca provê funções genéricas para manipulação de tabelas. Ela provê todas as suas funções na tabela table.

A maioria das funções na biblioteca de tabelas assume que a tabela representa um array ou uma lista. Para estas funções, quando falamos sobre o "tamanho" de uma tabela estamos falando sobre o resultado do operador de tamanho.

---

table.concat (table [, sep [, i [, j]]])

Dado um array onde todos os elementos são cadeias ou números, retorna table[i]..sep..table[i+1] ··· sep..table[j]. O valor padrão para sep é a cadeia vazia, o padrão para i é 1 e o padrão para j é o tamanho da tabela. Se i é maior do que j, retorna a cadeia vazia.

table.insert (table, [pos,] value)

Insere o elemento value na posição pos de table, deslocando os outros elementos para abrir espaço, se necessário. O valor padrão para pos é n+1, onde n é o tamanho da tabela (ver B.2.6.6), de modo que uma chamada table.insert(t,x) insere x no fim da tabela t.

table.maxn (table)

Retorna o maior índice numérico positivo da tabela fornecida ou zero se a tabela não possui índices numéricos positivos. (Para realizar seu trabalho esta função faz um percorrimento linear da tabela inteira.)

table.remove (table [, pos])

Remove de table o elemento na posição pos, deslocando os outros elementos para preencher o espaço, se necessário. Retorna o valor do elemento removido. O valor padrão para pos é n, onde n é o tamanho da tabela, de modo que uma chamada table.remove(t) remove o último elemento da tabela t.

table.sort (table [, comp])

Ordena os elementos da tabela em uma dada ordem, *in-place*, de table[1] até table[n], onde n é o tamanho da tabela. Se comp é fornecido, então ele deve ser uma função que recebe dois elementos da tabela e retorna true quando o primeiro é menor do que o segundo (de modo que not comp(a[i+1],a[i]) será verdadeiro após a ordenação). Se comp não é fornecido, então o operador padrão de Lua < é usado em seu lugar.

O algoritmo de ordenação não é estável; isto é, elementos considerados iguais pela ordem fornecida podem ter suas posições relativas trocadas pela ordenação.

B.5.8 Funções matemáticas

Esta biblioteca é uma interface para a biblioteca matemática de C padrão. Ela provê todas as suas funções na tabela math.

math.abs (x)

Retorna o valor absoluto de x.

math. acos (x)

Retorna o arco co-seno de x (em radianos).

math.asin (x)

Retorna o arco seno de x (em radianos).

math.atan (x)

Retorna o arco tangente de x (em radianos).

math.atan2 (x)

Retorna o arco tangente de y/x (em radianos), mas usa o sinal dos dois parâmetros para achar o quadrante do resultado. (Também trata corretamente o caso de x ser zero.)

math.aceil (x)

Retorna o menor inteiro maior ou igual a x.

math.cos (x)

Retorna o co-seno de x (assume que x está em radianos).

math.cosh (x)

Retorna o co-seno hiperbólico de x.

math.deg (x)

Retorna o ângulo x (dado em radianos) em graus.

math.exp (x)

Retorna o valor de e^x.

math.floor (x)

Retorna o maior inteiro menor ou igual a x.

math.fmod (x, y)

Retorna o resto da divisão de x por y.

math.frexp (x)

Retorna m e e tais que $x = m2^e$, e é um inteiro e o valor absoluto de m está no intervalo *[0.5, 1)* (ou zero quando x é zero).

math.huge (x)

O valor de HUGE_VAL, um valor maior ou igual a qualquer outro valor numérico.

math.ldexp (m, e)

Retorna $m2^e$ (e deve ser um inteiro).

math.log (x)

Retorna o logaritmo natural de x.

math.log10 (x)

Retorna o logaritmo base-10 de x.

math.max (x, …)

Retorna o valor máximo entre os seus argumentos.

math.min (x, …)

Retorna o valor mínimo entre os seus argumentos.

math.modf (x)

Retorna dois números, a parte integral de x e a parte fracionária de x.

math.pi

O valor de *pi*.

math.pow (x, y)

Retorna x^y (também pode-se usar a expressão x^y para computar este valor).

math.rad (x)

Retorna o ângulo x (dado em graus) em radianos.

math.random ([m [,n]])

Esta função é uma interface para a função geradora pseudo-randômica simples rand fornecida por ANSI C. (Nenhuma garantia pode ser dada para suas propriedades estatísticas.)

Quando chamada sem argumentos, retorna um número real pseudo-randômico no intervalo *[0,1)*. Quando chamada com um número m, math.random retorna um inteiro pseudo-randômico no intervalo *[1, m]*. Quando chamada com dois números m e n, math.random retorna um inteiro pseudo-randômico no intervalo *[m, n]*.

math.randomseed (x)

Estabelece x como a "semente" para o gerador pseudo-randômico: sementes iguais produzem seqüências iguais de números.

math.sin (x)

Retorna o seno de x (assume que x está em radianos).

math.sinh (x)

Retorna o seno hiperbólico de x.

math.sqrt (x)

Retorna a raiz quadrada de x (também se pode usar a expressão x^0.5 para computar este valor).

math.tan (x)

Retorna a tangente de x (assume que x está em radianos).

math.tanh (x)

Retorna a tangente hiperbólica de x.

B.5.9 Facilidades de entrada e saída

A biblioteca de E/S provê dois estilos diferentes para manipulação de arquivos. O primeiro usa descritores de arquivo implícitos; isto é, há operações para estabelecer um arquivo de entrada padrão e um arquivo de saída padrão e todas as operações de entrada/saída são realizadas sobre estes arquivos. O segundo estilo usa descritores de arquivo explícitos.

Quando se usa descritores de arquivo implícitos, todas as operações são providas pela tabela io. Quando se usa descritores de arquivo explícitos, a operação *io.open* retorna um descritor de arquivo e então todas as operações são providas como métodos do descritor de arquivo.

