NORMA
BRASILEÑA

ABNT NBR 15606-2

Segunda edición
27.05.2011

Válida a partir de
27.06.2011

Televisión digital terrestre — Codificación de datos y especificaciones de transmisión para radiodifusión digital
Parte 2: Ginga-NCL para receptores fijos y móviles – Lenguaje de aplicación XML para codificación de aplicacione

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

ICS 33.160.01

ISBN 978-85-07-02877-2

Número de referencia
ABNT NBR 15606-2:2007
294 páginas

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

Impreso en Brasil

Índice

Página

Prefacio

La Associação Brasileira de Normas Técnicas (ABNT) es el Fórum Nacional de Normalización. Las Normas Brasileñas, cuyo contenido es responsabilidad de los Comités Brasileños (ABNT/CB), de los Organismos de Normalización Sectorial (ABNT/ONS) y de las Comisiones de Estudios Especiales (ABNT/CEE), son elaboradas por Comisiones de Estudio (CE), formadas por representantes de sus sectores implicados de los que forman parte: productores, consumidores y neutrales (universidades, laboratorios y otros).

Los Documentos Técnicos ABNT se elaboran de acuerdo con las reglas de Directivas ABNT, Parte 2.

La Associação Brasileira de Normas Técnicas (ABNT) llama la atención sobre la posibilidad de que algunos de los elementos de este documento pueden ser objeto de derechos de patente. La ABNT no debe ser considerada responsable por la identificación de cualesquiera derechos de patente.

La ABNT NBR 15606-2 fue elaborada por la Comisión de Estudio Especial de Televisión Digital (ABNT/CEE-00:001.85). El Proyecto circuló en Consulta Nacional según Edicto nº 09, de 06.09.2007 a 05.11.2007, con el número de Proyecto 00:001.85-006/2.

En caso que surja cualquier duda con relación a la interpretación de la versión en español siempre deben prevalecer las prescripciones de la versión en portugués

Esta Norma está basada en los trabajos del Fórum del Sistema Brasileiro de Televisão Digital Terrestre, según establece el Decreto Presidencial nº 5.820, de 29/06/2006.

La ABNT NBR 15606, bajo el título general “Televisión digital terrestre – Codificación de datos y especificaciones de transmisión para radiodifusión digital”, está previsto que contenga las siguientes partes:

— Parte 1: Codificación de datos;

— Parte 2: Ginga-NCL para receptores fijos y móviles – Lenguaje de aplicación XML para codificación de aplicaciones;

— Parte 3: Especificación de transmisión de datos;

— Parte 4: Ginga-J – Ambiente para la ejecución de aplicaciones procedurales;

— Parte 5: Ginga-NCL para receptores portátiles – Lenguaje de aplicación XML para codificación de aplicaciones;

— Parte 6: JavaDTV 1.3;

— Parte 7: Ginga-NCL – Directrices operacionales para las ABNT NBR 15606-2 y ABNT NBR 15606-5.

Esta segunda edición incorpora la Enmienda 1 de 27.05.2011 y cancela y sustituye la edición anterior (ABNT NBR 15606-2:2007).

Esta versión en español es equivalente a la ABNT NBR 15606-2:2011.

Esta versión en español fue publicada en 27.05.2011.

Introducción

La Associação Brasileira de Normas Técnicas (ABNT) llama la atención sobre el hecho de que la exigencia de conformidad con este documento ABNT puede involucrar el uso de una patente relativa a NCL, conforme se menciona en 5.1.

La ABNT no se posiciona respecto a evidencias, validez y alcance de ese derecho de patente.

El propietario de este derecho de patente aseguró a la ABNT que él está preparado para negociar licencias sobre términos y condiciones razonables y no discriminatorias con los solicitantes. Sobre esto, hay una declaración del propietario de esta patente registrada con la ABNT. Informaciones pueden obtenerse con:

Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Transferência de Tecnologia

Rua Marquês de São Vicente, 225 – Gávea, 22451-900 - Rio de Janeiro - RJ - Brasil.

La ABNT quiere resaltar la posibilidad de que algunos de los elementos de este documento ABNT pueden ser objeto de otros derechos de patente además de los identificados arriba. La ABNT no debe ser considerada responsable por la identificación de cualesquiera derechos de patente.

Esta Norma estandariza un lenguaje de aplicación XML que permite a los autores escribir presentaciones multimedia interactivas. Este componente de la ABNT NBR 15606 forma parte de las especificaciones de codificación de datos para el Sistema Brasileño de Televisión Digital Terrestre (SBTVD) y comprende la especificación del lenguaje utilizado por la máquina de presentación Ginga-NCL del *middleware* SBTVD, llamado Ginga.

A través de este lenguaje, denominado NCL (*Nested Context Language* – Lenguaje de Contextos Anidados), un autor puede describir el comportamiento temporal de una presentación multimedia, asociar *hyperlinks* (interacción del usuario) a objetos de media, definir alternativas para presentación (adaptación) y describir el formato de la presentación en múltiples dispositivos.

Esta Norma está primordialmente destinada a las entidades que están especificando terminales y/o estándares basados en el Ginga. También está destinada a los desarrolladores de aplicaciones que usan las funcionalidades del Ginga y de sus API. El *middleware* Ginga tiene como objetivo garantizar la interoperabilidad de las aplicaciones en diferentes implementaciones de plataformas que lo soportan.

Las aplicaciones Ginga están clasificadas en dos categorías, dependiendo de si la aplicación inicialmente procesada tiene contenido de naturaleza declarativa o imperativa. Estas categorías de aplicaciones son llamadas aplicaciones declarativas y aplicaciones procedurales, respectivamente. Los ambientes de aplicación son igualmente clasificados en dos categorías, dependiendo de si procesan aplicaciones declarativas o procedurales, siendo entonces llamados Ginga-NCL y Ginga-J, respectivamente.

Es importante observar que a una implementación únicamente Ginga-NCL o únicamente Ginga-J, ya sea en receptores fijos o móviles, les está prohibida la reivindicación de cualquier tipo de conformidad con el SBTVD. Esto garantiza que el Ginga ofrezca perfiles siempre compatibles con versiones anteriores.

Esta Norma no especifica la forma cómo los ambientes de aplicación deben implementarse en un receptor en conformidad. Un fabricante de receptores puede implementar los dos ambientes como un único subsistema; alternativamente, los ambientes pueden implementarse como subsistemas distintos, con interfaces internos bien definidos entre los ambientes.

Televisión digital terrestre — Codificación de datos y especificaciones de transmisión para radiodifusión digital
Parte 2: Ginga-NCL para receptores fijos y móviles – Lenguaje de aplicación XML para codificación de aplicaciones

1 Alcance

Esta parte de la ABNT NBR 15606 especifica un lenguaje de aplicación XML denominada NCL *(Nested Context Language),* el lenguaje declarativo del *middleware* Ginga, la codificación y la transmisión de datos para radiodifusión digital.

2 Referencias normativas

Los documentos indicados a continuación son indispensables para la aplicación de este documento. Para las referencias fechadas, se aplican solamente las ediciones citadas. Para las referencias sin fecha, se aplican las ediciones más recientes del documento citado (incluyendo enmiendas).

ABNT NBR 15601, *Televisión digital terrestre – Estándar de transmisión*

ABNT NBR 15603-2:2007*, Televisión digital terrestre – Multiplexación y servicios de información (SI) – Parte 2: Estructura de datos y definiciones de la información básica de SI*

ABNT NBR 15606-1, *Televisión digital terrestre – Codificación de datos y especificaciones de transmisión para radiodifusión digital – Parte 1: Codificación de datos*

ABNT NBR 15606-3, *Televisión digital terrestre – Codificación de datos y especificaciones de transmisión para radiodifusión digital – Parte 3: Especificación de transmisión de datos*

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 Términos y definiciones

Para los efectos de esta parte de la ABNT NBR 15606, se aplican los siguientes términos y definiciones.

3.1
ambiente de aplicación
contexto o ambiente de *software* en el cual se procesa una aplicación

3.2
ambiente de aplicación declarativa
ambiente que soporta el procesamiento de aplicaciones declarativas

NOTA Un formateador *(user agent)* NCL es un ejemplo de ambiente de aplicación declarativa.

3.3
ambiente de aplicación procedural
ambiente que soporta el procesamiento de aplicaciones procedurales

3.4
API DON
API que define la estructura lógica de un documento XML y la forma de acceder, o manejar, un documento XML

NOTA Esta API es una interfaz independiente de plataformas y lenguajes y sigue el Modelo DOM (*Document Object Model*).

3.5
aplicación
información que expresa un conjunto específico de comportamientos observables

3.6
aplicación declarativa
aplicación que utiliza principalmente, y como punto de partida, información declarativa para expresar su comportamiento

NOTA Una instancia de documento NCL es un ejemplo de aplicación declarativa.

3.7
aplicación híbrida
aplicación híbrida declarativa o aplicación híbrida procedural

3.8
aplicación híbrida declarativa
aplicación declarativa que contiene contenido de objeto activo

NOTA Un documento NCL con un Java Xlet embutido es un ejemplo de aplicación híbrida declarativa.

3.9
aplicación híbrida procedural
aplicación procedural con contenido declarativo

NOTA Un Java Xlet que crea y causa la exhibición de una instancia de documento NCL es un ejemplo de aplicación híbrida procedural.

3.10
aplicación nativa
función intrínseca implementada por una plataforma receptora

NOTA Una exhibición en *closed caption* es un ejemplo de aplicación nativa.

3.11
aplicación procedural
aplicación que utiliza principalmente, y como punto de partida, informaciones procedurales para expresar su comportamiento

NOTA Un programa en Java es un ejemplo de una aplicación procedural.

3.12
almacenamiento persistente
memoria disponible que puede ser leída o grabada por una aplicación y puede ser mantenida por más tiempo que el tiempo de vida de la misma aplicación

NOTA El almacenamiento persistente puede ser volátil o no volátil.

3.13
atributo
parámetro para representar la naturaleza de una propiedad

3.14
atributo de un elemento
propiedad de un elemento XML

3.15
audio principal
audio básico
flujo básico de audio cuya component_tag es igual a 0x10 para el receptor *full-seg* y 0x83 ó 0x85 para el receptor *one-seg*

3.16
autor
persona que escribe documentos NCL

3.17
canal de interactividad
canal de retorno
mecanismo de comunicación que suministra conexión entre el receptor y un servidor remoto

3.18
carácter
"letra" específica u otro símbolo identificable

EJEMPLO “A”

3.19
carrusel de datos
método que envía cualquier conjunto de datos en forma cíclica, para que esos datos se puedan obtener, vía radiodifusión, en un intervalo de tiempo tan largo como sea necesario

[ISO/IEC 13818-6:2001]

3.20
ciclo de vida de una aplicación
caracteriza el período de tiempo, desde el momento en que la aplicación es cargada hasta el momento en que es destruída

3.21
codificación de caracteres
mapeo entre un valor de entrada entero y el carácter textual, representado por ese mapeo

3.22
contenido de objeto activo
tipo de contenido que toma la forma de un programa ejecutable
NOTA Un Xlet Java compilado es un ejemplo de contenido de objeto activo.

3.23
contenido NCL
conjunto de informaciones que consiste en un documento NCL y en un grupo de datos, incluyendo objetos (de media o de ejecución), que acompañan el documento NCL

3.24
digital storage media command and control
DSM-CC
método de control que suministra acceso a un archivo o flujo en servicios digitales interactivos

[ISO/IEC 13818-6:2001]

3.25
document type definition
DTD
declaración que describe un tipo de documento XML

3.26
ECMA*Script*
lenguaje de programación definido en la ECMA 262

3.27
elemento
unidad de estructuración del documento delimitada por tags

NOTA Un elemento es usualmente delimitado por una *tag* inicial y una *tag* final, excepto un elemento vacío que es delimitado por una *tag* de elemento vacío.

3.28
elemento *property*
elemento NCL que define un nombre de propiedad y su valor asociado

3.29
exhibidor de media
media player
componente de un ambiente de aplicación que decodifica o ejecuta un tipo específico de contenido

3.30
evento
ocurrencia en el tiempo que puede ser instantánea o tener duración mensurable

3.31
exhibidor de media
media player
componente identificable de un ambiente de aplicación que descodifica o ejecuta un tipo específico de contenido

3.32
eXtensible HTML
XHTML
versión extendida del HTML como aplicación XML

NOTA En la especificación XHTML, un documento HTML es reconocido como aplicación XML.

3.33
herramienta de autoría
herramienta para ayudar a los autores a crear documentos NCL

3.34
fuente
mecanismo que permite la renderización específica de un carácter

EJEMPLO Tiresias, 12 puntos.

NOTA En la práctica, un formato de fuente incorpora aspectos de la codificación de un carácter.

3.35
formateador NCL
componente de software responsable por recibir la especificación de un documento NCL y controlar su presentación, intentando garantizar que las relaciones entre los objetos de media, especificados por el autor, sean respetados

NOTA Renderizador *(renderer)* de documentos, agente del usuario (*user agent*) y exhibidor son otros nombres que se utilizan con el mismo significado del formateador de documentos.

3.36
flujo de transporte
se refiere a la sintaxis del flujo de transporte MPEG-2 para empaquetado y multiplexación de video, audio y señales de datos en sistemas de radiodifusión digital

3.37
flujo elemental
elementary stream
ES
flujo básico que contiene datos de video, audio, o datos privados

NOTA Un único flujo elemental se transporta en una secuencia de paquetes PES con un y sólo un identificador *(stream_id)*.

3.38
gestor de aplicaciones
entidad responsable por la administración del ciclo de vida de las aplicaciones y que administra las aplicaciones, funcionando tanto en la máquina de presentación como en la máquina de ejecución

3.39
identificador de paquete
PID
valor entero único utilizado para asociar los flujos elementales de un programa, tanto en un flujo de transporte único como en multiprograma

3.40
información de servicio SI
datos que describen programas y servicios

3.41
informaciones específicas del programa
program specific information
PSI
datos normativos necesarios para demultiplexar los flujos de transporte y regenerar los programas

3.42
interfaz de programación de la aplicación
API
bibliotecas de software que ofrecen acceso uniforme a los servicios del sistema

3.43
lenguaje de marcación
formalismo que describe una clase de documentos que emplean marcación para delinear la estructura, apariencia u otros aspectos del documento

3.44
lenguaje de *script*
lenguaje utilizado para describir un contenido de objeto activo incorporado en documentos NCL y en documentos HTML

3.45
localizador
identificador que suministra una referencia a una aplicación o recurso

3.46
máquina de presentación
subsistema en un receptor que analiza y presenta aplicaciones declarativas, con contenidos como audio, video, gráficos y texto, con base en reglas definidas en la máquina de presentación

NOTA Una máquina de presentación es responsable por el control del comportamiento de la presentación y por iniciar otros procesos en respuesta a entradas del usuario y otros eventos.

EJEMPLO Navegador HTML y formateador NCL.

3.47
máquina de ejecución
subsistema en un receptor que evalúa y ejecuta aplicaciones procedurales, compuestas por instrucciones en lenguaje de computadora, contenido de medias asociadas y otros datos

NOTA Una máquina de ejecución se puede implementar con un sistema operativo, compiladores de lenguaje de computadora, interpretadores e interfaces de programación de aplicaciones (API), que una aplicación procedural puede utilizar para presentar contenido audiovisual, interactuar con el usuario o ejecutar otras tareas que no sean evidentes al usuario.

EJEMPLO Ambiente de *software* JavaTV, utilizando lenguaje de programación Java e interpretador *bytecode*, API JavaTV y máquina virtual Java para ejecución del programa.

3.48
método
función asociada a un objeto autorizado para manejar los datos del objeto

3.49
nudo NCL
elemento <media>, <context>, <body> o <switch> de NCL

3.50
normal play time
NPT
coordenada temporal absoluta que representa la posición en un flujo

3.51
objeto de media
colección de pedazos de datos identificados por nombre que puede representar un contenido de media o un programa escrito en lenguaje específico

3.52
objeto de media visual
cualquier objeto de media NCL, representado por el elemento <media>, cuyo contenido, cuando el objeto es iniciado, genera una presentación visual

3.53
perfil
especificación de una clase de capacidades, ofreciendo diferentes niveles de funcionalidades en un receptor

3.54
perfil *one-seg*
caracteriza el servicio que puede ser recibido por un sintonizador de banda estrecha (430 KHz) y por lo tanto con ahorro en el consumo de batería

NOTA El perfil *one-seg* también es conocido como perfil portátil.

3.55
perfil *full-seg*
caracteriza el servicio que necesita necesariamente un demodulador de banda ancha (5,7 MHz) para ser recibido

NOTA Dependiendo de las configuraciones de transmisión y de funcionalidad específicas del receptor, puede ser recibido en movimiento o apenas por receptores fijos, aunque sin el beneficio del ahorro de energía. La resolución del video transmitido puede ser o no de alta definición.

3.56
plug-in
conjunto de funcionalidades que se puede agregar a una plataforma genérica para suministrar funcionalidad adicional

3.57
plataforma receptora
plataforma
hardware, sistema operativo y bibliotecas de *software* nativas del receptor, elegidos por el fabricante

3.58
recurso
objeto de datos o un servicio de la red que es identificado unívocamente

3.59
sistema de archivos local
sistema de archivos suministrado por la plataforma receptora local

3.60
tiempo de vida de una aplicación
período de tiempo entre el momento en que una aplicación se carga y el momento en que se destruye

3.61
uniform resource identifier
URI
método de encaminamiento que permite el acceso a objetos en una red

3.62
user agent
cualquier programa que interpreta un documento NCL

NOTA Un *user agent* puede exhibir un documento, intentando garantizar que las relaciones especificadas por el autor entre objetos de media sean respetadas, pronunciarlo en audio sintetizado, convertirlo en otro formato etc.

3.63
usuario
persona que interactúa con un formateador para visualizar, oír o utilizar de otra forma un documento NCL

3.65
usuario final
individuo que opera o interactúa con un receptor

4 Abreviaturas

Para los efectos de esta parte de la ABNT NBR 15606, se aplican las siguientes 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*
NIT	*Network Information Table*
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 Brasileño de Televisión Digital Terrestre
SMIL	*Synchronized Multimedia Integration Language*
TS	*Transport Stream*
UCS	*Universal (Coded) Character Set*
URI	*Universal Resource Identifier*
URL	*Universal Resource Locator*
XHTM	*eXtensible HTML*
L XML	*Extensible Markup Language*
W3C	*World-Wide Web Consortium*

5 Arquitectura Ginga

5.1 Ginga main modules

El universo de las aplicaciones Ginga se puede dividir en un conjunto de aplicaciones declarativas y un conjunto de aplicaciones procedurales. Una aplicación declarativa es aquella donde el tipo del contenido de la entidad inicial es declarativo. Por otro lado, una aplicación procedural es aquella cuyo tipo de contenido de la entidad inicial es procedural. Una aplicación declarativa pura es aquella en la cual el contenido de todas las entidades es del tipo declarativo. Una aplicación procedural pura es aquella en la cual el contenido de todas las entidades es del tipo procedural. Una aplicación híbrida es aquella cuyo conjunto de entidades posee tanto contenido del tipo declarativo como procedural. Una aplicación Ginga no necesita ser puramente declarativa o procedural.

En particular, las aplicaciones declarativas frecuentemente utilizan *scripts,* cuyo contenido es de modalidad procedural. Además, una aplicación declarativa puede hacer referencia a un código Java TV Xlet incorporado. Del mismo modo, una aplicación procedural puede hacer referencia a una aplicación declarativa, conteniendo, por ejemplo, contenido gráfico, o puede construir e iniciar la presentación de aplicaciones con contenido declarativo. Por lo tanto, ambos tipos de aplicación Ginga pueden utilizar las facilidades de los ambientes de aplicación declarativo y procedural.

Ginga-NCL es un subsistema lógico del sistema Ginga responsable por el procesamiento de documentos NCL[1)]. Un componente clave del Ginga-NCL es la máquina de interpretación del contenido declarativo (formateador NCL). Otros módulos importantes son el exhibidor *(user agent)* XHTML, que incluye interpretadores CSS y ECMAScript, y la máquina de presentación Lua, que es responsable por la interpretación de los *scripts* Lua (ver Anexo B).

Ginga-J es un subsistema lógico del sistema Ginga responsable por el procesamiento de contenidos activos. Un componente clave del ambiente de aplicación procedural es la máquina de ejecución del contenido procedural, compuesta por una máquina virtual Java.

Decodificadores de contenidos comunes sirven tanto para las aplicaciones procedurales con respecto a las declarativas que necesitan decodificar y presentar tipos comunes de contenido como PNG, JPEG, MPEG y otros formatos. El núcleo común Ginga (Ginga *Common Core)* está compuesto por los decodificadores de contenido comunes y por procedimientos para lograr contenidos transportados en flujos de transporte (*transport streams*) MPEG-2 y a través del canal de interactividad. El núcleo común Ginga también debe soportar obligatoriamente el modelo conceptual de exhibición, tal como se describe en la ABNT NBR 15606-1.

---

1) NCL es marca registrada y su especificación es propiedad intelectual de la PUC-Rio (INPI Departamento de Transferencia Tecnológica - No. 0007162-5; 20/12/2005).

La arquitectura (ver Figura 1) y facilidades Ginga fueron proyectadas para ser aplicadas en sistemas de transmisión y recepción terrestres de radiodifusión. Adicionalmente, la misma arquitectura y facilidades se pueden aplicar a sistemas que utilizan otros mecanismos de transporte de datos (como sistemas de televisión vía satélite o por cable).

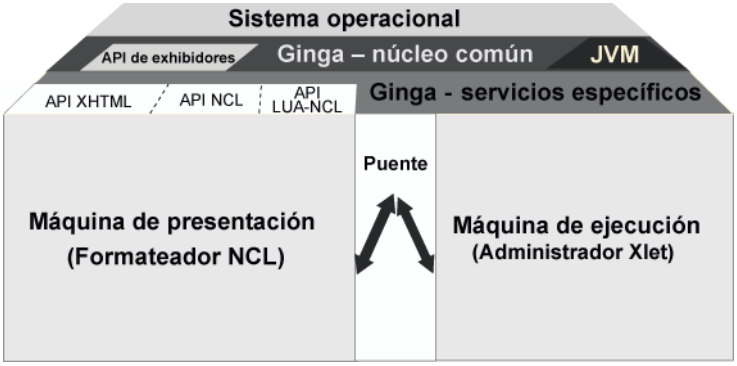

Figura 1 — Arquitectura Ginga

5.2 Interacción con el ambiente nativo

En general, Ginga es independiente de cualesquiera aplicaciones nativas que pueden también optar por utilizar el plano gráfico. Eso incluye, pero no se limita a, aplicaciones como: *closed caption*, mensajes del sistema de acceso condicional (CA), menús del receptor y guías nativas de programación.

Las aplicaciones nativas pueden tener prioridad sobre las aplicaciones Ginga. El *closed caption* y los mensajes de emergencia deben tener obligatoriamente prioridad sobre el sistema Ginga.

Algunas aplicaciones nativas, como el *closed caption,* representan un caso especial en el cual la aplicación nativa puede estar activa por largos períodos en conjunto con las aplicaciones Ginga.

6 Interoperabilidad con ambientes declarativos definidos en otros sistemas de televisión digital - Objetos XHTML incorporados en presentaciones NCL

6.1 NCL como lenguaje cola

Todas las máquinas de presentación de los tres principales sistemas de televisión digital utilizan un lenguaje basado en XHTML.

XHTML es un lenguaje declarativo basado en media, lo que significa que su estructura es definida por las relaciones entre objetos XHTML (documentos XHTML u objetos insertos en documentos XHTML) que están incorporados en el contenido de la media del documento. XHTML puede, entonces, ser clasificada como lenguaje de marcación: un formalismo que describe una clase de documentos que emplean marcación para delinear la estructura, apariencia y otros aspectos de los documentos.

Las relaciones de referencia definidos por los enlaces XHTML son el foco de ese lenguaje declarativo. Otros tipos de relaciones, como relaciones de sincronización espacio-temporal y relaciones alternativas (adaptación de la media), son comúnmente definidos a través de un lenguaje imperativo (por ejemplo, ECMAScript).

Diferentemente de XHTML o HTML, NCL define una separación bien delimitada entre el contenido y la estructura de un documento (o aplicativo), probando un control no intrusivo del enlace entre el contenido y su presentación y layout.

El foco del lenguaje declarativo NCL es más amplio que el ofrecido por la XHTML. La sincronización espacio-temporal, definida genéricamente por los enlaces NCL; adaptabilidad, definida por los elementos *switch* y *descriptor switch* de la NCL; y soporte a múltiples dispositivos de exhibición, definidos por regiones NCL, es el foco de ese lenguaje declarativo. La interacción del usuario se trata sólo como un caso particular de sincronización temporal.

Como la NCL tiene una separación más exacta entre el contenido y la estructura, la misma no define ninguna media en sí. Al contrario, define la cola que pega la media en presentaciones multimedia.

Un documento NCL sólo define cómo los objetos de media son estructurados y relacionados en el tiempo y espacio. Como un lenguaje de cola, la misma no restringe o prescribe los tipos de contenido de los objetos de media. En ese sentido, se pueden tener objetos de imagen (GIF, JPEG etc.), de video (MPEG, MOV etc.), de audio (MP3, WMA etc.), de texto (TXT, PDF etc.), de ejecución (Xlet, Lua etc.), entre otros, como objetos de media NCL. Qué objetos de media son soportados depende de los exhibidores de media que están acoplados al formateador NCL (exhibidor NCL). Uno de esos exhibidores es el decodificador/exhibidor MPEG-4, normalmente implementado en *hardware* en el receptor de televisión digital. De esa forma, el video y el audio MPEG-4 principal se tratan como todos los demás objetos de media que pueden estar relacionados utilizando NCL.

Otro objeto de media NCL que debe ser obligatoriamente soportado es el objeto de media basado en XHTML. La NCL no reemplaza, sino incorpora documentos (u objetos) basados en XHTML. Como ocurre con otros objetos de media, qué lenguaje basado en XHTML tiene soporte en un formateador NCL es una elección de implementación y, por lo tanto, depende de qué navegador XHTML, incorporado en el formateador NCL, actúa como exhibidor de esa media.

Como consecuencia, es posible tener navegadores BML, DVB-HTML y ACAP-X individualmente incorporados en un exhibidor de documento NCL. Es posible, además, tenerlos todos. Es igualmente posible recibir el código de un programa navegador a través de la difusión de datos e instalarlo como *plug-in* (normalmente un *plug-in* Java).

También es posible tener un navegador genérico implementado y, en caso de ser necesario, recibir la parte complementar (específica) como un *plug-in,* para convertir el exhibidor XHTML genérico en un exhibidor específico de uno de los diversos estándares de navegador DTV.

En último caso, un documento NCL puede ser reducido para contener un sólo objeto de media XHTML. En ese caso, el exhibidor del documento NCL actúa casi como un navegador XHTML, es decir, como cualquier otro navegador de los estándares mencionados.

No importa el caso, la implementación del navegador XHTML debe ser una consecuencia de las siguientes exigencias:

— interoperabilidad;

— robustez;

— conformidad con las normas del W3C;

— rechazo de contenido no conforme;

— compatibilidad con el modelo de seguridad Ginga;

— minimización de la redundancia con la tecnología Ginga-J existente;

— minimización de la redundancia con las facilidades NCL existentes;

— mecanismos necesarios de control del layout del contenido;

— soporte a diferentes razones de aspecto de las unidades de exhibición *(pixels).*

Para brindar soporte a las facilidades del navegador XHTML definidas por otros estándares DTV, se recomienda que todas las especificaciones SBTVD relacionadas con la difusión de datos soporten también las facilidades definidas para tales navegadores, como el transporte de eventos de flujos (*stream events*), por ejemplo.

Aunque un navegador XHTML deba ser obligatoriamente soportado, se recomienda que la utilización de elementos XHTML para definir relaciones (incluso *links* XHTML) sea evitada en la autoría de documentos NCL. Se recomienda que la autoría con base en la estructura sea priorizada por razones conocidas y ampliamente divulgadas en la literatura.

Durante la exhibición del contenido de objetos de media se generan varios eventos (ver 7.2.8). Algunos ejemplos son la presentación de parte del contenido de un objeto de media, la selección de parte del contenido de un objeto etc. Los eventos pueden generar acciones sobre otros objetos de media, como iniciar o terminar sus presentaciones. Por lo tanto, los eventos deben ser obligatoriamente relatados por los exhibidores de media al formateador NCL que, a su vez, puede generar acciones a ser aplicadas a ésos u otros exhibidores. Ginga-NCL define la API (ver Sección 8) de un adaptador con el alcance de estandarizar la interfaz entre el formateador Ginga-NCL y cada exhibidor específico.

Para que cualquier exhibidor de media, en particular un navegador XHTML, sea acoplado al formateador Ginga-NCL, debe soportar obligatoriamente la API de los adaptadores. Así, para algunos exhibidores de media, incluso navegadores XHTML, un módulo adaptador puede ser necesario para alcanzar la integración.

Para edición en vivo, el Ginga-NCL también define eventos de flujo NCL para ofrecer soporte a los eventos generados en vivo sobre flujos de media, en particular sobre el flujo de video del programa principal. Esos eventos son una generalización del mismo concepto encontrado en otras normas, como, por ejemplo, los *bevents* de BML. Aunque un navegador XHTML deba ser obligatoriamente soportado, se recomienda evitar la utilización de elementos XHTML para definir relaciones (incluso eventos de flujo) durante la creación de documentos NCL por la misma razón, es decir, se recomienda que la autoría con base en la estructura sea priorizada por razones conocidas y ampliamente divulgadas en la literatura.

6.2 Formato de contenido XHTML

Formatos comunes de contenido deben ser obligatoriamente adoptados para la producción e intercambio de contenido multimedia, como definido en la ABNT NBR 15606-1. Además, en el ambiente de aplicación declarativa también se exige la especificación de formatos comunes de contenidos XHTML para las aplicaciones de televisión interactiva.

NOTA Esta Norma sigue la ITU Recommendation J.201 para identificar las funcionalidades comunes entre los ambientes de aplicación declarativa para aplicaciones de televisión interactiva especificadas por DVB-HTML, ACAP-X y BML.

Conviene que se especifiquen los elementos comunes y API en el nivel sintáctico de objetos de media XHTML incorporados en aplicaciones NCL para ayudar a los autores en la creación de contenido XHTML.

Cualquier implementación de objeto de media XHTML de acuerdo con esta Norma debe obligatoriamente dar soporte a, por lo menos, todas las marcaciones XML y propiedades de hojas de estilo comunes a los servicios básicos BML ("perfil terminal fijo"), ACAP-X y DVB-HTML, como definido en 6.3. Se recomienda que facilidades de objetos nativos ECMAScript y API DOM, comunes a los servicios básicos BML ("perfil terminal fijo"), ACAP-X y DVB-HTML, también tengan soporte.

6.3 Armonización del formato de contenido XHTML

6.3.1 Marcaciones XML

NOTA Objetos de media NCL basados en XHTML siguen la recomendación W3C *"Modularization of XHTML"* y sus marcaciones XML se definen en la ITU Recommendation J.201.

Los módulos comunes a marcaciones XML pueden 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*.

Las colecciones de atributo XHTML se definen de acuerdo con la Tabla 1. Las marcaciones XML comunes a los estándares servicios básicos BML (“perfil de terminal fijo”), ACAP-X y DVB-HTML, que deben ser obligatoriamente soportadas por cualquier implementación, se listan en la Tabla 2, en conjunto con las extensiones Ginga obligatorias.

Tabla 1 — Colecciones de atributos

Nombre de la colección	Atributos en la colección	Condición del 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	

Tabla 2 — Elementos de marcación XML en común

Módulo		Elemento	Condición del elemento	Atributo	Condición del 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	–		

Tabla 2 (continuación)

Módulo		Elemento	Condición del elemento	Atributo	Condición del 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	–		

Tabla 2 (continuación)

Módulo		Elemento	Condición del elemento	Atributo	Condición del 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	–		

Tabla 2 (continuación)

Módulo	Elemento	Condición del elemento	Atributo	Condición del 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	%I1 8n.attrib	Requerido
			id	–
			type	Requerido
			media	Requerido
			title	–
Style attribute		Requerido		
Link	link	Requerido		
Base	base	–		

6.3.2 Hojas de estilo

Las propiedades de hojas de estilo (CSS) se listan en la Tabla 3.

Tabla 3 — Propiedades de hojas de estilo en común

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	----

Las propiedades de hojas de estilo comunes a los estándares servicios básicos BML, ACAP-X y DVB-HTML, que deben ser obligatoriamente soportadas por cualquier implementación, se listan en la Tabla 4.

Tabla 4 — Propiedades de hojas de estilo CSS 2 en común

Propiedad	Condición de la propiedad
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	–

Tabla 4 (continuación)

Propiedad	Condición de la propiedad
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	–

Tabla 4 (continuación)

Propiedad	Condición de la propiedad
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	–

Tabla 4 (continuación)

Propiedad	Condición de la propiedad
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	–

Las siguientes restricciones se deberán aplicar obligatoriamente a las propiedades de exhibición:

— solamente elementos de bloque se pueden aplicar para <p>, , <body>, <input> y <object>;

— solamente valores definidos en el mismo elemento HTML se pueden aplicar para
, <a> y .

Además, las siguientes restricciones se deberán aplicar obligatoriamente a las propiedades de posición:

— solamente valores absolutos se pueden aplicar para <p>, , <input> y <object>;

— solamente valores estáticos se pueden aplicar para
, y <a>.

Los selectores CSS comunes a los estándares servicios básicos BML, ACAP-X y DVB-HTML, que deben ser obligatoriamente soportados por cualquier implementación, son los siguientes:

— *universal;*

— *type;*

— *class;*

— *id;*

— *dynamic* (*active and :focus*).

6.3.3 ECMAScript

Una vez implementada, es altamente recomendable que la máquina ECMAScript dé soporte a los objetos nativos en común de los estándares de servicios básicos BML, ACAP-X y DVB-HTML, listados en la Tabla 5. Como restricción, los tipos numéricos sólo soportan operaciones enteras.

Tabla 5 — Objetos nativos en común

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, item 1, . . .)	Requerido
	new Array(item0, item 1, . . .)	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

Tabla 5 (continuación)

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)	–

Tabla 5 (continuación)

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 [,	Requerido
	new Date([year, month [, date [, hours [,	Requerido
	Date(value)	–
	new Date(value)	–
	Date.parse(string)	–
	Date.UTC([year [, month [, date [, hours	–
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

Tabla 5 (continuación)

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()	–

Dependiendo de la implementación del *middleware,* es posible tener funciones ECMAScript mapeadas para las API suministradas por el Ginga-J, logrando acceso a algunos recursos de la plataforma receptora y facilidades Ginga. En ese caso, se recomienda que la API suministrada en el ECMAScript siga la misma especificación presentada para el ambiente procedural del Ginga-J.

6.3.4 API DOM

Las API DOM nivel 1 son las siguientes:

— DOM Exception;

— DOMImplementation;

— DocumentFragment;

— Document;

— Node;

— NodeList;

— NamedNodeMap;

— CharacterData;

— Attr;

— Element;

— Text;

— Comment.

Las API DOM nivel 1, cuando son implementadas, es altamente recomendable que sigan las API comunes del DOM nivel 1 para para servicios básicos BML, ACAP-X y DVB-HTML, listadas en la Tabla 6.

Tabla 6 — API DOM nivel 1 en común

Interfaz	Atributo/Método	Condición de la operación
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 - Lenguaje declarativo XML para especificación de presentaciones multimedia interactivas

7.1 Lenguajes modulares y perfiles de lenguajes

7.1.1 Módulos NCL

El abordaje modular ha sido utilizado en varios lenguajes recomendados por el W3C.

Módulos son colecciones de elementos, atributos y valores de atributos XML semánticamente relacionados que representan una unidad funcional. Los módulos se definen en conjuntos coherentes. Esa coherencia se expresa por medio de la asociación de un mismo *namespace* a los elementos de esos módulos.

NOTA *Namespaces* se discuten en Namespaces in XML:1999.

Un perfil de lenguaje es una combinación de módulos. Los módulos son atómicos, es decir, no se pueden subdividir cuando son incluidos en un perfil de lenguaje. Además, la especificación de un módulo puede incluir un conjunto de requisitos para integración, con el cual los perfiles de lenguaje, que incluyen el módulo, deben ser compatibles obligatoriamente.

NCL fue especificada de forma modular, permitiendo la combinación de sus módulos en perfiles de lenguaje. Cada perfil puede agrupar un subconjunto de módulos NCL, permitiendo la creación de lenguajes orientados hacia las necesidades específicas de los usuarios. Además, los módulos y perfiles NCL se pueden combinar con módulos definidos en otros lenguajes, permitiendo la incorporación de características de la NCL en esos lenguajes y viceversa.