A tabela io também fornece três descritores de arquivo pré-definidos com os seus significados usuais de C: io.stdin, io.stdout e io.stderr.

A menos que dito de modo contrário, todas as funções de E/S retornam **nil** em caso de falha (mais uma mensagem de erro como segundo resultado e um código de erro dependente do sistema como um terceiro resultado), ou algum valor diferente de **nil** em caso de sucesso.

io.close ([file])

Equivalente a file:close(). Quando não recebe file, fecha o arquivo de saída padrão.

io.flush ()

Equivalente a file:flush no arquivo de saída padrão.

io.input ([file])

Quando chamada com um nome de arquivo, abre o arquivo com aquele nome (em modo texto) e estabelece seu manipulador como o arquivo de entrada padrão. Quando chamada com um manipulador de arquivo, simplesmente estabelece este manipulador de arquivo como o arquivo de entrada padrão. Quando chamada sem parâmetros, retorna o arquivo de entrada padrão corrente.

Em caso de erros esta função dispara o erro, ao invés de retornar um código de erro.

io.lines ([filename])

Abre o nome de arquivo fornecido em modo de leitura e retorna uma função iteradora que, cada vez que é chamada, retorna uma nova linha do arquivo. Portanto, a construção

```
for line in io.lines(filename) do body end
```

irá iterar sobre todas as linhas do arquivo. Quando a função iteradora detecta o fim do arquivo, ela retorna **nil** (para finalizar o laço) e automaticamente fecha o arquivo.

A chamada io.lines() (sem nenhum nome de arquivo) é equivalente a io.input():lines(); isto é, ela itera sobre as linhas do arquivo de entrada padrão. Neste caso ela não fecha o arquivo quando o laço termina.

io.open (filename [, mode])

Esta função abre um arquivo, no modo especificado na cadeia mode. Ela retorna um novo manipulador de arquivo ou, em caso de erros, **nil** mais uma mensagem de erro.

A cadeia de caracteres mode pode ser qualquer uma das seguintes:

- **“r”:** modo de leitura (o padrão);
- **“w”:** modo de escrita;
- **“a”:** modo de adição;
- **“r+”:** modo de atualização, todos os dados anteriores são preservados;
- **“w+”:** modo de atualização, todos os dados anteriores são preservados;
- **“a+”:** modo de atualização de adição, dados anteriores são preservados, a escrita somente é permitida no fim do arquivo.

A cadeia mode também pode ter um 'b' no fim, que é necessário em alguns sistemas para abrir o arquivo em modo binário. Esta cadeia é exatamente o que é usado na função padrão de C fopen.

io.output ([file])

Similar a *io.input*, mas opera sobre o arquivo de saída padrão.

io.popen ([prog [, mode]])

Inicia o programa prog em um processo separado e retorna um manipulador de arquivo que pode ser usado para ler dados deste programa (se mode é "r", o padrão) ou escrever dados para este programa (se mode é "w").

Esta função é dependente do sistema e não está disponível em todas as plataformas.

io.read (format1, ...)

Equivalente a io.input():read.

io.tmpfile ()

Retorna um manipulador para um arquivo temporário. Este arquivo é aberto em modo de atualização e é automaticamente removido quando o programa termina.

io.type (obj)

Verifica se obj é um manipulador de arquivo válido. Retorna a cadeia "file" se obj é um manipulador de arquivo aberto, "close file" se obj é um manipulador de arquivo fechado ou **nil** se obj não é um manipulador de arquivo.

io.write (value1, ...)

Equivalente a io.output():write.

file:close ()

Fecha file. Arquivos são automaticamente fechados quando seus manipuladores são coletados pelo coletor de lixo, mas leva uma quantidade indeterminada de tempo para isso acontecer.

file:flush ()

Salva qualquer dado escrito para file.

file:lines ()

Retorna uma função iteradora que, cada vez que é chamada, retorna uma nova linha do arquivo. Portanto, a construção

```
for line in file:lines() do ... end
```

irá iterar sobre todas as linhas do arquivo. (Ao contrário de *io.lines*, essa função não fecha o arquivo quando o laço termina.)

file:read (format1, ...)

Lê o arquivo file, de acordo com os formatos fornecidos, os quais especificam o que deve ser lido. Para cada formato, a função retorna uma cadeia (ou um número) com os caracteres lidos ou **nil** se ela não pode retornar dados com o formato especificado. Quando chamada sem formatos, ela usa o formato padrão que lê a próxima linha toda.

Os formatos disponíveis são:

- **"*n":** lê um número; este é o único formato que retorna um número ao invés de uma cadeia.
- **"*a":** lê o arquivo inteiro, iniciando na posição corrente. Quando está no final do arquivo, retorna a cadeia vazia.
- **"*l":** lê a próxima linha (pulando o fim de linha), retornando **nil** ao final do arquivo. Este é o formato padrão.
- ***número*:** lê uma cadeia até este número de caracteres, retornando **nil** ao final do arquivo. Se o número fornecido é zero, a função não lê nada e retorna uma cadeia vazia ou **nil** quando está no fim do arquivo.

file:seek ([whence] [, offset])

Estabelece e obtém a posição do arquivo, medida a partir do início do arquivo, até a posição dada por offset mais uma base especificada pela cadeia whence, como a seguir:

- **"set" :** base é a posição 0 (o início do arquivo);
- **"cur" :** base é a posição corrente;
- **"end" :** base é o fim do arquivo;

Em caso de sucesso, a função seek retorna a posição final do arquivo, medida em bytes a partir do início do arquivo. Se esta função falha, ela retorna **nil**, mais uma cadeia descrevendo o erro.