Normalmente, hay un perfil de lenguaje que incorpora casi todos los módulos asociados a un único *namespace.* Ése es el caso del perfil *Lenguaje NCL.*

Otros perfiles de lenguaje se pueden especificar como subconjuntos de un perfil mayor o incorporar una combinación de módulos asociados a diferentes *namespaces.* Ejemplos del primer caso son los perfiles TVD Básico (perfil BDTV) y TVD Avanzado (perfil EDTV) de la NCL.

Subconjuntos de los módulos del perfil Lenguaje NCL utilizados en la definición de los perfiles TVD Básico y TVD Avanzado se definen para ajustar el lenguaje a las características del ambiente de radiodifusión de televisión, con sus varios dispositivos de presentación: Aparato de televisión, dispositivos móviles etc.

NOTA Un abordaje análogo también se encuentra en otros lenguajes (SMIL 2.1 Specification:2005 y XHTML 1.0:2002).

El principal alcance de la conformidad con perfiles de lenguaje es aumentar la interoperabilidad. Los módulos obligatorios se definen de tal forma que cualquier documento, especificado de conformidad con un perfil de lenguaje, da como resultado una presentación razonable cuando se presenta en un perfil distinto de aquél para el cual fue especificado. El formateador de documentos, soportando el conjunto de módulos obligatorios, ignoraría todos los otros elementos y atributos desconocidos.

NOTA Renderizador de documentos, agente del usuario y exhibidor son otros nombres atribuidos al formateador de documentos.

La versión NCL 3.0 revisa las funcionalidades contenidas en la NCL 2.3 (NCL Main 13 Profile:2005) y se divide en 15 áreas funcionales, que se dividen nuevamente en módulos. A partir de las 15 áreas funcionales, 14 se utilizan para definir los perfiles TVD Avanzado y TVD Básico. Dos áreas funcionales tienen módulos con la misma semántica definida por SMIL 2.0. Las 14 áreas funcionales utilizadas y sus módulos correspondientes son:

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 *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) *Trasition Effects*

Módulo *TransitionBase*

Módulo *Trasition*

14) *Meta-Information*

7.1.2 Identificadores para módulos y perfiles de lenguaje de la NCL 3.0

Se recomienda que cada perfil NCL declare explícitamente el URI del *namespace* que será usado para identificarlo.

Documentos creados en perfiles de lenguaje que incluyen el módulo *Structure* de NCL pueden ser asociados con el tipo *MIME* “application/x-ncl+xml”. Los documentos que utilizan el tipo *MIME* “application/x-ncl+xml” deben obligatoriamente estar de conformidad con el lenguaje hospedero.

Los identificadores de *namespace* XML para el conjunto completo de módulos, elementos y atributos NCL 3.0 están contenidos en el siguiente *namespace:* http://www.ncl.org.br/NCL3.0/.

Cada módulo NCL posee un identificador único a él asociado. Los identificadores de los módulos NCL 3.0 deben estar de acuerdo obligatoriamente con la Tabla 7.

Módulos también pueden ser identificados colectivamente. Las siguientes colecciones de módulos se definen:

— módulos utilizados por el perfil Lenguaje NCL 3.0: http://www.ncl.org.br/NCL3.0/LanguageProfile;

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

— módulos utilizados por el perfil DTV Avanzado NCL 3.0: http://www.ncl.org.br/NCL3.0/EDTVProfile;

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

Tabla 7 — Identificadores de los 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/ConnectorCausal Expression
ConnectorAssessmentExpression	http://www.ncl.org.br/NCL3.0/ConnectorAssessmentExpression
ConnectorBase	http://www.ncl.org.br/NCL3.0/ConnectorBase
ConnectorCommonPart	http://www.ncl.org.br/NCL3.0/ConnectorCommonPart
ContentControl	http://www.ncl.org.br/NCL3.0/ContentControl
Context	http://www.ncl.org.br/NCL3.0/Context
Descriptor	http://www.ncl.org.br/NCL3.0/Descriptor
DescriptorControl	http://www.ncl.org.br/NCL3.0/DescriptorControl
EntityReuse	http://www.ncl.org.br/NCL3.0/EntityReuse
ExtendedEntityReuse	http://www.ncl.org.br/NCL3.0/ExtendedEntityReuse
Import	http://www.ncl.org.br/NCL3.0/Import
Layout	http://www.ncl.org.br/NCL3.0/Layout
Linking	http://www.ncl.org.br/NCL3.0/Linking
Media	http://www.ncl.org.br/NCL3.0/Media
MediaContentAnchor	http://www.ncl.org.br/NCL3.0/MediaContentAnchor
KeyNavigation	http://www.ncl.org.br/NCL3.0/KeyNavigation
PropertyAnchor	http://www.ncl.org.br/NCL3.0/PropertyAnchor
Structure	http://www.ncl.org.br/NCL3.0/Structure
SwitchInterface	http://www.ncl.org.br/NCL3.0/SwitchInterface
TestRule	http://www.ncl.org.br/NCL3.0/TestRule
TestRuleUse	http://www.ncl.org.br/NCL3.0/TestRuleUse
Timing	http://www.ncl.org.br/NCL3.0/Timing
TransitionBase	http://www.ncl.org.br/NCL3.0/TransitionBase
Transition	http://www.ncl.org.br/NCL3.0/Transition
Metainformation	http://www.ncl.org.br/NCL3.0/MetaInformation

Tres módulos SMIL [SMIL 2.1 Specification, 2005] se usaron como base para la definición de los módulos NCL Transition y Metainformation. Los identificadores de estos módulos SMIL 2.0 se presentan en la Tabla 8.

Tabla 8 — Identificadores de los módulos de SMIL 2.0

Módulos	Identificadores
BasicTransitions	http://www.w3.org/2001/SMIL20/BasicTransitions
TransitionModifiers	http://www.w3.org/2001/SMIL20/TransitionModifiers
Metainformation	http://www.w3.org/2001/SM I L20/Metainformation

7.1.3 Informaciones sobre versiones de la NCL

Las siguientes instrucciones de procesamiento se deben incluir obligatoriamente en un documento NCL. Éstas identifican documentos NCL que contengan sólo los elementos definidos en esta Norma, y la versión NCL con la cual el documento está de acuerdo.

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

<ncl id="qualquer string" xmlns="http://www.ncl.org.br/NCL3.0/profileName">

El atributo *id* del elemento <ncl> puede recibir cualquier cadena de caracteres como valor, que sea compatible con el tipo NCName [*Namespaces* in XML: 1999]. O sea, puede tener como valor cualquier cadena de caracteres que comience con una letra o subrayado ("_") y que contenga sólo letras, dígitos, "." y "_".

El número de versión de una especificación NCL consiste en un número principal y otro secundario, separados por un punto. Los números son representados como una cadena de caracteres formada por números decimales, en la cual los ceros a la izquierda se suprimen. El número de versión inicial del estándar es 3.0.

Nuevas versiones de la NCL deben ser obligatoriamente publicadas de acuerdo con la siguiente política de versión:

— si los receptores compatibles con versiones más antiguas aún pueden recibir un documento con base en la especificación revisada, con relación a correcciones de error o por motivos operativos, la nueva versión de la NCL debe ser obligatoriamente publicada con el número secundario actualizado;

— si los receptores compatibles con versiones más antiguas no pueden recibir un documento basado en las especificaciones revisadas, el número principal debe ser obligatoriamente actualizado.

Una versión específica está definida bajo el URI http://www.ncl.org.br/NCL3.0/profileName, en que el número de la versión se escribe inmediatamente después de la sigla “NCL”.

El nombre del perfil (*profileName*) en URI debe obligatoriamente ser *EDTVProfile* (Perfil TVD Avanzado), *BDTVProfile* (Perfil TVD Básico) o *CausalConnectorProfile* (Perfil Conector causal).

7.2 Módulos NCL

7.2.1 Observaciones generales

Las principales definiciones de cada uno de los módulos NCL 3.0 presentes en los perfiles NCL TVD Básico y TVD Avanzado se proveen en 7.2.2 a 7.2.15.

La definición completa de los módulos NCL 3.0, utilizando XML *Schema,* se presenta en el Anexo A. Cualquier ambigüedad encontrada en este texto puede ser aclarada por medio de la consulta a los esquemas XML.

Tras discutir cada módulo, una tabla es presentada para indicar los elementos del módulo y sus atributos. Para determinado perfil, los atributos y contenidos (elementos hijos) de los elementos pueden ser definidos en el propio módulo o en el perfil del lenguaje que agrupa los módulos. El valor de un atributo no puede contener comillas (“). Cuando el valor es una string, puede ser cualquier cadena de caracteres que sea compatible con el tipo NCName [*Namespaces* in XML: 1999]. O sea, el valor puede ser cualquier cadena de caracteres que comience con una letra o subrayado ("_") y que contenga sólo letras, dígitos, "." y "_".

Las tablas descritas en 7.2.2 a 7.2.15 muestran los atributos y contenidos que vienen del perfil NCL DTV Avanzado, además de los definidos en los mismos módulos. Las tablas descritas en 7.3.3 muestran los atributos y contenidos que vienen del perfil NCL TVD Básico, además de los definidos en los mismos módulos. Los atributos de elementos que son obligatorios están subrayados. En las tablas, se emplean los siguientes símbolos: (?) opcional (cero o una ocurrencia), (|) o, (*) cero o más ocurrencias, (+) una o más ocurrencias. El orden de los elementos hijos no se especifica en las tablas.

7.2.2 Área funcional *Structure*

El área funcional *Structure* tiene un sólo módulo, llamado *Structure,* que define la estructura básica de un documento NCL. Este área define el elemento raíz, denominado <ncl>, el elemento <head> y el elemento <body>, siguiendo la terminología adoptada por otros estándares W3C. El elemento <body> de un documento NCL es tratado como un nudo de contexto NCM (NCMCore:2005).

En el NCM, el modelo conceptual de datos de la NCL, un nudo puede ser un contexto, un switch o un objeto de media. Todos los nudos NCM están representados por elementos NCL correspondientes. Los nudos de contexto, conforme definido en 7.2.4, contienen otros nudos y eslabones NCM.

Casi todos los elementos NCL pueden poseer un atributo *id*, que puede recibir como valor cualquier cadena de caracteres, que sea compatible con el tipo NCName [*Namespaces* in XML: 1999]. O sea, puede tener como valor cualquier cadena de caracteres que comience con una letra o subrayado ("_") y que contenga sólo letras, dígitos, "." y "_". El atributo *id* identifica unívocamente un elemento dentro de un documento. Su valor es un identificador XML. En particular, el elemento <ncl> debe obligatoriamente definir un atributo *id*. En el elemento <body> ese atributo *id* es opcional.

El atributo *title* de <ncl> ofrece información adicional sobre el elemento. Los valores del atributo *title* pueden ser utilizados por agentes de usuarios de varias formas.

El atributo *xmlns* declara un namespace XML, es decir, declara la colección primaria de construcciones XML utilizada por el documento. El valor del atributo es el URL *(Uniform Resource Locator),* que identifica donde el *namespace* está oficialmente definido. Tres valores se permiten para el atributo *xmlns:* http://www.ncl.org.br/NCL3.0/EDTVProfile y http://www.ncl.org.br/NCL3.0/BDTVProfile, para los perfiles TVD Avanzado y Básico, respectivamente, y http://www.ncl.org.br/NCL3.0/CausalConnectorProfile, para el perfil Conector Causal. Un formateador NCL debe obligatoriamente saber que la localización de los esquemas para tales *namespaces* es, por *default,* respectivamente:

http://www.ncl.org.br/NCL3.0/profiles/NCL30EDTV.xsd,

http://www.ncl.org.br/NCL3.0/profiles/NCL30BDTV.xsd, y
http://www.ncl.org.br/NCL3.0/profiles/NCL30CausalConnector.xsd

Los elementos hijos de <head> y <body> se definen en otros módulos NCL. Es altamente recomendable que los elementos hijos de <head> se declaren en el orden siguiente: importedDocumentBase?, ruleBase?, transitionBase?, regionBase*, descriptorBase?, connectorBase?, meta*, metadata*.

Los elementos de este módulo, sus elementos hijos y sus atributos deben estar de acuerdo obligatoriamente con la Tabla 9.

Tabla 9 — Módulo *Structure* extendido

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

7.2.3 Área funcional *Layout*

El área funcional *Layout* tiene un único módulo, llamado *Layout*, lo cual especifica elementos y atributos que pueden definir cómo los objetos serán inicialmente presentados dentro de regiones de dispositivos de salida. De hecho, este módulo puede definir valores iniciales para propiedades NCL homónimas definidas en los elementos <media>, <body> y <context> (ver 7.2.4).

Un elemento <regionBase>, que puede ser declarado en el elemento <head> del documento NCL, define un conjunto de elementos <región>, cada cual pudiendo contener otro conjunto de elementos <región> anidados, y así en adelante, recurrentemente.

El elemento <regionBase> puede tener un atributo *id.* Elementos <region> deben tener obligatoriamente un atributo *id.* Como esperado, el atributo *id* identifica en forma unívoca un elemento dentro de un documento.

Cada elemento <regionBase> está asociado a una clase de dispositivos donde sucederá la presentación. Para identificar la asociación, el elemento <regionBase> define el atributo *device*, que puede tener los valores: "systemScreen (i)" o "systemAudio(i)", donde *i* es un número entero superior o igual a 1. La clase elegida define las variables globales del ambiente: system.screenSize(i), system.screenGraphicSize(i) y system.audioType(i), como se define en la Tabla 12 (ver 7.2.4). Cuando el atributo no es especificado, la presentación debe obligatoriamente ser hecha en el mismo dispositivo que ejecuta el formateador NCL.

NOTA 1 Existen dos tipos diferentes de clases de dispositivos: activa y pasiva. En una clase activa, un dispositivo es capaz de ejecutar las funciones de exhibidores de media. De un dispositivo de exhibición que se registra en una clase del tipo "pasiva" no se exige la capacidad de ejecutar las funciones de exhibidores de media. Éste sólo debe ser capaz de presentar el mapa de memoria de video que le es pasado y exhibir las muestras de audio pasada por otro dispositivo. En el SBTVD, systemScreen (1) y systemAudio (1) se reservan para clases del tipo pasiva; systemScreen (2) y systemAudio (2) se reservan para las clases del tipo activa.

NOTA 2 El elemento <regionBase> que define una clase pasiva también puede tener un atributo *region*. Este atributo usado para identificar un elemento <region> en otra <regionBase> asociada a una clase activa donde está registrado el dispositivo que genera el mapa de memoria de video enviado para los dispositivos de la clase pasiva; en la región especificada, el mapa de memoria también debe ser exhibido.

Se recomienda que la interpretación del anidado de las regiones dentro de un <regionBase> sea realizada por el *software* responsable por la orquestación de la presentación del documento (es decir, el formateador N CL). Para los efectos de esta Norma, un primer nivel de anidado debe ser interpretado obligatoriamente como si fuese el área del dispositivo donde ocurrirá la presentación; el segundo nivel como ventanas (por ejemplo, áreas de presentación en la pantalla) del área-padre; y los otros niveles como regiones dentro de esas ventanas.

Una <region> también puede definir los siguientes atributos: *title, left, right, top, bottom, height, width* e *zIndex.* Todos esos atributos poseen el significado usual W3C.

La posición de una región, conforme es especificada por sus atributos *top, bottom, left* y *right,* es siempre relativa a la geometría-padre, que es definida por el elemento <region> padre o por el área total del dispositivo, en el caso de las regiones en el primer nivel de anidado. Los valores de los atributos pueden ser valores porcentuales no negativos, o unidades de pixels. Para valores en pixels, el autor puede omitir el calificador de unidad "px" (por ejemplo, "100"). Para valores porcentuales, por otro lado, el símbolo "%" debe ser indicado obligatoriamente (por ejemplo, "50%"). El porcentaje es siempre relativo al ancho del padre, en el caso de las definiciones de los atributos *right, left* and *width,* y a la altura del padre, para las definiciones de los atributos *bottom, top* y *height.*

Los atributos *top* y *left* son los atributos primarios de posicionamiento de la región. Posicionan el canto superior izquierdo de la región en la distancia especificada del margen superior izquierdo de la región-padre (o margen superior izquierdo del dispositivo, en el caso de la región en el primer nivel de anidado). Algunas veces, puede ser útil ajustar explícitamente los atributos *bottom* y *right*. Sus valores establecen la distancia entre el ángulo inferior derecho de la región y el ángulo inferior derecho de la región-padre (o el margen inferior derecho del dispositivo, en el caso de la región en el primer nivel de anidado) (ver Figura 2).

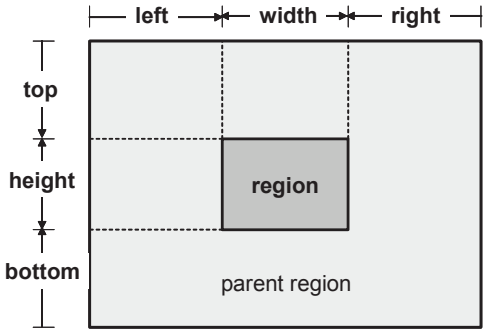

Figura 2 — Atributos de posicionamiento de la región

Con relación a los tamaños de la región, cuando se especifican declarando los atributos *width* y *height* usando la anotación "%", el tamaño de la región es relativo al tamaño de la geometría de su padre, como mencionado anteriormente. Los tamaños declarados como valores absolutos en pixels mantienen tales valores absolutos. El tamaño intrínseco de una región es igual al tamaño de la geometría lógica del padre. Eso significa que, si una región anidada no especifica cualquier posicionamiento o valores de tamaño, debe ser obligatoriamente asumido que tiene la misma posición y valores de tamaño que su región-padre. En particular, cuando una región de primer nivel no especifica cualquier posicionamiento o valores de tamaño, debe ser obligatoriamente asumida como teniendo toda el área de presentación del dispositivo.

Cuando el usuario especifica informaciones sobre *top, bottom* y *height* para una misma <region>, pueden ocurrir inconsistencias espaciales. En este caso, los valores de *top* y *height* deben obligatoriamente preceder el valor de *bottom.* De forma análoga, cuando el usuario especifica valores inconsistentes para los atributos *left, right* y *width* de la <region>, los valores de *left* y *width* se deben utilizar obligatoriamente para calcular un nuevo valor de *right.* Cuando cualquiera de esos atributos no se especifica y no puede tener su valor calculado desde otros atributos, ese valor debe ser obligatoriamente heredado del valor correspondiente definido en el padre de esa <region>. Otra restricción es que las regiones-hijas no pueden quedar fuera del área establecida por sus regiones-padres.

El atributo *zIndex* especifica la prioridad de sobreposición de la región. Regiones con mayores valores de *zIndex* deben ser obligatoriamente empiladas en el tope de regiones con valores de *zIndex* inferiores. Si dos presentaciones generadas por los elementos A y B tuvieron el mismo nivel de apilamiento, caso la exhibición de un elemento B comience después de la exhibición de un elemento A, la presentación de B debe obligatoriamente ser sobrepuesta a la presentación de A (orden temporal); por otro lado, si la exhibición de los elementos comenzar al mismo tiempo, la orden de sobreposición es elegida arbitrariamente por el formateador. Cuando omitido, el valor por default del zIndex es igual a 0 (cero).

El módulo *Layout* también define el atributo *region,* que es utilizado por un elemento <descriptor> (ver 7.2.6) para referirse a un elemento <region> de *Layout.*

Los elementos de este módulo, sus elementos-hijos y sus atributos deben estar de acuerdo obligatoriamente con la Tabla 10.

Tabla 10 — Módulo *Layout* extendido

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

7.2.4 Área funcional *Components*

El área funcional *Components* se divide en dos módulos, denominados *Media y Context.*

El módulo Medía define los tipos básicos de objetos de media. Para definir objetos de media, este módulo define el elemento <media>. Cada objeto de media tiene dos atributos principales, además del atributo id*: src,* que define un URI del contenido del objeto, y *type,* que define el tipo de objeto.

Los URI *(Uniform Resource Identifier)* deben estar de acuerdo obligatoriamente con la Tabla 11.

Tabla 11 — URI permitidos

Esquema	Parte específica del esquema	Uso
file:	///file_path/#fragment_identifier	Para archivos locales
http:	//server_identifier/file_path/#fragment_identifier	Para archivos remotos buscados por el canal de interactividad usando el protocolo http
https:	//server_identifier/file_path/#fragment_identifier	Para archivos remotos buscados por el canal de interactividad usando el protocolo https
rtsp:	//server_identifier/file_path/#fragment_identifier	Para flujos (*streams*) obtenidos por el canal de interactividad usando el protocolo rtsp
rtp:	//server_identifier/file_path/#fragment_identifier	Para flujos (*streams*) logrados por el canal de interactividad usando el protocolo rtp
Ncl-mirror:	//media_element_identifier	Para un flujo de contenido idéntico a uno que esté en presentación por otro elemento de media
sbtvd-ts:	//program_number.component_tag	Para flujos elementales recibidos por el flujo de transporte (TS)

Un URI absoluto contiene todas las informaciones necesarias para localizar su recurso. Los URI relativos también son permitidos. URI relativos son direcciones incompletas que se aplican a una URI base para completar la localización. Las partes omitidas son el esquema URI, el servidor y, también, en algunos casos, parte del camino del URI.

El beneficio principal de utilizar URI relativas es la posibilidad de mover o copiar hacia otros locales los documentos y directorios contenidos en el URI, sin exigir el cambio de los valores de los atributos URI dentro de los documentos. Eso es especialmente interesante cuando se transportan documentos del servidor (normalmente radiodifusores) hacia los receptores. Los caminos relativos del URI son típicamente utilizados como un medio rápido de localización de archivos de media almacenados en el mismo directorio del documento NCL actual, o en un directorio próximo a él. Frecuentemente consisten sólo en el nombre del archivo (opcionalmente con un identificador de fragmento dentro del archivo). También pueden tener un camino relativo de directorio antes del nombre del archivo.

Conviene enfatizar que las referencias para los recursos de flujos de video o audio no pueden, obligatoriamente, causar la ocurrencia de sintonización (*tuning*). Las referencias que implican sintonización para acceder a un recurso deben obligatoriamente portarse como si el recurso estuviera indisponible.

Los valores permitidos para el atributo *type* dependen del perfil NCL y deben seguir obligatoriamente el formato MIME *Media Types* (o, simplemente, *mimetypes*). Un *mimetype* es una cadena de caracteres que define la clase de media (audio, video, imagen, texto, aplicación) y un tipo de codificación de media (como jpeg, mpeg etc.). Los *mimetypes* pueden ser registrados o informales. Los *mimetypes* registrados son controlados por la IANA

(*Internet Assigned Numbers Authority*). Los *mimetypes* informales no son registrados por la IANA, pero se definen pactadamente; normalmente tienen una “x-“ antes del nombre del tipo de media.

Cinco tipos especiales son definidos: application/x-ginga-NCL (o application/x-ncl-NCL), application/x-ginga-NCLua (o application/x-ncl-NCLua), application/x-ginga-NCLet (o application/x-ncl-NCLet), application/x-ginga-settings (application/x-ncl-settings) y application/x-ginga-time (o application/x-ncl-time).

El tipo application/x-ginga-NC debe ser obligatoriamente aplicado a elementos <media> con código NCL (así, una aplicación NCL puede ser embutida en otra aplicación NCL). El tipo application/x-ginga-NCLua debe ser obligatoriamente aplicado a elementos <media> con código procedural Lua como contenido (ver Sección 10). El tipo application/x-ginga-NCLet debe ser obligatoriamente aplicado a elementos <media> con código procedural Xlet como contenido (ver Sección 11).

El tipo application/x-ginga-settings debe ser obligatoriamente aplicado a un elemento <media> especial (puede existir solamente uno en un documento NCL) cuyas propiedades son variables globales definidas por el autor del documento o variables de ambiente reservadas, que pueden ser manejadas por el procesamiento del documento NCL. La Tabla 12 establece las variables ya definidas y su semántica.

Tabla 12 — Variables de ambiente

Grupo	Variable	Semántica	Valores posibles
system • grupo de variables mantenidas por el sistema receptor; • se pueden leer, pero no pueden tener sus valores alterados por una aplicación NCL, un procedimiento Lua o un procedimiento Xlet; • programas nativos en el receptor pueden alterar los valores de las variables; • persisten durante todo el tiempo de vida de un receptor	system.language	Lenguaje de audio	ISO 639-1 code
	system.caption	Lenguaje de *caption*	ISO 639-1
	system.subtitle	Lenguaje de leyenda	ISO 639-1
	system.returnBitRate(i)	Tasa de bits total del canal de interactividad (i) en Kbps	real
	system.screenSize	Tamaño de la pantalla del dispositivo de exhibición, en (líneas, pixels/línea), cuando una clase no es definida	(entero, entero)
	system.screenGraphicSize	Resolución configurada para el plano gráfico de la pantalla del dispositivo de exhibición, en (líneas, pixels/línea), cuando una clase no es definida	(entero, entero)
	system.audioType	Tipo de audio del dispositivo exhibición, cuando una clase no es definida	“mono” | “stereo” | “5.1”
	system.screenSize (i)	Tamaño de la pantalla de la clase (i) de dispositivos de exhibición, en (líneas, pixels/línea)	(entero, entero)
	system.screenGraphicSize (i)	Resolución configurada para el plano gráfico de la pantalla de la clase (i) de dispositivos de exhibición, en (líneas, pixels/línea)	(entero, entero)
	system.audioType(i)	Tipo de audio de la clase (i) de dispositivos de exhibición	“mono” | “stereo” | “5,1”
	system.devNumber(i)	Número de dispositivos de exhibición registrados en la clase (i)	entero
	system.classType(i)	Tipo de la clase (i)	(“passive” | “ative”)
	system.info(i)	Lista de exhibidores de media de la clase (i) de dispositivos de exhibición	string
	system.classNumber	Número de clases de dispositivos de exhibición definidas	entero
	system.CPU	Desempeño de la CPU en MIPS, considerando toda su capacidad para ejecución de aplicaciones	real
	system.memory	Espacio de la memoria mínimo ofrecido para aplicaciones, en Mbytes	entero

Tabla 12 (continuación)

Grupo	Variable	Semántica	Valores posibles
	system.operatingSystem	Tipo de sistema operativo	string a ser definida
	system.javaConfiguration	Tipo y versión de la configuración soportada por la JVM del receptor	string (tipo seguido de la versión, sin espacio, por ejemplo: "CLDC1.1")
	system.javaProfile	Tipo y versión del perfil soportado por la JVM del receptor	string (tipo seguido de la versión, sin espacio – ej.: "MIDP2.0"
	system.luaVersion	Versión de la máquina Lua del receptor	string
	system.ncl.version	Versión del lenguaje NCL	string
	system.GingaNCL.version	Versión del ambiente Ginga-NCL	string
	system.GingaJ.version	Versión del ambiente Ginga-J	string
	system.xxx	Cualquier variable prefijada por "system" debe ser obligatoriamente reservada para uso futuro	
user • grupo de variables mantenidas por el sistema receptor; • se pueden leer, pero no pueden tener sus valores alterados por una aplicación NCL, un procedimiento Lua o un procedimiento Xlet; • programas nativos en el receptor pueden alterar los valores de las variables; • persisten durante todo el tiempo de vida de un receptor	user.age	Edad del usuario	entero
	user.location	Localización del usuario	string
	user.genre	Género del usuario	"m"| "f"
	user.xxx	Cualquier variable prefijada por "user" debe ser obligatoriamente reservada para uso futuro	
default • grupo de variables mantenidas por el sistema receptor; • se pueden leer y tener sus valores alterados por una aplicación NCL o por un procedimiento Lua o Xlet; • programas nativos en el receptor pueden alterar los valores de las variables; • persisten durante todo el tiempo de vida de un receptor, sin embargo, vuelven a su valor inicial al cambiar de canal	default.focusBorderColor	Color *default* aplicado al margen de un elemento en foco	"white" | "black" | "silver" | "gray" | "red" | "maroon" | "fuchsia" | "purple" | "lime" | "green" | "yellow" | "olive" |"blue" |"navy" |"aqua" |"teal"
	default.selBorderColor	Color *default* aplicado al margen de un elemento en foco cuando es activado	"White" | "black" | "silver" | "gray" | "red" | "maroon" | "fuchsia" | "purple" | "lime" | "green" | "yellow" | "olive" | "blue" | "navy" | "aqua" |"teal"
	default.focusBorderWidth	Ancho *default* (en pixels) aplicado al margen de un elemento en foco	entero
	default.focusBorderTransparency	Transparencia *default* aplicada al borde de un elemento en foco	valor real entre 0 y 1, o valor real en la banda [0,100] terminando con el carácter "%" (por ejemplo, 30 %), con "1", ó "100 %" significando máxima transparencia y "0", ó "0 %" significando ninguna transparencia
	default.xxx	Cualquier variable prefijada por "*default*" debe obligatoriamente ser reservada para uso futuro	

Tabla 12 (continuación)

Grupo	Variable	Semántica	Valores posibles
service • grupo de variables mantenidas por el formateador NCL; • se pueden leer y, en general, tener sus valores alterados por una aplicación NCL del mismo servicio; • se pueden leer, pero no pueden tener sus valores alterados por un programa Lua o Xlet del mismo servicio; la escritura debe hacerse a través de comandos NCL; • persisten, como mínimo, durante todo el tiempo de duración de un servicio	service.currentFocus	Valor del atributo *focusIndex* del elemento <media> en foco	entero
	service.currentKeyMaster	Identificador *(id)* del elemento <media> que detenta el control de las claves de navegación; si el elemento no está siendo presentado o no está pausado, el control es del formateador	string
	service.xxx	Cualquier variable prefijada por “service” debe obligatoriamente seguir las reglas especificadas para el grupo	
si • grupo de variables mantenidas por el middleware; • se pueden leer pero no pueden tener sus valores alterados por una aplicación NCL, un procedimiento Lua o un procedimiento Xlet; • persisten, como mínimo, durante todo el tiempo de sintonía de un canal	si.numberOfServices	Número de servicios disponibles, en el país, para el canal sintonizado. NOTA Es recomendable que el valor de esta variable se obtenga a partir del número de tablas PMT encontradas en la tabla PAT del flujo de transporte recibido por el canal sintonizado (conforme norma ISO/IEC 138-18-1:2007). En el cálculo del valor de esta variable, es recomendable que sólo se consideren las tablas cuyo campo country_code, disponible en el descriptor country_avaliability_descriptor (Sección 8.3.6 de la norma ABNT NBR 15603-2:2007) relativo a la tabla, corresponda al contenido de la variable de ambiente *user.location*.	entero
	si.numberOfPartialServices	Número de servicios disponibles, en el país, para el canal sintonizado. NOTA Es recomendable que el valor de esta variable se obtenga a partir del número de tablas PMT encontradas en la tabla PAT del flujo de transporte recibido por el canal sintonizado (conforme norma ISO/IEC 138-18-1:2007). En el cálculo del valor de esta variable, es recomendable que sólo se consideren las tablas PMT cuyo campo country_code, disponible en el descriptor country_availability_descriptor (Sección 8.3.6 de la ABNT NBR 15603-2:2007) corresponda al contenido de la variable de ambiente *user.location* y también cuyos campos program_number de las tablas correspondan a los campos service_id de los partial_reception_descriptor relativos a las tablas NIT	entero
	si.channeNumber	Número del canal sintonizado. NOTA El valor de esta variable debe obtenerse obligatoriamente a través del campo remote_control_key-id del descriptor ts_information_descriptor (Sección 8.3.42 de la norma ABNT NBR 15603-2:2007) de la tabla NIT (Sección 7.2.4 de la norma ABNT NBR15603-2:2007) que describe el servicio corriente.	entero
	si.xxx	Cualquier variable prefijada por “si” debe obligatoriamente seguir las reglas especificadas para el grupo.	

Tabla 12 (continuación)

Grupo	Variable	Semántica	Valores posibles
channel • grupo de variables mantenidas por el formateador NCL; • se pueden leer y tener sus valores alterados por una aplicación NCL del mismo canal; • pueden ser leídas pero no pueden tener sus valores alterados por una aplicación NCL del mismo canal; • persisten, por lo menos, durante todo el tiempo de sintonía de un canal	channel.keyCapture	Requisición de teclas alfanuméricas por aplicaciones NCL	(string)
	cannel.virtualKeyboard	Requisición del teclado virtual por aplicaciones NCL	
	cannel.keyboardBounds	Región de exhibición del teclado virtual (left, top, width, height	(entero, entero, entero, entero)
	channel.xxx	Cualquier variable prefijada por “channel” debe obligatoriamente seguir las reglas especificadas para el grupo	
shared • grupo de variables mantenidas por el formateador NCL; • se pueden leer y tener sus valores alterados por un programa NCL; • pueden ser leídas pero no pueden tener sus valores alterados por un programa Lua o Xlet; se debe escribir a través de comandos NCL; • persisten, por lo menos, durante todo el tiempo del servicio que la definió;	shared.xxx	Cualquier variable prefijada por “shared” debe obligatoriamente seguir las reglas especificadas para el grupo	

NOTA 1 El objeto de media del tipo application/x-ginga-settings no posee contenido para exhibición. Después de iniciada una aplicación NCL, las propiedades de ese objeto de media están disponibles para evaluación de las reglas definidas en los elementos <rule>. Para que las propiedades sean usadas en su definición, estas deben ser explícitamente declaradas.

El tipo application/x-ginga-time debe ser obligatoriamente aplicado a un elemento <media> especial (puede existir solamente uno en un documento NCL) cuyo contenido es el Universal Time Coordinated (UTC). Cualquier elemento <media> continuo sin fuente se puede utilizar para definir un reloj, relativo al tiempo de inicio de ese elemento <media>.

NOTA 2 El contenido de un elemento <media> del tipo application/x-ginga-time es especificado de acuerdo con la siguiente sintaxis: Año“:”Mes“:”Día“:”Hora“:”Minutos“:”Segundos“.”Fracción, donde Año es un entero; Mes es un entero en el intervalo [1,12]; Día es un entero en el intervalo [1,31]; Horas es un entero en el intervalo [0, 23]; Minutos es un entero en el intervalo [0,59]; Segundos es un entero en el intervalo [0,59]; y Fracción es un entero positivo. Después de iniciado, el contenido UTC no es exhibido, pero sobre él pueden ser definidas anclas por medio de elemento <área>.

La Tabla 13 muestra algunos valores posibles del atributo *type* para los perfiles TVD Avanzado y Básico y las extensiones de archivos asociadas. Los tipos obligatorios se definen en la ABNT NBR 15601. El atributo *type* es opcional (excepto para los elementos <media> sin atributo *src* definido) y se recomienda su uso para guiar la elección del exhibidor de media (herramienta de presentación) por el formateador. Cuando el atributo *type* no se especifica, se recomienda que el formateador use la extensión del contenido especificado en el atributo *src* para hacer la elección del exhibidor.

Cuando hay más de un exhibidor para el tipo soportado por el formateador, la propiedad *player* del elemento <media> puede especificar cuál será utilizado para la presentación. Caso contrario, el formateador debe obligatoriamente utilizar un exhibidor *default* para aquel tipo de media.

Tabla 13 — Tipos de media MIME para formateadores Ginga-NCL

Tipo de media	Extensión de archivo
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-NCL	ncl
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)*

El módulo *Context* es responsable por la definición de nudos de contexto a través de elementos <context>. Un nudo de contexto NCM es un tipo particular de nudo de composición NCM y se define conteniendo un conjunto de nudos y un conjunto de eslabones. Como normalmente ocurre, el atributo id identifica en forma unívoca cada elemento <context> y <media> dentro de un documento.

Los atributos *Instance, refer* y *descriptor* son extensiones definidas en otros módulos y son discutidos en la definición de tales módulos.

Elementos <media> del tipo application/x-ginga-NCL no pueden tener los atributos *instance* y *refer*.

Los elementos de los dos módulos de la funcionalidad *Components,* sus elementos-hijos y sus atributos deben estar de acuerdo obligatoriamente con las Tablas 14 y 15.

Tabla 14 — Módulo Media extendido

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

Tabla 15 — Módulo *Context* extendido

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

7.2.5 Área funcional *Interfaces*

El área funcional *Interfaces* permite la definición de interfaces de nudos (objetos de media o nudos de composición) que serán utilizadas en relaciones con otras interfaces de nudos. Este área funcional se divide en cuatro módulos:

— *MediaContentAnchor,* que permite definiciones de anclas de contenido (o área) para nudos de media (elementos <media>);

— *CompositeNodeInterface,* que permite definiciones de puertos para nudos de composición (elementos <context> y <switch>);

— *PropertyAnchor,* que permite la definición de propiedades de nudos como interfaces de nudos; y

— *SwitchInterface,* que permite la definición de interfaces especiales para elementos <switch>.

El módulo *MediaContentAnchor* define el elemento <area>, que permite la definición de anclas de contenido representando porciones espaciales, a través de atributo *coords* (como en XHTML); la definición de anclas de contenido representando porciones temporales, a través de los atributos *begin* y *end*; y la definición de anclas de contenido representando porciones espacio-temporales, a través de los atributos *coords*, *begin* y *end*. Además, el elemento <area> permite la definición de anclas textuales, a través de los atributos *beginText*, *beginPosition*, *endText*, *endPosition*, que definen una cadena de caracteres y la ocurrencia de esa cadena en el texto, respectivamente. Adicionalmente, el elemento <area> puede también definir un ancla de contenido con base en el número de muestras de audio o *frames* de video, a través de los atributos *first* y *last*, que deben obligatoriamente indicar la muestra/*frame* inicial y final. El elemento <área> también puede definir un ancla de contenido basada en el atributo *label*, que especifica una cadena de caracteres que debe ser utilizada por el exhibidor de medias para identificar una región de contenido. Finalmente, el elemento <area> también puede definir un ancla de contenido basada en el atributo *clip,* que especifica una triple que debe ser utilizada por el exhibidor de medias para identificar un clip de contenido en un objeto de media con contenido hipermedia declarativo.

Los atributos *first* y *last* se especifican de acuerdo con una de las siguientes sintaxis:

a) muestras”s”, donde Muestras es un entero positivo;

b) cuadros”f”, donde Cuadros es un entero positivo;

c) NPT”npt”, donde NPT es el valor tiempo normal de exhibición (Normal Play Time value).

Si el atributo *begin* es definido, pero el atributo *end* no fuera especificado, el final de toda presentación del contenido de media debe obligatoriamente ser considerado como cierre del ancla. Por otro lado, si el atributo *end* es definido sin una definición explícita de *begin*, el inicio de toda la presentación del contenido de media debe obligatoriamente ser considerado como el inicio del ancla. Comportamiento semejante es esperado con relación a los atributos *first* y *last*. En el caso de un elemento <media> del tipo application/x-ginga-time, los atributos *begin* y *end* deben obligatoriamente ser siempre especificados y corresponden a un tiempo absoluto de Universal Time Coordinated (UTC). En anclas de contenido textual, si el final de la región del ancla no fuera definido, se debe obligatoriamente asumir el final del contenido del texto; si el inicio de la región del ancla no fuera definido, se debe obligatoriamente asumir el inicio del contenido del texto..

Con excepción del elemento <media> del tipo application/x-ginga-time, los atributos *begin* y *end* deben especificarse obligatoriamente con una de las siguientes sintaxis:

a) Horas”:”Minutos”:”Segundos”.”Fracción, donde Horas es un entero en el intervalo[0,23]; Minutos es un entero en el itervalo [0,59]; Segundos es un entero en el intervalo [0,59] y fracción es un entero posivo

b) Segundos”s”, donde Segundos es un entero positivo.

Para el elemento <media> del tipo application/x-ginga-time, los atributos *begin* y *end* deben especificarse obligatoriamente con la siguiente sintaxis: Año”:”Mes”:”Día”:”Hora”:”Minutos”:”Segundos”.”Fracción, de acuerdo con la zona del tiempo del país. El Formateador NCL es responsable de traducir ese valor para un valor correspondiente de UTC.

Como normalmente ocurre, los elementos <area> deben tener obligatoriamente un atributo id*,* que identifica en forma unívoca el elemento dentro de un documento.

El elemento <area> y sus atributos deben estar de acuerdo obligatoriamente con la Tabla 16.

Tabla 16 — Módulo *MediaContentAnchor* extendido

Elementos	Atributos	Contenido
area	*id, coords, begin, end, beginText, beginPosition endText, endPosition, first, last, label, clip*	vacío

El módulo *CompositeNodeInterface* define el elemento <port>, que especifica un puerto de un nudo de composición con su respectivo mapeo para una interfaz (atributo *interfaz*) de uno de sus componentes (especificado por el atributo *component*).

En NCM, todo nodo (media o contexto) debe obligatoriamente poseer un ancla con una región representando el contenido total del nodo. Esa ancla es llamada ancla de contenido total y es declarada por omisión (default) en NCL. Con excepción del elemento de media con código imperativo (por ejemplo, <media type="application/x-ginga-NCLua" ...>, cada vez que un componente NCL es referenciado sin especificar una de sus anclas, se debe obligatoriamente asumir el ancla de contenido total.

El elemento <port> y sus atributos deben estar de acuerdo obligatoriamente con la Tabla 17.

Tabla 17 — Módulo *CompositeNodeInterface* extendido

Elementos	Atributos	Contenido
port	*id, component, interface*	vacío

El módulo PropertyAnchor define un elemento denominado <property>, que se puede utilizar para definir una propiedad o grupo de propiedades de un nudo, como una de sus interfaces (ancla). El elemento <property> define el atributo *name,* que indica el nombre de la propiedad o grupo de propiedades, y el atributo *value,* atributo opcional que define un valor inicial para la propiedad *name.* El elemento padre no puede tener elementos <property> con los mismos valores para el atributo *name.*

Es posible tener exhibidores de documentos NCL (formateadores) que definen algunas propiedades de nudos e interfaces de nudos, implícitamente. Sin embargo, por lo general, es de buena práctica definir explícitamente las interfaces.

Los elementos <body>, <context> y <media> pueden tener varias propiedades embutidas, que no son explícitamente declaradas por los elementos <property>. Ejemplos de esas propiedades pueden ser encontrados entre aquellas que definen el local a ser colocado el objeto de media durante una presentación, la duración de la presentación y otras que definen características adicionales de la presentación: *top*, *left*, *bottom*, *right*, *width*, *height*, *plan*, *explicitDur*, *background*, *transparency*, *visible*, *fit*, *scroll*, *style*, *soundLevel*, *balanceLevel*, *trebleLevel*, *bassLevel*, *fontColor, fontFamily, fontStyle, fontSize, fontVariant, fontWeight, reusePlayer, playerLife* etc. Tales propiedades asumen como sus valores iniciales los definidos en atributos homónimos del descriptor y región asociados al nodo (ver 7.2.3 y 7.2.6). Algunas propiedades tienen su valor definido por el propio sistema, como la propiedad *contentId,* asociada a un objeto de media continua cuyo contenido se refiere a un flujo elementar, que inicialmente tiene el valor nulo, pero apenas el objeto es iniciado asume el valor del identificador (contenido en el campo también llamado contentId) transportado en el descriptor de referencia NPT. Otro ejemplo es la propiedad *standby*, que debe obligatoriamente asumir el valor "true" mientras un objeto de media continua ya iniciado, y cuyo contenido se refiere a un flujo elementar, esté con su contenido temporalmente interrumpido por otro contenido entrelazado en el mismo flujo elementar. Entretanto, cualquiera que sea el caso, cuando una propiedad embutida es utilizada en una relación, esta debe obligatoriamente ser explícitamente declarada como elemento <property> (interfaz). Por lo tanto, cada propiedad posee un atributo booleano oculto denominado *externable,* que es definido como "false" por *default* y como "true" cuando la propiedad es explícitamente declarada en un elemento <property>.

NOTA 1 Todas las propiedades (y sus valores iniciales) de un objeto NCL pueden ser definidas usando solamente elementos <property>. Los elementos <descriptor>, <descriptorParam> y <región> son sólo una opción más (opción de reuso) para definir valores iniciales para las propiedades.

NOTA 2 A la propiedad de *standby* se le puede atribuir el valor "true" cuando el valor del identificador transportado en el NPT reference descriptor (en el campo contentId) señalizado como "no pausado" sea diferente del valor de la propiedad *contentId* del mismo objeto.

NOTA 3 La propiedad *visible* también puede ser asociada a un elemento <context> y <body>. En esos casos, cuando la propiedad tiene su valor igual a "true", vale la especificación de *visible* de cada nudo hijo. Cuando tiene su valor igual a "false", todos los elementos de la composición son exhibidos de forma invisible. Particularmente, cuando un documento tiene su elemento <body> con la propiedad *visible* = "false" y su evento de presentación en estado= "paused", se dice que la aplicación está en "espera" (*stand-by).* Cuando una aplicación entra en *stand-by,* el video principal del servicio vuelve a ocupar el 100% de la dimensión de la pantalla en la que es exhibido y el audio principal al 100% de su volumen.

Un grupo de propiedades del nodo también puede ser explícitamente declarado como un elemento único <property> (interfaz), permitiendo que los autores especifiquen el valor de varias propiedades con una propiedad única. Los siguientes grupos deben obligatoriamente ser reconocidos por un formateador NCL: *location*, agrupando (*left*, *top*), en esa orden; *size*, agrupando (*width*, *height*), en esa orden; y *bounds*, agrupando (left, top, *width*, *height*), en esa orden.

Cuando un formateador trata una alteración en un grupo de propiedad, debe obligatoriamente probar sólo la consistencia del proceso al final. Las palabras *top, left, bottom, right, width, height, explicitDur, background, transparency, visible, fit, scroll, style, soundLevel, balanceLevel, trebleLevel, bassLevel, fontColor, fontFamily, fontStyle, fontSize, fontVariant, font Weight, reusePlayer, playerLife, location, size* y *bounds* son palabras reservadas para valores del mismo atributo name del elemento <property>.

El elemento <property> y sus atributos deben estar de acuerdo obligatoriamente con la Tabla 18.

Tabla 18 — Módulo *PropertyAnchor* extendido

Elementos	Atributos	Contenido
property	*name, value*	vacío

El módulo *SwitchInterface* permite la creación de interfaces de elemento <switch> (ver 7.2.4), que se pueden mapear a un conjunto de interfaces alternativas de nudos internos, permitiendo a un eslabón anclar en el componente elegido cuando el <switch> es procesado (NCM Core:2005). Ese módulo introduce el elemento <switchPort>, que contiene un conjunto de elementos de mapeo. Un elemento de mapeo define un camino desde el <switchPort> para una interfaz (atributo *interfaz)* de uno de los componentes del <switch> (especificados por su atributo *component).*

Es importante mencionar que cada elemento representando una interfaz del objeto (<area>, <port>, <property> y <switchPort>) debe obligatoriamente tener un identificador (atributo *id* o *name*).

O elemento <switchPort>, sus elementos-hijos y sus atributos deben estar de acuerdo obligatoriamente con la Tabla 19.

Tabla 19 — Módulo *SwitchInterface* extendido

Elementos	Atributos	Contenido
switchPort	*id*	mapping+
mapping	*component, interface*	vacío

7.2.6 Área funcional *Presentation Specification*

El área funcional *Presentation Specification* tiene un único módulo denominado *Descriptor.* El alcance de ese módulo es especificar informaciones espacio-temporales necesarias para la presentación de cada componente del documento. Esas informaciones son modeladas por los objetos descriptores.

El módulo *Descriptor* permite la definición de elementos <descriptor>, que contienen un conjunto de atributos opcionales, agrupando las definiciones espacio-temporales que se deben usar de acuerdo con el tipo de objeto a ser presentado. La definición de elementos <descriptor> se debe incluir obligatoriamente en el encabezamiento del documento, dentro del elemento <descriptorBase>, que especifica el conjunto de descriptores de un documento. El elemento <descriptor> debe tener obligatoriamente el atributo *id;* y el elemento <descriptorBase> puede tener el atributo *id,* que, como normalmente ocurre, identifica en forma unívoca los elementos dentro de un documento.

Un elemento <descriptor> puede tener atributos temporales: *ExplicitDur* y *freeze,* definidos por el módulo *Timing* (ver 7.2.10); un atributo denominado *player,* que identifica la herramienta de presentación a ser utilizada; un atributo denominado *region,* que se refiere a la región definida por los elementos del módulo *Layout* (ver 7.2.3); atributos para navegación: *moveLeft, moveRight, move Up ; moveDown, focusIndex, focusBorderColor; focusBorderWidth; focusBorderTransparency, focusSrc, selBorderColor,* y *focusSelSrc,* definidos por el módulo KeyNavigation (ver 7.2.12); y atributos de transición: *transIn* y *transOut* (ver 7.2.14).

NOTA Un elemento <descriptor> de un elemento <media> del tipo application/x-ginga-NCL no puede contener el atributo *player.* En ese caso, un exhibidor NCL específico para cada dispositivo de exhibición es definido por el middleware.

Un elemento <descriptor> también puede tener elementos <descriptorParam> como elementos hijo, que se utilizan para parametrizar el control de la presentación del objeto asociado con el elemento descriptor. Estos parámetros pueden, por ejemplo, redefinir algunos valores de atributos definidos por los atributos de la región. Los mismos también pueden definir nuevos atributos, tales como *plan,* en qué plano de una pantalla estructurada se colocará un objeto; *rgbChromakey,* definido por un color RGB que se exhibirá como transparente; *background,* especificando el color de fondo utilizado para rellenar el área de una región de exhibición de la media, cuando toda la región no es rellenada por la media en sí; *visible,* permitiendo que la presentación del objeto sea visualizada u ocultada; *fit,* indicando como un objeto será presentado; *scroll,* que permite la especificación de como un autor le gustaría configurar el desplazamiento vertical en una región; *transparency,* indicando el grado de transparencia de la presentación de un objeto; *style,* que se refiere a una hoja de estilo [Cascading Style Sheets, 1998] con informaciones, por ejemplo, para presentación del texto; además de especificar atributos para objetos de audio, tales como *soundLevel, balanceLevel, trebleLevel* y *bassLevel.* Además, los elementos-hijos <descriptorParam> pueden determinar si un nuevo exhibidor debe ser obligatoriamente solicitado, o si un exhibidor ya solicitado se debe utilizar obligatoriamente *(reusePlayer),* y especificar qué pasará a instancias del exhibidor al final de la presentación *(playerLife).* Las palabras Top, left, bottom, right, width, height, explicitDur, location, size, bounds, background, visible, fit, scroll, style, soundLevel, balanceLevel, trebleLevel, bassLevel, reusePlayer y playerLife son palabras reservadas para valores del atributo *name* del elemento <descriptorParam>. Algunos valores posibles para los nombres reservados de parámetros/atributos son presentados en la Tabla 20.

Tabla 20 — Parámetros/atributos reservados y valores posibles

Nombre del parámetro/atributo	Valor
top, left, bottom, right, width, height	Número real en la banda [0,100] terminando con el carácter “%” (por ejemplo, 30 %), o un valor entero especificando el atributo en pixels (en el caso de *weight* y *height,* un entero no negativo)
location	Dos números separados por coma, cada uno siguiendo las reglas de valor especificadas para parámetros *left y top,* respectivamente
size	Dos valores separados por coma. Cada valor debe seguir obligatoriamente las mismas reglas especificadas para parámetros de *width* y *height,* respectivamente
bounds	Cuatro valores separados por coma. Cada valor debe seguir obligatoriamente las mismas reglas especificadas para parámetros *left, top, width* y *height,* respectivamente
background	Nombres de colores reservados: “white”, “black”, “silver”, “gray”, red”, “maroon”, fuchsia”, “purple”, “lime”, “green”, “yellow”, “olive”, “blue”, “navy”, “aqua”, o “teal”. Otra opción para especificar el valor del color se especifica en la ABNT NBR 15606-1. El valor del color de fondo también puede tener valor reservado “transparent”. Eso puede ser útil para presentar imágenes transparentes, como GIF transparentes, superpuestos sobre otras imágenes o videos.
visible	“true” o “false”. Cuando no se especifica, el atributo asume el valor “true”.
transparency	Un número real en la banda [0,1], o un número real en la banda [0,100] terminando con el carácter “%” (por ejemplo, 30%), especificando el grado de transparencia de una presentación de objeto (“1” ó “100 %” significa transparencia total y ”0“ ó “0 %” significa opaco)
fit	“fill”, “hidden”, “meet”, “meetBest”, “slice”. “fill”: redimensiona el contenido del objeto de media para que toque todos los bordes de la caja definida por los atributos de ancho y altura del objeto “hidden”: si la altura/longitud intrínseca del contenido de media es menor que el atributo de altura/longitud, el objeto debe ser obligatoriamente creado empezando del margen superior/izquierdo y tener la altura/longitud restante rellenada con el color de fondo; si la altura/longitud intrínseca del contenido de media es mayor que el atributo altura/longitud, el objeto debe ser obligatoriamente creado empezando por el margen superior/izquierdo hasta que la altura/longitud definida en el atributo sea alcanzada, y tener la parte del contenido de media abajo/a la derecha de la altura/longitud cortada “meet”: redimensiona el objeto de media visual mientras preserva su relación de aspecto hasta que su altura o anchura es igual al valor especificado por los atributos height o width. El ángulo superior izquierdo del contenido de media se sitúa en las coordenadas superiores izquierdas de la caja, el espacio vacío a la derecha o en la base debe ser obligatoriamente rellenado con el color del telón “meetBest”: La semántica es idéntica a la del “meet”, excepto que la imagen no es redimensionada en más de 100 % en cualquier dimensión “slice”: redimensiona el contenido de la media visual preservando su relación de aspecto, hasta que su altura o anchura sea igual al valor especificado en los atributos de altura y anchura, y que la caja de presentación definida esté completamente rellenada. Algunas partes del contenido pueden ser cortadas. El exceso de anchura es cortado desde la derecha del objeto de media. El exceso de altura es cortado desde la base del objeto de media
scroll	“none”, “horizontal”, “vertical”, “both”, o “automatic”
style	Localizador de un archivo de hoja de estilo
soundLevel, balanceLevel, trebleLevel, bassLevel	Un número real en la banda [0, 1], o un número real en la banda [0,100] terminando con el carácter “%” (por ejemplo, 30 %)
zIndex	Un número entero en la banda [0, 255], siendo que regiones con mayor valor de *zIndex* se sitúan sobre regiones con menor valor de *zIndex*
fontFamily	Una lista priorizada de nombres de familia de fuentes y/o nombres genéricos de familias
fontStyle	Estilo de la fuente (“normal”, o “italic”)
fontSize	Tamaño de la fuente
fontVariant	Forma de exhibición del texto: fuente en “small-caps” o “normal”
fontWeight	Peso de la fuente (“normal”, o “bold”)
fontColor	Color de la fuente (“white”, “black”, “silver”, “gray”, red”, “maroon”, fuchsia”, “purple”, “lime”, “green”, “yellow”, “olive”, “blue”, “navy”, “aqua”, o “teal”)
reusePlayer	Valor booleano: “false”, “true”. Valor default = “false”
playerLife	“keep”, “close”. Valor default = “close”

Además de todos los atributos mencionados, el elemento <descriptor> también puede tener atributos definidos en el área funcional *transition effects* (ver 7.2.14).

NOTA Si se especifican varios valores para una misma propriedad, el valor definido en el elemento <property> tiene precedencia sobre el valor definido en un elemento <descriptorParam>, que, a su vez, tiene preferencia sobre el valor definido en un atributo del elemento <descriptor> (incluyendo el atributo *region)*.

Además del elemento <descriptor>, el módulo *Descriptor* define un atributo homónimo, que se refiere a un elemento del conjunto de descriptores del documento. Cuando un perfil de lenguaje utiliza el módulo *Descriptor,* debe obligatoriamente determinar cómo los descriptores estarán asociados con los componentes del documento. Siguiendo las directrices NCM, esta Norma establece que el atributo *descriptor* está asociado con cualquier nudo de media a través de elementos <media> y a través de las extremidades de los eslabones (elementos <bind>) (ver 8.2.1).

El conjunto de descriptores de un documento puede contener elementos <descriptor> o elementos <descriptorSwitch>, que permiten especificar descriptores alternativos (ver 7.2.9).

Los elementos del módulo *Descriptor,* sus elementos-hijos y sus atributos deben estar de acuerdo obligatoriamente con la Tabla 21.

Tabla 21 — Módulo *Descriptor* extendido

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

7.2.7 Área funcional *Linking*

El área funcional *Linking* define el módulo *Linking,* responsable por la definición de los eslabones, que utilizan conectores. Un elemento <enlace> puede tener un atributo *id,* que identifica en forma unívoca el elemento dentro de un documento y debe tener obligatoriamente un atributo *xconnector,* que se refiere al URI de un conector hipermedia. La referencia debe tener obligatoriamente el formato *alias#connector_id,* o *documentURI_value#connector_id,* para conectores definidos en un documento externo (ver 7.2.11), o simplemente *connector_id,* para conectores definidos en el propio documento.

El elemento <link> contiene elementos-hijos denominados <bind>, que permiten asociar nudos a papeles *(roles)* del conector (ver 7.2.8). Para hacer esta asociación, un elemento <bind> tiene cuatro atributos básicos. El primero se denomina *role,* que se utiliza para hacer referencia a un papel del conector. El segundo se denomina *component,* que se utiliza para identificar el nudo. El tercero es un atributo opcional denominado *interfaz,* usado para hacer referencia a una interfaz del nudo. El cuarto es un atributo opcional denominado *descriptor,* usado para hacer referencia a un descriptor a ser asociado con el nudo, conforme definido por el módulo *Descriptor* (ver 7.2.6).

NOTA El atributo *interfaz* puede referirse a cualquier interfaz del nudo, es decir, un ancla, una propiedad o un puerto, si es un nudo de composición. El atributo *interfaz* es opcional. Cuando no se especifica, la asociación se realiza con todo el contenido del nudo (ver 7.2.5).

Si el elemento conector define parámetros (ver 7.2.8), conviene a los elementos <bind> o <link> definir valores para esos parámetros, a través de sus elementos-hijos denominados <bindParam> y <enlaceParam>, respectivamente, ambos con atributos *name* y *value.* En ese caso, el atributo *name* debe obligatoriamente hacer referencia al nombre de un parámetro del conector, mientras que el atributo *value* debe definir obligatoriamente un valor a ser atribuido al respectivo parámetro.

Los elementos del módulo *Linking,* sus atributos y sus elementos-hijos deben estar de acuerdo obligatoriamente con la Tabla 22.

Tabla 22 — Módulo *Linking* extendido

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

7.2.8 Area funcional *Connectors*

El área funcional *Connectors* de la NCL 3.0 se divide en siete módulos básicos: ConnectorCommonPart, ConnectorAssessmentExpression, ConnectorCausalExpression, CausalConnector, ConstraintConnector (no considerado en esta Norma), ConnectorBase y CompositeConnector (tampoco considerado en esta Norma).

Los módulos del área funcional *Connectors* son totalmente independientes de los demás módulos NCL. Estos módulos forman el núcleo de un lenguaje nuevo de aplicación XML (de hecho, otros perfiles NCL 3.0) para la definición de conectores, que se pueden utilizar para especificar relaciones de sincronización espacio-temporales, tratando relaciones de referencia (de interacción con el usuario) como un caso particular de relaciones de sincronización temporal.

Además de los módulos básicos, el área funcional *Connectors* también define módulos que agrupan conjuntos de módulos básicos, para facilitar la definición del perfil de lenguaje. Ése es el caso del módulo CausalConnectorFunctionality, utilizado en la definición de los perfiles EDTV, BDTV y CausalConnector. El módulo CausalConnectorFunctionality agrupa los siguientes módulos: ConnectorCommonPart, ConnectorAssessmentExpression, ConnectorCausalExpression y CausalConnector.

Un elemento <causalConnector> representa una relación causal que puede ser utilizada por elementos <link> en documentos. En una relación causal, una condición debe ser obligatoriamente satisfecha para disparar una acción.

Un <causalConnector> especifica una relación independientemente de las relaciones, es decir, no especifica cuáles nudos (representados por elementos <media>, <context>, <body> y <switch>) interactúan a través de la relación. Un elemento <link>, a su vez, representa una relación, del tipo definido por su conector, interconectando a diferentes nudos. Los eslabones representan el mismo tipo de relación, pero interconectando a diferentes nudos, pueden reutilizar el mismo conector, reutilizando todas las especificaciones. Un <causalConnector> especifica, a través de sus elementos-hijos, un conjunto de puntos de la interfaz, denominados papeles. Un elemento <link> se refiere a un <causalConnector> y define un conjunto de mapeos (elementos <bind> hijos del elemento <link>), que asocian cada extremidad del eslabón (interfaz de nudo) a un papel del conector utilizado.

Las relaciones en NCL se basan en eventos. Un evento es una ocurrencia en el tiempo que puede ser instantánea o tener duración mensurable. La NCL 3.0 define los siguientes tipos de eventos:

— evento de presentación, que es definido por la presentación de un subconjunto de las unidades de información de un objeto de media, especificado en NCL por el elemento <área>, o por el nodo de media en sí (presentación de todo el contenido). Los eventos de presentación también pueden ser definidos sobre nodos de composición (representados por un elemento <body>, <context> o <switch>), representando la presentación de las unidades de información de cualquier nodo dentro del nodo de composición;

— evento de selección, que es definido por la selección de un subconjunto de las unidades de información de un objeto de media en exhibición, especificado en NCL por el elemento <area>, o por el propio nodo de media (presentación del contenido total);

— en la NCL por el elemento <area>, o por el propio nudo de media (presentación del contenido total);

— evento de atribución, que es definido por la atribución de un valor a una propiedad de un nudo (representado por un elemento <media>, <body>, <context> o <switch>), que se debe declarar obligatoriamente en un elemento <property>, hijo del nudo; y

— evento de composición, que es definido por la presentación de la estructura de un nodo de composición (representado por un elemento <body>, <context> o <switch>). Los eventos de composición son utilizados para presentar el mapa de la composición (organización de la composición). El recurso para presentar esa estructura, sin embargo, es opcional.

Cada evento define una máquina de estados que se recomienda ser controlada por el formateador NCL, como demostrado en la Figura 3. Además, todo evento tiene un atributo asociado, denominado *occurrences,* que cuenta cuántas veces el evento va del estado ocurriendo *(occurring)* al estado preparado *(sleeping),* durante la presentación de un documento. Eventos como presentación y atribución también tienen un atributo denominado *repetitions,* que cuenta cuántas veces el evento debe ser obligatoriamente reiniciado (pasó del estado ocurriendo para el estado preparado) por el formateador. Ese atributo puede contener el valor “indefinite”, llevando a una repetición infinita (loop) de las ocurrencias del evento hasta que ocurra alguna interrupción externa.

Los nombres de transición para la máquina de estado de evento deben estar de acuerdo obligatoriamente con la Tabla 23.

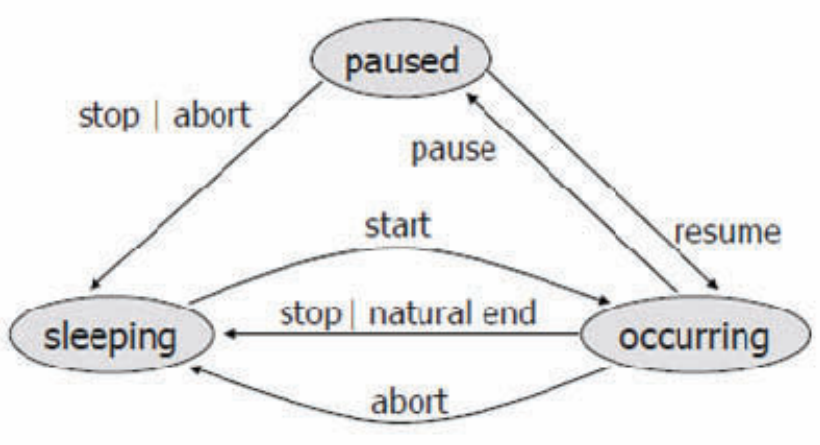

Figura 3 — Máquina de estado de evento

Tabla 23 — Nombres de las transiciones para una máquina de estado de evento

Transición (causada por acción)	Nombre de la transición
sleeping → occurring (start)	*starts*
occurring → sleeping (stop or natural end)	*stops*
occurring → sleeping (abort)	*aborts*
occurring → paused (pause)	*pauses*
paused → occurring (resume)	*resumes*
paused → sleeping (stop)	*stops*
paused → sleeping (abort)	*aborts*

Un evento de presentación asociado a un nodo de media, representado por un elemento <media>, inicia en el estado preparado (*sleeping*). En el inicio de la exhibición de sus unidades de información, el evento irá al estado ocurriendo (*occurring*). Si la exhibición fuera temporalmente suspendida, el evento permanece en el estado pausado (*paused*), mientras dure esa situación.

Un evento de presentación puede cambiar de ocurriendo para preparado como consecuencia de la finalización natural de la duración de la presentación, o debido a una acción que termina el evento. En ambos casos, el atributo *occurrences* se incrementa, y el atributo *repetitions* se disminuye en uno. Si, después de disminuido, el valor del atributo *repetitions* es mayor que cero, el evento es automáticamente reiniciado (vuelve nuevamente al estado ocurriendo).

Cuando la presentación de un evento se interrumpe bruscamente, a través de un comando para abortar la presentación, el evento también va al estado preparado, pero sin incrementar el atributo *occurrences* y actualizando el valor del atributo *repetitions* para cero. La duración de un evento es el tiempo en que permanece en el estado ocurriendo. Tal duración puede ser intrínseca al objeto de media, explícitamente especificada por un autor (atributo *explicitDur* de un elemento <descriptor>), o derivado de una relación.

Un evento de presentación asociado con un nudo de composición representado por un elemento <body> o <context> permanece en el estado ocurriendo mientras por lo menos un evento de presentación asociado con cualquiera de los nudos hijos de esa composición está en el estado ocurriendo, o mientras por lo menos un eslabón-hijo del nudo de composición esté siendo evaluado.

Un evento de presentación asociado con un nudo de composición representado por un elemento <body> o <context> está en el estado pausado si por lo menos un evento de presentación asociado con cualquiera de los nudos hijos de la composición está en el estado pausado y todos los otros eventos de presentación asociados con los nudos hijos de composición están en el estado preparado o pausado. En caso contrario, el evento de presentación está en el estado preparado.

NOTA Otros detalles sobre el comportamiento de las máquinas de estado de evento de presentación para nudos de media y de composición se suministran en la Sección 8.

Un evento de presentación asociado con un nudo *switch,* representado por un elemento <switch>, permanece en el estado ocurriendo mientras el elemento-hijo del *switch,* elegido (nudo seleccionado) a través de las reglas de enlace (*bind rules*)*,* esté en el estado ocurriendo. Está en el estado pausado si el nudo seleccionado está en el estado pausado. En caso contrario, el evento de presentación está en el estado preparado.

Un evento de selección es iniciado en el estado preparado. Permanece en el estado ocurriendo mientras el ancla correspondiente (subconjunto de las unidades de información del objeto de media) esté siendo seleccionada.

Los eventos de atribución permanecen en el estado ocurriendo mientras los valores de propiedad correspondientes estén siendo modificados. Obviamente, eventos instantáneos, como eventos para simple atribución de valor, permanecen en el estado ocurriendo sólo durante un período infinitesimal de tiempo.

Un evento de composición (asociado a un nudo de composición representado por un elemento <body>, <context> o <switch>) permanece en el estado ocurriendo mientras el mapa de la composición está siendo presentado.

Las relaciones se definen con base en los estados de los eventos, en las alteraciones sobre las máquinas de estado de evento, sobre los valores de atributos de los eventos, y sobre los valores de propiedad de los nudos (elemento <media>, <body>, <context> o <switch>). El módulo *CausalConnectorFunctionality* permite sólo la definición de relaciones causales, definidas por el elemento <causalConnector> del módulo *CausalConnector.*

Un elemento <causalConnector> tiene una expresión de cola *(glue expression),* que define una expresión de condición y una de acción. Cuando la expresión de condición es satisfecha, la expresión de acción debe ser obligatoriamente ejecutada. El elemento <causalConnector> debe tener obligatoriamente un atributo id*,* que identifica únicamente el elemento dentro de un documento.

Una expresión de condición puede ser simple (elemento <simpleCondition>) o compuesta (elemento <compoundCondition>), ambos elementos definidos por el módulo *ConnectorCausalExpression.*

El elemento <simpleCondition> tiene un atributo *role,* cuyo valor debe ser obligatoriamente único en el conjunto de *roles* del conector. Un *role* (papel) es un punto de interfaz del conector, que puede ser asociado a interfaces de nudos por un eslabón que hace referencia al conector. Un elemento <simpleCondition> también define un tipo de evento (atributo *eventType)* y a qué transición se refiere la condición (atributo *transition).* Los atributos *eventType* y *transition* son opcionales. Pueden ser inferidos por el valor del atributo *role* si se utilizan valores reservados. En caso contrario, atributos *event Type* y *transition* son obligatorios.

Los valores reservados utilizados para definir los *roles* en <simpleCondition> se establecen en la Tabla 24. Si un valor de *eventType* es “selection”, el *role* también puede definir sobre a qué dispositivo se refiere la selección (por ejemplo, teclas de un teclado o control remoto), a través de su atributo *key.* Por lo menos los siguientes valores (que son sensibles al tipo mayúscula o minúscula) deben ser obligatoriamente aceptados por el 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”.

Tabla 24 — Valores reservados para especificación de la condición del role asociados a las máquinas de estado de evento

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

Varias palabras reservadas están definidas para valores de papeles-condición. Entretanto, se debe observar que hay más valores que aquellos definidos por las palabras reservadas. Las palabras reservadas son solamente otros y, en el futuro, otros nuevos pueden ser agregados. Por ejemplo, en el caso de un evento de selección, palabras reservadas pueden ser definidas para el inicio y para el final de una selección.

La cardinalidad del *role* especifica el número mínimo (atributo *min)* y máximo (atributo *Max)* de los participantes que pueden ejercer el *papel* (número de *binds)* cuando el <causalConnector> se utiliza para crear un <link>. El valor mínimo de la cardinalidad debe ser obligatoriamente siempre un valor finito positivo, mayor que cero y menor o igual al valor máximo de la cardinalidad. Si la cardinalidad mínima y la máxima no se informan, “1” debe ser obligatoriamente asumido como valor (default) para ambos parámetros. Cuando el valor máximo de cardinalidad es mayor que uno, varios participantes pueden ejecutar el mismo *papel (role),* es decir, puede haber varios enlaces *(binds)* conectando diversos nudos al mismo *papel.* El valor “unbounded” puede ser dado al atributo *max,* si el *role* tuviera *binds* ilimitados asociados a él. En los dos últimos casos, conviene que un atributo *qualifier* sea especificado para informar la relación lógica entre los *binds* de condición simple. Como descrito en la Tabla 25, los valores posibles para el atributo *qualifier* son: “Or” (o) o “and” (y). Si el calificador (atributo *qualifier)* establece un operador lógico “or”, el eslabón será accionado durante la ocurrencia de cualquier condición. Si el calificador establece un operador lógico “and”, el eslabón será accionado después de la ocurrencia de todas las condiciones simples. En caso de no ser especificado, se debe asumir obligatoriamente el valor (default) “or”.

Tabla 25 — Valores del calificador para condiciones simples

Elemento role	Calificador	Semántica
<simpleCondition>	“or”	Verdadero siempre que ocurre cualquier condición simple asociada
simpleCondition>	“and”	Verdadero inmediatamente después de la ocurrencia de todas las condiciones simples asociadas

Un atributo de retardo (*delay*) también puede ser definido para una <simpleCondition>, especificando que la condición es verdadera después de un período de retardo desde el momento en que ocurre la transición.

El elemento <compoundCondition> tiene un atributo *operator* con el valor Booleano (“and” u “or”) relacionando sus elementos-hijos: <simpleCondition>, <compoundCondition>,<assessmentStatement> y <compoundStatement>. Un atributo *delay* también puede ser definido, especificando que la condición compuesta es verdadera después que un tiempo de retardo del momento en que la expresión, relacionada a sus elementos-hijos, sea verdadera. Los elementos <assessmentStatement> y <compoundStatement> son definidos por el módulo *ConnectorAssessmentExpression.*

NOTA Cuando una condición “and” compuesta se relaciona a más de una condición de disparo (es decir, una condición solamente satisfecha en un instante de tiempo infinitesimal – como, por ejemplo, el final de la presentación de un objeto), la condición compuesta debe ser obligatoriamente considerada verdadera en el instante inmediatamente después de la satisfacción de todas las condiciones de disparo.

Una expresión de acción capta acciones que pueden ser ejecutadas en relaciones causales, pudiendo ser compuestas por un elemento <simpleAction> o <compoundAction>, también definido por el módulo *ConnectorCausalExpression.*

El elemento <simpleAction> tiene un atributo *role,* que debe ser obligatoriamente único en el conjunto de *papeles (roles)* del conector. Como siempre, un papel es un punto de interfaz del conector, que está asociado a las interfaces de nudos por un <link> que se refiere al conector. Un elemento <simpleAction> también define un tipo de evento (atributo *event Type)* y qué transición de estado de evento él dispara (*action Type*)*.*

Los atributos *eventType* y *actionType* son opcionales. Pueden ser inferidos por el valor de *role,* si se utilizan valores reservados. En caso contrario, son obligatorios atributos *eventType* y *actionType*. Los valores reservados utilizados para definir los *roles* de un elemento <simpleAction> se establecen en la Tabla 26.