O valor padrão para whence é "cur" e para offset é 0. Portanto, a chamada file:seek() retorna a posição do arquivo corrente, sem modificá-la; a chamada file:seek("set") estabelece a posição para o início do arquivo (e retorna 0); e a chamada file:seek("end") estabelece a posição para o fim do arquivo e retorna seu tamanho.

file:setvbuf (mode [, size])

Define o modo de bufferização para um arquivo de saída. Há três modos disponíveis:

- **"no" :** nenhuma bufferização; o resultado de qualquer operação de saída aparece imediatamente;
- **"full" :** bufferização completa; a operação de saída é realizada somente quando o buffer está cheio (ou quando explicitamente se descarrega o arquivo (ver *io.flush*));
- **"line" :** bufferização de linha; a saída é bufferizada até que uma nova linha é produzida ou há qualquer entrada a partir de alguns arquivos especiais (como um dispositivo de terminal).

Para os últimos dois casos, size especifica o tamanho do buffer, em bytes. O padrão é um tamanho apropriado.

file:write (value1, ...)

Escreve o valor de cada um de seus argumentos para file. Os argumentos devem ser cadeias de caracteres ou números. Para escrever outros valores, use *tostring* ou *string.format* antes de write.

B.5.10 Facilidades do sistema operacional

Esta biblioteca é implementada através da tabela os.

os.clock ()

Retorna uma aproximação da quantidade de tempo de CPU, em segundos, usada pelo programa.

os.date ([format [, time]])

Retorna uma cadeia ou uma tabela contendo data e hora, formatada de acordo com a cadeia format fornecida.

Se o argumento time está presente, este é o tempo a ser formatado (ver a função *os.time* para uma descrição deste valor). Caso contrário, date formata a hora corrente.

Se format começa com '!', então a data é formatada no Tempo Universal Coordenado. Após esse caractere opcional, se format é a cadeia "*t", então date retorna uma tabela com os seguintes campos: year (quatro dígitos), month (1--12), day (1--31), hour (0--23), min (0--59), sec (0--61), yday (dia do ano) e isdst (flag que indica o horário de verão, um booleano).

Se format não é "*t", então date retorna a data como uma cadeia de caracteres, formatada de acordo com as mesmas regras da função C strftime.

Quando chamada sem argumentos, date retorna uma representação aceitável da data e da hora que depende do sistema hospedeiro e do idioma (*locale*) corrente. (isto é, os.date() é equivalente a os.date("%c")).

os.difftime (t2, t1)

Retorna o número de segundos a partir do tempo t1 até o tempo t2. Em POSIX, Windows e alguns outros sistemas, este valor é exatamente t2-t1.

os.execute ([command])

Esta função é equivalente à função C system. Ela passa command para ser executado por um interpretador de comandos do sistema operacional. Ela retorna um código de status, que é dependente do sistema. Se command está ausente, então a função retorna um valor diferente de zero se um interpretrador de comandos está disponível e zero caso contrário.

os.exit ([code])

Chama a função C exit, com um código code opcional, para terminar o programa hospedeiro. O valor padrão para code é o código de sucesso.

os.getenv (varname)

Retorna o valor da variável de ambiente do processo varname ou **nil** se a variável não está definida.

os.remove (filename)

Remove um arquivo ou diretório com o nome fornecido. Diretórios devem estar vazios para serem removidos. Se esta função falha, ela retorna **nil**, mais uma cadeia descrevendo o erro.

os.rename (oldname, newname)

Renomeia um arquivo ou diretório chamado oldname para newname. Se esta função falha, ela retorna **nil**, mais uma cadeia descrevendo o erro.

os.setlocale (locale [, category])

Estabelece o idioma (*locale*) corrente do programa. locale é uma cadeia de caracteres especificando um idioma; category é uma cadeia opcional descrevendo para qual categoria deve-se mudar: "all", "collate", "ctype", "monetary", "numeric" ou "time"; a categoria padrão é "all". Esta função retorna o nome do novo idioma ou **nil** se a requisição não pode ser honrada.

Se locale é a cadeia vazia, estabelece-se o idioma corrente como um idioma nativo definido pela implementação. Se locale é a cadeia "C", estabelece-se o idioma corrente como o idioma padrão de C.

Quando chamada com **nil** como o primeiro argumento, esta função retorna somente o nome do idioma corrente para a categoria fornecida.

os.time ([table])

Retorna o tempo corrente quando chamada sem argumentos ou um tempo representando a data e a hora especificados pela tabela fornecida. Esta tabela deve ter campos year, month e day e pode ter campos hour, min, sec e isdst (para uma descrição destes campos, ver a função *os.date*).

O valor retornado é um número, cujo significado depende do seu sistema. Em POSIX, Windows e alguns outros sistemas, este número conta o número de segundos desde algum tempo de início dado (a "era"). Em outros sistemas, o significado não é especificado e o número retornado por time pode ser usado somente como um argumento para date e difftime.

os.tmpname ()

Retorna uma cadeia de caracteres com o nome de um arquivo que pode ser usado para um arquivo temporário. O arquivo deve ser explicitamente aberto antes de ser usado e explicitamente removido quando não for mais necessário.

B.5.11 Biblioteca de depuração

Esta biblioteca provê as funcionalidades da interface de depuração para programas Lua. Deve-se ter cuidado ao usar esta biblioteca. As funções fornecidas aqui devem ser usadas exclusivamente para depuração e tarefas similares, tais como medição (*profiling*). Por favor resista à tentação de usá-las como uma ferramenta de programação usual: elas podem ser muito lentas. Além disso, várias dessas funções violam algumas suposições a respeito do código Lua (e.g., que variáveis locais a uma função não podem ser acessadas de fora

da função ou que meta-tabelas de objetos userdata não podem ser modificadas por código Lua) e portanto podem comprometer código que, de outro modo, seria seguro.