Tabla 26 — Valores de role de acción reservados asociados a las 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	*start*	*attribution*

Si un valor *eventType* es “attribution”, el elemento <simpleAction> también debe definir obligatoriamente el valor que será atribuido, a través de su atributo *value.* Si ese valor se especifica como “&anyName”, (donde el $ es símbolo reservado a anyName es cadena de caracteres, excepto uno de los nombres reservados para apeles), el valor a ser atribuido se debe obtener de la propiedad vinculada a *role=”anyName”, definida en un elemento <bind> del elemento <link> que utiliza el conector.* Si ese valor no se puede obtener, no se debe realizar ninguna atribución.

NOTA 1 Declarar el atributo *role*=“anyName” en un elemento <bind> de un <link>, implica tener un papel implícitamente declarado como: <attributeAssessment role=“anyName” eventType=“attribution” attributeType=“nodeProperty”/>. Ese es el único caso posible de un elemento <bind> referirse a un papel no definido explícitamente en un conector.

NOTA 2 En el caso de value=“$anyName”, el valor que se atribuirá debe ser obligatoriamente el valor de una propiedad (elemento <property>) de un componente de la misma composición en la que el eslabón (elemento <link>) que referencia el evento es referido, o una propiedad de composición en la que el eslabón es definido, o una propiedad de un elemento accesible a través de un puerto de composición en el que el eslabón es definido o, aún, una propiedad de un elemento

accesible a través de un puerto de una composición (elementos <port> o <switchPort> anidada en la misma composición en que el eslabón es definido.

Como en el caso de los elementos <simpleCondition>, la cardinalidad del *role* especifica el número mínimo (atributo *min*) y máximo (atributo *Max*) de los participantes de un *role* (número de *binds*), cuando el <causalConnector> se utiliza para crear un eslabón. Cuando el valor máximo de la cardinalidad es mayor que uno, varios participantes pueden participar de un mismo role. Cuando tiene valor “unbounded”, el número de *binds* es ilimitado. En los dos últimos casos, un calificador debe, obligatoriamente, ser especificado. La Tabla 27 presenta los posibles valores de calificador.

Tabla 27 — Valores del calificador de acción

Elemento role	Calificador	Semántica
<simpleAction>	“par”	Todas las acciones se deben ejecutar obligatoriamente en paralelo
<simpleAction>	“seq”	Todas las acciones se deben ejecutar obligatoriamente en la secuencia de las declaraciones dada por el *bind*

Un atributo *delay* también puede ser definido por un <simpleAction>, especificando, cuando sea definido, que la acción debe ser obligatoriamente disparada sólo después de esperar por el tiempo especificado. Además de ello, el <simpleAction> también puede definir un atributo *repeat* para ser aplicado al atributo *repetitions* del evento, y un atributo *repeatDelay,* para ser aguardado antes de la repetición de la acción.

Además de todos los atributos mencionados, el elemento <simpleAction> también puede tener atributos definidos en el área funcional *Animation* (atributos *duration* y *by),* si su valor de *event Type* es “attribution” (ver 7.2.13).

El elemento <compoundAction> tiene un atributo *operator* (“par” o “seq”) relacionando sus elementos-hijos: <SimpleAction> y <compoundAction>. Acciones compuestas paralelas ("par") y secuenciales ("seq") especifican que la ejecución de las acciones se deberá realizar obligatoriamente en cualquier orden o en un orden específico, respectivamente. Un atributo de *delay* también puede ser definido, especificando que la acción compuesta se debe aplicar obligatoriamente después del retardo especificado.

Cuando el operador secuencial se utiliza, las acciones deben ser iniciadas obligatoriamente en el orden especificado. Sin embargo, una acción no necesita esperar hasta que la anterior sea completada para entonces ser iniciada.

El módulo ConnectorAssessmentExpression define cuatro elementos: <assessmentStatement>, <attributeAssessment>, <valueAssessment> y <compoundStatement>.

El <attributeAssessment> tiene un atributo *role,* que debe ser obligatoriamente único en el conjunto de *roles* del conector. Como normalmente ocurre, un *role* es un punto de interfaz del conector, que es asociado a las interfaces de los nudos por un <link> que hace referencia al conector. Un <attributeAssessment> también define un tipo de evento (atributo *event Type).*

Si el valor de *eventType* es “selection” (selección), conviene al <attributeAssessment> también definir sobre a qué equipo se refiere la selección (por ejemplo, teclas de un teclado o control remoto), a través de su atributo *key.* Si el valor *eventType* es “presentation”, el atributo *attributeType* especifica el atributo de evento (“occurrences” o “repetitions”) o el estado del evento (“state”); si el valor *eventType* es “selection”, el atributo *attribute Type* es opcional y, en caso de estar presente, puede tener el valor “occurrences” (default) o “state”; si el *eventType* es “attribution” el *attributeType* es opcional y puede tener el valor “nodeProperty” (default), “occurrences”, “repetition” o “state”. En el primer caso, el evento representa una propiedad del nudo a ser evaluada, en los otros, el evento representa la evaluación de la propiedad de evento de atribución correspondiente o el estado del evento de atribución. Un valor de compensación (*offset*) se puede agregar a un <attributeAssessment> antes de la comparación (por ejemplo, una compensación se puede agregar a una evaluación de atributo para especificar: “la posición vertical de la pantalla más 50 pixels”.

El elemento <valueAssessment> tiene un atributo *value* que debe asumir obligatoriamente un valor de estado de evento, o cualquier valor a ser comparado con una propiedad del nudo o atributo de evento.

El elemento <assessmentStatement> tiene un atributo *comparator* que compara los valores inferidos desde sus elementos-hijos (elementos <attributeAssessment> y <valueAssessment>):

d) en el caso de <attributeAssessment>: un valor de propiedad del nudo [*eventType* = “attribution” y el *attribute Type* = “nodeProperty”]; o un valor de atributo de evento *[eventType* = (“presentation”, “attribution” o “selection”) y el *attributeType* = (“occurrences” o “repetition”)]; o un estado de evento [*event Type* = (“presentation”, “attribution” o “selection”) y el *attribute Type* = “state”];

e) en el caso del <valueAssessment>: un valor de su atributo *value.*

El elemento <compoundStatement> tiene un atributo *operator* con un valor Booleano (“and” u “or”) relacionando sus elementos-hijos: <AssessmentStatement> o <compoundStatement>. Un atributo *isNegated* también se puede definir para especificar si el elemento-hijo del <compoundStatement> debe ser obligatoriamente negado antes que la operación Booleana sea evaluada.

El elemento <causalConnector> puede tener elementos-hijos (elementos <connectorParam>), que se utilizan para parametrizar valores de los atributos de los conectores. El módulo *ConnectorCommonPart* define el tipo de elemento <connectorParam>, que tiene atributos *name* y *type.*

Para especificar cuáles atributos reciben valores de parámetro definidos por el conector, sus valores se especifican como el nombre del parámetro, precedido por el símbolo $.

EJEMPLO Para parametrizar el atributo *delay,* un parámetro denominado *actionDelay* se define (<connectorParam name="actionDelay" type=”unsignedLong”/>) y el valor “$actionDelay” se utiliza en el atributo (delay=”$actionDelay”).

Los elementos del módulo CausalConnectorFunctionality, sus elementos-hijos y sus atributos deben estar de acuerdo obligatoriamente con la Tabla 28.

Tabla 28 — Módulo *CausalConnectorFunctionality* extendido

Elementos	Atributos	Contenido
causalConnector	*id*	(connectorParam*, (simpleCondition | compoundCondition), (simpleAction | compoundAction))
connectorParam	*name, type*	vacío
simpleCondition	*role, delay, eventType, key, transition, min, max, qualifier*	vacío
compoundCondition	*operator, delay*	((simpleCondition | compoundCondition)+, (assessmentStatement | compoundStatement)*)
simpleAction	*role, delay, event Type, action Type, value, min, max, qualifier, repeat, repeatDelay, duration, by*	vacío
compoundAction	*operator, delay*	(simpleAction | compoundAction)+
assessmentStatement	*comparator*	(attributeAssessment, (attributeAssessment | valueAssessment)
attributeAssessment	*role, eventType, key, attributeType, offset*	vacío
valueAssessment	*value*	vacío
compoundStatement	*operator, isNegated*	(assessmentStatement | compoundStatement)+

El módulo *ConnectorBase* define un elemento denominado <connectorBase>, que permite la agrupación de conectores. Como normalmente ocurre, se recomienda que el elemento <connectorBase> tenga un atributo *id,* que identifica únicamente el elemento dentro de un documento.

El contenido exacto de una base de conectores es especificado por un perfil de lenguaje que utiliza las facilidades ofrecidas por los conectores. Sin embargo, como la definición de conectores no es fácilmente realizada por usuarios inexpertos, la idea es tener usuarios experimentados definiendo los conectores, almacenándolos en bibliotecas (bases de conectores) que puedan ser importadas, tornándolas disponibles a otros usuarios para la creación de eslabones. El Anexo C suministra un ejemplo de definiciones de conectores que pueden ser importadas.

Los elementos del módulo *ConnectorBase,* sus elementos-hijos, y sus atributos deben estar de acuerdo obligatoriamente con la Tabla 29.

Tabla 29 — Módulo *ConnectorBase* extendido

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

7.2.9 Área funcional *Presentation Control*

El alcance del área funcional *Presentation Control* es especificar alternativas de contenido y presentación para un documento. Ese área funcional se divide en cuatro módulos, denominados *TestRule, TestRuleUse, ContentControl* y *DescriptorControl.*

El módulo *TestRule* permite la definición de reglas que, cuando satisfechas, seleccionan alternativas para la presentación del documento. La especificación de reglas en NCL 3.0 se realiza en un módulo separado, porque son útiles para definir tanto componentes cuanto descriptores alternativos.

El elemento <ruleBase> especifica un conjunto de reglas y se debe definir obligatoriamente como elemento-hijo del elemento <head>. Esas reglas pueden ser simples, definidas por el elemento <rule>, o compuestas, definidas por el elemento <compositeRule>. Las reglas simples definen un identificador (atributo *id*), una variable (atributo *var*), un valor (atributo *value*) y un comparador (atributo *comparator*) relacionando la variable a un valor.

La variable debe ser obligatoriamente una propiedad del nudo *settings* (elemento <media> del tipo application/x-ginga-settings), es decir, el atributo *var* debe poseer obligatoriamente el mismo valor del atributo *name* de un elemento <property>, definido como hijo del elemento <media> del tipo application/x-ginga-settings. Las reglas compuestas tienen un identificador (atributo *id)* y un operador Booleano (“and” u “or” – atributo *operator)* relacionando sus reglas-hijas. Como normalmente ocurre, el atributo *id* identifica únicamente los elementos <rule> y <compositeRule> dentro de un documento.

Los elementos del módulo *TestRule,* sus elementos-hijos y sus atributos deben estar de acuerdo obligatoriamente con la Tabla 30.

Tabla 30 — Módulo TestRule extendido

Elementos	Atributos	Contenido
ruleBase	*id*	(importBase|rule|compositeRule)+
Rule	*id, var, comparator, value*	vacío
compositeRule	*id, operator*	(rule | compositeRule)+

El *TestRuleUse* define el elemento <bindRule>, que se utiliza para asociar reglas con componentes de un elemento <switch> o <descriptorSwitch>, a través de sus atributos *rule* y *constituent,* respectivamente.

El elemento del módulo *TestRuleUse* y sus atributos deben estar de acuerdo obligatoriamente con la Tabla 31.

Tabla 31 — Módulo TestRuleUse extendido

Elementos	Atributos	Contenido
bindRule	*constituent, rule*	vacío

El módulo *ContentControl* especifica el elemento <switch>, permitiendo la definición de nudos alternativos a ser elegidos en tiempo de presentación del documento. Las reglas de prueba utilizadas para escoger el componente del switch a ser presentado se definen por el módulo *TestRule,* o son reglas de prueba específicamente definidas e incorporadas en una implementación del formateador NCL. El módulo *ContentControl* también define el elemento <defaultComponent>, cuyo atributo *component* (del tipo IDREF) identifica el elemento *(default)* que debe ser obligatoriamente seleccionado si ninguna de las reglas *bindRule* es evaluada como verdadera.

Para permitir la definición de eslabones que se conectan a anclas del componente elegido después de la evaluación de las reglas de un *switch,* se recomienda que un perfil del lenguaje también incluya el módulo *SwitchInterface,* que permite la definición de interfaces especiales, denominadas <switchPort>.

Los elementos <switch> deben tener obligatoriamente un atributo *id,* que identifica únicamente el elemento dentro de un documento. El atributo *refer* es una extensión definida en el módulo *Reuse* (ver 7.2.11).

Cuando un <context> se define como elemento-hijo de un <switch>, los elementos <link> en forma recursiva contenidos en el elemento <context> deben ser obligatoriamente considerados por un exhibidor NCL apenas si <context> es seleccionado después de la evaluación del switch. En caso contrario, se recomienda que los elementos <link> sean considerados desactivados y obligatoriamente no deben interferir en la presentación del documento.

Los elementos del módulo *ContentControl,* sus elementos-hijos y sus atributos deben estar de acuerdo obligatoriamente con la Tabla 32.

Tabla 32 — Módulo *ContentControl* extendido

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

El módulo *DescriptorControl* especifica el elemento <descriptorSwitch>, que contiene un conjunto de descriptores alternativos a ser asociado a un objeto. Los elementos <descriptorSwitch> deben tener obligatoriamente un atributo *id,* que identifica únicamente el elemento dentro de un documento. Análogamente al elemento <switch>, la elección <descriptorSwitch> se realiza en tiempo de presentación, utilizando reglas de prueba definidas por el módulo *TestRule,* o reglas de prueba específicamente definidas e incorporadas en una implementación del formateador NCL. El módulo *DescriptorControl* también define el elemento <defaultDescriptor>, cuyo atributo *descriptor* (del tipo IDREF) identifica el elemento (default) que debe ser obligatoriamente seleccionado si ninguna de las reglas *bindRule* es evaluada como verdadera.

Los elementos del módulo *DescriptorControl,* sus elementos-hijos y sus atributos deben estar de acuerdo obligatoriamente con la Tabla 33.

Tabla 33 — Módulo *DescriptorControl* extendido

Elementos	Atributos	Contenido
descriptorSwitch	*id*	(defaultDescriptor?, (bindRule | descriptor)*)
defaultDescriptor	*descriptor*	vacío

Durante la presentación de un documento, desde el momento en el que un <switch> es evaluado, se considera resuelto hasta el final de su presentación, es decir, mientras su evento de presentación correspondiente está en el estado ocurriendo o pausado. Durante la presentación de un documento, desde el momento en el que un <descriptorSwitch> es evaluado, se considera resuelto hasta el final de la presentación del elemento <media> que le fue asociado, es decir, mientras cualquier evento de presentación asociado con el elemento <media> esté en el estado ocurriendo o pausado.

NOTA Se recomienda que los formateadores NCL aplacen la evaluación del switch hasta el momento que un eslabón anclado al switch necesite ser evaluado. Conviene que la evaluación del descriptor Switch sea atrasada hasta el momento en el que el objeto refiriéndose a él necesite estar listo para ser presentado.

7.2.10 Área funcional *Timing*

El área funcional *Timing* define el módulo *Timing.* El módulo *Timing* permite la definición de atributos temporales para componentes de un documento. Básicamente, ese módulo define atributos para especificar qué pasa con un objeto al final de su presentación *(freeze)* y la duración ideal de un objeto *(explicitDur).* Esos atributos pueden ser incorporados por los elementos <descriptor>.

7.2.11 Área funcional *Reuse*

NCL permite una gran reutilización de sus elementos. El área funcional *Reuse de NCL* se divide en tres módulos: *Import, EntityReuse* y *ExtendedEntityReuse.*

Para permitir que una base de entidades sea incorporada a otra base ya existente, el módulo *Import* define el elemento <importBase>, que tiene dos atributos: *DocumentURI* y *alias.* El atributo *documentURI* se refiere a un URI correspondiente al documento NCL conteniendo la base a ser importada. El atributo *alias* especifica un nombre a ser utilizado como código cuando sea necesario referirse a elementos de esa base importada.

El nombre del atributo *alias* debe ser obligatoriamente único en un documento y su alcance está supeditado al documento que lo definió. La referencia tendría el formato: *alias#element_id.* La operación de importación es transitiva, es decir, si la baseA importa la baseB que importa la *baseC,* entonces la baseA importa la baseC. Sin embargo, el alias definido para la baseC dentro de la *baseB* obligatoriamente no debe ser tenido en cuenta por la baseA.

Cuando un perfil de lenguaje utiliza el módulo *Import,* se permiten las siguientes especificaciones:

— el elemento <descriptorBase> puede tener un elemento-hijo <importBase> refiriéndose a un URI correspondiente a otro documento NCL conteniendo la base de descriptores a ser importada (en realidad sus elementos-hijos) y anidada. Cuando una base de descriptores es importada, la base de regiones y la base de reglas, si existen en el documento importado, son también automáticamente importadas para las bases de regiones y de reglas del documento correspondiente que realiza la importación;

— el elemento <connectorBase> puede tener un elemento-hijo <importBase> refiriéndose a un URI correspondiente a otra base de conectores a ser importada (en realidad sus elementos-hijos) y anidada;

— el elemento <transitionBase> puede tener un elemento-hijo <importBase> refiriéndose a un URI correspondiente a otra base de transiciones a ser importada (en realidad sus elementos-hijos) y anidada;

— el elemento <ruleBase> puede tener un elemento-hijo <importBase> refiriéndose a un URI correspondiente a otro documento NCL conteniendo la base de reglas que será importada (en verdad sus elementos-hijos) y anidada;

— el elemento <regionBase> puede tener un elemento-hijo <importBase> refiriéndose a un URI correspondiente a otro documento NCL conteniendo la base de regiones que será importada (en verdad sus elementos-hijos) y anidada. Como dicho documento puede tener más de una base de regiones, la base que será importada es identificada por la atribución de su identificador al atributo *baseId*. Aunque NCL defina su modelo de layout, nada impide que un documento NCL utilice otros modelos de layout, desde que estos definan regiones donde los objetos pueden ser presentados, como, por ejemplo, modelos de layout SMIL 2.1. Al importar una <regionBase>, un atributo opcional denominado *región* puede ser especificado, dentro de un elemento <importBase>. Cuando presente, el atributo debe obligatoriamente identificar el *id* de un elemento <region> declarado en el elemento <regionBase> del documento hospedero (documento que hizo la operación de importación). Como consecuencia, todos los elementos <región>, hijos de <regionBase> importados, deben obligatoriamente ser considerados elementos <región> hijos de la región referida por el atributo *region* de <importBase>. Si no se especifica, los elementos <región>, hijos de <regionBase> importado, deben obligatoriamente ser considerados hijos directos del elemento <regionBase> del documento hospedero.

: El elemento <importedDocumentBase> especifica un conjunto de documentos NCL importados y debe obligatoriamente ser definido como elemento-hijo del elemento <head>. El elemento <importedDocumentBase> puede tener un atributo *id*, que identifica únicamente el elemento dentro de un documento.

Un documento NCL puede ser importado a través de un elemento <importNCL>. Todas las bases definidas dentro de un documento NCL, así como el elemento <body> del documento, se importan todos a la vez, a través del elemento <importNCL>. Las bases son tratadas como si cada una hubiese sido importada por un elemento <importBase>. El elemento <body> importado será tratado como un elemento <context>. El elemento <importNCL> no “incluye” el documento NCL referido, sino que sólo hace visible el documento referido para que sus componentes puedan ser reutilizados por el documento que definió el elemento <importNCL>. Así, el <Body> importado, así como cualesquiera de sus nudos, puede ser reutilizado dentro del elemento <body> del documento NCL que realizó la importación.

El elemento <importNCL> tiene dos atributos: *DocumentURI* y *alias.* El *documentURI* se refiere a un URI correspondiente al documento a ser importado. El atributo *alias* especifica un nombre que se usa cuando se realiza una referencia a elementos de ese documento importado. Como en el elemento <importBase>, el nombre debe ser obligatoriamente único (type=ID) y su alcance se supedita al documento que definió el atributo *alias.* La referencia tendría el formato: *alias#element_id.* Es importante notar que el mismo *alias* conviene que se utilice cuando es necesario referirse a elementos definidos en las bases del documento importado (<regionBase>, <connectorBase>, <descriptorBase> etc.).

La operación del elemento <importNCL> también tiene propiedad transitiva, es decir, si el *documentoA* importa *documentoB* que importa *documentoC,* entonces el *documentoA* importa el *documentoC.* Sin embargo, el alias definido para el *documentoC* dentro del *documentoB* obligatoriamente no se debe tener en cuenta por el *documentoA.*

Los elementos del módulo *Import,* sus elementos-hijos y sus atributos deben estar de acuerdo obligatoriamente con la Tabla 34.

Tabla 34 — Módulo *Import* extendido

Elementos	Atributos	Contenido
importBase	*alias, documentURI, region, baseId*	vacío
imported DocumentBase	*id*	(importNCL)+
importNCL	*alias, documentURI*	vacío

El módulo *EntityReuse* permite que un elemento NCL sea reutilizado. Ese módulo define el atributo *refer,* que hace referencia a un elemento *id* que será reutilizado. Sólo <media>, <context>, <body> o <switch> se pueden reutilizar. Un elemento que hace referencia a otro elemento no puede ser reutilizado; es decir, su *id* no puede ser el valor de un atributo *refer.*

Si el nudo referido se define dentro de un documento D importado, el valor del atributo *refer* debe tener obligatoriamente el formato “alias#iD”, donde “alias” es el valor del atributo *alias* asociado al documento D importado.

Cuando un perfil del lenguaje utiliza el módulo *EntityReuse,* puede agregar el atributo *refer* a:

— un elemento <media> o <switch>. En ese caso, el elemento referido debe ser obligatoriamente, respectivamente, un elemento <media> o <switch>, que representa el mismo nudo previamente definido en el propio <body> del documento o en un <body> externo importado. El elemento referido debe contener, obligatoriamente, directamente la definición de todos sus atributos y elementos-hijos;

— un elemento <context>. En ese caso, el elemento referido debe ser obligatoriamente un elemento <context> o <body>, que representa el mismo contexto previamente definido en el <body> del propio documento o en un <body> externo importado. El elemento referido debe contener, obligatoriamente, directamente la definición de todos sus atributos y elementos-hijos.

Cuando un elemento declara un atributo *refer,* todos los atributos y elementos-hijos definidos por el elemento referido son heredados. Todos los otros atributos y elementos-hijos, si son definidos por el elemento que realiza la referencia, deben ser obligatoriamente ignorados por el formateador, excepto el atributo *id* que se debe definir obligatoriamente. La única otra excepción es para elementos <media>, para los cuales nuevos elementos-hijos <area> y <property> pueden ser agregados, y se puede definir un nuevo atributo, *instance*.

Si el nuevo elemento <property> agregado tiene el mismo atributo *name* de un elemento <property> ya existente (definido en el elemento <media> reutilizado), el nuevo elemento <property> agregado debe ser obligatoriamente ignorado. Igualmente, si el nuevo elemento <area> agregado tiene el mismo atributo *id* de un elemento <area> ya existente (definido en el elemento <media> reutilizado), el nuevo elemento <area> agregado debe ser obligatoriamente ignorado. El atributo *instance* se define en el módulo *ExtendedEntityReuse* y tiene “new” como su valor *default* de *string.*

El elemento referido y el elemento que le hace referencia deben ser obligatoriamente considerados el mismo, con relación a sus estructuras de datos. En otras palabras, eso significa que un nudo NCM puede ser representado por más que un elemento NCL. Como nudos contenidos en un nudo de composición NCM definen un conjunto, un nudo NCM puede ser representado por no más que un elemento NCL dentro de una composición. Eso significa que el atributo *id* de un elemento NCL representando un nudo NCM no es sólo un identificador único para el elemento, sino el único identificador correspondiente al nudo NCM en la composición.

NOTA Se puede consultar la NCMCore:2005 para otras informaciones.

EJEMPLO Suponiendo que el elemento NCL (node1) representa un nudo NCM, los elementos NCL que lo mencionan (node1ReuseA, node1ReuseB) representan el mismo nudo NCM. En otras palabras, el nudo NCM único es representado por más de un elemento NCL (node1, node1ReuseA, y node1ReuseB). Más aún, como los nudos contenidos en un nudo de composición NCM definen un conjunto, cada uno de los elementos NCL, node1, node1ReuseA y node1ReuseB, debe ser declarado dentro de una composición diferente.

El elemento referido y el elemento que le hace referencia también deben ser obligatoriamente considerados el mismo nudo con relación a su estructura de presentación, si el atributo *instance* recibe un valor “instSame” o “gradSame”. Por lo tanto, la siguiente semántica debe ser obligatoriamente respetada. Considere el conjunto de elementos <media> compuesto por el elemento <media> referido y todos los elementos <media> que lo mencionan. Si cualquier elemento del subconjunto formado por el elemento <media> referido y todos los otros elementos <media> que tienen el atributo *instance* igual a “instSame” o “gradSame” estuvieran programados para ser presentados, se presume que todos los otros elementos en ese subconjunto, que no son descendentes de un elemento <switch>, están también programados para presentación y, más que eso, cuando éstos se presentan, deben ser obligatoriamente representados por la misma instancia de presentación. Los elementos descendentes de un elemento <switch> también deben tener obligatoriamente el mismo comportamiento, si todas las reglas necesarias para presentarlos se cumplen; en caso contrario, no deben ser programados para presentación. Si el atributo *instance* fuera igual a “instSame”, todos los nudos programados del subconjunto deben ser presentados obligatoriamente en una única instancia, inmediatamente (la instrucción star aplicada en todos los elementos del subconjunto simultáneamente). Si el atributo *instance* fuera igual a “gradSame”, todos los nudos programados son presentados en una única instancia, pero gradualmente, a medida que van sufriendo la instrucción star,

procedente de eslabones etc.. La instancia común en presentación debe obligatoriamente notificar todos los eventos asociados con los elementos <area> y <property> definidos en todos los elementos <media> del subconjunto que fueron programados para presentación. Por otro lado, los elementos <media> en el conjunto que tiene valores de atributo *instance* iguales a "new" obligatoriamente no deben ser programados para presentación. Cuando son individualmente programados para presentación, ningún otro elemento del conjunto tiene su comportamiento afectado, y más aún, nuevas instancias independientes de presentación deben ser obligatoriamente creadas a cada inicio individual de presentación.

7.2.12 Área funcional *Navigational Key*

El área funcional *Navigational Key* define el módulo *KeyNavigation* que ofrece las extensiones necesarias para describir las operaciones de movimiento del foco, definido por un dispositivo, como un control remoto. Básicamente, el módulo define atributos que pueden ser incorporados por elementos <descriptor>.

El atributo *focusIndex* especifica un índice para el elemento <media> al cual el foco puede ser aplicado, cuando ese elemento esté en exhibición. El *focusIndex* puede ser definido en un elemento <property> o en un elemento <descriptor>. Cuando la propiedad no sea definida, el objeto es considerado como si no pudiera recibir el foco. En determinado momento de la presentación, si el foco no hubiera sido aún definido, o si estuviera perdido, un foco será inicialmente aplicado al elemento que se esté presentando que tenga el menor valor de índice.

Los valores del atributo *focusIndex* deben obligatoriamente ser únicos en un documento NCL. Caso contrario, atributos repetidos deben obligatoriamente ser ignorados, si en determinado momento hubiera más de un elemento <media> para ganar foco. Además, cuando un elemento <media> se refiere a otro elemento <media> (utilizando el atributo *refer* especificado en 7.2.11), éste debe obligatoriamente ignorar el *focusIndex* asociado al elemento <media> referido.

El atributo *moveUp* especifica un valor igual al valor del *focusIndex* asociado al elemento sobre el cual será aplicado el foco cuando sea presionada la tecla "flecha hacia arriba" (*"up arrow key"*). El atributo *moveDown* especifica un valor igual al valor del *focusIndex* asociado al elemento sobre el cual el foco será aplicado cuando sea presionada la tecla "flecha hacia abajo" (*"down arrow key"*).

El atributo *moveRight* especifica un valor igual al valor del *focusIndex* asociado al elemento sobre el cual el foco será aplicado cuando la tecla "flecha para la derecha *"*(*"right arrow key"*) sea presionada. El atributo *moveLeft* especifica un valor igual al valor del *focusIndex* asociado al elemento sobre el cual el foco será aplicado cuando la tecla "flecha para la izquierda" (*"left arrow key"*) sea presionada.

Cuando el foco se aplica a un elemento con la propiedad de visibilidad con el valor "false", o a un elemento que no está siendo presentado, el foco actual no debe moverse.

El atributo *focusSrc* puede especificar un contenido alternativo a ser presentado, en vez del contenido de la presentación actual, si un elemento recibe el foco. Ese atributo sigue las mismas reglas del atributo *src* del elemento <media>.

Cuando un elemento recibe un foco, el marco *(box)* definido por los atributos de posicionamiento del elemento debe ser obligatoriamente destacado. El atributo *focusBorderColor* define el color de destaque y puede recibir los nombres reservados de color: "white", "black", "silver", "gray", "red", "maroon", "fuchsia", "purple", "lime", "green", "yellow", "olive", "blue", "navy", "aqua", o "teal". El atributo *focusBorder Width* define el ancho en pixels del borde en destaque (0 significa que no aparece ningún borde, valores positivos significan que el borde está fuera del contenido del objeto y valores negativos significan que el borde se dibuja sobre el contenido del objeto); el atributo *focusBorderTransparency* define la transparencia del color de destaque. El *focusBorderTransparency* debe ser obligatoriamente un valor real entre 0 1, o un valor real en la banda [0,100] terminando con el carácter "%" (por ejemplo, 30 %), con "1" ó "100 %" significando transparencia total y "0" ó "0 %" significando ninguna transparencia. Cuando el *focusBorderColor,* el *focusBorderWidth* o el *focusBorderTransparency* no están definidos, los valores *default* deben obligatoriamente ser asumidos. Esos valores son especificados en las propiedades del elemento <media> de tipo *application/x-ginga-settings*: *default.focusBorderColor*, *default.focusBorderWidth* y *default.focusTransparency*, respectivamente.

Cuando un elemento en foco es seleccionado presionando la tecla de activación, el atributo *focusSelSrc* puede especificar un contenido de media alternativo a ser presentado, en vez de la presentación actual. Este atributo sigue las mismas reglas del atributo *src* del elemento <media>. Cuando esté seleccionada, la caja definida por los atributos de posicionamiento de elemento debe ser obligatoriamente destacada con el color definido por el atributo *selBorderColor* (valor-*default* especificado por el *defaultSelBorderColor* del elemento <media> del tipo *application/xginga-settings),* el ancho del borde en destaque es definido por el atributo *focusBorder Width* y la transparencia del color del borde es definida por el atributo *focusBorderTransparency.*

Cuando un elemento en foco es seleccionado presionando la tecla de activación, el control del foco debe ser obligatoriamente pasado al exhibidor del elemento <media> (*player*). El exhibidor puede entonces seguir sus propias reglas para navegación. El control de foco debe ser obligatoriamente devuelto al formateador NCL cuando la tecla “back” es presionada. En ese caso, el foco va para el elemento identificado por el atributo *service.currentFocus* del nudo settings (elemento <media> del tipo application/x-ginga-settings.

El control del foco puede también ser pasado alterando el valor del atributo service.currentKeyMaster del nodo *settings* (elemento <media> del tipo application/x-ginga-settings). Esto puede ser hecho vía acción de un enlace(<link> element), vía un comando de edición NCL ejecutado por un código imperativo de un nodo (objeto NCLua o NCLet). El exhibidor del nodo que posee el control corriente no puede cambiar directamente el valor de ese atributo.

7.2.13 Área funcional *Animation*

Animación es en realidad una combinación de dos factores: soporte al dibujo del objeto y soporte al movimiento del objeto o, más propiamente, soporte para la alteración del objeto en función del tiempo.

NCL no es un formato de contenido y, como tal, no tiene soporte para la creación de objetos de media y no tiene un método generalizado para alterar el contenido del objeto de media. Al contrario, NCL es un formato de escalonamiento y orquestación. Eso significa que NCL no se puede utilizar para hacer dibujos animados, pero se puede utilizar para exhibir objetos de dibujo animado en el contexto de una presentación general y para alterar las propiedades de sincronización y exhibición de un objeto de un dibujo animado (o cualquier otro) globalmente, mientras el objeto esté siendo exhibido.

Las primitivas de animación NCL permiten que los valores de propiedades de los nudos sean alterados durante una duración determinada. Como la animación NCL puede ser computacionalmente intensa, solamente es soportada por el perfil EDTV y sólo las propiedades que definen valores numéricos y colores pueden ser animadas.

El área funcional *Animation* define el módulo *Animation* que suministra las extensiones necesarias para describir qué pasa cuando el valor de una propiedad de un nudo es alterado. Básicamente, el módulo define atributos que pueden ser incorporados por los elementos <simpleAction> de un conector, si su valor *eventType* es “attribution”. Dos nuevos atributos son definidos: *duration* e *by.*

Al atribuir un nuevo valor para una propiedad, la alteración es instantánea por *default* (*duration=”0”*), pero la alteración también se puede realizar durante un período explícitamente declarado, especificado por el atributo *duration.*

Además de ello, al atribuir un nuevo valor a una propiedad, la alteración del valor antiguo para el nuevo puede ser lineal por *default* (*by*=”indefinite”), o hecha paso a paso, con el paso especificado por el atributo *by.*

La combinación de las definiciones de los atributos *duration* y *by* ofrece la definición de cómo (de forma discreta o lineal) la alteración se debe realizar obligatoriamente y su intervalo de transformación.

7.2.14 Área funcional SMIL *Transition Effects*

El área funcional *Transition Effects* se divide en dos módulos: *TransitionBase* y *Transition.*

NOTA Se puede consultar el SMIL 2.1 Specification:2005 para otras informaciones.

El módulo *TransitionBase* es definido por la NCL 3.0 y consiste en un elemento <transitionBase> que especifica un conjunto de efectos de transición y se debe definir obligatoriamente como elemento-hijo del elemento <head>.

El elemento <transitionBase>, sus elementos-hijos y sus atributos deben estar de acuerdo obligatoriamente con la Tabla 35.

Tabla 35 — Módulo *TransitionBase* extendido

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

El módulo *Transition* está basado en las especificaciones SMIL 2.1. El módulo solamente tiene un elemento llamado <transition>.

NOTA Se puede consultar el SMIL 2.1 Specification, 2005 para otras informaciones.

En el perfil NCL 3.0 DTV Avanzado, el elemento <transition> es especificado en el elemento <transitionBase> y permite que un estándar (*template*) de transición sea definido. Cada elemento <transition> define un estándar único de transición y debe obligatoriamente tener un atributo *id* para que pueda ser una referencia.

Siete atributos del elemento <transition> son derivados de la especificación del módulo BasicTransitions de SMIL: *type; subtype; dur; startProgress; endProgress; direction;* y *fadeColor.*

Las transiciones se clasifican de acuerdo con una taxonomía de dos niveles: tipos y subtipos. Cada tipo de transición describe un grupo de transiciones que están íntimamente relacionadas. Dentro de ese tipo, cada una de las transiciones individuales se asocia a un subtipo que enfatiza las características distintas de la transición.

El atributo *type* es obligatorio y utilizado para especificar la transición general. Si el tipo nominado no es soportado por el formateador NCL, la transición debe ser obligatoriamente ignorada. Eso no es una condición de error, pues las implementaciones son libres para ignorar transiciones.

El atributo *subtype* suministra el control específico para la transición. Ese atributo es opcional y, si es especificado, debe ser obligatoriamente uno de los subtipos de transición apropiados para el tipo correspondiente. Si ese atributo no se especifica, la transición debe ser obligatoriamente revertida para el subtipo *default* del tipo especificado. Sólo los subtipos para los cinco tipos de transición obligatorios, listados en la Tabla 36, deben ser obligatoriamente soportados; los otros tipos y subtipos definidos en la especificación SMIL son opcionales.

Tabla 36 — Tipos y subtipos de transición exigidos

Tipo de transition	Subtipo default
barWipe	leftToRight
irisWipe	rectangle
clockWipe	clockwiseTwelve
snakeWipe	topLeftHorizontal
fade	crossfade

El atributo *dur* especifica la duración de la transición. La duración *default* es de 1 s.

El atributo *startProgress* especifica la cantidad de efecto de transición del inicio de la ejecución. Los valores permitidos son números reales en la banda [0.0,1.0]. Por ejemplo, puede quererse iniciar un *crossfade* con imagen destino ya transparente en 30 % inicialmente. Para ese caso, el *startProgress* sería 0.3. El valor *default* es 0.0.