Todas as funções nesta biblioteca são fornecidas na tabela debug. Todas as funções que operam sobre um objeto do tipo thread possuem um primeiro argumento opcional que é o objeto thread sobre o qual a função deve operar. O padrão é sempre o fluxo de execução corrente.

---

debug.debug ()

Entra em um modo interativo com o usuário, executando cada cadeia de caracteres que o usuário entra. Usando comandos simples e outros mecanismos de depuração, o usuário pode inspecionar variáveis globais e locais, mudar o valor delas, avaliar expressões, etc. Uma linha contendo somente a palavra cont termina esta função, de modo que a função chamadora continua sua execução.

Os comandos para debug.debug não são aninhados de modo léxico dentro de nenhuma função e, portanto, não possuem acesso direto a variáveis locais.

---

debug.getfenv (o)

Retorna o ambiente do objeto o.

---

debug.gethook ()

Retorna as configurações de gancho correntes do fluxo de execução como três valores: a função de gancho corrente, a máscara de ganho corrente e a contagem de ganho corrente (como estabelecido pela função *debug.sethook*).

---

debug.getinfo (function [, what])

Retorna uma tabela com informação sobre uma função. Pode-se fornecer a função diretamente ou pode-se fornecer um número como o valor de function, que significa a função executando no nível function da pilha de chamadas do fluxo de execução fornecido: nível 0 é a função corrente (a própria getinfo); nível 1 é a função que chamou getinfo; e assim por diante. Se function é um número maior do que o número de funções ativas, então getinfo retorna **nil**.

A tabela retornada pode conter todos os campos retornados por *lua_getinfo*, com a cadeia what descrevendo quais campos devem ser preenchidos. O padrão para what é obter todas as informações disponíveis, exceto a tabela de linhas válidas. Se presente, a opção 'f' adiciona um campo chamado func com a própria função. Se presente, a opção 'L' adiciona um campo chamado activelines com a tabela de linhas válidas.

Por exemplo, a expressão debug.getinfo(1,"n").name retorna uma tabela com um nome para a função corrente, se um nome razoável pode ser encontrado, e a expressão debug.getinfo(print) retorna uma tabela com todas as informações disponíveis sobre a função *print*.

---

debug.getlocal (level, local)

Esta função retorna o nome e o valor da variável local com índice local da função no nível level da pilha (o primeiro parâmetro ou variável local possui índice 1 e assim por diante, até a última variável local ativa). A função retorna **nil** se não existe uma variável local com o índice fornecido e dispara um erro quando chamada com um level fora da faixa de valores válidos (pode-se chamar *debug.getinfo* para verificar se o nível é válido).

Nomes de variáveis que começam com '(' (abre parênteses) representam variáveis internas (variáveis de controle de laços, temporários e locais de funções C).

debug.getmetatable (object)

Retorna a meta-tabela do object fornecido ou **nil** se ele não possui uma meta-tabela.

debug.getregistry ()

Retorna a tabela de registro (ver B.3.6).

debug.getupvalue (func, up)

Esta função retorna o nome e o valor do upvalue com índice up da função func. A função retorna **nil** se não há um upvalue com o índice fornecido.

debug.setfenv (object, table)

Estabelece a tabela table como o ambiente do object fornecido. Retorna object.

debug.sethook (hook, mask [, count])

Estabelece a função fornecida como um gancho. A cadeia mask e o número count descrevem quando o gancho será chamado. A cadeia mask pode ter os seguintes caracteres, com o respectivo significado:

- **"c" :** o gancho é chamado toda vez que Lua chama uma função;
- **"r" :** o gancho é chamado toda vez que Lua retorna de uma função;
- **"l" :** o gancho é chamado toda vez que Lua entra uma nova linha de código.

Com um count diferente de zero, o gancho é chamado após cada count instruções.

Quando chamada sem argumentos, *debug.gethook* desabilita o gancho.

Quando o gancho é chamado, seu primeiro parâmetro é uma cadeia de caracteres descrevendo o evento que disparou a sua chamada: "call", "return" (ou "tail return"), "line" e "count". Para eventos de linha, o gancho também obtém o novo número de linha como seu segundo parâmetro. Dentro do gancho, é possível chamar getinfo com nível 2 para obter mais informação sobre a função sendo executada (nível 0 é a função getinfo e nível 1 é a função de gancho), a menos que o evento seja "tail return". Neste caso, Lua está somente simulando o retorno e uma chamada a getinfo retornará dados inválidos.

debug.setlocal (level, local, value)

Esta função atribui o valor value à variável local com índice local da função no nível level da pilha. A função retorna **nil** se não há uma variável local com o índice fornecido e dispara um erro quando chamada com um level fora da faixa de valores válidos (pode-se chamar getinfo para verificar se o nível é válido). Caso contrário, a função retorna o nome da variável local.

debug.setmetatable (object, table)

Estabelece table como a meta-tabela do object fornecido (table pode ser **nil**).

debug.setupvalue (func, up, value)

Esta função atribui o valor value ao upvalue com índice up da função func. A função retorna **nil** se não há um upvalue com o índice fornecido. Caso contrário, a função retorna o nome do upvalue.

debug.traceback ([message])

Retorna uma cadeia de caracteres com um traço da pilha de chamadas. Uma cadeia opcional message é adicionada ao início do traço. Um número opcional level diz em qual nível iniciar o traço (o padrão é 1, a função chamando traceback).

B.6 O interpretador Lua autônomo

Embora Lua tenha sido projetada como uma linguagem de extensão, para ser embutida em um programa C hospedeiro, Lua também é freqüentemente usada como uma linguagem auto-suficiente. Um interpretador para Lua como uma linguagem auto-suficiente, chamado simplesmente lua, é fornecido com a distribuição padrão. Esse interpretador inclui todas as bibliotecas padrão, inclusive a biblioteca de depuração. Seu uso é:

```
lua [options] [script [args]]
```

As opções são:

- **-e *stat* :** executa a cadeia *stat*;
- **-l *mod* :** "requisita" *mod*;
- **-i :** entra em modo interativo após executar *script*;
- **-v :** imprime informação de versão;
- **-- :** pára de tratar opções;
- **- :** executa stdin como um arquivo e pára de tratar opções.

Após tratar suas opções, Lua executa o *script* fornecido, passando para ele os *args* fornecidos como cadeias de argumentos. Quando chamado sem argumentos, lua comporta-se como lua -v -i quando a entrada padrão (stdin) é um terminal e como lua - em caso contrário.

Antes de executar qualquer argumento, o interpretador verifica se há uma variável de ambiente LUA_INIT. Se seu formato é *@filename*, então lua executa o arquivo. Caso contrário, lua executa a própria cadeia de caracteres.

Todas as opções são manipuladas na ordem dada, exceto -i. Por exemplo, uma invocação como

```
$ lua -e'val=1' -e 'print(val)' script.lua
```

irá primeiro atribuir 1 a a, depois imprimirá o valor de a (que é '1') e finalmente executará o arquivo script.lua sem argumentos (aqui $ é o *prompt* do interpretador de comandos. Pode-se ter um prompt diferente).

Antes de começar a executar o script, lua guarda todos os argumentos fornecidos na linha de comando em uma tabela global chamada arg. O nome do script é armazenado no índice 0, o primeiro argumento após o nome

do script fica no índice 1 e assim por diante. Quaisquer argumentos antes do nome do script (isto é, o nome do interpretador mais as opções) ficam em índices negativos. Por exemplo, na chamada

```
$ lua -la b.lua t1 t2
```

o interpretador primeiro executa o arquivo a.lua, depois cria a tabela

```
arg = { [-2] = "lua", [-1] = "-la",
        [0] = "b.lua",
        [1] = "t1", [2] = "t2" }
```

e finalmente executa o arquivo b.lua. O script é chamado com arg[1], arg[2], ··· como argumentos; ele também pode acessar estes argumentos com a expressão vararg '...'.

Em modo interativo, se for escrito um comando incompleto, o interpretador espera que o complete e indica isto através de um prompt diferente.

Se a variável global _PROMPT contém uma cadeia de caracteres, então o seu valor é usado como o prompt. De maneira similar, se a variável global _PROMPT2 contém uma cadeia, seu valor é usado como o prompt secundário (mostrado durante comandos incompletos). Portanto, os dois prompts podem ser modificados diretamente na linha de comando ou em quaisquer programas Lua fazendo uma atribuição a _PROMPT. Ver o exemplo a seguir:

```
$ lua -e"_PROMPT='myprompt> '" –i
```

O par de aspas mais externo é para o interpretador de comandos e o par mais interno é para Lua. O uso de -i para entrar em modo interativo; caso contrário, o programa iria terminar silenciosamente logo após a atribuição a _PROMPT.

Para permitir o uso de Lua como um interpretador de scripts em sistemas Unix, o interpretador de linha de comando pula a primeira linha de um trecho de código se ele começa com #. Portanto, scripts Lua podem ser usados como programas executáveis usando chmod +x e a forma #!, como em

```
#!/usr/local/bin/lua
```

É claro que a localização do interpretador Lua pode ser diferente na sua máquina. Se lua está em seu PATH, então

```
#!/usr/bin/env lua
```

é uma solução mais portável.

B.7 Incompatibilidades com a versão 5.0

NOTA As incompatibilidades que podem ser encontradas quando se passa um programa de Lua 5.0 para Lua 5.1 são listadas nesta seção. Pode-se evitar a maioria das incompatibilidades compilando Lua com opções apropriadas (ver o arquivo luaconf.h). Contudo, todas essas opções de compatibilidade serão removidas na próxima versão de Lua.

B.7.1 Mudanças na linguagem

A seguir são listadas as alterações na linguagem introduzidas em Lua 5.1:

- o sistema de vararg mudou do pseudo-argumento arg com uma tabela com os argumentos extras para a expressão vararg (ver a opção de tempo de compilação LUA_COMPAT_VARARG em luaconf.h);
- houve uma mudança sutil no escopo das variáveis implícitas do comando **for** e do comando **repeat**;
- a sintaxe de cadeia longa/comentário longo ([[*string*]]) não permite aninhamento. Pode-se usar a nova sintaxe ([=[*string*]=]) nesses casos (ver a opção de tempo de compilação LUA_COMPAT_LSTR em luaconf.h.)

B.7.2 Mudanças nas bibliotecas

A seguir são listadas as alterações nas bibliotecas-padrão introduzidas em Lua 5.1:

- a função string.gfind foi renomeada para *string.gmatch* (ver a opção de tempo de compilação LUA_COMPAT_GFIND em luaconf.h);
- quando *string.gsub* é chamada com uma função como seu terceiro argumento, sempre que esta função retorna **nil** ou **false** a cadeia de substituição é o casamento inteiro, ao invés da cadeia vazia;
- A função table.setn está ultrapassada e não deve ser usada. A função table.getn corresponde ao novo operador de tamanho (#); use o operador ao invés da função (ver a opção de tempo de compilação LUA_COMPAT_GETN em luaconf.h.);
- a função loadlib foi renomeada para *package.loadlib* (ver a opção de tempo de compilação LUA_COMPAT_LOADLIB em luaconf.h);
- a função math.mod foi renomeada para *math.fmod* (ver a opção de tempo de compilação LUA_COMPAT_MOD em luaconf.h);
- as funções table.foreach e table.foreachi estão ultrapassadas e não devem ser usadas. Pode-se usar um laço **for** com pairs ou ipairs ao invés delas;
- houve mudanças substanciais na função *require* devido ao novo sistema de módulos. O novo comportamento é basicamente compatível com o antigo, porém agora require obtém o caminho de *package.path* e não mais de LUA_PATH;
- a função *collectgarbage* possui argumentos diferentes. A função gcinfo está ultrapassada e não deve ser usada; usar collectgarbage("count") ao invés dela.