El atributo *endProgress* especifica la cantidad de efecto de transición al término de la ejecución. Los valores permitidos son números reales en la banda [0.0,1.0] y el valor de ese atributo debe ser obligatoriamente mayor o igual al valor del atributo *startProgress.* Si *endProgress* fuera igual a *startProgress*, entonces la transmisión permanece en un valor fijo por la duración de la transmisión. El valor *default* es 1.0.

El atributo *direction* especifica la dirección en que ocurrirá la transición. Los valores permitidos son “forward” “reverse”. El valor *default* es “forward”. No todas las transiciones tendrán interpretaciones reversas significantes.

Por ejemplo, un *crossfade* no es una transición geométrica y, por lo tanto, no tiene interpretación de dirección reversa. Las transiciones que no tienen interpretación reversa deben tener el atributo *direction* ignorado y el valor *default* "forward" asumido.

Si el valor del atributo *type* es “fade” y el valor del atributo *subtype* es “fadeToColor” o “fadeFromColor” (valores de subtipo que no son obligatorios en una implementación Ginga), entonces el atributo *fadeColor* especifica el color final o inicial del *fade.* Si el valor del atributo *type* no es “fade”, o si el valor del atributo *subtype* no es “fadeToColor” o “fadeFromColor”, entonces el atributo *fadeColor* debe ser obligatoriamente ignorado. El valor *default* es “black”.

El módulo *Transition* también define los atributos a ser utilizados en los elementos <descriptor>, para los estándares de transición definidos por los elementos <transition>: atributos *transIn* y *transOut.* Las transiciones especificadas con un atributo *transIn* empezarán en el comienzo de la duración activa de los elementos de media (cuando la presentación del objeto empieza). Las transiciones especificadas con un atributo *transOut* terminarán al final de la duración activa de los elementos de media (cuando la presentación del objeto pasa del estado ocurriendo a preparado).

Los atributos *transIn* y *transOut* son agregados a los elementos <descriptor>. El valor por *default* de ambos atributos es una *string* vacía, que indica que, obligatoriamente, ninguna transición debe ser realizada. Esas propiedades también pueden ser definidas usando elementos <property>.

El valor de los atributos *transIn* y *transOut* es una lista separada por punto y coma de los identificadores de transición. Cada uno de los identificadores debe corresponder obligatoriamente al valor del identificador XML de uno de los elementos de transición anteriormente definidos en el elemento <transitionBase>. El alcance de la lista separada por punto y coma es permitir que los autores especifiquen un conjunto de transiciones alternativas (*fallback*) si la transición preferida no está disponible.

La primera transición en la lista se debe realizar si el agente del usuario implementa esta transición. Si esa transición no está disponible, entonces la segunda transición en la lista se debe realizar, y así sucesivamente. Si el valor del atributo *transIn* o *transOut* no corresponde al valor del identificador XML de ninguno de los elementos de transición previamente definidos, entonces hay un error. En caso de ocurrir ese error, el valor del atributo que se debe considerar como siendo una string vacía y, entonces, obligatoriamente, ninguna transición se debe realizar.

Todas las transiciones definidas en el módulo *Transition* aceptan cuatro atributos adicionales (procedentes de la especificación del módulo TranstionModifiers de SMIL) que se pueden utilizar para controlar la apariencia de las transiciones. El atributo *horzRepeat* especifica cuántas veces se realizará el estándar de transiciones a lo largo del eje horizontal. El valor *default* es 1 (el estándar ocurre una vez horizontalmente). El atributo *vertRepeat* especifica cuántas veces se realizará el estándar de transición a lo largo del eje vertical. El valor *default* es 1 (el estándar ocurre una vez verticalmente). El atributo *borderWidth* especifica el ancho de un borde generado a lo largo de un área apagada. Los valores permitidos son enteros mayores o iguales a 0. Si el valor de *borderWidth* es igual a 0, entonces conviene que ningún borde se genere a lo largo del área apagada. El valor *default* es 0. Si el valor del atributo *type* no es "fade", entonces el atributo *borderColor* especifica el contenido de un borde generado a lo largo de un área apagada. Si el valor de ese atributo es un color, entonces el borde generado se rellena con el color definido. Si el valor de este atributo es “blend”, entonces el borde generado es una mezcla aditiva (o *blur*) de las fuentes de media. El valor *default* para ese atributo es “black”.

El elemento del módulo *Transition* extendido, sus elementos-hijos y sus atributos deben estar de acuerdo obligatoriamente con la Tabla 37.

Tabla 37 — Módulo *Transition* extendido

Elementos	Atributos	Contenido
transition	*id, type, subtype, dur, startProgress, endProgress, direction, fadeColor, horzRepeat, vertRepeat, borderWidth, borderColor*	vacío

7.2.15 Área funcional Metainformation

Una metainformación no contiene informaciones de contenido utilizadas o exhibidas durante la presentación de un documento. En vez de eso, contiene informaciones sobre el contenido utilizado o exhibido. El área funcional *Metainformation* está compuesta por el módulo *metainformation, derivado del módulo* Metainformation SMIL.

El elemento <meta> especifica un único par de propiedad/valor en los atributos *name* y *content,* respectivamente. El elemento <metadata> contiene informaciones que también se relacionan con la metainformación del documento. Actúa como el elemento raíz del árbol RDF. El elemento <metadata> puede tener como elementos-hijos: Elementos RDF y sus subelementos

NOTA Se puede consultar la RDF:1999 para otras informaciones.

Los elementos del módulo *Metainformation,* sus elementos-hijos y sus atributos deben estar de acuerdo obligatoriamente con la Tabla 38.

Tabla 38 — Módulo *Meta-Information* extendido

Elementos	Atributos	Contenido
meta	*name, content*	vacío
metadata	*empty*	RDF tree

7.3 Perfiles del lenguaje NCL para el SBTVD

7.3.1 Módulos de perfiles

Cada perfil NCL puede agrupar un subconjunto de módulos NCL, permitiendo la creación de lenguajes de acuerdo con las necesidades de los usuarios.

Cualquier documento de conformidad con los perfiles NCL debe tener obligatoriamente el elemento <ncl> como su elemento-raíz.

El perfil NCL 3.0 completo, también denominado perfil Lenguaje NCL 3.0, es el “perfil completo” del lenguaje NCL 3.0. Comprende todos los módulos NCL (incluso los discutidos en 7.2) y suministra todas las facilidades para la autoría declarativa de documentos NCL.

Los perfiles definidos para el SBTVD son:

a) perfil NCL 3.0 DTV Avanzado. Incluye los siguientes módulos de NCL 3.0: *Structure, Layout, Media, Context, MediaContentAnchor, CompositeNodeInterface, PropertyAnchor, SwitchInterface, Descriptor, Linking, CausalConnectorFunctionality, ConnectorBase, TestRule, TestRuleUse, ContentControl, DescriptorControl, Timing, Import, EntityReuse, ExtendedEntityReuse KeyNavigation, Animation, TransitionBase, Transition* y *Meta-Information*. Las tablas presentadas en 7.2 muestran cada elemento de cada módulo, ya extendidos por los atributos y elementos hijos heredados de otros módulos por ese perfil (ver los esquemas XML en 7.3.2);

b) perfil NCL 3.0 CausalConnector. Permite la creación de conectores hipermedia simples. El perfil incluye los siguientes módulos: *Structure, CausalConnectorFunctionality* y *ConnectorBase*. En el perfil, el elemento <body> del módulo *Structure* no es utilizado (ver los esquemas XML en 7.3.3);

c) perfil NCL 3.0 DTV Básico. Incluye los siguientes módulos: *Structure, Layout, Media, Context, MediaContentAnchor, CompositeNodeInterface, PropertyAnchor, SwitchInterface, Descriptor, Linking, CausalConnectorFunctionality, ConnectorBase, TestRule, TestRuleUse, ContentControl, DescriptorControl, Timing, Import, EntityReuse, ExtendedEntityReuse* y *KeyNavigation*. Las tablas presentadas en 7.3.4 muestran cada elemento de cada módulo de ese perfil, ya extendidos por los atributos y elementos-hijos heredados de otros módulos (ver los esquemas XML en 7.3.5).

7.3.2 Esquema del perfil NCL 3.0 DTV avanzado

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.ncl.org.br/NCL3.0/Metainformation"
  xmlns:transition="http://www.ncl.org.br/NCL3.0/Transition"
  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.ncl.org.br/NCL3.0/Metainformation"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30Metainformation.xsd"/>
<import namespace="http://www.ncl.org.br/NCL3.0/Transition"
  schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30Transition.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"/>
        <element ref="profile:meta"/>
        <element ref="profile:metadata"/>
      </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"/>
        <element ref="profile:meta"/>
        <element ref="profile:metadata"/>
      </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="transition:transAttrs"/>
      </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"/>

  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = -->
  <!-- Transition                                    -->
  <!-- = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =  -->

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

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

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

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

7.3.3 Esquema del 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/NCL3.0/Import"
schemaLocation="http://www.ncl.org.br/NCL3.0/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 y elementos del perfil NCL 3.0 DTV básico

Los elementos y sus atributos, utilizados en el perfil NCL 3.0 DTV Básico, se presentan en las Tablas 39 a 55. Se quiere resaltar que los atributos y contenidos (elementos-hijos) de elementos se pueden definir en el módulo en sí o en el perfil NCL DTV Básico que agrupa los módulos. Los atributos obligatorios están subrayados. En las Tablas 39 a 55, se emplean los siguientes símbolos: (?) opcional (cero o una ocurrencia), (|) o (*) cero o más ocurrencias, (+) una o más ocurrencias.

Tabla 39 — Elementos y atributos del módulo *Structure* extendido utilizados en el perfil DTV Básico

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

Tabla 40 — Elementos y atributos del módulo *Layout* extendido utilizados en el perfil DTV Básico

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

Tabla 41 — Elementos y atributos del Módulo Media extendido utilizados en el perfil DTV Básico

Elementos	Atributos	Contenido
media	*id, src, refer, instance, type, descriptor*	(area|property)*

Tabla 42 — Elementos y atributos del módulo *Context* extendido utilizados en el perfil DTV Básico

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

Tabla 43 — Elementos y atributos del módulo *MediaContentAnchor* extendido utilizados en el perfil DTV Básico

Elementos	Atributos	Contenido
área	*id, coords, begin, end, beginText, beginPosition endText, endPosition, first, last, label, clip*	vacío

Tabla 44 — Elementos y atributos del módulo *CompositeNodeInterface* extendido utilizados en el perfil DTV Básico

Elementos	Atributos	Contenido
port	*id, component, interface*	vacío

Tabla 45 — Elementos y atributos del módulo PropertyAnchor extendido utilizados en el perfil DTV Básico

Elementos	Atributos	Contenido
property	*name, value*	vacío

Tabla 46 — Elementos y atributos del módulo *SwitchInterface* extendido utilizados en el perfil DTV Básico

Elementos	Atributos	Contenido
switchPort	*id*	mapping+
mapping	*component, interfaz*	vacío

Tabla 47 — Elementos y atributos del módulo *Descriptor* extendido utilizados en el perfil DTV Básico

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

Tabla 48 — Elementos y atributos del módulo *Linking* extendido utilizados en el perfil DTV Básico

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

Tabla 49 — Elementos y atributos del módulo *CausalConnectorFunctionality* extendido utilizados en el perfil DTV Básico

Elementos	Atributos	Contenido
causalConnector	*id*	(connectorParam*, (simpleCondition | compoundCondition), (simpleAction | compoundAction))
connectorParam	*name, type*	vacío
simpleCondition	*role, delay, event Type, key, transition, min, max, qualifier*	vacío
compoundCondition	*operator, delay*	((simpleCondition | compoundCondition)+, (assessmentStatement | compoundStatement)*)
simpleAction	*role, delay, eventType, actionType, value, min, max, qualifier, repeat, repeatDelay*	vacío
compoundAction	*operator, delay*	(simpleAction | compoundAction)+
assessmentStatement	*comparator*	(attributeAssessment, (attributeAssessment | valueAssessment))
attributeAssessment	*role, eventType, key, attributeType, offset*	vacío
valueAssessment	*value*	vacío
compoundStatement	*operator, isNegated*	(assessmentStatement | compoundStatement)+

Tabla 50 — Elementos y atributos del módulo *ConnectorBase* extendido utilizados en el perfil DTV Básico

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

Tabla 51 — Elementos y atributos del módulo *TestRule* extendido utilizados en el perfil DTV Básico

Elementos	Atributos	Contenido
ruleBase	*id*	(importBase|rule|compositeRule)+
Rule	*id, var, comparator, value*	vacío
compositeRule	*id, operator*	(rule | compositeRule)+

Tabla 52 — Elementos y atributos del módulo *TestRuleUse* extendido utilizados en el perfil DTV Básico

Elementos	Atributos	Contenido
bindRule	*constituent, rule*	vacío

Tabla 53 — Elementos y atributos del módulo *ContentControl* extendido utilizados en el perfil DTV Básico

Elementos	Atributos	Contenido
switch	*id, refer*	(defaultComponent?, (switch Port| bindRule|media| context | switch)*)
defaultComponent	*component*	vacío

Tabla 54 — Elementos y atributos del módulo *DescriptorControl* extendido utilizados en el perfil DTV Básico

Elementos	Atributos	Contenido
descriptorSwitch	*id*	(defaultDescriptor?, (bindRule | descriptor)*)
defaultDescriptor	*descriptor*	vacío

Tabla 55 — Elementos y atributos del módulo *Import* extendido utilizados en el perfil DTV Básico

Elementos	Atributos	Contenido
importBase	*alias, documentURI, región*	vacío
imported DocumentBase	*id*	(importNCL)+
importNCL	*alias, documentURI,*	vacío

7.3.5 Esquema del 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 media en presentaciones NCL

8.1 Implementación modular de Ginga-NCL

La presentación de un documento NCL requiere el control de la sincronización de varios objetos de media (especificados a través del elemento <media>). Para cada objeto de media, un *player* (exhibidor de media) puede ser cargado para el control del objeto y de sus eventos NCL. Un exhibidor de media (o su adaptador) debe ser capaz obligatoriamente de recibir los comandos de presentación, controlar las máquinas de estado de los eventos del objeto de media controlado y responder a las exigencias del formateador.

Para favorecer la incorporación de players de terceros dentro de la implementación de la arquitectura Ginga, se recomienda un proyecto modular del Ginga-NCL para separar los players de la máquina de presentación (formateador NCL).

La Figura 4 sugiere una organización modular para la implementación Ginga-NCL. Los exhibidores son módulos *plug-in* de la máquina de presentación. Por ser interesante utilizar los *players* ya existentes, que puedan tener interfaces propietarias que no sean compatibles con las exigidas por la máquina de presentación, será necesario desarrollar módulos para hacer las adaptaciones necesarias. En ese caso, el exhibidor será constituido por un adaptador además del exhibidor no-conforme en sí.

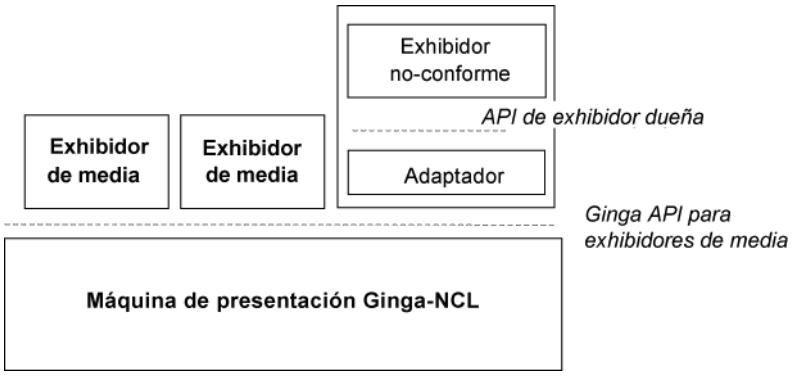

Figura 4 — API para integrar los *players* a la implementación de la máquina de presentación NCL

NOTA Como la arquitectura e implementación Ginga-NCL es una elección de cada receptor, esta subsección no tiene la intención de estandarizar la sintaxis de la API de la máquina de presentación. El objetivo de esta sección es sólo definir el comportamiento esperado del exhibidor de media cuando se está controlando objetos que forman parte de un documento NCL.

8.2 Comportamiento esperado de los exhibidores básicos de media

8.2.1 Instrucción *start* para eventos de presentación

Antes de enviar una instrucción *start*, se recomienda que el formateador encuentre el exhibidor de media más apropiado, con base en el tipo de contenido que será exhibido. Para ello, el formateador tiene en cuenta el atributo *player* asociado al objeto de media que será exhibido. Si ese atributo no fuera especificado, el formateador debe obligatoriamente tener en cuenta el atributo *type* del elemento <media>. Si ese atributo tampoco es especificado, el formateador debe obligatoriamente considerar la extensión del archivo especificado en el atributo *src* del elemento <media>.

La instrucción *start* emitida por un formateador debe obligatoriamente informar al exhibidor de media los siguientes parámetros: el objeto de media que será controlado, sus descriptores asociados (cuando sean especificados), una lista de eventos (presentación, selección o atribución) que tienen que ser monitoreados por el exhibidor de media, el evento de presentación que será iniciado (llamado evento principal), un tiempo de compensación (*offset-time*) y un tiempo de retardo opcionales.

Un objeto de media debe ser obligatoriamente derivado de un elemento <media>, cuyo atributo *src* se debe usar obligatoriamente, por el exhibidor de media, para localizar el contenido e iniciar la presentación. Si el contenido no se puede localizar, o si el exhibidor de media no sabe cómo manejar el tipo de contenido, el exhibidor de media debe obligatoriamente encerrar la operación de iniciación sin realizar ninguna acción.

El descriptor que será utilizado debe obligatoriamente ser elegido por el formateador siguiendo las directrices especificadas en el documento NCL. Si la instrucción *start* resulta de una acción de un enlace que tenga un descriptor explícitamente declarado en su elemento <bind> (atributo *descriptor* del elemento <bind>), el descriptor resultante debe obligatoriamente mezclar los atributos del descriptor especificado por <bind> con los atributos del descriptor especificado en el elemento <media> correspondiente. Para atributos en común, la información del descriptor de <bind> debe obligatoriamente sobreponer los datos del descriptor de <media>. Si el elemento <bind> no contiene un descriptor explícito, el descriptor calculado por el formateador debe obligatoriamente ser el descriptor especificado por el elemento <media>, si el atributo hubiera sido especificado. Caso contrario, un descriptor *default* para el tipo de <media> específico debe obligatoriamente ser elegido por el formateador.

Conviene que la lista de eventos a ser monitoreados por un exhibidor de media también sea computada por el formateador, teniendo en cuenta la especificación del documento NCL. Debe obligatoriamente chequear todos los eslabones de los cuales participa el objeto de media y el descriptor resultante. Al computar los eventos a ser monitoreados, el formateador debe considerar obligatoriamente la perspectiva del objeto de media, es decir, el camino de los varios elementos <body> y <context> para alcanzar en profundidad el elemento <media>

correspondiente. Conviene que apenas eslabones contenidos en esos elementos <body> y <context> sean considerados en la computación de los eventos monitoreados.

El parámetro *offset-time* es opcional y tiene “cero” como su valor *default.* El parámetro es significativo solamente para media continua o estática con duración explícita. En ese caso, el parámetro define un tiempo de compensación, desde el inicio *(beginning-time)* del evento principal, desde el cual la presentación de ese evento debe ser inmediatamente iniciada (es decir, comanda el exhibidor para saltar para el *beginning-time + offset-time).* Obviamente, el valor del *offset-time* debe ser obligatoriamente menor que la duración del evento principal.

Si el *offset-time* es mayor que cero, el exhibidor de media debe obligatoriamente colocar el evento principal en el estado ocurriendo (*occurring*), pero la transición de inicio (*starts*) del evento obligatoriamente no debe ser notificada. Si el *offset-time* es cero, el exhibidor de media debe obligatoriamente colocar el evento principal en el estado ocurriendo y notificar la ocurrencia de la transición de inicio. Los eventos que tendrían sus tiempos de finalización anteriores al tiempo de inicio del evento principal y eventos que tendrían sus tiempos de inicio después del tiempo de finalización del evento principal no necesitan ser monitoreados por el exhibidor de media (conviene que el formateador haga esa verificación cuando construya la lista de eventos monitoreados).

Los eventos monitoreados que tienen tiempos de inicio antes del tiempo de inicio del evento principal y tiempos de término tras el tiempo de arranque (*beginning-time + offset-time*) del evento principal deben obligatoriamente ser colocados en el estado de ocurriendo, pero sus transiciones de inicio obligatoriamente no pueden ser notificadas (ellos que dependen de esa transición obligatoriamente no pueden ser disparados). Los eventos monitoreados que teriamndrían sus tiempos de término tras el tiempo de inicio del evento principal, pero antes del tiempo de partida (*beginning-time + offset-time*), deben obligatoriamente tener su atributo *occurrences* incrementado, pero las transiciones de inicio y término (stops) no deben ser notificadas. Los eventos monitoreados que tienen su tiempo de inicio antes del momento de arranque (*beginning time + offset-time*) y tiempo de término tras el tiempo de arranque deben obligatoriamente ser colocados en el estado de ocurriendo, pero la transición de inicio correspondiente obligatoriamente no debe ser notificada.

El tiempo de retardo también es un parámetro opcional y su valor *default* también es “cero”. Si es mayor que cero, ese parámetro contiene un tiempo a ser esperado por el exhibidor de media antes de iniciar su presentación.

Si un exhibidor de media recibe una instrucción de *start* para un objeto ya siendo presentado (pausado o no), debe obligatoriamente ignorar la instrucción y mantener el control de la presentación en marcha. En este caso, el elemento <simpleAction> que causó la instrucción *start* no debe causar cualquier transición en la máquina de estados del evento a él asociado.

8.2.2 Instrucción stop para eventos de presentación

La instrucción *stop* solamente tiene que identificar un objeto de media que ya está siendo controlado. Identificar el objeto de media significa identificar el elemento <media> y los descriptores correspondientes. Así, si un elemento <simpleAction> con *actionType* igual a “stop” es unido por un enlace a una interfaz de nodo, la interfaz debe ser ignorada cuando la acción sea ejecutada.

Si el objeto no está siendo presentado (es decir, si ninguno de los eventos en la lista de eventos del objeto está en el estado *occurring* o *paused*) y el exhibidor de media no está aguardando debido a una instrucción atrasada de *start,* la instrucción *stop* debe ser obligatoriamente ignorada. Si el objeto está siendo presentado, el evento principal (evento pasado como parámetro cuando el objeto de media fue iniciado) y todos los eventos monitoreados en el estado *occurring* o *paused,* con tiempo de finalización igual o anterior al tiempo de término del evento principal deben obligatoriamente transitar para el estado *sleeping* y sus transiciones *stops* deben ser obligatoriamente notificadas.

Los eventos monitoreados en el estado *occurring* o *paused* con tiempo de finalización posterior al tiempo de finalización del evento principal deben ser obligatoriamente colocados en el estado *sleeping,* pero sus transiciones *stops* no deben ser notificadas y su atributo *occurrences* no se debe incrementar. La presentación del contenido del objeto debe ser obligatoriamente parada. Si el atributo *repetitions* del evento es mayor que cero, debe ser obligatoriamente disminuido en uno y la presentación del evento principal debe obligatoriamente reiniciar después del tiempo entre repeticiones (el tiempo de retardo entre repeticiones debe obligatoriamente haber sido transmitido al exhibidor de media como parámetro de retardo de inicio). Si el objeto de media está esperando para

ser presentado después de una instrucción *start* atrasada y si es emitida una instrucción *stop*, la instrucción de *start* anterior debe ser obligatoriamente retirada.

"NOTA Cuando todos los objetos de media que se refieren al flujo elemental que transporta el video principal de un programa estén en estado *sleeping,* la exhibición del video principal ocupa toda la pantalla. Solamente por medio de un objeto de media en ejecución se puede redimensionar el video principal. Lo mismo acontece con el audio principal de un programa, cuando todos los objetos de media que se refieren al flujo elemental que transporta el audio principal de un programa están en estado *sleeping,* la exhibición del audio principal adquiere el 100% de su volumen.

8.2.3 Instrucción abort para eventos de presentación

La instrucción *abort* necesita sólo identificar un objeto de media que ya está siendo controlado. Si un elemento <simpleAction> con el action Type igual a "abort" se conecta por un eslabón a una interfaz de nudo, la interfaz debe ser ignorada cuando la acción es ejecutada.

Si el objeto no está siendo presentado y no está esperando ser presentado después de una instrucción de *start* atrasada, la instrucción *abort* debe ser obligatoriamente ignorada. Si el objeto está siendo presentado, evento principal y todos los eventos monitoreados en el estado *occurring* o *paused* deben obligatoriamente transitar para el estado *sleeping,* y sus transiciones *aborts* deben ser obligatoriamente notificadas. Cualquier presentación de contenido debe obligatoriamente parar.

Si el atributo *repetitions* del evento es mayor que cero, debe ser colocado obligatoriamente en cero y la presentación del objeto de media no debe ser reiniciada. Si el objeto de media está esperando para ser presentado después de una instrucción *start* atrasada y es emitida una instrucción *abort*, la instrucción *start* debe ser obligatoriamente retirada.

8.2.4 Instrucción pause para eventos de presentación

La instrucción *pause* sólo necesita identificar un objeto de media que ya está siendo controlado. Si un elemento <simpleAction> con el *action Type* igual a "pause" se conecta por un eslabón a una interfaz de nudo, la interfaz debe ser ignorada cuando la acción es ejecutada.

Si el objeto no está siendo presentado (si el evento principal, pasado como parámetro cuando el objeto de media fue iniciado, no está en el estado *occurring)* y el exhibidor de media no está esperando por el retardo de inicio, la instrucción debe ser obligatoriamente ignorada. Si el objeto está siendo presentado, el evento principal y todos los eventos monitoreados en el estado *occurring* deben obligatoriamente pasar para el estado *paused y* sus transiciones *pauses* deben ser obligatoriamente notificadas. La presentación del objeto debe ser obligatoriamente pausada y el tiempo de pausa transcurrido obligatoriamente no se debe tener en cuenta como parte de la duración del objeto. Como ejemplo, si un objeto tiene duración explícita de 30 s, y después de 25 s se pausa, incluso si el objeto permanece pausado durante 5 min, después del reinicio, el evento principal del objeto debe obligatoriamente permanecer ocurriendo por 5 S. Si el evento principal aún no está ocurriendo porque el exhibidor de media está esperando el retardo de inicio, el objeto de media debe obligatoriamente esperar por una instrucción *resume* para continuar aguardando el retardo de inicio.

8.2.5 Instrucción resume para eventos de presentación

La instrucción resume necesita apenas identificar un objeto de media que ya está siendo controlado. Si un elemento <simpleAction> con el *actionType* igual a "resume" se conecta por un eslabón a una interfaz de nudo, la interfaz debe ser ignorada cuando la acción es ejecutada.

Si el objeto no está pausado (si el evento principal, pasado como parámetro cuando el objeto de media fue iniciado, no está en el estado *paused*) y el exhibidor de media no está pausado (esperando por el retardo de inicio), la instrucción debe ser obligatoriamente ignorada.

Si el exhibidor de media está pausado aguardando el retardo de inicio, debe obligatoriamente retomar la exhibición desde el instante en que fue pausado. Si el evento principal está en el estado *paused,* el evento principal y todos los eventos monitoreados en el estado *paused* deben ser obligatoriamente colocados en el estado *occurring* y sus transiciones *resumes* deben ser obligatoriamente notificadas.

8.2.6 Instrucción *start* para eventos de atribución

La instrucción *start* puede ser aplicada a un atributo de un objeto, independientemente del hecho de que el objeto se está o no presentando (en ese último caso, aunque el objeto no se esté presentando, su exhibidor de media ya debe obligatoriamente estar instanciado). La instrucción *start* debe identificar el objeto de media siendo controlado, un evento de atribución monitoreado, un valor que será atribuido al atributo que definió el evento, la duración de la atribución y el paso de la atribución.

Al aplicar un valor al atributo, el exhibidor de media debe obligatoriamente pasar la máquina del estado de evento de atribución al estado *occurring* y, después de terminada la atribución, nuevamente para el estado *sleeping,* generando la transición *starts* y a continuación la transición *stops.*

Para cada evento de atribución monitoreado, si el exhibidor de media altera, por su propia cuenta, el valor correspondiente de un atributo, éste debe proceder obligatoriamente como si hubiera recibido una instrucción de *start* externa.

8.2.7 Instrucción *addEvent*

La instrucción *addEvent* es emitida en el caso de edición de un comando de edición NCL *addInterface* (ver Sección 11). La instrucción necesita apenas identificar un objeto de media que ya esté siendo controlado y un nuevo evento que deba ser obligatoriamente incluido para ser monitoreado.

Todas las reglas aplicadas a la intersección de eventos monitoreados con el evento principal se deberán aplicar obligatoriamente al nuevo evento. Si el tiempo de inicio del nuevo evento es anterior al tiempo actual del objeto y el tiempo de finalización del nuevo evento es posterior al tiempo actual del objeto, el nuevo evento debe ser colocado obligatoriamente en el mismo estado del evento principal (*occurring* o *paused*), sin notificar la transición correspondiente.

8.2.8 Instrucción *removeEvent*

La instrucción *removeEvent* es emitida en el caso de recepción de un comando de NCL *removeInterface*. La instrucción necesita identificar un objeto de media que ya esté siendo controlado y un nuevo evento que ya no tiene que ser controlado. El estado del evento debe ser colocado obligatoriamente en el estado *sleeping* sin generar ninguna transición.

8.2.9 Finalización natural de una presentación

Los eventos de un objeto, con duración explícita o intrínseca, normalmente terminan sus presentaciones naturalmente, sin necesitar instrucciones externas. En ese caso, el exhibidor de media debe obligatoriamente pasar el evento para el estado *sleeping* y notificar la transición *stops.* Lo mismo debe ser obligatoriamente hecho para eventos monitoreados en el estado *occurring* con el mismo tiempo de finalización del evento principal o con tiempo de finalización desconocido, cuando el evento principal termina. Los eventos en el estado *occurring* con tiempo de finalización posterior al tiempo de finalización del evento principal deben ser obligatoriamente colocados en el estado *sleeping* sin generar la transición *stops* y sin incrementar el atributo *occurences.* Es importante resaltar que, si el evento principal corresponde a un ancla temporal interna al objeto, cuando la presentación de ese ancla termine, toda la presentación del objeto de media debe obligatoriamente terminar.

8.3 Comportamiento esperado de los exhibidores de media después de instrucciones aplicadas a los objetos de composición

NOTA Los conceptos establecidos en esta subsección también se aplican a los elementos <media> del tipo “application/x-ginga-NCL”, que se comportarán como si fueran nudos de composición constituidos por sus elementos <body>.

8.3.1 Eslabones refiriendo nudos de composición

Un <simpleCondition> o <simpleAction> con valor del atributo *eventType* igual a “presentation” puede ser asociado por un eslabón a un nudo de composición (representado por un elemento <context> o <body>) como un todo (es decir, sin que una de sus interfaces sea informada). Como normalmente ocurre, la máquina de estado del

evento de presentación definido por el nudo de composición debe ser obligatoriamente controlada como especificado en 7.2.8.

De forma análoga, un <attributeAssessment>, con valor de atributo *eventType* igual a “presentation” y *attributeType* igual a “state”, “occurrences” o “repetitions” puede ser asociado por un eslabón a un nudo de composición (representado por un elemento <context> o <body>) como un todo. Se recomienda que el valor del atributo derive de la máquina de estado del evento de presentación definido por el nudo de composición.

Si una <simpleAction> con valor de atributo *eventType* igual a “presentation” es asociada por un enlace a un nodo de composición (representado por un elemento <context> o <body>) como un todo (o sea, sin que una de sus interfaces sea informada), la instrucción debe obligatoriamente ser reflejada en las máquinas de estado de evento de los nodos hijos de la composición.

8.3.2 Empezando la presentación de un contexto

Si un elemento <context> o <body> participa en un papel (*role*) *action* cuyo tipo de acción es "start", cuando esa acción es accionada, la instrucción *start* también se debe aplicar obligatoriamente a todos los eventos de presentación mapeados por los puertos de los elementos <context> o <body>.

Si el autor quisiera iniciar la presentación usando un puerto específico, también debe indicar obligatoriamente el *id* de <port> como valor de *interfaz* <bind>.

8.3.3 Parando la presentación de un contexto

Si un elemento <context> o <body> participa en un papel (*role*) *action* cuyo tipo de acción es "stop", cuando esa acción es accionada, la instrucción *stop* también se debe aplicar obligatoriamente a todos los eventos de presentación de los nudos hijos de la composición.

Si la composición contiene eslabones siendo evaluados (o con su evaluación pausada), las evaluaciones deben ser obligatoriamente suspendidas y obligatoriamente ninguna acción debe ser accionada.

8.3.4 Abortando la presentación de un contexto

Si un elemento <context> o <body> participa en un papel (*role*) *action* cuyo tipo de acción es "abort", cuando esa acción es accionada, la instrucción *abort* también se debe aplicar obligatoriamente a todos los eventos de presentación de los nudos hijos de la composición.

Si la composición contiene eslabones siendo evaluados (o con su evaluación pausada), las evaluaciones deben ser obligatoriamente suspendidas y obligatoriamente ninguna acción debe ser accionada.

8.3.5 Pausando la presentación de un contexto

Si un elemento <context> o <body> participa en un papel *(role) action* cuyo tipo de acción es "pause", cuando esa acción es accionada, la instrucción *pause* también se debe aplicar obligatoriamente a todos los eventos de presentación de los nudos hijos de la composición que estén en el estado *occurring.*

Si la composición contiene eslabones siendo evaluados, todas las evaluaciones deben ser obligatoriamente suspendidas hasta que una acción *resume, stop* o *abort* sea emitida.

Si la composición contiene nudos-hijos con eventos de presentación en el estado *paused* cuando la acción *paused* en la composición es emitida, ésos nudos deben ser obligatoriamente identificados, porque, si la composición recibe una instrucción *resume,* esos eventos obligatoriamente no deben ser retomados.

8.3.6 Retomando la presentación de un contexto

Si un elemento <context> o <body> participa en un papel (*role*) *action* cuyo tipo de acción es "resume", cuando esa acción sea accionada, la instrucción *resume* también se debe aplicar obligatoriamente a todos los eventos de presentación de los nudos hijos de la composición que estén en el estado *paused,* excepto los que ya estaban pausados antes de la composición ser pausada.

Si la composición contiene eslabones con evaluaciones pausadas, deben ser obligatoriamente reiniciadas.

8.4 Relación entre las máquinas de estado de eventos de presentación de un nudo y la máquina de estado del evento de presentación de su nudo de composición padre

NOTA Los conceptos establecidos en esta sub-sección también se aplican a elementos <media> del tipo “application/x-ginga-NCL”, que se comportarán como si fueran nudos de composición constituidos por sus elementos <body>.

Siempre que el evento de presentación de un nudo (media o composición) va al estado *occurring,* el evento de presentación del nudo de composición que contiene el nudo también debe obligatoriamente entrar en el estado *occurring.*

Cuando todos los nudos-hijos de un nudo de composición tengan sus eventos de presentación en el estado *sleeping,* el evento de presentación del nudo de composición también debe obligatoriamente estar en el estado *sleeping.*

Los nudos de composición no necesitan inferir transiciones *aborts a partir de* sus nudos-hijos. Esas transiciones en los eventos de presentación de nudos de composición deben ocurrir obligatoriamente sólo cuando se aplican instrucciones directamente a su evento de presentación (ver 8.3).

Cuando todos los nudos hijos de un nudo de composición tienen sus eventos de presentación en un estado diferente de *occurring* y al menos uno de los nudos tiene su evento principal en el estado *paused,* el evento de presentación del nudo de composición también debe estar en el estado *paused.*

Si un elemento <switch> es iniciado, pero no define un componente *default* y ninguna de las reglas <bindRule> referenciadas es evaluada como verdadera, la presentación *switch* obligatoriamente no debe entrar en el estado *occurring.*

8.5 Comportamiento esperado de los exhibidores de media imperativa en aplicativos NCL

Objetos procedurales pueden ser insertados en documentos NCL, trayendo capacidades computacionales adicionales a los documentos declarativos. La forma de agregar objetos procedurales en documentos NCL es definir un elemento <media> cuyo contenido (localizado por el atributo *src)* es el código procedural a ser ejecutado. Los perfiles EDTV y BDTV de NCL 3.0 permiten que dos tipos de media sean asociados con el elemento <media>: Application/x-ginga-NCLua, para códigos procedurales Lua (extensión de archivo .lua); y application/x-ginga-NCLet, para códigos procedurales Java (Xlet) (extensión de archivo .class o .jar).