B.7.3 Mudanças na API

A seguir são listadas as alterações na API C introduzidas em Lua 5.1:

- as funções luaopen_* (para abrir bibliotecas) não podem ser chamadas diretamente, como uma função C comum. Elas devem ser chamadas através de Lua, como uma função Lua;
- a função lua_open foi substituída por *lua_newstate* para permitir que o usuário defina uma função de alocação de memória. Pode-se usar *luaL_newstate* da biblioteca padrão para criar um estado com uma função de alocação padrão (baseada em realloc);
- a função lua_open foi substituída por *lua_newstate* para permitir que o usuário defina uma função de alocação de memória. Pode-se usar *luaL_newstate* da biblioteca padrão para criar um estado com uma função de alocação padrão (baseada em realloc);
- a função luaL_openlib foi substituída por *luaL_register*;
- a função luaL_checkudata agora dispara um erro quando o valor fornecido não é um objeto userdata do tipo esperado (em Lua 5.0 ela retornava NULL).

B.8 Sintaxe completa de Lua

A função luaL_checkudata agora dispara um erro quando o valor fornecido não é um objeto userdata do tipo esperado (em Lua 5.0 ela retornava NULL).

```
chunk ::= {stat [`;´]} [laststat[`;´]]

block ::= chunk

stat ::=  varlist1 `=´ explist1  |
   functioncall  |
   do block end  |
   while exp do block end  |
   repeat block until exp  |
   if exp then block {elseif exp then block}[else block] end |
   for Name `=´ exp `,´ exp [`,´ exp] do block end  |
   for namelist in explist1 do block end  |
   function funcname funcbody  |
   local function Name funcbody  |
   local namelist [`=´ explist1]

laststat ::= return [explist1]  |  break

funcname ::= Name {`.´ Name} [`:´ Name]

varlist1 ::= var {`,´ var}

var ::=  Name  |  prefixexp `[´ exp `]´  |  prefixexp `.´ Name

namelist ::= Name {`,´ Name}

explist1 ::= {exp `,´} exp

exp ::= nil  |  false  |  true  |  Number  |  String  |  `...´  |
     function  |  prefixexp  |  tableconstructor  |
     exp binop |  exp  |  unop exp

prefixexp ::= var  |  functioncall  |  `(´ exp `)´

functioncall ::=  prefixexp args  |  prefixexp `:´ Name args

args ::=  `(´ [explist1] `)´  |  tableconstructor  |  String

function ::= function funcbody

funcbody ::= `(´ [parlist1] `)´ block end

parlist1 ::= namelist [`,´ `...´]  |  `...´

tableconstructor ::= `{´ [fieldlist] `}´

fieldlist ::= field {fieldsep field} [fieldsep]

field ::= `[´ exp `]´ `=´ exp  |  Name `=´ exp  |  exp

fieldsep ::= `,´  |  `;´

binop ::= `+´  |  `-´  |  `*´  |  `/´  |  `^´  |  `%´  |  `..´  |
       `<´  |  `<=´  |  `>´  |  `>=´  |  `==´  |  `~=´  |
       and  |  or

unop ::= `-´  |  not  |  `#´
```

Anexo C
(informativo)

Base de conectores

Esta base de conectores pode ser importada por qualquer documento NCL 3.0.

```
<!--

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/NCL3.0/connectorBases/causalConnBase.ncl
Author: TeleMidia Laboratory
Revision: 19/09/2006

-->

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

<head>
<connectorBase>

<!-- OnBegin -->

<causalConnector id="onBeginStart">
  <simpleCondition role="onBegin"/>
  <simpleAction role="start"/>
</causalConnector>

<causalConnector id="onBeginStop">
  <simpleCondition role="onBegin"/>
  <simpleAction role="stop"/>
</causalConnector>

<causalConnector id="onBeginPause">
  <simpleCondition role="onBegin"/>
  <simpleAction role="pause"/>
</causalConnector>

<causalConnector id="onBeginResume">
  <simpleCondition role="onBegin"/>
  <simpleAction role="resume"/>
</causalConnector>

<causalConnector id="onBeginSet">
  <connectorParam name="var"/>
  <simpleCondition role="onBegin"/>
  <simpleAction role="set" value="$var"/>
</causalConnector>
```

```
<!-- OnEnd -->

<causalConnector id="onEndStart">
  <simpleCondition role="onEnd"/>
  <simpleAction role="start"/>
</causalConnector>

<causalConnector id="onEndStop">
  <simpleCondition role="onEnd"/>
  <simpleAction role="stop"/>
</causalConnector>

<causalConnector id="onEndPause">
  <simpleCondition role="onEnd"/>
  <simpleAction role="pause"/>
</causalConnector>

<causalConnector id="onEndResume">
  <simpleCondition role="onEnd"/>
  <simpleAction role="resume"/>
</causalConnector>

<causalConnector id="onEndSet">
  <connectorParam name="var"/>
  <simpleCondition role="onEnd"/>
  <simpleAction role="set" value="$var"/>
</causalConnector>


<!-- OnMouseSelection -->

<causalConnector id="onSelectionStart">
  <simpleCondition role="onSelection"/>
  <simpleAction role="start" />
</causalConnector>

<causalConnector id="onSelectionStop">
  <simpleCondition role="onSelection"/>
  <simpleAction role="stop" />
</causalConnector>

<causalConnector id="onSelectionPause">
  <simpleCondition role="onSelection"/>
  <simpleAction role="pause" />
</causalConnector>

<causalConnector id="onSelectionResume">
  <simpleCondition role="onSelection"/>
  <simpleAction role="resume" />
</causalConnector>

<causalConnector id="onSelectionSetVar">
  <connectorParam name="var" />
  <simpleCondition role="onSelection"/>
  <simpleAction role="set" value="$var"/>
</causalConnector>

<!-- OnKeySelection -->

<causalConnector id="onKeySelectionStart">
  <connectorParam name="keyCode"/>
```

```
  <simpleCondition role="onSelection" key="$keyCode"/>
  <simpleAction role="start"/>
</causalConnector>

<causalConnector id="onKeySelectionStop">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <simpleAction role="stop"/>
</causalConnector>

<causalConnector id="onKeySelectionPause">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <simpleAction role="pause"/>
</causalConnector>

<causalConnector id="onKeySelectionResume">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <simpleAction role="resume"/>
</causalConnector>

<causalConnector id="onKeySelectionSetVar">
  <connectorParam name="keyCode"/>
  <connectorParam name="var"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <simpleAction role="set" value="$var"/>
</causalConnector>

<!-- OnBeginAttribution -->

<causalConnector id="onBeginAttributionStart">
  <simpleCondition role="onBeginAttribution"/>
  <simpleAction role="start"/>
</causalConnector>

<causalConnector id="onBeginAttributionStop">
  <simpleCondition role="onBeginAttribution"/>
  <simpleAction role="stop"/>
</causalConnector>

<causalConnector id="onBeginAttributionPause">
  <simpleCondition role="onBeginAttribution"/>
  <simpleAction role="pause"/>
</causalConnector>

<causalConnector id="onBeginAttributionResume">
  <simpleCondition role="onBeginAttribution"/>
  <simpleAction role="resume"/>
</causalConnector>

<causalConnector id="onBeginAttributionSet">
  <connectorParam name="var"/>
  <simpleCondition role="onBeginAttribution"/>
  <simpleAction role="set" value="$var"/>
</causalConnector>

<!-- OnEndAttribution -->
```

```
<causalConnector id="onEndAttributionStart">
  <simpleCondition role="onEndAttribution"/>
  <simpleAction role="start"/>
</causalConnector>

<causalConnector id="onEndAttributionStop">
  <simpleCondition role="onEndAttribution"/>
  <simpleAction role="stop"/>
</causalConnector>

<causalConnector id="onEndAttributionPause">
  <simpleCondition role="onEndAttribution"/>
  <simpleAction role="pause"/>
</causalConnector>

<causalConnector id="onEndAttributionResume">
  <simpleCondition role="onEndAttribution"/>
  <simpleAction role="resume"/>
</causalConnector>

<causalConnector id="onEndAttributionSet">
  <connectorParam name="var"/>
  <simpleCondition role="onEnd"/>
  <simpleAction role="set" value="$var"/>
</causalConnector>


<!-- OnBegin multiple actions -->

<causalConnector id="onBeginStartStop">
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
     <simpleAction role="start"/>
     <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStartPause">
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
     <simpleAction role="start"/>
     <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStartResume">
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
     <simpleAction role="start"/>
     <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStartSet">
  <connectorParam name="var"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
     <simpleAction role="start"/>
     <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>
```

```
<causalConnector id="onBeginStopStart">
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStopPause">
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStopResume">
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginStopSet">
  <connectorParam name="var"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginSetStart">
  <connectorParam name="var"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginSetStop">
  <connectorParam name="var"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginSetPause">
  <connectorParam name="var"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>
```

```
<causalConnector id="onBeginSetResume">
  <connectorParam name="var"/>
  <simpleCondition role="onBegin"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>


<!-- OnEnd multiple actions -->

<causalConnector id="onEndStartStop">
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStartPause">
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStartResume">
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStartSet">
  <connectorParam name="var"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStopStart">
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStopPause">
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>
```

```
<causalConnector id="onEndStopResume">
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndStopSet">
  <connectorParam name="var"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndSetStart">
  <connectorParam name="var"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndSetStop">
  <connectorParam name="var"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="stet" value="$var"/>
    <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndSetPause">
  <connectorParam name="var"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndSetResume">
  <connectorParam name="var"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>


<!-- OnMouseSelection multiple actions -->

<causalConnector id="onSelectionStartStop">
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
```

```
    <simpleAction role="start"/>
    <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStartPause">
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStartResume">
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStartSet">
  <connectorParam name="var"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStopStart">
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStopPause">
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStopResume">
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStopSet">
  <connectorParam name="var"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>
```

```
<causalConnector id="onSelectionSetStart">
  <connectorParam name="var"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
     <simpleAction role="set" value="$var"/>
     <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionSetStop">
  <connectorParam name="var"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
     <simpleAction role="stet" value="$var"/>
     <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionSetPause">
  <connectorParam name="var"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
     <simpleAction role="set" value="$var"/>
     <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onSelectionSetResume">
  <connectorParam name="var"/>
  <simpleCondition role="onSelection"/>
  <compoundAction operator="seq">
     <simpleAction role="set" value="$var"/>
     <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>


<!-- OnKeySelection multiple actions -->

<causalConnector id="onKeySelectionStartStop">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
     <simpleAction role="start"/>
     <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStartPause">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
     <simpleAction role="start"/>
     <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStartResume">
```

```
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStartSet">
  <connectorParam name="var"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStopStart">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStopPause">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStopResume">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStopSet">
  <connectorParam name="var"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionSetStart">
  <connectorParam name="var"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="start"/>
```

```
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionSetStop">
  <connectorParam name="var"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionSetPause">
  <connectorParam name="var"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionSetResume">
  <connectorParam name="var"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>


<!-- OnBeginAttribution multiple actions -->

<causalConnector id="onBeginAttributionStartStop">
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStartPause">
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStartResume">
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>
```

```
<causalConnector id="onBeginAttributionStartSet">
  <connectorParam name="var"/>
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStopStart">
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStopPause">
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStopResume">
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionStopSet">
  <connectorParam name="var"/>
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionSetStart">
  <connectorParam name="var"/>
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionSetStop">
  <connectorParam name="var"/>
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionSetPause">
```

```
  <connectorParam name="var"/>
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onBeginAttributionSetResume">
  <connectorParam name="var"/>
  <simpleCondition role="onBeginAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>


<!-- OnEndAttribution multiple actions -->

<causalConnector id="onEndAttributionStartStop">
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStartPause">
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStartResume">
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStartSet">
  <connectorParam name="var"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="start"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStopStart">
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="start"/>
  </compoundAction>
```

```
</causalConnector>

<causalConnector id="onEndAttributionStopPause">
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStopResume">
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionStopSet">
  <connectorParam name="var"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="set" value="$var"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionSetStart">
  <connectorParam name="var"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionSetStop">
  <connectorParam name="var"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="stet" value="$var"/>
    <simpleAction role="stop"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionSetPause">
  <connectorParam name="var"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="pause"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionSetResume">
  <connectorParam name="var"/>
  <simpleCondition role="onEndAttribution"/>
  <compoundAction operator="seq">
    <simpleAction role="set" value="$var"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>
```

```
<!--Miscellaneous-->

<causalConnector id="onKeySelectionStopResizePauseStart">
  <connectorParam name="width"/>
  <connectorParam name="height"/>
  <connectorParam name="left"/>
  <connectorParam name="top"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="setWidth" value="$width"/>
    <simpleAction role="setHeight" value="$height"/>
    <simpleAction role="setLeft" value="$left"/>
    <simpleAction role="setTop" value="$top"/>
    <simpleAction role="pause"/>
    <simpleAction role="start"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onEndResizeResume">
  <connectorParam name="left"/>
  <connectorParam name="top"/>
  <connectorParam name="width"/>
  <connectorParam name="height"/>
  <simpleCondition role="onEnd"/>
  <compoundAction operator="seq">
    <simpleAction role="setLeft" value="$left"/>
    <simpleAction role="setTop" value="$top"/>
    <simpleAction role="setWidth" value="$width"/>
    <simpleAction role="setHeight" value="$height"/>
    <simpleAction role="resume"/>
  </compoundAction>
</causalConnector>

<causalConnector id="onKeySelectionStopSetPauseStart">
  <connectorParam name="bounds"/>
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <compoundAction operator="seq">
    <simpleAction role="stop"/>
    <simpleAction role="set" value="$bounds"/>
    <simpleAction role="pause"/>
    <simpleAction role="start"/>
  </compoundAction>
</causalConnector>
```