Autores pueden definir eslabones NCL para iniciar, parar, pausar, retomar o abortar la ejecución de un código procedural. Un exhibidor procedural (máquina de ejecución del lenguaje) debe proveer obligatoriamente la interfaz del ambiente de ejecución procedural con el formateador NCL.

Análogamente a lo realizado por los exhibidores de contenidos de media convencional, los exhibidores procedurales deben obligatoriamente controlar las máquinas de estado de los eventos asociados con el nudo procedural NCL (NCLua o NCLet). Como ejemplo, si un código termina su ejecución, el exhibidor debe obligatoriamente generar la transición *stops* en la máquina de estado de presentación del evento correspondiente a la ejecución procedural.

NCL permite que la ejecución del código procedural sea sincronizada con otros objetos NCL (procedural o no). Un elemento <media> conteniendo un código procedural también puede definir anclas (a través de elementos <area>) y propiedades (a través de elementos <property>).

Un código procedural puede ser asociado a elementos <area> (a través del atributo *label).* Si eslabones externos empiezan, paran, pausan, retoman o abortan la presentación del ancla, *callbacks* en el código procedural deben ser obligatoriamente disparados. La forma de cómo esos *callbacks* se definen es responsabilidad de cada código procedural asociado con el objeto procedural NCL.

Por otro lado, un código procedural puede también comandar el inicio, parada, pausa, reconquista o aborto de esas anclas, a través de una API ofrecida por el lenguaje procedural. Esas transiciones se pueden utilizar como condiciones de eslabones NCL para disparar acciones en otros objetos NCL del mismo documento. Así, una sincronización de dos vías puede ser establecida entre el código procedural y el resto del documento NCL.

La otra forma que un código procedural puede ser sincronizado con otros objetos NCL es a través de elementos <property>. Un elemento <property> definido como hijo de un elemento <media>, representando un código procedural, puede ser mapeado para un trecho de código (función, método etc.) o para un atributo del código. Cuando es mapeado para un trecho de código, una acción de eslabón "set" aplicada a la propiedad debe obligatoriamente causar la ejecución del código con los valores atribuidos interpretados como parámetros de entrada. El atributo *name* del elemento <property> se debe utilizar obligatoriamente para identificar el trecho de código procedural.

Un elemento <property> definido como hijo de un elemento <media> representa un código procedural, también puede estar asociado a un *assessment role* de un eslabón NCL. En ese caso, el formateador NCL debe obligatoriamente cuestionar el valor de la propiedad para evaluar la expresión del eslabón. Si el elemento <property> es mapeado para un atributo de código, su valor debe ser retornado obligatoriamente por el exhibidor procedural al formateador NCL. Si el elemento <property> es mapeado para un atributo de código, su valor debe obligatoriamente ser retornado por el exhibidor procedural al formateador NCL. Si el elemento <property> es mapeado para un trecho de código, debe obligatoriamente ser llamado y el valor del resultado de su ejecución debe obligatoriamente ser retomado por el exhibidor procedural al formateador NCL.

La instrucción *start* emitida por un formateador debe obligatoriamente informar al exhibidor imperativo los siguientes parámetros: el objeto imperativo que será controlado, sus descriptores asociados, una lista de eventos (definidos por los elementos <area> y <property>, hijos del elemento <media> que define el objeto procedural) que necesitan ser monitoreados por el exhibidor procedural, el identificador *(id)* del elemento <area> asociado al código procedural a ser ejecutado, y un tiempo de retardo, opcional. Desde el atributo *src,* el exhibidor procedural debe intentar localizar el código procedural e iniciar su ejecución. Si el contenido no se puede localizar, el exhibidor procedural debe obligatoriamente finalizar la operación de inicialización sin realizar ninguna acción.

La lista de eventos a ser monitoreados por un exhibidor procedural se aconseja también que sea computada por el formateador, teniendo en cuenta la especificación del documento NCL. Debe obligatoriamente chequear todos los eslabones de los cuales participa el objeto de media procedural y el descriptor resultante. Al computar los eventos a ser monitoreados, el formateador debe considerar obligatoriamente la perspectiva del objeto de media procedural, es decir, el camino de los varios elementos <body> y <context> para alcanzar en profundidad el elemento <media> correspondiente. Apenas eslabones contenidos en esos elementos <body> y <context> deben ser considerados en la computación de los eventos monitoreados.

Como con todos los tipos de elemento <media>, el tiempo de retardo es un parámetro opcional, y su valor *default* es "cero". Si es mayor que cero, ese parámetro contiene un tiempo a ser esperado por el exhibidor procedural antes de iniciar la ejecución.

A diferencia de los procedimientos realizados para otros tipos de elementos <media>, si un exhibidor procedural recibe una instrucción *start* para un evento asociado a un elemento <area> y ese evento esté en el estado *sleeping,* debe dar inicio a la ejecución del código procedural asociado al elemento, incluso si otra parte del código procedural del objeto de media está en ejecución (pausado o no). Sin embargo, si el evento asociado al elemento <area> destino está en el estado *occurring* o *paused,* la instrucción *start* debe ser ignorada por el exhibidor procedural que continuará controlando la ejecución anteriormente iniciada. Como consecuencia, a diferencia de lo que ocurre para los otros elementos <media>, una acción <simpleAction> con el atributo *actionType* igual a "stop", "pause", "resume" o "abort" debe conectarse, a través de un eslabón, a una interfaz del nudo procedural, que no debe ser ignorada cuando la acción se aplica.

La instrucción *start* emitida por un formateador para un evento asociado a un elemento <property> puede ser aplicada a un objeto imperativo, independientemente del hecho de que esté siendo ejecutado o no (en ese último caso, aunque el objeto no esté siendo ejecutado, su exhibidor imperativo debe obligatoriamente ya haber sido instanciado). La instrucción *start* necesita identificar el objeto imperativo en ejecución, un evento de atribución monitoreado y un valor que será pasado al código imperativo asociado al evento.

Para cada evento de atribución monitoreado, si el exhibidor procedural cambia por sí mismo el valor del atributo, debe proceder obligatoriamente como si hubiese recibido una instrucción externa de *start*.

Se recomienda que lenguajes procedurales ofrezcan una API que permita que los códigos procedurales cuestionen cualquier valor de propiedades predefinidas o dinámicas del nudo *settings* NCL (elemento <media> del tipo "application/x-ginga-settings"). Sin embargo, se debe observar que no se permite atribuir valores a esas propiedades directamente. Las propiedades de los nudos del tipo application/x-ginga-settings sólo pueden ser modificadas a través del uso de eslabones NCL.

9 Transmisión de contenido y eventos NCL

9.1 Bases privadas

El núcleo de la máquina de presentación Ginga-NCL está compuesto por el formateador NCL y su módulo Gestor de Base Privada.

El formateador NCL es responsable por recibir un documento NCL y controlar su presentación, intentando garantizar que las relaciones especificadas entre los objetos de media sean respetadas. El formateador trata con documentos NCL que son recolectados dentro de una estructura de datos conocida como base privada. Ginga asocia una base privada a un canal de televisión. Los documentos NCL en una base privada pueden ser iniciados, pausados, retomados, parados y pueden referirse unos a los otros.

El Gestor de Base Privada es responsable por recibir comandos de edición de documentos NCL y por la edición de los documentos NCL activos (documentos siendo presentados).

Esta subsección trata solamente de comandos de edición enviados por el canal de difusión terrestre.

NOTA Los mismos comandos de edición también pueden ser recibidos por el canal de interactividad o por eventos generados por los objetos imperativos NCLua y NCLet.

Ginga adopta secciones MPEG-2 específicas (identificadas por el campo tableID de la Sección MPEG-2) para transportar los comandos de edición en flujos elementales MPEG-2 TS, cuando los comandos son enviados por el canal de difusión terrestre.

Los Comandos de Edición son empaquetados en una estructura llamada *descriptor de evento.* Cada dsriptor de evento tiene una esteructura compuesta, básicamente, por un *id* (identificación), una referencia de tiempo y un campo de datos privados. La identificación define de forma unívoca cada evento.

La referencia de tiempo indica el exacto momento de disparar el evento. Un tiempo de referencia igual a cero informa que el evento debe, obligatoriamente, ser inmediatamente disparado después de ser recibido (eventos transportando este tipo de referencia de tiempo son comúnmente conocidos como eventos *"do it now"*). El campo de datos privados ofrece soporte para parámetros del evento, como presenta la Figura 5.

Sintaxis	Número de bits
EventDescriptor ( ) {	
eventId	16
eventNPT	33
privateDataLenght	8
commandTag	8
sequenceNumber	7
finalFlag	1
privateDataPayload	8 a 1928
FCS	8
}	

Figura 5 – Descriptor de evento para comandos de edición

El campo *commandTag* identifica en forma unívoca los comandos de edición, como especificado en la Tabla 56. Para permitir enviar un comando completo con más de 255 bytes en más de un descriptor de evento, todos los descriptores de un mismo comando deben obligatoriamente ser numerados y enviados en secuencia (o sea, no puede ser multiplexado con otros comandos de edición con el mismo *commandTag),* con el *finalFlag* igual a 1, excepto para el último receptor, que debe obligatoriamente tener el campo *finalFlag* igual a 0. El *privateDataPayload* contiene los parámetros de comando de edición. Y por último, el campo FCS contiene un *checksum* de todo el campo *privateData,* incluso el *privateDataLength*.

EJEMPLO Eventos de flujo DSM-CC pueden ser usados para transportar descriptores de eventos. El protocolo de carrusel de objetos DSM-CC permite la transmisión cíclica de objetos de eventos de flujo y sistemas de archivo. Los objetos de eventos de flujo son utilizados para mapear nombres de eventos de flujo en *ids* de eventos de flujo, definidos por los descriptores de evento. Los objetos de eventos de flujo son utilizados para informar a Ginga sobre eventos de flujo DSM-CC que pueden ser recibidos. Los nombres de los eventos permiten especificar tipos de eventos, ofreciendo mayor nivel de abstracción a las

aplicaciones de *middleware*. En este caso, conviene que e el Gestor de la Base Privada, así como los objetos de ejecución imperativos NCL (ejemplo, NCLua, NCLet), sean registrados como observadores de los eventos de flujo con los cuales tratan, utilizando nombres de evento.

Los archivos de documento NCL y los contenidos de objeto de media NCL se organizan en estructuras de sistemas de archivos. Los parámetros de comando de edición, basados en XML, pueden ser directamente transportados en el *playload* de un descriptor de evento o, alternativamente, organizados en estructuras de sistema de archivos a ser transportadas en el canal de difusión de datos o, incluso, ser recibidas por el canal de interactividad.

EJEMPLO Un generador de carrusel DSM-CC podía usarse para unir los sistemas de archivos y los objetos de eventos de flujo en un flujo elemental de datos. Cuando un comando de edición de documentos NCL precisara ser enviado, podría criarse un objeto de eventos DSM-CC, mapeando la *string "nclEditingCommand"* en una *id* de evento de flujo, y entonces colocado en un carrusel de objetos DSM-CC. Uno o más descriptores de evento de flujo DSM-CC con la *id* previamente seleccionada podrían entonces ser creados y enviados en otro flujo elemental MPEG-2 TS. Estos eventos de flujo podrían tener su referencia de tiempo fijada en cero, pero también podrían ser aplazados para ser ejecutados en un tiempo específico. El Gestor de la Base Privada debería registrarse como un oyente "nclEditingCommand" y sería notificado cuando estos eventos de flujo llegasen.

El *commandTag* recibido es utilizado por el Gestor de la Base Privada para interpretar la semántica de la *command string.* Si el *command parameter* basado en XML es lo suficientemente corto, puede ser transportado directamente en el *payload* de los descriptores de evento. Si no, el *privateDatePayload* transporta un conjunto de pares de referencia. En el caso de archivos recibidos por el canal de difusión (documentos en nudos NCL enviados sin solicitación), cada par relaciona un conjunto de caminos de archivos y su respectiva localización en el sistema de transporte. En el caso de archivos recibidos sobre pedido por el canal de interactividad o localizados en el propio receptor, ningún par de referencias necesita ser enviado, excepto el par {uri, "null"} asociado al documento NCL o a la especificación XML del nudo NCL que deberá ser adicionado según el comando de adición correspondiente.

La Tabla 56 muestra las *strings* de comando y, entre paréntesis, los parámetros transportados como contenido *payload* del descriptor de evento *nclEditingCommand*. Los comandos son divididos en tres grupos: el primero para operación de la base privada (para abrir, activar, desactivar, cerrar y guardar bases privadas); el segundo para manejo de documentos (para agregar, remover y guardar un documento en una base privada abierta y para iniciar, pausar, retomar y detener presentaciones de documentos en una base privada activa); y la última para manejar entidades NCL en una base privada abierta. Para cada entidad NCL, fueron definidos los comandos *add* y *remove*. Si una entidad ya existe, el comando *add* tiene la semántica de actualización (alteración).

NOTA El primer grupo de comandos de operación normalmente no viene a través del canal de difusión de datos. Como ya fue mencionado, comandos de edición también pueden ser recibidos por el canal de interactividad o venir de eventos generados por objetos imperativos NCLua y NCLet. Los comandos de edición también pueden ser generados internamente por el middleware.

Tabla 56 — Comandos de edición para el Gestor de la Base Privada Ginga

String de comando	*Tag* de comando	Descripción
openBase (baseId, location)	0x00	Abre una base privada existente localizada por el parámetro location. Si la base privada no existe o si el parámetro location no es informado, una nueva base es creada con el identificador baseId. El parámetro location debe obligatoriamente especificar el dispositivo de almacenaje en el ambiente del receptor y el camino donde está la base que será abierta
activateBase (baseId)	0x01	Activa una base privada abierta. Todas las aplicaciones estarán entonces aptas para ser iniciadas
deactivateBase (baseId)	0x02	Desactiva una base privada abierta. Todas las aplicaciones deben ser terminadas
saveBase (baseId, location)	0x03	Guarda todo el contenido de la base privada en un dispositivo de almacenaje persistente (si lo hubiera). El parámetro location debe obligatoriamente especificar el dispositivo y el camino para guardar la base
closeBase (baseId)	0x04	Cierra la base privada abierta y descarta todo el contenido de la base privada
addDocument (baseId, {uri, id}+)	0x05	Agrega un documento NCL a una base privada abierta. Los archivos del documento NCL pueden ser: i) enviados sin solicitación por la red de difusión de datos; en ese caso, el par {uri, id} es usado para relacionar un conjunto de caminos de archivos especificados en el documento NCL con sus respectivas localizaciones en el sistema de transporte (ver ejemplos en la Sección 12); Los conjuntos de pares de referencia deben obligatoriamente ser suficientes para que el middleware pueda mapear cualquier referencia a archivos presentes en la especificación del documento NCL en su localización concreta en la memoria del dispositivo receptor. ii) recibidos por el canal de interactividad bajo demanda, o ya ser residentes en el receptor; para esos archivos, ningún par {uri, id} tiene que ser enviado, excepto el par {uri, “null”} asociado al documento NCL que debe ser agregado a la base baseId, si el documento NCL no es recibido sin solicitación (pushed file).
removeDocument (baseId, documentId)	0x06	Remueve un documento NCL de una base privada abierta
saveDocument (baseId, documented, location)	0x2E	Grava un documento NCL en un dispositivo de almacenamiento persistente (si disponible). El parámetro location debe especificar el dispositivo y el camino en el dispositivo en el que el documento será gravado. Si el documento NCL está siendo exhibido, primero debe pararse (todos los eventos en el estado *occurring* se deben parar)
startDocument (baseId, documentId, interfaceId, offset, nptBaseId, nptTrigger) NOTA 1 El parámetro offset especifica un valor de tiempo. NOTA 2 El nptTrigger es obligatoriamente un valor de NPT, y nptBaseId es obligatoriamente un identificador de una base de tiempo NPT	0x07	Inicia la reproducción de n documento NCL en una base privada activa, iniciando la presentación a partir de una interfaz específica del documento. La referencia del tiempo transportada al campo nptTrigger establece el punto de inicio del documento, con respecto a la base de tiempo NPT identificada en nptBaseId. Tres casos pueden ocurrir: 1) Si nptTrigger es diferente de cero y mayor o igual al valor de NPT corriente de la base temporal identificada por nptBaseId, se espera hasta que NPT alcance el valor dado en nptTrigger y comienza la exhibición del documento desde su punto inicial en el tiempo + offset 2) Si nptTrigger es diferente de cero y menor que el valor del NPT corriente de la base temporal NPT identificada por nptBaseId, el inicio de la exhibición del documento es inmediato y desplazado en el tiempo de su punto incial del valor “offset + $(NPT - nptTrigger)_{seconds}$”; Solamente en ese caso, el parámetro offset puede recibir un valor negativo, pero offset + $(NPT - nptTrigger)_{seconds}$ debe obligatoriamente ser un valor positivo. 3) Si nptTrigger es igual a 0, la exhibición del documento es inmediata y a partir de su punto inicial en el tiempo + offset
stopDocument (baseId, documentId)	0x08	Detiene la presentación de un documento NCL en una base privada activa. Todos los eventos del documento que están en marcha deben obligatoriamente ser detenidos
pauseDocument (baseId, documentId)	0x09	Pausar la presentación de un documento NCL en una base privada activa. Todos los eventos del documento que están en marcha deben obligatoriamente ser pausados

Tabla 56 (continuación)

***String* de comando**	***Tag* de comando**	**Descripción**
resumeDocument (baseId, documentId)	0x0A	Retoma la presentación de un documento NCL en una base privada activa. Todos los eventos del documento que fueron previamente pausados por el comando de edición pauseDocument deben obligatoriamente ser retomados
saveDocument (baseId, documented, location)	0x2E	Guarda un documento NCL de una base privada abierta en un dispositivo de almacenaje persistente (si hubiera). El parámetro location debe especificar el dispositivo y el camino en el dispositivo donde el documento será guardado. Si el documento NCL está siendo exhibido, debe primero ser detenido (todos los eventos en estado *occurring* deben ser detenidos)
addRegion (baseId, documentId, regionBaseId, regionId, xmlRegion)	0x0B	Agrega un elemento <region> como hijo de otro <region> en el <RegionBase>, o como hijo del <regionBase> (regionId=”null”) de un documento NCL en una base privada
removeRegion (baseId, documentId, regionId)	0x0C	Remueve un elemento <region> de un <regionBase> de un documento NCL en una base privada
addRegionBase (baseId, documentId, xmlRegionBase)	0x0D	Agrega un elemento <regionBase> al elemento <head> de un documento NCL en una base privada. Si la especificación XML del regionBase es enviada en un sistema de transporte como un sistema de archivo, el parámetro xmlRegionBase es sólo una referencia para ese contenido
removeRegionBase (baseId, documentId, regionBaseId)	0x0E	Remueve un elemento <regionBase> del elemento <head> de un documento NCL en una base privada
addRule (baseId, documentId, xmlRule)	0x0F	Agrega un elemento <rule> al <ruleBase> de un documento NCL en una base privada
removeRule (baseId, documentId, ruleId)	0x10	Remueve un elemento <rule> del <ruleBase> de un documento NCL en una base privada
addRuleBase (baseId, documentId, xmlRuleBase)	0x11	Agrega un elemento <ruleBase> al elemento <head> de un documento NCL en una base privada. Si la especificación XML del ruleBase es enviada en un sistema de transporte como un sistema de archivo, el parámetro xmlRuleBase es apenas una referencia para ese contenido
removeRuleBase (baseId, documentId, ruleBaseId)	0x12	Remueve un elemento <ruleBase> del elemento <head> de un documento NCL en una base privada
addConnector (baseId, documentId, xmlConnector)	0x13	Agrega un elemento <connector> al <connectorBase> de un documento NCL en una base privada
removeConnector (baseId, documentId, connectorId)	0x14	Remueve un elemento <connector> del <connectorBase> de un documento NCL en una base privada
addConnectorBase (baseId, documentId, xmlConnectorBase)	0x15	Agrega un elemento <connectorBase> al elemento <head> de un documento NCL en una base privada. Si la especificación XML del connectorBase es enviada en un sistema de transporte como un sistema de archivo, el parámetro xmlConnectorBase es apenas una referencia para ese contenido
removeConnectorBase (baseId, documentId, connectorBaseId)	0x16	Remueve un elemento <connectorBase> del elemento <head> de un documento NCL en una base privada
addDescriptor (baseId, documentId, xmlDescriptor)	0x17	Agrega un elemento <descriptor> al <descriptorBase> de un documento NCL en una base privada
removeDescriptor (baseId, documentId, descriptorId)	0x18	Remueve un elemento <descriptor> del <descriptorBase> de un documento NCL en una base privada
addDescriptorSwitch (baseId, documentId, xmlDescriptorSwitch)	0x19	Agrega un elemento <descriptorSwitch> al <descriptorBase> de un documento NCL en una base privada. Si la especificación XML del descriptorSwitch es enviada en un sistema de transporte como un sistema de archivo, el parámetro xmlDescriptorSwitch es apenas una referencia para ese contenido
removeDescriptorSwitch (baseId, documentId, descriptorSwitchId)	0x1A	Remueve un elemento <descriptorSwitch> del <descriptorBase> de un documento NCL en una base privada
addDescriptorBase (baseId, documentId, xmlDescriptorBase)	0x1B	Agrega un elemento <descriptorBase> al elemento <head> de un documento NCL en una base privada. Si la especificación XML del descriptorBase es enviada en un sistema de transporte como un sistema de archivo, el parámetro xmlDescriptorBase es apenas una referencia para ese contenido

Tabla 56 (continuación)

***String* de comando**	***Tag* de comando**	**Descripción**
removeDescriptorBase (baseId, documentId, descriptorBaseId)	0x1C	Remueve un elemento <descriptorBase> del elemento <head> de un documento NCL en una base privada
addTransition (baseId, documentId, xmlTransition)	0x1D	Agrega un elemento <transition> al <transitionBase> de un documento NCL en una base privada
removeTransition (baseId, documentId, transitionId)	0x1E	Remueve un elemento <transition> del <transitionBase> de un documento NCL en una base privada
addTransitionBase (baseId, documentId, xmlTransitionBase)	0x1F	Agrega un elemento <transitionBase> al elemento <head> de un documento NCL en una base privada. Si la especificación XML del transitionBase es enviada en un sistema de transporte como un sistema de archivo, el parámetro xmlTransitionBase es apenas una referencia para ese contenido
removeTransitionBase (baseId, documentId, transitionBaseId)	0x20	Remueve un elemento <transitionBase> del elemento <head> de un documento NCL en una base privada
addImportBase (baseId, documentId, docBaseId, xmlImportBase)	0x21	Agrega un elemento <importBase> a la base (elemento <regionBase>, <descriptorBase>, <ruleBase>, <transitionBase>, o <connectorBase>) de un documento NCL en una base privada
removeImportBase (baseId, documentId, docBaseId, documentURI)	0x22	Remueve un elemento <importBase>, cuyo atributo documentURI es identificado por el parámetro documentURI, desde la base (elemento <regionBase>, <descriptorBase>, <ruleBase>, <transitionBase>, o <connectorBase>) de un documento NCL en una base privada
addImportedDocumentBase (baseId, documentId, xmlImportedDocumentBase)	0x23	Agrega un elemento <importedDocumentBase> al elemento <head> de un documento NCL en una base privada
removeImportedDocumentBase (baseId, documentId, importedDocumentBaseId)	0x24	Remueve un elemento <importedDocumentBase> del elemento <head> de un documento NCL en una base privada
addImportNCL (baseId, documentId, xmlImportNCL)	0x25	Agrega un elemento <importNCL> al elemento <importedDocumentBase> de un documento NCL en una base privada
removeImportNCL (baseId, documentId, documentURI)	0x26	Remueve un elemento <importNCL>, cuyo atributo documentURI es identificado por el parámetro documentURI, desde el <importedDocumentBase> de un documento NCL en una base privada
addNode (baseId, documentId, compositeId, {uri, ior}+)	0x27	Agrega un nudo (elemento <media>, <context> o <switch>) a un nudo *de composición* (elemento <body>, <context> o <switch>) de un documento NCL en una base privada. La especificación XML del nudo y su contenido de media pueden: a) ser enviados sin solicitación por la red de difusión de datos; en ese caso, el par {uri, id} es usado para relacionar un conjunto de caminos de archivos, definidos en el documento XML de la especificación del nudo, con sus respectivas localizaciones en el sistema de transporte (ver ejemplos en la Sección 12); NOTA Los conjuntos de pares de referencia deben obligatoriamente ser suficientes para que el middleware pueda mapear cualquier referencia a archivos, presentes en la especificación del nudo, en su localización concreta en la memoria del dispositivo receptor. b) ser recibidos por el canal de interactividad sobre pedido o ya ser residentes en el receptor; para estos archivos, ningún par {uri, id} necesita ser enviado, excepto el par {uri, "null"} asociado a la especialización XML del nudo NCL que deberá ser adicionado en compositeId, en el caso de que el documento XML no se reciba sin solicitación (pushed file)
removeNode(baseId, documentId, compositeId, nodeId)	0x28	Remueve un nudo (elemento <media>, <context> o <switch>) de un nudo de composición (elemento <body>, <context> o <switch>) de un documento NCL en una base privada
addInterface (baseId, documentId, nodeId, xmlInterface)	0x29	Agrega una interfaz (<port>, <area>, <property> o <switchPort>) a un nudo (elemento <media>, <body>, <context> o <switch>) de un documento NCL en una base privada
removeInterface (baseId, documentId, nodeId, interfaceId)	0x2A	Remueve una interfaz (<port>, <area>, <property>, o <switchPort>) de un nudo (elemento <media>, <body>, <context> o <switch>) de un documento NCL en una base privada. La interfaceID debe identificar obligatoriamente un atributo name de un elemento <property> o un atributo id de un elemento <port>, <area> o <switchPort>
addLink (baseId, documentId, compositeId, xmlLink)	0x2B	Agrega un elemento <link> a un nudo de composición (elemento <body>, <context> o <switch>) de un documento NCL en una base privada

Tabla 56 (continuación)

***String* de comando**	***Tag* de comando**	**Descripción**
removeLink (baseId, documentId, compositeId, linkId)	0x2C	Remueve un elemento <link> de un nudo de composición (elemento <body>, <context> o <switch>) de un documento NCL en una base privada
setPropertyValue(baseId, documentId, nodeId, propertyId, value)	0x2D	Atribuye el valor a una propiedad. La propertyId debe identificar obligatoriamente un atributo *name* de un elemento <property> o un atributo *id* de elemento <switchPort>. El <property> o <switchPort> debe obligatoriamente pertenecer a un nudo (elemento <body>, <context>, <switch> o <media>) de un documento NCL en una base privada identificada por los parámetros

Los receptores que solamente implementan el perfil NCL DTV Básico pueden no manejar los siguientes comandos: *pauseDocument, resumeDocument, addTransition, remove Transition, add TransitionBase* y *remove TransitionBase.*

Ginga asocia por lo menos una base privada a cada canal de televisión. Cuando un canal es sintonizado, su base privada correspondiente es abierta y activada por el Gestor de la Base Privada; otras bases privadas deben ser desactivadas. Por razones de seguridad, sólo una única base privada puede estar activa por vez. El modo más simple y restrictivo de gestionar bases privadas es tener una única base privada abierta por vez. Así, si el usuario cambia el canal seleccionado, es recomendable cerrar la base privada actual. En este caso, el comando *openBase* es siempre seguido por el comando *activeBase* y el comando *deactiveBase* nunca se utiliza. Sin embargo, el número de bases privadas que pueden mantenerse abiertas es una decisión de implementación del *middleware*.

Los comandos *add* tienen entidades NCL como sus argumentos (parámetros de comando basados en XML). Si la entidad especificada ya existe o no, la consistencia del documento debe ser obligatoriamente mantenida por el formateador NCL, en el sentido de que todos los atributos de identidad clasificados como obligatorios se deben definir obligatoriamente. Las entidades se definen utilizando una notación sintáctica idéntica a aquella usada por los esquemas NCL, con excepción del comando *addInterface:* el atributo *begin* de un elemento <area> puede recibir el valor “now”, especificando el NPT actual del *nodeId,* que debe ser obligatoriamente el video principal MPEG siendo reproducido por el decodificador de *hardware.*

Los identificadores utilizados en los comandos deben estar de acuerdo obligatoriamente con la Tabla 57.

Tabla 57 — Identificadores que se utilizan en los comandos de edición

Identificadores	**Definición**
baseId	Identificadores de canal de radiodifusión especificados por el SBTVD
DocumentId	Atributo *id* de un elemento <ncl> de un documento NCL
nptTrigger	Un valor de NPT
nptBaseId	Identificador *contentId* de una base de tiempo NPT
regionId	Atributo *id* de un elemento <region> de un documento NCL
ruleId	Atributo *id* de un elemento <rule> de un documento NCL
connectorId	Atributo *id* de un elemento <connector> de un documento NCL
descriptorId	Atributo *id* de un elemento <descriptor> de un documento NCL
descriptorSwitchId	Atributo *id* de un elemento <descriptorSwitch> de un documento NCL
transitionId	Atributo *id* de un elemento <transition> de un documento NCL
regionBaseId	Atributo *id* de un elemento <regionBase> de un documento NCL
ruleBaseId	Atributo *id* de un elemento <ruleBase> de un documento NCL

Tabla 57 (continuación)

Identificadores	Definición
connectorBaseId	Atributo *id* de un elemento <connectorBase> de un documento NCL
descriptorBaseId	Atributo *id* de un elemento <descriptorBase> de un documento NCL
transitionBaseId	Atributo *id* de un elemento <transitionBase> de un documento NCL
docBaseId	Atributo *id* de un elemento <regionBase>, <ruleBase>, <connectorBase>, <descriptorBase>, o <transitionBase> de un documento NCL
documentU RI	Atributo documentURI de un elemento <importBase> o un elemento <importNCL> de un documento NCL
importedDocumentBaseId	Atributo *id* de un elemento <importedDocumentBase> de un documento NCL
compositeID	Atributo *id* de un elemento <body>, <context> o <switch> de un documento NCL
nodeId	Atributo *id* de un elemento <body>, <context>, <switch> o <media> de un documento NCL
interfaceId	Atributo *id* de un elemento <port>, <area>, <property> o <switchPort> de un documento NCL
linkId	Atributo *id* de un elemento <link> de un documento NCL
propertyId	Atributo *id* de un elemento <property>, o <switchPort> de un documento NCL

9.2 Esquema XML de los parámetros de comando

Las entidades NCL utilizadas en los comandos de edición deben ser obligatoriamente un documento de conformidad con el perfil de Comando NCL 3.0 definido por el esquema XML a continuación. Conviene que los receptores que apenas implementan el perfil Básico NCL DTV ignoren los elementos y atributos XML relacionados a las funcionalidades de Meta-information y Transition Efects.

Observar que, diferentemente de los documentos NCL, diversos elementos NCL pueden tener el elemento raíz en los parámetros 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.ncl.org.br/NCL3.0/Metainformation"
xmlns:transition="http://www.ncl.org.br/NCL3.0/Transition"
  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"/>
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.ncl.org.br/NCL3.0/Metainformation"
    schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30Metainformation.xsd"/>
..<import namespace="http://www.ncl.org.br/NCL3.0/Transition"
….schemaLocation="http://www.ncl.org.br/NCL3.0/modules/NCL30Transition.xsd"/>

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

  <!--This is a pseudo-element, only defined to show the elements that may 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"/>
  <element ref="profile:meta"/>
  <element ref="profile:metadata"/>
</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"/>
<element ref="profile:meta"/>
<element ref="profile:metadata"/>
</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"/>
```

```
  <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="transition:transAttrs"/>
      </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"/>

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

<!-- Transition -->
<!-- ========================================================== -->
<element name="transition" substitutionGroup="transition:transition"/>

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

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

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

</schema>
```

10 Objetos procedurales Lua en presentaciones NCL

10.1 Lenguaje Lua - Funciones retiradas de la biblioteca de Lua

El lenguaje de *script* adoptado por el Ginga-NCL es Lua (elementos *<media>* del tipo application/x-ginga-NCLua). La definición completa de Lua se presenta en el Anexo B.

Las funciones a continuación son dependientes de plataforma y fueron retiradas:

a) en el módulo *package: loadlib*;

b) en el módulo *os*: *clock, execute, exit, getenv, remove, rename, tmpname* y *setlocale*;

c) en el módulo *debug:* todas las funciones.

10.2 Modelo de ejecución

El ciclo de vida de un objeto NCLua es controlado por el formateador NCL. El formateador es responsable por iniciar la ejecución de un objeto NCLua y por mediar la comunicación entre ese objeto y otros objetos en un documento NCL, como definido en 8.5.

Como con todos los exhibidores de objetos de media, un exhibidor Lua, una vez referido, debe ejecutar los procedimientos de iniciación del objeto NCL que controlará. Sin embargo, diferentemente de los demás exhibidores de media, el código de iniciación debe también ser especificado por el autor del objeto NCLua. Los procedimientos de iniciación se ejecutan una sola vez, para cada instancia, y crean funciones y objetos que se pueden usar durante la ejecución del objeto NCLua y, en particular, registra uno o más tratadores de eventos para la comunicación con el formateador NCL.

Después de la iniciación, la ejecución del objeto NCLua se torna orientada a evento, en ambas direcciones. Es decir, cualquier acción comandada por el formateador NCL es dirigida a los tratadores de evento registrados, y cualquier notificación de cambio de estado de eventos NCL es enviada como un evento al formateador NCL (como por ejemplo, el fin de la ejecución del código imperativo). El exhibidor Lua estará entonces listo para ejecutar cualquier intrusión de *start* o *set* (ver 8.5).

10.3 Módulos adicionales

10.3.1 Módulos obligatorios

Además de la biblioteca estándar de Lua, los siguientes módulos deben ser obligatoriamente ofrecidos y automáticamente cargados:

a) módulo *canvas:* ofrece una API para dibujar primitivas gráficas y manejar imágenes;

b) módulo *event*: permite que aplicaciones NCLua se comuniquen con el *middleware* a través de eventos (eventos NCL, pointer y de teclas);

c) módulo *settings:* exporta una tabla con variables definidas por el autor del documento NCL y variables de ambiente reservadas en un nudo "application/x-ginga-settings";

d) módulo *persistent:* exporta una tabla con variables persistentes, que están disponibles para manipulación apenas por objetos imperativo.

La definición de cada función en los módulos mencionados respeta la siguiente nomenclatura:

funcname (parnamel: partypel [; optnamel: opttypel]) -> retname: rettype

10.3.2 Módulo *canvas*

10.3.2.1 Objeto *canvas*

Cuando un objeto de media NCLua es iniciado, la región del elemento <media> correspondiente (del tipo application/x-ginga-NCLua) queda disponible como la variable global *canvas* para el *script* Lua. Si el elemento de <media> no tiene ninguna región especificada (propiedades *left, right, Top and bottom),* entonces el valor para *canvas* debe ser establecido como “nil”.

Ejemplo: para una región definida en un documento NCL como:

```
<Region iD="luaRegion" width="300" height="100" Top="200" left="20"/>
```

La variable '`canvas`' en un objeto de media asociado a la región “luaRegion” es conectada a un objeto canvas de tamaño 300x100, asociada con la región (20,200).

Un canvas ofrece una API gráfica para ser usada por aplicaciones NCLua. A través de ella es posible dibujar líneas, rectángulos, fuentes, imágenes etc.

Un canvas guarda en su estado atributos bajo los cuales operan las primitivas de dibujo, por ejemplo, si su atributo de color es azul, una llamada a `canvas:drawLine()` dibuja una línea azul en el canvas.

Las coordenadas pasadas son siempre relativas al punto más a la izquierda y a la parte superior del canvas (0,0).

10.3.2.2 Constructores

A través de cualquier canvas es posible crear nuevos canvas y combinarlos a través de operaciones de composición.

---

canvas:new (image_path: String) -> canvas: object

Argumentos

image_path	Camino de la imagen

Valor de retorno

canvas	Canvas representando la imagen

Descripción

Retorna un nuevo canvas cuyo contenido es la imagen recibida como parámetro.

El nuevo canvas debe obligatoriamente mantener los aspectos de transparencia de la imagen original.

---

canvas:new (width, height: Number) -> canvas: object

Argumentos

width	Ancho del canvas
height	Altura del canvas

Valores de retorno

canvas	Nuevo canvas

Descripción

Retorna un nuevo canvas con el tamaño recibido.

Inicialmente todos los pixels deben ser transparentes, obligatoriamente.

10.3.2.3 Atributos

Todos los métodos de atributos tienen el código attr y sirven tanto para leer como para alterar un atributo (con algunas excepciones).

Cuando el método es llamado sin parámetros de entrada el valor corriente del atributo es retornado, en contrapartida, cuando es llamado con parámetros, ésos deben ser los nuevos valores del atributo.

canvas:attrSize () -> width, height: number

Argumentos

Valores de retorno

width	Ancho del canvas
height	Altura del canvas

Descripción

Retorna las dimensiones del canvas.
Es importante observar que no se permite alterar las dimensiones de un canvas.

canvas:attrColor (R, G, B, A: Number)

Argumentos

R	Componente rojo del color
G	Componente verde del color
B	Componente azul del color
A	Componente Alpha del color

Descripción

Altera el color del canvas.
Los colores se pasan en RGBA, donde A varia de 0 (totalmente transparente) a 255 (totalmente opaco). Las primitivas (ver 10.3.3.4) son dibujadas con el color de ese atributo del canvas.

El valor inicial es ‘0,0,0,255’ (negro).

canvas:attrColor (clr_name: string)

Argumentos

clr_name	Nombre del color

Altera el color del canvas.

Los colores pasan a través de una string correspondiendo a uno de los 16 colores NCL predefinidas: 'white', 'aqua', 'lime', 'yellow', 'red', 'fuchsia', 'purple', 'maroon',

'blue', 'navy', 'teal', 'green', 'olive', 'silver', 'gray', 'black'

Para valores en string, el Alpha es opaco (correspondiendo a “A = 255”).

Las primitivas (ver 10.3.3.4) son dibujadas con el color de ese atributo del canvas.

El valor inicial es 'black'.

canvas:attrColor () -> R, G, B, A: number

Valores de retorno

R	Componente rojo del color
G	Componente verde del color
B	Componente azul del color
A	Componente alpha del color

Descripción

Retorna el color del canvas.

canvas:attrFont (face: string; size: number; style: string)

Argumentos

face	Nombre de la fuente, o el camino y el nombre del tipo de la fuente
size	Tamaño de la fuente
style	Estilo de la fuente

Descripción

Altera la fuente del canvas.
Las siguientes fuentes deben estar disponibles obligatoriamente: 'Tiresias'.

El tamaño es en pixels y representa la altura máxima de una línea escrita con la fuente elegida.

Los estilos posibles son: 'bold', 'italic' o 'bold-italic'. El valor `nil` presupone que ninguno de los estilos será usado.

Cualquier valor pasado no soportado debe obligatoriamente generar un error.

El valor inicial de la fuente es indeterminado.

canvas:attrFont () -> face: string; size: number; style: string

Valores de retorno

face	Nombre de la fuente
size	Tamaño de la fuente
style	Estilo de la fuente

Descripción

Retorna la fuente del canvas.

canvas:attrClip (x, y, width, height: Number)

Argumentos

x	Coordenada del área de *clipping*
y	Coordenada del área de *clipping*
width	Ancho del área de *clipping*
height	Altura del área de *clipping*

Descripción

Altera el área de *clipping* del canvas.

Las primitivas de dibujo (ver 10.3.3.4) y el método `canvas:compose()` solo operan dentro de esta región de clipping.

El valor inicial es el canvas entero.

canvas:attrClip () -> x, y, width, height: number

Valores de retorno

x	Coordenada del área de *clipping*
y	Coordenada del área de *clipping*
width	Ancho del área de *clipping*
height	Altura del área de *clipping*

Descripción

Retorna el área de *clipping* del canvas.

canvas:attrCrop (x, y, w, h: number)

Argumentos

x	Coordenada de la región de crop
y	Coordenada de la región de crop
w	Anchura de la región de crop
h	Altura de la región de crop

Descripción

Altera la región de crop del canvas.

Solamente la región configurada pasa a ser usada en operaciones de composición.

El valor inicial es el canvas entero.

canvas:attrCrop () -> x, y, w, h: number

Valores de retorno

x	Coordenada de la región de crop
y	Coordenada de la región de crop
w	Anchura de la región de crop
h	Altura de la región de crop

Descripción

Retorna la región de crop del canvas.

canvas:attrFlip (horiz, vert: boolean)

Argumentos

horiz	Si el espejado es horizontal
vert	Si el espejado es vertical

Descripción

Configura el espejado del canvas usado en funciones de composición

El canvas principal no puede tener su valor alterado pues es controlado por el formateador NCL.

canvas:attrFlip () -> horiz, vert: boolean

Argumentos

horiz	Si el espejado es horizontal
vert	Si el espejado es vertical

Descripción

Retorna la configuración de espejado del canvas.

canvas:attrOpacity (opacity: number)

Argumento

opacity	Nuevo valor de opacidad del canvas

Descripción

Altera la opacidad del canvas.

El valor de la opacidad varía entre 0 (transparente) y 255 (opaco).

El canvas principal no puede tener su valor alterado pues es controlado por el formateador NCL.

canvas:attrOpacity () -> opacity: number

Valor de retorno

opacity	Opacidad del canvas

Descripción

Retorna el valor de la opacidad del canvas.

canvas:attrRotation (degrees: number)

Argumento

degrees	Rotación del canvas en grados

Descripción

Configura el atributo de rotación del canvas, que debe ser múltiplo de 90 grados.

El canvas principal no puede tener su valor alterado pues es controlado por el formateador NCL.

canvas:attrRotation () -> degrees: number

Valor de retorno

degrees	Rotación del canvas en grados

Descripción

Retorna el atributo de rotación del canvas.

canvas:attrScale (w, h: number)

Argumentos

w	Anchura de escalonamiento del canvas
h	Altura de escalonamiento del canvas

Descripción

Escalona el canvas con nueva anchura y altura.

Uno de los valores puede ser *true,* indicando que la proporción del canvas debe mantenerse.

El atributo de escalonamiento es independiente del atributo de tamaño, o sea, el tamaño del canvas se mantiene.

El canvas principal no puede tener su valor alterado pues es controlado por el formateador NCL.

canvas:attrScale () -> w, h: number|boolean

Valor de retorno

w	Anchura de escalonamiento del canvas
h	Altura de escalonamiento del canvas

Descripción

Retorna los valores de escalonamiento del canvas.

10.3.2.4 Primitivas

Todos los métodos siguientes tienen en cuenta los atributos del canvas.

---

canvas:drawLine (x1, y1, x2, y2: number)

Argumentos

x1	Extremidad 1 de la línea
y1	Extremidad 1 de la línea
x2	Extremidad 2 de la línea
y2	Extremidad 2 de la línea

Descripción

Dibuja una línea con sus extremidades en (x1,y1) y (x2,y2).

---

canvas:drawRect (mode: string; x, y, width, height: Number)

Argumentos

mode	Modo de dibujo
x	Coordenada del rectángulo
y	Coordenada del rectángulo
width	Ancho del rectángulo
height	Altura del rectángulo

Descripción

Función para dibujo y rellenado de rectángulos.

El parámetro mode puede recibir 'frame' para dibujar apenas el marco del rectángulo o 'fill' para rellenarlo.

---

canvas:drawRoundRect (mode: string; x, y, width, height, arcWidth, arcHeight: number)

Argumentos

mode	Modo de dibujo
x	Coordenada del rectángulo
y	Coordenada del rectángulo
width	Anchura del rectángulo
height	Altura del rectángulo
arcWidth	Anchura del arco de la esquina redondeada
arcHeight	Altura del arco de la esquina redondeada

Descripción

Función para dibujo y rellenado de rectángulos redondeados.

El parámetro mode puede recibir 'frame' para dibujar apenas el contorno o 'fill' para rellenarlo.

canvas:drawPolygon (mode: string) -> drawer: function

Argumentos

mode	Modo de dibujo

Valores de retorno

f	Función de dibujo

Descripción

Método para dibujo y rellenado de polígonos.

El parámetro mode recibe el valor 'open', para dibujar el polígono sin conectar el último punto al primero; 'close', para dibujar el polígono conectando el último punto al primero; o 'fill', para dibujar el polígono enlazando el último punto al primero y colorear la región interior.
La función canvas:drawPolygon retorna una función anónima "drawer" con la firma:

```
function (x, y) end
```

La función retornada recibe las coordenadas del próximo vértice del polígono y retorna a sí mismo con resultado. Ese procedimiento recurrente facilita la composición:

```
canvas:drawPolygon('fill')(1,1)(10,1)(10,10)(1,10)()
```

Al recibir nil la función "drawer" efectúa la operación encadenada. Cualquier llamada subsiguiente debe obligatoriamente generar un error.

canvas:drawEllipse (mode: string; xc, yc, width, height, ang_start, ang_end: number)

Argumentos

mode	Modo de dibujo
xc	Centro de la elipsis
yc	Centro de la elipsis
width	Ancho de la elipsis
height	Altura de la elipsis
ang_start	Ángulo de inicio
ang_end	Ángulo de fin

Descripción

Dibuja elipsis y otras primitivas análogas tales como círculos, arcos y sectores.

El parámetro mode puede recibir 'arc' para dibujar apenas la circunferencia o 'fill' para rellenado interno.

canvas:drawText (x, y: number; text: string)

Argumentos

x	Coordenada del texto
y	Coordenada del texto
text	Texto a ser dibujado

Descripción

Dibuja el texto pasado en la posición (x,y) del canvas utilizando la fuente configurada en `canvas:attrFont()`.

10.3.2.5 Miscelánea

canvas:clear (x, y, w, h: number)

Argumentos

x	Coordenada del área de clear
y	Coordenada del área de clear
w	Anchura del área de clear
h	Altura del área de clear

Descripción

Limpia el canvas con el color configurado en *attrColor.*

Caso no sean pasados los parámetros del parea, se asume que se limpiará el canvas entero.

canvas:flush ()

Descripción

Actualiza el canvas después de operaciones de dibujo y composición.

Es suficiente llamarlo una sola vez después de una secuencia de operaciones.

canvas:compose (x, y: number; src: canvas; [src_x, src_y, src_width, src_height: number])

Argumentos

x	Posición de la composición
y	Posición de la composición
src	Canvas a ser compuesto
src_x	Posición de la composición en el canvas src
src_y	Posición de la composición en el canvas src
src_width	Ancho de la composición en el canvas src
src_height	Altura de la composición en el canvas src

Descripción

Hace sobre el canvas (canvas de destino), en su posición (x,y), la composición pixel a pixel con src (canvas de origen).
Los otros parámetros son opcionales e indican qué parte del canvas src componer. Cuando ausentes, es compuesto el canvas entero.
Esa operación llama la src:flush() automáticamente antes de la composición.

La composición satisface la ecuación:

Cd = Cs*As + Cd*(255 - As)/255

Ad = As*As + Ad*(255 - As)/255

donde:

Cd = color del canvas de destino (canvas)

Ad = alfa del canvas de destino (canvas)

Cs = color del canvas de origen (src)

As = alfa del canvas de origen (src)

Después de la operación el canvas de destino tiene el resultado de la composición y src no sufre ninguna alteración.

canvas:pixel (x, y, R, G, B, A: number)

Argumentos

x	Posición del pixel
y	Posición del pixel
R	Componente rojo del color
G	Componente verde del color
B	Componente azul del color
A	Componente alpha del color

Descripción

Altera el color de un pixel del canvas.

canvas:pixel (x, y: number) -> R, G, B, A: number

Argumentos

x	Posición del pixel
y	Posición del pixel

Valores de retorno

R	Componente rojo del color
G	Componente verde del color
B	Componente azul del color
A	Componente alpha del color

Descripción

Retorna el color de un pixel del canvas.

canvas:measureText (text: string) -> dx, dy: number

Argumentos

text	Texto a ser medido

Valores de retorno

dx	Anchura del texto

dy	Altura del texto

Descripción

Retorna las coordenadas limítrofes para el texto pasado, en el caso que el mismo fuese dibujado en la posición (x,y) del canvas con la fuente configurada en canvas:attrFont().

10.3.3 Módulo *event*

10.3.3.1 Visión general

Este módulo ofrece una API para tratamiento de eventos. A través del mismo el formateador NCL y una aplicación NCLua pueden comunicarse de manera asíncrona.

Una aplicación también puede disfrutar de ese mecanismo internamente, a través de eventos de la clase "user".

Probablemente el uso más común de aplicaciones NCLua será tratando eventos: ya sean éstos eventos NCL (ver 7.2.8) o de interacción con el usuario (por el control remoto, por ejemplo).

Durante su iniciación, antes de volverse orientado a eventos, un *script* Lua debe obligatoriamente registrar una función de tratamiento de eventos. Después de la iniciación, cualquier acción tomada por la aplicación es solamente en respuesta a un evento enviado por el formateador NCL a la función "handler".

```
=== example.lua ===

...             -- código de iniciación

function handler (evt)

...             -- código tratador

end

event.register(handler) -- registro del tratador en el middleware

=== fin ===
```

Entre los tipos de eventos que pueden llegar al handler están todos los eventos generados por el formateador NCL. Un *script* Lua también es capaz de generar eventos, llamados "espontáneos", con una llamada a la función `event.post(evt)`.

10.3.3.2 Funciones

event.post ([dst: string]; evt: event) -> sent: boolean; err_msg: string

Argumentos

dst	Destinatario del evento
evt	Evento a ser posteado

Valores de retorno

sent	Si el evento es enviado con éxito
err_msg	Mensaje de error en caso de fallo

Descripción

Envía el evento pasado.

El parámetro "dst" es el destinatario del evento y puede asumir los valores "in" (envío para la propia aplicación) y "out" (envío para el formateador NCL). El valor default es 'out'.

event.timer (time: number, f: function) -> cancel: function

Argumentos

time	Tiempo en milisegundos
f	Función de *callback*

Valor de retorno

unreg	Función para cancelar el timer

Descripción

Crea un timer que expira después de time (en milisegundos) y entonces llama la función F.

La firma de f es simple, sin recibimiento de parámetros:

```
function f () end
```

El valor de 0 milisegundos es válido. En ese caso, event.timer() debe, obligatoriamente, retornar inmediatamente y `f` debe ser llamada de inmediato.

event.register ([pos: number]; f: function; [class: string]; […: any])

Argumentos

pos	Posición del registro (opcional)
f	Función de *callback*
class	Filtro de clase (opcional)
…	Filtro dependiente de la clase (opcional)

Descripción

Registra la función pasada como un *listener* de eventos, es decir, siempre que ocurra un evento, `f` será llamada. La función `f` es, así, la función de tratamiento de eventos (function "handler").

El parámetro *pos* es opcional e indica la posición en que *f* es registrada. Caso no sea pasada, la función es registrada en último lugar.
El parámetro *class* también es opcional y cuando se pasa indica qué clase de eventos debe recibir la función. Si

el pámetro se especifica, pueden definirse otros filtros dependientes de la clase. El valor *nil* en cualquier posición indica que el parámetro no se debe filtrar.

La firma de `f` es:

```
function f (evt) end -> handled: boolean
```

Donde `evt` es el evento que, al ocurrir, activa la función.

La función puede retornar “true”, para señalizar que el evento fue tratado y, por lo tanto, no debe ser enviado a otros tratadores.

Se recomienda que la función, definida por la aplicación, regrese rápidamente, ya que, mientras esté siendo ejecutada, ningún otro evento será procesado.

El formateador NCL debe garantizar obligatoriamente que las funciones reciban los eventos en el orden en que fueron registradas, y si ninguna de las mismas retorna el valor “true”, el formateador NCL debe notificar los otros tratadores registrados.

event.unregister (f: function)

Argumentos

f	Función de *callback*

Descripción

Saca del registro la función pasada como un *listener,* es decir, nuevos eventos dejarán de ser pasados a `f`.

event.uptime () -> ms: number

Valores de retorno

ms	Tiempo en milisegundos

Descripción

Retorna el número de milisegundos transcurridos desde el inicio de la aplicación.

10.3.3.3 Clases de eventos

La función `event.post()` y el “handler” registrado en `event.register()` reciben eventos como parámetros.

Un evento se describe por una tabla Lua normal, donde el campo `class` es obligatorio e identifica la clase del evento.

Son definidas las siguientes clases de eventos:

Classe `key:`

evt = { class='key', type: string, key: string}

* type puede ser 'press' o 'release'.

* key es el valor de la tecla en cuestión

EJEMPLO evt = { class='key', type='press', key=”0”}

NOTA En la clase key, el filtro dependiente de la clase puede ser *type* y *key*, en ese orden.

Classe `pointer:`

evt = { class=’pointer’, type: string, x=number, y=number }

* type puede ser 'press', 'release' o ‘move’

.

* X e Y se refieren as coordenadas da ocurrencia del evento puntero

NOTA En la clase pointer, el filtro dependiente de la clase solo puede ser *type*.

EJEMPLO evt = { class=’pointer’, type=’press’, x=20, y=50 }

Classe `ncl:`

Las relaciones entre los nudos de media NCL se basan en eventos. Lua tiene acceso a esos eventos a través de `Classe ncl.`

Los eventos pueden actuar en las dos direcciones, es decir, el formateador puede enviar eventos de acción para alterar el estado del exhibidor Lua, y éste, a su vez, puede disparar eventos de transición para indicar cambios de estado.

En los eventos, el campo type debe asumir obligatoriamente uno de los tres valores:

'presentation', 'selection' o 'attribution'

Eventos pueden ser encaminados a anclas específicas o al nodo como un todo, o sea, identificado en el campo label, que asume el nodo entero, cuando ausente.

En el caso de un evento generado por el formateador, el campo action debe tener obligatoriamente uno de los valores:

'start', 'stop', 'abort', 'pause', o 'resume'

Tipo ‘presentation’:

evt = { class='ncl', type='presentation', label='?', action='?' }

Tipo ‘attribution’:

evt = { class='ncl', type='attribution', name='?', action='?', value=’?’ }

Para eventos generados por el exhibidor Lua, el campo "action" debe obligatoriamente asumir uno de los valores:

'start', 'stop', 'abort', 'pause', o 'resume'

Tipo ‘presentation’:

evt = { class='ncl', type='presentation', label='?', action='start'/'stop'/'abort'/'pause'/'resume' }

Tipo ‘selection’:

evt = { class='ncl', type='selection', label='?', action='stop' }

Tipo ‘attribution’:

evt = { class='ncl', type='attribution', name='?', action=’start’/'stop'/’abort’/’pause’/’resume’, value=’?’ }

NOTA En la clase ncl, el filtro dependiente de la clase puede ser *type, label* y *action*, en ese orden.

Classe `edit`:

Esta clase espeja los comandos de edición para el Gerenciador de Bases Privadas (ver Sección 9). Sin embargo, hay una diferencia entre los comandos de edición procedentes de los eventos de flujo DSM-CC y los comandos de edición realizados por los scripts Lua (objetos NCLua). Los primeros alteran, no sólo la presentación de un documento NCL, sino también la especificación de un documento NCL. O sea, al final del proceso es generado un nuevo documento NCL, incorporando todos los resultados de la edición. Por otro lado, los comandos de edición procedentes de los objetos de media NCLua alteran solamente la presentación del documento NCL. El documento original es preservado durante todo el proceso de edición.

Así como en las otras clases de evento, un comando de edición es representado por una tabla Lua. Todo evento debe obligatoriamente cargar el campo *command:* un string con el nombre del comando de edición. Los otros campos dependen del tipo de comando, conforme Tabla 56. La única diferencia es con relación al campo que define los pares de referencia {uri, id} denominado *data* en la clase edit. El valor del campo acepta, no sólo los pares de referencia {uri,id} mencionados en la Tabla 56, sino también *strings* XML con el contenido a ser adicionado.

EJEMPLO

```
evt = {
   command = 'addNode',
   composited = 'someId',
   data = '<media>…',
}
```

Los campos *baseId* y *documentId* (cuando son aplicables) son opcionales y asumen por *default* el identificador de la base y del documento en el que NCLua está ejecutando.

El evento describiendo el comando de edición también puede recibir una referencia de tiempo como parámetro opcional *time*. Este parámetro opcional se puede usar para especificar el momento exacto en que el comando de edición debe ser ejecutado. Si este parámetro no se suministra en la llamada de función, el comando de edición debe ser ejecutado inmediatamente. Cuando se suministra, el parámetro puede tener dos tipos de valores distintos, con diferentes significados. Si es un valor numérico, define la cantidad de tiempo, en segundos, que la ejecución del comando debe ser postergada. Sin embargo, ese parámetro también puede especificar el momento exacto para la ejecución del comando, en valores absolutos. Para ello, el parámetro debe obligatoriamente ser una tabla con los siguientes campos: *year* (cuatro dígitos), *month* (1 a 12), *day* (1 a 31), *hour* (0 a 23), *minute* (0 a 59), *second* (0 a 61) e *isdst* (señalizador del horario de verano, un booleano)

Classe tcp:

El uso del canal de interactividad se realiza por medio de esta clase de eventos.

Para el envío o recibimiento de datos tcp, se debe establecer obligatoriamente una conexión inicialmente, postando un evento en laforma:

Evt = {class='tcp', type='connect', host=addr, port=number, [timeout=number]}

El resultado de la conexión es retornado en un tratador d eventos pre-registrado para la clase. El evento retornado tiene la siguiente forma:

Evt = { class='tcp', type='connect', host=addr, port=number, connection=identifier, error=<err_msg>}

Los campos *error* y *connection* son mutuamente exclusivos. Cuando hay un error de comunicación, debe ser retornado un mensaje en el campo *error.* Cuando la comunicación es satisfactoria, el identificador de la conexión es retornado en el campo *connection*.

Una aplicación NCLua envía datos por un canal de interactividad, enviando eventos en la forma:

evt = { class='tcp', type='data', connection=identifier, value=string, [timeout=number] }

De forma análoga, una aplicación NCLua recibe datos transportados por un canal de interactividad haciendo uso de eventos en la forma:

evt = { class='tcp', type='data', connection=identifier, value=string, error=msg }

Los campos *error* y *value* son mutuamente exclusivos. Cuando hay un error de comunicación, es enviado un mensaje por el campo *error*. Cuando la comunicación tiene éxito, el mensaje transportado es pasado por el campo *value*.

Para cerrar una conexión, se debe postar obligatoriamente el siguiente evento:

evt = { class='tcp', type='disconnect', connection=identifier }

NOTA 1 Se recomienda que cuestiones tales como autenticación, sean tratadas en una implementación específica del *middleware.*

NOTA 2 En la clase tcp, el filtro dependiente de la clase sólo puede ser *connection.*

Classe sms:

Una aplicación NCLua envía datos, utilizando SMS, por medio de publicación de eventos de la siguiente manera:

evt = { class='sms', type='send', to='string', value=string [, id:string]}

El campo `to` contiene el número de destino (número del teléfono o número de la cuenta). Si no fueran especificados, la región y los prefijos de código de país recibirán la región y los códigos del país desde donde el mensaje está siendo enviado.

El campo `value` posee el contenido del mensaje.

El campo `id` puede ser usado para identificar el SMS que será enviado. La aplicación es responsable por definir el valor de `id` y garantizar su unicidad.

Un evento de confirmación debe obligatoriamente ser enviado de vuelta a la aplicación NCLua, acompañado el formato:

evt = { class='sms', type='send', to:string, sent:Boolean [,error:string] [, id:string] }

En el mensaje de confirmación, el campo `to` debe obligatoriamente tener el mismo valor que en el evento original publicado por la aplicación NCLua. El campo `sent` notifica si SMS fue despachado por el dispositivo (verdadero) o no. El campo `error` es opcional. Si el valor del campo `sent` fuera falso, podría contener un mensaje de error detallado. Si el SMS original fuera publicado con el campo `id` definido, el evento de confirmación debe llegar obligatoriamente con el mismo valor de id. De este modo, la aplicación NCLua puede hacer una asociación entre ambos eventos y tratar con varios mensajes SMS enviados simultáneamente.

De forma similar, una aplicación NCLua se registra para recibir mensajes SMS, publicando eventos en la forma:

evt = { class='sms', type ='register', port:number }

El campo `port` debe recibir obligatoriamente un número válido de puerta TCP. Para el cumplimiento de las normas GSM (3GPP TS 23.040 V6.8.1, de 2006-10), ese valor debe obligatoriamente estar en el intervalo [16000,16999].

Eventos recibidos por tratador poseen el siguiente formato:

evt = { class='sms', type='receive', from:string, port:number, value:string }

El campo `port` es definido como en el tipo = 'register'. El campo `from` contiene el número del mensaje de origen (número del teléfono o número de la cuenta). El prefijo de la región y el código del país pueden ser omitidos si fueran iguales a los del receptor. El campo `value` posee el contenido del mensaje.

En cualquier momento, la aplicación puede solicitar para dejar de recibir mensajes SMS en determinada puerta, registrando el evento:

evt = { class='sms', type='unregister', port:number }

El campo puerta es definido como en el tipo = 'register'.

En el momento en que la presentación de la media NCLua se detenga, la implementación de *middleware* debe obligatoriamente asegurar que todas las puertas sean desregistradas.

NOTA 1 Se recomienda que una implementación específica de *middleware* trate cuestiones como autenticación, etc.

NOTA 2 En la clase sms, el filtro dependiente de la clase sólo puede ser *from* y *port*, en ese orden.

NOTA 3 La finalidad del número de puerta es evitar conflictos entre los mensajes comunes SMS recibidos por un usuario y los mensajes SMS que deben obligatoriamente ser tratadas solamente por la aplicación.

Una implementación de Ginga-NCL debe obligatoriamente retornar *false* inmediatamente a cada llamada para event.post () que usa una clase de evento no soportada. se recomienda que la aplicación NCLua capture esa condición de error para verificar si el envío de SMS falló.

Clase si:

La clase de eventos 'si' permite el acceso a un conjunto de informaciones multiplexadas en un flujo de transporte y transmitidas periódicamente por difusión.

El proceso de adquisición de las informaciones debe ser obligatoriamente realizado en dos pasos:

1) una requisición realizada por una llamada al `event.post()`, que no bloquea la ejecución del script;

2) un evento, posteriormente repasado a los tratadores de eventos registrados del script NCLua, cuyo campo data contiene un conjunto de subcampos que depende del tipo de información exigida, y que es representado por una tabla lua.

NOTA En la clase si, el filtro dependiente de la clase sólo puede ser *type*.

Son definidos cuatro tipos de eventos por los siguientes tipos de tablas:

type = 'services'

La tabla del tipo 'services' consiste en un conjunto de vectores. Cada vector posee informaciones relativas a un servicio multiplexado del flujo de transporte sintonizado.

Cada requisición para una tabla del tipo de evento 'services' debe ser obligatoriamente realizada a través de la siguiente llamada:

event.post('out', { class='si', type='services'[, index=N][, fields={field_1, field_2,…, field_j}]}),

donde:

a) el campo index, si es especificado, indica el índice delo servicio; en caso de que no sea especificado, todos los servicios deben ser obligatoriamente retornados;

b) el campo fields puede tener como valor cualquier subconjunto de los subcampos definidos para la tabla retornada en el campo data del evento de respuesta (así, field_i representa uno de los subcampos de la tabla data). En caso de que el campo fields no sea especificado, todos los subcampos de la tabla retornada en el campo data deben ser obligatoriamente rellenados.

El evento de respuesta es generado después de que todas las informaciones exigidas hayan sido procesadas por el middleware (informaciones no transmitidas por difusión dentro de un intervalo máximo de tiempo especificado en la ABNT NBR 15603-2:2007, Tabla 6, del estándar son retornadas como nulas). La tabla del campo data es retornada en el evento, como sigue:

```
evt = {
   class = 'si',
   type = 'services',
   data = {
      [i] = {  -- cada servicio está en una posición
         id                          = <number>,
         isAvailable                 = <boolean>,
         isPartialReception          = <boolean>,
         parentalControlRating       = <number>,
         runningStatus               = <number>,
         serviceType                         = <number>,
         providerName                = <string>,
         serviceName                 = <string>,
         stream = {
            [j] = {
               pid = <number>,
               componentTag = <number>,
               type = <number>,
               regionSpecType = <number>,
               regionSpec     = <string>,
            }
      }
   }
}
```

Para la obtención de las respuestas a envíos de eventos del tipo ‘services’, se recomienda que los subcampos de la tabla data se calculen con base en tablas SI y descriptores asociados al servicio [i].

Se recomienda que los valores de los subcampos id y runningStatus de la tabla data se computen conforme los valores de los campos service_id y running_status, respectivamente, de la tabla SDT (ver ABNT NBR 15603-2:2007, Tabla 13, que describe el servicio [i].

Se recomienda que sea establecida la misma relación entre los subcampos providerName y serviceName de la tabla data y los campos service_name y service_provider_name, respectivamente, del descriptor service_descriptor (conforme la ABNT NBR 15603-2:2007).

Se recomienda que el subcampo parentalControlRating de la tabla data se calcule con el valor del campo rating del descriptor parentalControlRating, en el cual el campo country_code presente el mismo país referente al valor de la variable de ambiente (nudo Settings) user.location.

Se recomienda que el subcampo isAvailable de la tabla data se calcule basado en el valor del campo country_code (con el grupo de países disponibles) del descriptor country_availability_descriptor (ver ABNT NBR 15603-2:2007, Subsección 8.3.6) relativo al servicio [i]. El valor “true” es atribuido solamente si el campo country_code contiene el mismo país referente al valor de la variable de ambiente (nudo Settings) user.location.

Se recomienda que el subcampo isPartualReception de la tabla data se calcule con base en el valor del campo service_id del descriptor partial_reception_descriptor (ver ABNT NBR 15603-2:2007, Subsección 8.3.32).

Se recomienda que la semántica del subcampo serviceType de la tabla data sea definida por la ABNT NBR 15603-2:2007, Tabla H.2.

Se recomienda que la semántica del subcampo runningStatus sea definida de acuerdo con la ABNT NBR 15603-2:2007, Tabla 14.

Se recomienda que el subcampo pid de la tabla stream tenga el valor del campo pid del encabezado del paquete del flujo elemental [i] (conforme ISO/IEC 13818-1).

Se recomienda que el valor del subcampo componentTag de la tabla stream se obtenga a través del campo component_tag del descriptor stream_identifier_descriptor (ver ABNT NBR 15603-2:2007, Subsección 8.3.16) relacionado al flujo elemental [i].

Se recomienda que el subcampo type de la tabla stream tenga la semántica definida en la Tabla 2-34 de la ISO/IEC 13818-1: 2008, relacionada al flujo elemental [i].

Se recomienda que el subcampo regionSpecType de la tabla stream defina el método de codificación del campo regionSpec de la misma tabla, conforme la semántica definida en la ABNT NBR 15603-2:2007, Tabla 53.

Se recomienda que el campo regionSpec de la tabla stream defina la región para la cual es designado el flujo elemental [i].

Se recomienda también que los campos regionSpec y regionSpecType se obtengan a través del descriptor target_region_descriptor especificado en la ABNT NBR 15603-2:2007.

type = ‘mosaic’

La tabla del tipo ‘mosaic’ consiste en un subconjunto de informaciones para el mosaico que se suministra como una matriz. La tabla es, sin embargo, opcional.

Cuando se suministra la tabla del tipo ‘mosaic’, la requisición de la tabla debe ser obligatoriamente realizada a través de la siguiente llamada:

event.post('out', { class='si', type='mosaic'[, fields={field_1, field_2,…, field_j}]}),

donde el campo fields del evento de requisición puede tener como valor cualquier subconjunto de los subcampos definidos para la tabla retornada en el campo data del evento de respuesta (así, field_i representa uno de los subcampos de la tabla data, como se especifica a continuación). En el caso de que el campo fields no se especifique, todos los subcampos de la tabla retornada en el campo data deben ser obligatoriamente rellenados.

El evento de respuesta es generado después de que todas las informaciones exigidas sean procesadas por el middleware (informaciones no transmitidas por difusión dentro de un intervalo máximo de tiempo especificado en la ABNT NBR 15603-2:2007, Tabla 6 del estándar, son retornadas como nulas). La tabla del campo data es retornada en el evento, como sigue:

```
evt = {
   class = 'si',
```

```
    type = 'mosaic',
    data = {
       [i] = {
          [j] = {
             logicalId       = <number>,
             presentationInfo = <number>,
             id              = <number>,
             linkageInfo     = <number>,
             bouquetId       = <number>,
             networkId       = <number>,
             tsId            = <number>,
             serviceId       = <number>,
             eventId         = <number>,
          }
       }
    }
}
```

NOTA Para la obtención de respuestas a envíos de eventos del tipo ‘mosaic’, se recomienda que los subcampos de la tabla data sean calculados con base en tablas SI y descriptores asociados al mosaico. Se recomienda que los valores máximos de [i] y [j], así como los valores de los subcampos logicalId, presentationInfo, id, linkageInfo, bouquetId, networkId, tsId, serviceId y eventId de la tabla data se obtengan a través de los campos number_of_horizontal_elementary_cells, number_of_vertical_elementary_cells, logical_cell_id, logical_cell_presentation_info, id, cell_linkage_info, bouquet_id, original_network_id, transport_stream_id, service_id y event_id del descriptor mosaic_descriptor (especificado en la ABNT NBR 15603-2:2007, Subsección 8.3.9), respectivamente.

type = ‘epg’

La tabla del tipo ‘epg’ consiste en un conjunto de vectores. Cada vector posee informaciones relativas a un evento del contenido transmitido.

Una requisición de la tabla del tipo ‘epg’ debe ser obligatoriamente realizada a través de una de las posibles llamadas a seguir:

1) event.post('out', { class='si', type='epg', stage=’current’[, fields={field_1, field_2,…, field_j}]})

donde el campo fields del evento de requisición puede tener como valor cualquier subconjunto de los subcampos definidos para la tabla retornada en el campo data del evento de respuesta (así, field_i representa uno de los subcampos de la tabla data, como se especifica a continuación). En caso que el campo fields no se especifique, todos los subcampos de la tabla retornada en el campo data deben ser obligatoriamente rellenados.

Descripción: obtiene las informaciones del evento corriente de la programación.

2) event.post('out', {class='si', type='epg', stage='next'[, eventId=<number>][, fields={field_1, field_2,…, field_j}]})

Donde:

a) el campo eventId, cuando es especificado, identifica el evento inmediatamente anterior al evento del que se quieren las informaciones. Cuando no se especifica indica que las informaciones deseadas son las del evento siguiente al evento corriente de la programación;

b) el campo fields del evento de requisición puede tener como valor cualquier subconjunto de los subcampos definidos para la tabla retornada en el campo data del evento de respuesta (así, field_i representa uno de los subcampos de la tabla data, como se especifica a continuación). En caso que el campo fields no se especifique, todos los subcampos de la tabla retornada en el campo data deben ser obligatoriamente rellenados.

Descripción: obtiene las informaciones del evento siguiente al evento identificado en *eventId* o, en el caso de que eventId no se especifique, del evento siguiente al evento corriente de la programación.

3) event.post('out', {class='si', type='epg', stage='schedule', startTime=<date>, endTime=<date>[, fields={field_1, field_2,…, field_j}]})

donde el campo fields del evento de requisición puede tener como valor cualquier subconjunto de los subcampos definidos para la tabla retornada en el campo data del evento de respuesta (así, field_i representa uno de los subcampos de la tabla data, como se especifica a continuación). En caso que el campo fields no se especifique, todos los subcampos de la tabla retornada en el campo data deben ser obligatoriamente rellenados.

Descripción: obtiene las informaciones de los eventos comprendidos en la banda de tiempo pasada en los campos startTime y endTime que tienen como valor tablas en formato de <date>.

El evento de respuesta es generado después que todas las informaciones exigidas sean procesadas por el middleware (informaciones no transmitidas por difusión dentro de un intervalo máximo de tiempo especificado en la ABNT NBR 15603-2:2007, Tabla 6, del estándar son retornadas como nulas). La tabla del campo data es retornada en el evento, como sigue:

```
evt = {
   class = 'si',
   type = 'epg',
   data = {
      [i] – {
         startTime               = <date>,
         endTime                 = <date>,
         runningStatus           = <number>,
         name                    = <string>,
         originalNetworkId       = <number>,
         shortDescription        = <string>,
         extendedDescription     = <string>,
         copyrightId             = <number>,
         copyrightInfo           = <string>,
         parentalRating          = <number>,
         parentalRatingDescription = <string>,
         audioLanguageCode       = <string>,
         audioLanguageCode2      = <string>,
         dataContentLanguageCode = <string>,
         dataContentText         = <string>,
         hasInteractivity        = <boolean>,
         logoURI                 = <string>,
         contentDescription      = {
            [1] = <content_nibble_1>,
            [2] = <content_nibble_2>,
            [3] = <user_nibble_1>,
            [4] = <user_nibble_2> }
```