Bibliografia

[1] ITU Recommendation J.201:2004, Harmonization of declarative content format for interactive television applications

[2] ARIB STD-B24:2004, Data coding and transmission specifications for digital broadcasting

[3] Cascading Style Sheets, Cascading Style Sheets, level 2, Bert Bos, Håkon Wium Lie, Chris Lilley, Ian Jacobs. W3C Recommendation 12. Maio de 1998, disponínel em <http://www.w3.org/TR/REC-CSS2>

[4] DVB-HTML, Perrot P. DVB-HTML - An Optional Declarative Language within MHP 1.1, EBU Technical Review. 2001

[5] Namespaces in XML, Namespaces in XML, W3C Recommendation. Janeiro de 1999

[6] NCM Core, Soares L.F.G; Rodrigues R.F. Nested Context Model 3.0: Part 1 – NCM Core, Technical Report, Departamento de Informática PUC-Rio. Maio de 2005, ISSN: 0103-9741. Também disponível em <http://www.ncl.org.br>

[7] NCL Digital TV Profiles, Soares L.F.G; Rodrigues R.F. Part 8 – NCL (Nested Context Language) Digital TV Profiles, Technical Report, Departamento de Informática PUC-Rio, No. 35/06. Outubro de 2006, ISSN: 0103-9741. Também disponível em <http://www.ncl.org.br>

[8] NCL Live Editing Commands, Soares L.F.G; Rodrigues R.F; Costa, R.R.; Moreno, M.F. Part 9 – NCL Live Editing Commands. Technical Report, Departamento de Informática PUC-Rio, No. 36/06. Dezembro de 2006, ISSN: 0103-9741. Também disponível em <http://www.ncl.org.br>

[9] NCL-Lua, Cerqueira, R,; Sant'Anna, F. Nested Context Model 3.0: Part 11 – Lua Scripting Language for NCL, Technical Report, Departamento de Informática PUC-Rio. Maio de 2007, ISSN: 0103-9741.

[10] NCL Main Profile, Soares L.F.G; Rodrigues R.F; Costa, R.R. Nested Context Model 3.0: Part 6 – NCL (Nested Context Language) Main Profile, Technical Report, Departamento de Informática PUC-Rio. Maio de 2005, ISSN: 0103-9741. Também disponível em <http://www.ncl.org.br>

[11] RDF, Resource Description Framework (RDF) Model and Syntax Specification, Ora Lassila and Ralph R. Swick. W3C Recommendation. 22 de fevereiro de 1999. Disponível em <http://www.w3.org/TR/REC-rdf-syntax/>

[12] SMIL 2.1 Specification, SMIL 2.1 - Synchronized Multimedia Integration Language – SMIL 2.1 Specification, W3C Recommendation. Dezembro de 2005

[13] XHTML 1.0, XHTML™ 1.0 2º Edition - Extensible HyperText Markup Language, W3C Recommendation, Agosto de 2002

[14] Programming in Lua, Segunda Edição, de Roberto Ierusalimschy et al. Março de 2006, ISBN 85-903798-2-5.

[15] ACAP, Advanced Application Platform (ACAP), ATSC Standard: Document A/101. Agosto de 2005