```
},
linkage = {
   tsId      = <number>,
   networkId = <number>,
   serviceId = <number>,
   type      = <number>, (table 30, norma 3 vol 2)
   data      = <string>,
},
hyperlink = {
   type           = <number>,
   destinationType = <number>,
   tsId           = <number>,
   networkId      = <number>,
   eventId        = <number>,
   componentTag   = <number>,
   moduleId       = <number>,
   serviceId      = <number>,
   contentId      = <number>,
   url            = <string>,
},
series = {
   id              = <number>,
   repeatLabel     = <number>,
   programPattern  = <number>,
   episodeNumber   = <number>,
   lastEpisodeNumber = <number>,
   name = <string>,
},
eventGroup = {
   type = <number>,
   [j] = {
      id       = <number>,
      tsId     = <number>,
      networkId = <number>,
      serviceId = <number>,
   }
},
componentGroup = {
   type = <number>,
   [j] = {
      id          = <number>,
      totalBitRate = <number>,
      description  = <string>,
      caUnit = {
         id       = <number>,   -- (table 80, norma 3 vol 2)
         component = {
```

```
                    [k] = tag (<number>)
                }
            },
        }
    }
  }
}
```

Para la obtención de las respuestas a envíos de eventos del tipo ‘epg’, se recomienda que los campos de la tabla data sean calculados con base en tablas SI y descriptores asociados al evento [i].

Se recomienda que los valores de los campos startTime, endTime, runningStatus y originalNetworkId de la tabla data se obtengan a través de los campos start_time, (duration + start_time), running_status y original_network_id de la tabla SI event_information_section (conforme especificado en la ABNT NBR 15603-2:2007, Tabla 15), respectivamente.

Se recomienda que los valores de los campos name y shortDescription se obtengan a través de los campos event_name_char y text_char del descriptor short_event_descriptor (conforme especificado en la ABNT NBR 15603-2:2007, Subsección 8.3.15), respectivamente.

Se recomienda que el valor del campo extendedDescription sea obtenido a través del campo text_char del descriptor extended_event_descriptor (conforme especificado en la ABNT NBR 15603-2:2007, Subsección 8.3.7).

Se recomienda que los valores de los campos copyrightId y copyrightInfo se obtengan a través de los campos copyright_identifier y additional_copyright_info del descriptor copyright_descriptor (conforme la la ISO/IEC 13818-1:2008, Tabla 2-63), respectivamente.

El campo parentalRating de la tabla data tiene la semántica definida en la ABNT NBR 15603-2:2007, Tabla 32, y se recomienda que su valor sea calculado con base en el campo country_code del descriptor parental_rating_descriptor y en la variable de ambiente (Settings node) user.location.

El campo parentalRatingDescription de la tabla data tiene la semántica definida en la ABNT NBR 15603-2:2007, Tabla 33, y se recomienda que su valor se calcule con base en el campo country_code del descriptor parental_rating_descriptor y en la variable de ambiente (Settings node) user.location.

Se recomienda que los valores de los campos audioLanguageCode y audioLanguageCode2 se obtengan a través de los campos ISO_639_language_code e ISO_639_language_code2 del descriptor audio_component_descriptor (conforme ABNT NBR 15603-2:2007, Tabla 48), respectivamente.

Se recomienda que los valores de los campos dataContentLanguageCode y dataContextText se obtengan a través de los campos ISO_639_language_code y text_char del descriptor data_content_descriptor (conforme ABNT NBR 15603-2:2007, Tabla 54), respectivamente.

El campo hasInteractivity de la tabla data debe tener obligatoriamente el valor true cuando el evento [i] tenga una aplicación interactiva disponible.

Se recomienda que el valor del campo logoURI de la tabla data posea la localización del logotipo, transmitido por una tabla CDT (conforme ABNT NBR 15603-2:2007, Subsección 8.3.44).

Se recomienda que los valores de los campos de la tabla contentDescription sean obtenidos a través de los campos de igual nombre del descriptor content_descriptor (conforme ABNT NBR 15603-2:2007, Subsección 8.3.5).

Se recomienda que los valores de los campos tsId, networkId, serviceId, type y data de la tabla linkage se obtengan a través de los campos transport_stream_id, original_network_id, original_service_id, description_type y

user_defined del descriptor linkage_descriptor (conforme ABNT NBR 15603-2:2007, Subsección 8.3.40), respectivamente.

Se recomienda que los valores de los campos type, destinationType, tsId, networkId, eventId, componentTag, moduleId, contentId y url de la tabla hyperlink se obtengan a través de los campos hyper_linkage_type, link_destination_type, transport_stream_id, original_network_id, event_id, component_tag, moduleId, content_id y url_char del descriptor hyperlink_descriptor (conforme ABNT NBR 15603-2:2007, Subsección 8.3.29), respectivamente.

Se recomienda que los valores de los campos id, repeatLabel, programPattern, episodeNumber, lastEpisodeNumber y name de la tabla series se obtengan a través de los campos series_id, repeat_label, program_pattern, episode_number, last_episode_number y series_name_char del descriptor series_descriptor (conforme ABNT NBR 15603-2:2007, Subsección 8.3.33), respectivamente.

Se recomienda que los valores de los campos type, id, tsId, networkId y serviceId de la tabla eventGroup se obtengan a través de los campos group_type, event_id, transport_stream_id, original_network_id y service_id del descriptor event_group_descriptor (conforme ABNT NBR 15603-2:2007, Subsección 8.3.34), respectivamente.

Se recomienda que los valores de los campos type, id, totalBitRate, description, caUnit.id, caUnit.component[k].tag, tsId, networkId y serviceId de la tabla componentGroup se obtengan a través de los campos component_group_type, component_group_id, total_bit_rate, text_char, CA_unit_id y component_tag del descriptor component_group_descriptor (conforme ABNT NBR 15603-2:2007, Subsección 8.3.37), respectivamente.

type=’time’

La tabla del tipo ‘time’ consiste en informaciones sobre la fecha y hora corriente basadas en la UTC, pero en el horario oficial del país en que se encuentra el receptor.

La requisición de la tabla debe realizarse obligatoriamente a través de la siguiente llamada:

event.post('out', { class='si', type=’time’}),

El evento de respuesta es generado después que la información sobre fecha y hora corriente exigida sea procesada por el middleware (las informaciones no transmitidas por difusión dentro de un intervalo máximo de tiempo especificado en la ABNT NBR 15603-2:2007, Tabla 6 del estándar, son retornadas como nulas). La tabla del campo data es retornada en el evento, como sigue:

```
evt = {
   class = 'si',
   type = 'time',
   data = {
            year          = <number>,
            month         = <number>,
            day           = <number>,
            hours          = <number>,
            minutes       = <number>,
            seconds       = <number>
   }
```

NOTA Para la obtención de las respuestas a envíos de eventos del tipo ‘time’, se recomienda que los campos de la tabla data se calculen con base en la tabla TOT y en el descriptor local_time_offset_descriptor (conforme ABNT NBR 15603-2:2007, Subsección 7.2.9).

Classe user:

Utilizando Clase user, las aplicaciones pueden extender sus funcionalidades, creando sus propios eventos. En esa clase, ningún campo está definido (aparte, del campo class).

NOTA En la clase user, el filtro dependiente de la clase puede ser *type*, si se establece el filtro.

10.3.4 Módulo *settings*

Exporta la tabla *settings* con variables definidas por el autor del documento NCL y variables de ambiente reservadas, contenidas en el nudo application/x-ginga-settings.

No se permite atribuir valores a los campos representando variables en los nudos *settings.* Un error debe ser generado en el caso que ocurra un intento de atribución. Las propiedades de un nudo *settings* solo pueden ser modificadas por medio de eslabones NCL.

La tabla *settings* particiona sus grupos en varias subtablas, correspondiendo a cada grupo del nudo application/x-ginga-settings. Por ejemplo, en un objeto NCLua, la variable del nudo *settings* “system.CPU” es referida como settings.system.CPU.

Ejemplos de uso:

lang = settings.system.language

age = settings.user.age

val = settings.default.selBorderColor

settings.service.myVar = 10

settings.user.age = 18 --> ERRO!

En el caso de referencia a una propiedad del nodo application/x-ginga-settings indexada, la indexación (i) debe ser sustituida en el objeto NCLua por [i].

10.3.5 Módulo *persistent*

Aplicaciones NCLua pueden grabar datos en un área restringida del *middleware* y recuperarlos entre ejecuciones. El exhibidor Lua permite a una aplicación NCLua persistir un valor para ser posteriormente usado por la misma o por otro objeto imperativo. Para ello, el exhibidor define un área reservada, inaccesible a objetos NCL que no sean imperativo. Este área se divide entre los grupos *“service”, “channel”* y *“shared”,* con la misma semántica de los grupos homónimos del nudo NCL *settings.* No existe ninguna variable predefinida o reservada en esos grupos, y objetos imperativo pueden atribuir valores a esas variables directamente. Se recomienda que otros lenguajes imperativa, Java en particular para los objetos NCLets (<media> elements of type application/x-ginga-NCLet), ofrezcan una API dando acceso al mismo área.

En éste modulo *persistent,* Lua ofrece una API para exportar la tabla *persistent* con las variables definidas en el área reservada.

El uso de la tabla *persistent* es semejante al uso de la tabla *settings* excepto por el hecho de que, en este caso, el código imperativo puede alterar los valores de los campos.

Ejemplos de uso:

persistent.service.total = 10

color = persistent.shared.color

10.4 Lua-API para Ginga-J

10.4.1 Mapeo

Dependiendo de la configuración del *middleware,* es posible tener acceso en Lua a la misma API suministrada por el Ginga-J, a fin de tener acceso a algunos recursos del decodificador y facilidades del Ginga. La API para Ginga-J suministrada en Lua es opcional, pero cuando es ofrecida debe seguir obligatoriamente la misma especificación definida para el Ginga-J.

10.4.2 Paquetes

Las jerarquías de los paquetes Java que componen la API del Ginga-J se mapean para jerarquías equivalentes de los paquetes Lua que tienen un paquete raíz en común, llamado ginga. Más específicamente, un paquete "x" en la API del Ginga-J se mapea para un paquete Lua ginga.x equivalente. En ese contexto, un paquete Lua *equivalente* significa un paquete que contenga clases y subpaquetes equivalentes a los definidos en el paquete Java.

El conjunto de paquetes Ginga-J que estarán disponibles en el ambiente de ejecución de un *script* Lua puede ser restringido por políticas de seguridad. Si un paquete “x” de la API del Ginga-J está disponible en el ambiente Lua, ginga.x mantendrá una referencia para una tabla Lua con todas las definiciones relacionadas a "x" (clases y subpaquetes). En caso contrario, ginga.x es una referencia nil. Algunos ejemplos de mapeos de nombre de los paquetes del Ginga-J para paquetes Lua se presentan en la Tabla 58.

Tabla 58 – Ejemplos de mapeos de nombre entre los paquetes Ginga-J y paquetes Lua

Paquete Ginga-J	Paquete 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

Los tipos de datos básicos del Java, que se utilizan en la API Ginga-J, se mapean para los tipos de datos básicos de Lua. Esos mapeos se presentan en la Tabla 59. Además de los tipos primitivos de Java, la tabla también especifica el mapeo de *strings* y matrices.

Tabla 59 — Mapeo de tipos de datos 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 Clases

Toda clase Java da API Ginga-J es representada en Lua como una tabla, definida en su respectivo paquete. Por ejemplo, la clase

org.sbtvd.net.tuning. ChannelManager

se representa en Lua como una entrada ChannelManager en el paquete ginga.org.sbtvd.net.tuning, es decir, esa clase es accedida a través de

ginga.org.sbtvd.net.tuning.ChannelManager.

Todos los miembros estáticos de una clase Java se mapean para campos de la tabla Lua equivalente. Cada clase representada en Lua también tiene una operación *newInstance,* que desempeña el papel de un *constructor.*

10.4.5 Objetos

Siempre que el método *newInstance* suministrado por una clase representada en Lua se llama, retorna una nueva instancia *(objeto)* de esa clase. El objeto retornado es una tabla Lua que tiene todos los miembros de instancia especificados por su clase (campos y métodos públicos).

10.4.6 Objetos de *callback* (observadores)

Muchos métodos definidos en la API Ginga-J esperan recibir un objeto observador *(listener)* como parámetro. Esos objetos observadores pueden ser implementados en Lua como tablas que tienen todos los métodos especificados en la interfaz del observador.

10.4.7 Excepciones

Las excepciones del Java también se mapean para tablas Lua, siguiendo las mismas reglas para mapear objetos del Java para Lua. Para generar una excepción, se recomienda que un objeto observador implementado en Lua use la función error suministrada por Lua (ver Anexo B). Para capturar una excepción generada por un método de la API, se recomienda que el script Lua use la función pcall (ver Anexo B).

11 Puente

11.1 Revisión

El puente de doble mano entre el Ginga-NCL y el Ginga-J se realiza:

— en un sentido, a través de las relaciones del NCL, definidos en los elementos <link> que se refieren a los elementos <media> que representan los códigos Xlet (tipo application/x-ginga-NCLet) soportados por el Ginga-J; y a través de los *scripts* Lua (elementos <media> del tipo application/x-ginga-NCLua) que hacen referencia a los métodos del Ginga-J;

— en el camino inverso, a través de las funciones del Ginga-J que pueden monitorear cualquier evento NCL y también pueden comandar alteraciones en elementos y propiedades NCL, a través de relaciones definidas en elementos <link> o a través de comandos de edición del NCL.

11.2 Puente a través de los elementos NCL <link> y <media>

El Ginga-NCL puede actuar sobre el Ginga-J a través de elementos <link> y a través de elementos <media> del tipo application/x-ginga-NCLet.

De modo análogo al contenido de media convencional, el NCL permite que el código Xlet sea sincronizado con otros objetos NCL (imperativo o no). Los autores del NCL pueden definir eslabones NCL para iniciar, parar, pausar, retomar o abortar la ejecución de un código imperativo Xlet (representado por un elemento <media> del tipo application/x-ginga-NCLet) como hacen para contenidos de presentación usual (ver 8.5). Un *player* (exhibidor) NCLet (basado en la máquina Java) debe obligatoriamente hacer la interfaz del ambiente de ejecución imperativo con el formateador NCL (ver 8.5).

Un elemento <media> conteniendo un código Java puede definir anclas (a través de elementos <area> y atributos (a través de elementos <property>). El *player* debe obligatoriamente controlar la máquina de estado de los eventos asociados con esos elementos de interfaz.

El código Xlet puede ser asociado a elementos <area>. Si los eslabones externos inician, paran, pausan o retoman la presentación del ancla, las *callBacks* en el código Xlet deben ser obligatoriamente disparadas. Por otro lado, el código Xlet puede comandar el inicio, parada, pausa, reinicio o aborto de esas anclas a través de una API ofrecida por el lenguaje imperativa. Las transiciones causadas por esos comandos se pueden usar como condiciones de los eslabones NCL para disparar acciones en otros objetos NCL del mismo documento. Así, una sincronización de dos vías puede ser establecida entre el código Xlet y el resto del documento NCL.

Un elemento <property> definido como hijo del elemento <media> del tipo application/x-ginga-NCLet puede ser mapeado para un método del código Xlet o para un atributo del código Xlet. Cuando es mapeado para un método del código, una acción “set” del eslabón aplicada al atributo debe obligatoriamente causar la ejecución del método, con los valores definidos por la acción del eslabón siendo interpretados como parámetros de entrada del método. El atributo *name* del elemento <property> se debe usar obligatoriamente para identificar el método del código procedural. Cuando el elemento <property> es mapeado para un atributo del código Xlet, la acción “set” debe obligatoriamente atribuir un valor al atributo.

El elemento <property> puede también ser asociado a un *assessment role* de un eslabón NCL. En ese caso, el formateador NCL debe obligatoriamente consultar el valor del atributo, a fin de evaluar la expresión del eslabón. Si el elemento <property> es mapeado para un atributo del código, el valor del atributo debe ser retornado obligatoriamente por el player Xlet para el formateador NCL. Si el elemento <property> es mapeado para un método del código, el método debe ser obligatoriamente llamado y su valor debe ser retornado obligatoriamente por el player Xlet para el formateador NCL.

11.3 Puente a través de las funciones Lua y métodos del Ginga-J

Dependiendo de la configuración del *middleware,* es posible tener acceso en Lua a la misma API suministrada por el Ginga-J, a fin de tener acceso a algunos recursos del decodificador y facilidades del Ginga. La API suministrada en Lua debe seguir obligatoriamente la misma especificación presentada para el Ginga-J.

El Ginga-J también ofrece una API que permite que el código Xlet consulte cualquier valor de propiedad predefinido o dinámico del nudo de configuraciones del NCL (elemento <media> del tipo "application/x-ginga-settings”).

Además, el Ginga-J ofrece API que suministran un conjunto de métodos para dar soporte a los comandos de edición del NCL y comandos del Gestor de la Base Privada.

12 Requisitos de codificación de media y métodos de transmisión referidos en documentos NCL

12.1 Uso del canal de interactividad

Un formateador NCL debe obligatoriamente ignorar con éxito cualquier método de codificación o transmisión que no sea soportado por el navegador. Con el objeto de adquirir contenido de datos que sea referido por los elementos <media> a través de un protocolo de canal interactivo específico, los mecanismos especificados para el canal de interactividad del SBTVD se deben utilizar obligatoriamente.

12.2 Métodos de codificación y transmisión de video – Datos de video referidos en elementos <media>

12.2.1 Transmisión de video MPEG-1

12.2.1.1 Transmisión como flujo elemental de video

Para transmitir contenido de video MPEG-1 como un flujo elemental de video, los datos del video se deben transmitir obligatoriamente como *MPEG-2 packetized elementary stream* (video PES), con el tipo de flujo especificado de conformidad con la atribución de tipos de flujos ISO/IEC 13818-1 (valor 0x01 para video ISO/IEC 11172-2).

12.2.1.2 Transmisión en secciones MPEG-2

Para transmitir datos de video MPEG-1 por medio de secciones MPEG-2 específicas (ver atribución de tipos de flujos para secciones MPEG-2 en ISO/IEC 13818-1), uno de los siguientes métodos de transmisión se debe usar obligatoriamente:

a) como un archivo de flujo multiplexado en sistemas MPEG-1 (de acuerdo con la ISO/IEC 11172-1);

b) como un archivo de flujo elemental de video MPEG-1;

c) como un archivo de flujo multiplexado en el formato TS especificado en 12.4.

12.2.2 Transmisión de video MPEG-2

12.2.2.1 Transmisión como flujo elemental de video

Para transmitir contenido de video MPEG-2 como un flujo elemental de video, los datos del video se deben transmitir obligatoriamente como *MPEG-2 packetized elementary stream* (video PES), con el tipo de flujo especificado de acuerdo con la atribución de tipos de flujos de acuerdo con ISO/IEC 13818-1 (valor 0x02 para video de acuerdo con ISO/IEC 13818-2).

12.2.2.2 Transmisión en secciones MPEG-2

Para transmitir datos de video MPEG-2 por medio de secciones MPEG-2 específicas (ver atribución de tipos de flujospara secciones MPEG-2 en ISO/IEC 13818-1), uno de los métodos de transmisión siguientes se debe usar obligatoriamente:

a) Como un archivo de flujo elemental de video MPEG-2;

b) Como un archivo de flujo multiplexado en el formato TS especificado en 12.4.

12.2.3 Transmisión de video MPEG-4 y H.264|MPEG-4 AVC

12.2.3.1 Transmisión como flujo elemental de video

Para transmitir contenido de video MPEG-4 como un flujo elemental de video, los datos del video se deben transmitir obligatoriamente como *MPEG-2 packetized elementary stream* (video PES), con el tipo de flujo especificado de acuerdo con la atribución de tipos de flujos ISO/IEC 13818-1 (valor 0x10 para video de acuerdo con H.264|MPEG-4 AVC).

12.2.3.2 Transmisión en secciones MPEG-2

Para transmitir datos de video MPEG-4 ó H.264|MPEG-4 AVC por medio de secciones MPEG-2 específicas (ver atribución de tipos de flujos para secciones MPEG-2 en ISO/IEC 13818-1), uno de los métodos de transmisión siguientes se debe usar obligatoriamente:

a) Como un archivo de flujo elemental de video MPEG-4 (o H.264|MPEG-4 AVC);

b) Como un archivo de flujo multiplexado en el formato TS especificado en 12.4.

12.3 Métodos de codificación y transmisión de audio – datos de audio referidos en elementos <media>

12.3.1 Transmisión de audio MPEG-1

12.3.1.1 Transmisión como flujo elemental de audio

Para transmitir contenido de audio MPEG-1 como un flujo elemental de audio, los datos del audio se deben transmitir obligatoriamente como *MPEG-2 packetized elementary stream* (audio PES), con el tipo de flujo especificado de acuerdo con la atribución de tipos de flujos ISO/IEC 13818-1 (valor 0x03 para audio ISO/I EC 11172-3).

12.3.1.2 Transmisión en secciones MPEG-2

Para transmitir datos de audio MPEG-1 por medio de secciones MPEG-2 específicas (ver atribución de tipos de flujos para secciones MPEG-2 en ISO/IEC 13818-1), uno de los siguientes métodos de transmisión se debe usar obligatoriamente:

a) como un archivo de flujo multiplexado en sistemas MPEG-1 (de acuerdo con la ISO/IEC 11172-1);

b) como un archivo de flujo elemental de audio MPEG-1;

c) como un archivo de flujo multiplexado en el formato TS especificado en 12.4.

12.3.2 Transmisión de audio MPEG-2

12.3.2.1 Transmisión como flujo elemental de audio

Para transmitir contenido de audio MPEG-2 AAC como un flujo elemental de audio, los datos del audio se deben transmitir obligatoriamente como *MPEG-2 packetized elementary stream* (audio PES), con el tipo de flujo especificado de acuerdo con la atribución de tipos de flujos ISO/IEC 13818-1 (valor 0x0F para audio ISO/IEC 13818-7).

Para transmitir contenido de audio MPEG-2 BC como un flujo elemental de audio, los datos del audio se deben transmitir obligatoriamente como *MPEG-2 packetized elementary stream* (PES), con el tipo de flujo especificado de acuerdo con la atribución de tipos de flujos ISO/IEC 13818-1 (valor 0x04 para audio ISO/IEC 13818-3).

12.3.2.2 Transmisión en secciones MPEG-2

Para transmitir datos de audio MPEG-2 por medio de secciones MPEG-2 específicas (ver atribución de tipos de flujos para secciones MPEG-2 en ISO/IEC 13818-1), uno de los siguientes métodos de transmisión se debe usar obligatoriamente:

a) como un archivo de flujo elemental de audio MPEG-2;

b) como un archivo de flujo multiplexado en el formato TS especificado en 12.4.

12.3.3 Transmisión de audio MPEG-4

12.3.3.1 Transmisión como flujo elemental de audio

Para transmitir contenido de audio MPEG-4 como un flujo elemental de audio, los datos del audio se deben transmitir obligatoriamente como MPEG-2 *packetized elementary stream* (audio PES), con el tipo de flujo especificado en conformidad con la atribución de tipos de flujos ISO/IEC 13818-1 (valor 0x11 para audio ISO/IEC 14496-3).

12.3.3.2 Transmisión en secciones MPEG-2

Para transmitir datos de audio MPEG-4 por medio de secciones MPEG-2 específicas (ver atribución de tipos de flujos para secciones MPEG-2 en ISO/IEC 13818-1), uno de los métodos siguientes de transmisión se debe usar obligatoriamente:

a) como un archivo de flujo elemental de audio MPEG-4;

b) como un archivo de flujo multiplexado en el formato TS especificado en 12.4.

12.3.4 Transmisión de audio AC3

12.3.4.1 Transmisión como flujo elemental de audio

Para transmitir contenido de audio AC3 como un flujo elemental de audio, los datos de audio se deben transmitir obligatoriamente como MPEG-2 *packetized elementary stream* (audio PES) con el tipo de audio especificado como 0x81.

12.3.4.2 Transmisión en secciones MPEG-2

Para transmitir datos de audio AC3 por medio de secciones MPEG-2 específicas (ver atribución de tipos de flujos para secciones MPEG-2 en ISO/IEC 13818-1), uno de los métodos siguientes de transmisión se debe usar obligatoriamente:

a) como un archivo de flujo elemental de audio AC3;

b) como un archivo de flujo multiplexado en el formato TS especificado en 12.4.

12.3.5 Transmisión de audio PCM (AIFF-C)

Se recomienda que el audio AIFF-C PCM se transmita como un archivo a través de secciones MPEG-2 específicas (ver atribución de tipos de flujos para secciones MPEG-2 en ISO/IEC 13818-1).

12.4 Formato TS para transmisión de video/audio MPEG – Especificación de la codificación de datos

12.4.1 Transmisión de video y audio multiplexados

Para transmitir datos de video MPEG-1/2/4 ó H.264|MPEG-4 AVC junto con datos de audio MPEG-1/2/4 ó AC3 en archivos multiplexados en secciones MPEG-2 específicas, cada archivo de video/audio multiplexado se codifica en un formato TS, conforme definido en la ISO/IEC 13818-1.

12.4.2 PSI requerido

Una tabla PAT se debe describir obligatoriamente. Cualquier PAT se debe describir obligatoriamente con el *program_number* cuyo valor es diferente de 0 y este valor debe representar obligatoriamente un PID de la PMT. Los valores disponibles de *program_number* serán definidos en un reglamento de estándar operativo.

Una tabla PMT se debe describir obligatoriamente. Cualquier descriptor de identificación de flujo que indique un segundo *loop* debe contener obligatoriamente un descriptor PMT. En caso contrario, se puede insertar un descriptor conforme sea necesario.

Se recomienda que los valores disponibles para *component_tag* y sus reglas de ocurrencia en descriptores ES y PMT defaults en un segundo *loop* sean equivalentes a un reglamento operativo estándar dedicado al flujo principal del tipo de media responsable por la transmisión del flujo en cuestión.

En una implementación en la cual un flujo de transporte se decodifica desde un archivo que fue transmitido con base en la especificación de la codificación de datos definida en esta sección y se entrega en una interfaz digital de alta velocidad, una tabla SIT se debe describir obligatoriamente (ver la ABNT NBR 15606-1). En otros casos, las SIT no son necesarias, salvo especificación explícita en contrario.

Cualquier tabla diferente de PAT, PMT y SIT (por ejemplo: CAT, NIT, SDT, BAT, EIT, RST, TDT, TOT, PCAT, SDTT y St – ver la ABNT NBR 15601) obligatoriamente no debe ser descrita.

Una tabla PAT debe ocurrir obligatoriamente en un flujo a una frecuencia no menor que una vez a cada 100 ms. Una tabla PMT debe ocurrir obligatoriamente en un flujo a una frecuencia no menor que una vez cada 100 ms.

Por toda la duración de un archivo de formato TS, las tablas PAT y PMT obligatoriamente no deben ser modificadas o actualizadas.

12.4.3 Transmisión en secciones MPEG-2

Para transmitir un archivo codificado con la especificación de codificación de datos de acuerdo con 12.6 en secciones MPEG-2 específicas, la transmisión debe estar de acuerdo obligatoriamente con ABNT NBR 15606-3.

12.4.4 Restricciones en la reproducción

Para, al mismo tiempo, recibir un servicio por difusión y reproducir un archivo TS recibido a través de secciones MPEG-2 específicas, son necesarios dos sistemas de procesamiento de flujo de transporte separados. Las restricciones en la integración y coordinación de un contenido/evento recibido por un servicio de difusión con un archivo TS no son descritas en esta Norma.

12.5 Esquema de codificación y transmisión de imágenes estáticas y gráficos de *bitmap* referidos por elementos <media>

12.5.1 Transmisión de MPEG-2 I-frame, MPEG-4 I-VOP y H.264|MPEG-4 AVC I-picture

12.5.1.1 Transmisión en video PES para reproducción lineal

Para transmitir una imagen estática en cuadros MPEG-2 I por medio de un componente video PES, el esquema de codificación debe obligatoriamente cumplir las convenciones definidas en la ABNT NBR 15606-1. El componente PES se debe transmitir obligatoriamente como un flujo cuyo valor de tipo es igual a 0x02.

Para transmitir una imagen estática en MPEG-4 I-VOP por medio de un componente video PES, el esquema de codificación debe obligatoriamente cumplir las convenciones definidas en la ABNT NBR 15606-1. El componente PES se debe transmitir obligatoriamente como un flujo cuyo valor de tipo es igual a 0x10.

Para transmitir una imagen estática en H.264|MPEG-4 AVC I-picture por medio de un componente video PES, el esquema de codificación debe obligatoriamente cumplir las convenciones definidas en la ABNT NBR 15606-1. El componente PES se debe transmitir obligatoriamente como un flujo cuyo valor de tipo es igual a 0x1B.

12.5.1.2 Transmisión en secciones MPEG-2 para reproducción interactiva

Para transmitir una imagen estática en cuadros MPEG-2 I por medio de secciones MPEG-2, el esquema de codificación debe obligatoriamente estar de conformidad con la ABNT NBR 15606-1. La imagen estática se debe transmitir obligatoriamente como un archivo en la sección MPEG-2.

Para transmitir una imagen estática en MPEG4-I-VOP por medio de secciones MPEG-2, el esquema de codificación debe obligatoriamente estar de conformidad con la ABNT NBR 15606-1. La imagen estática se debe transmitir obligatoriamente como un archivo en la sección MPEG-2.

Para transmitir una imagen estática en H.264|MPEG-4 AVC I-picture por medio de secciones MPEG-2, el esquema de codificación debe obligatoriamente estar conforme con las convenciones en la ABNT NBR 15606-1. La imagen estática se debe transmitir obligatoriamente como un archivo en la sección MPEG-2.

En esos casos, el valor del tipo de flujo de la sección MPEG-2 debe obligatoriamente estar de acuerdo con la ISO/IEC 13818-1.

12.5.2 Transmisión de imagen estática JPEG

Las imágenes estáticas JPEG se deben transmitir obligatoriamente a través de secciones MPEG-2 específicas (ver la atribución de tipos de flujos para secciones MPEG-2 en la ISO/IEC 13818-1).

12.5.3 Esquema de codificación y transmisión del bitmap PNG

Para los datos de bitmap PNG que son exhibidos solamente bajo el control de datos CLUT especificados por separado de esta Norma, los datos de la paleta, dentro de los datos PNG, pueden ser abreviados.

El gráfico de bitmap PNG se debe transmitir obligatoriamente a través de secciones MPEG-2 específicas (ver la atribución de tipos de flujos para secciones MPEG-2 en la ISO/IEC 13818-1).

12.5.4 Esquema de codificación y transmisión de la animación MNG

Para que los datos de bitmap PNG en el formato de animación MNG que son exhibidos solamente bajo el control de datos CLUT especificados por separado de esta Norma, los datos de la paleta dentro de los datos PNG pueden ser omitidos. El gráfico de animación de bitmap MNG se debe transmitir obligatoriamente a través de secciones MPEG-2 específicas (ver atribución de tipos de flujos para secciones MPEG-2 en la ISO/IEC 13818-1).

12.5.5 Esquema de codificación y transmisión de datos y animación de gráficos GIF

Los datos de gráficos y animaciones GIF se deben transmitir obligatoriamente a través de secciones MPEG-2 específicas (ver atribución de tipos de flujos para secciones MPEG-2 en la ISO/IEC 13818-1).

12.6 Codificación y transmisión de caracteres - archivos de texto externos referidos por elementos <media>

Un archivo de texto codificado de acuerdo con la ISO 8859-1 se debe transmitir obligatoriamente por medio de secciones MPEG-2 específicas (ver atribución de tipos de flujos para secciones MPEG-2 en la ISO/IEC 13818-1).

12.7 Transmisión de documentos XML

12.7.1 Transmisión de documentos NCL y otros documentos XML que se utilizan en los comandos de edición

Para transmitir un documento NCL u otro archivo de documento XML usado en parámetros de Comandos de Edición NCL, uno de los métodos siguientes de transmisión se debe usar obligatoriamente:

a) por medio de un protocolo de canal de interactividad;

b) por medio de secciones MPEG-2 específicas.

Si un protocolo de canal interactivo se usa para bajar un documento NCL u otro archivo de Documento XML mencionado en un parámetro del comando de edición addNode, el parámetro *uri* del comando de edición addDocument o addNode (ver Sección 9) no puede tener su esquema igual a “x-sbtvd”, y su parámetro correspondiente *id* se debe definir obligatoriamente como NULL. El parámetro *uri* debe especificar obligatoriamente la localización del documento y el esquema de protocolo usado para transmitir el documento.

Si secciones MPEG-2 específicas fueran usadas, son posibles varias alternativas, como a seguir. La alternativa elegida por el SBTVD debe obligatoriamente seguirla especificaciónABNT NBR 15606-3.

12.7.2 Transmisión en Secciones MPEG-2

12.7.2.1 1Transporte de comandosde edición usando descriptores de evento de flujo y carrusel de objetos DSM-CC

En los ambientes de televisión digital es usual la adopción de protocolo DSM-CC para transporte de comandos de edición en flujos elementales MPEG-2.

Comandos de edición son transportados en descriptores de evento de flujo DSM-CC, que tienen una estructura muy parecida con la de los descriptores de eventos definidos por la Figura 5, como ilustra la Figura 6.

Sintaxis	Número de bits
StreamEventDescriptor ( ) {	
descriptorTag	8
descriptorLenght	8
eventId	16
Reserved	31
eventNPT	33
privateDataLenght	8
commandTag	8
sequenceNumber	7
finalFlag	1
privateDataPayload	8 a 1928
FCS	8
}	

Figura 6 – Descriptor de evento de flujo DSM-CC para comandos de edición

El protocolo de carrusel de objetos DSM-CC permite la transmisión cíclica de objetos de eventos de flujo y sistemas de archivos. Los objetos de eventos de flujo son utilizados para mapear nombres de eventos de flujo a *ids* de eventos de flujo, y son utilizados para informar a Ginga sobre eventos de flujo DSM-CC que pueden ser recibidos. Los nombres de los eventos permiten especificar tipos de eventos, ofreciendo mayor nivel de abstracción a las aplicaciones de *middleware*. En ese caso, conviene que el Gestor de la Base Privada, así como

los objetos de ejecución imperativos NCL (ejemplo, NCLua, NCLet), sean registrados como observadores de los eventos de flujo con los cuales tratan, utilizando nombres de evento, en el caso: "*nclEditingCommand*".

Además de los objetos de eventos de flujo, el protocolo de carrusel de objetos DSM-CC también es utilizado para transportar archivos organizados en directorios. El demultiplexador DSM-CC es responsable por montar el sistema de archivo en el dispositivo receptor.

Para transmitir los archivos de Documentos NCL u otros archivos de Documento XML, usados en los parámetros de comandos de edición, por medio de un carrusel de objetos, el tipo de flujo con valor 0x0B debe ser obligatoriamente usado. En el mismo carrusel de objetos que carga la especificación XML, un objeto de eventos de flujo debe ser obligatoriamente transmitido, para mapear el nombre "*nclEditingCommand*" para el *eventId* del descriptor de evento del flujo DSM-CC, que debe obligatoriamente cargar el comando de edición NCL (ver Sección 9).

El campo *privateDataPayload* del descriptor de evento del flujo debe obligatoriamente cargar un conjunto de pares de referencia *{uri, id}*. El parámetro uri del primer par debe obligatoriamente tener el esquema "x-sbtvd" y el camino absoluto del documento XML (el camino en el servidor de datos). El parámetro *id* correspondiente en el par debe obligatoriamente hacer referencia al IOR de especificación del documento XML *(carouselId, moduleId, objectKey;* de acuerdo con la ABNT NBR 15606-3 e ISO/IEC 13818-6) en el carrusel de objetos.

Si otros sistemas de archivos necesitan ser transmitidos usando otros carruseles de objeto, con el fin de completar el comando de edición con contenido de media (como es usual en los comandos *addDocument* y *addNode)*, otros pares *{uri, id}* deben estar presentes obligatoriamente en el comando. En ese caso, el parámetro *uri* debe tener obligatoriamente el esquema "x-sbtvd" y el camino local absoluto de la raíz del sistema de archivos (el camino en el servidor de transmisión de datos) y el respectivo parámetro *ior* en el par debe obligatoriamente hacer referencia al IOR *(carouselId, moduleId, objectKey;* de acuerdo con la ABNT NBR 15606-3 e ISO/IEC 13818-6) de cualquier archivo o directorio hijo de la raíz en el carrusel de objetos (el *service gateway* del carrusel).

La Figura 7 ilustra un ejemplo de transmisión de documento NCL por medio de un carrusel de objetos. En ese ejemplo, un proveedor de contenido quiere transmitir un programa interactivo llamado "weatherConditions.ncl" almacenado en uno de sus servidores de datos (sistema local de archivos, de acuerdo con la Figura 7).

Un carrusel de objetos debe entonces ser generado (dominio de servicio = 1, de acuerdo con la Figura 7) cargando todo el contenido del programa interactivo (archivo .ncl y todos los archivos de media) y también un objeto de eventos *(moduleId* = 2 y *objectKey* = 2, de acuerdo con la Figura 7), mapeando el nombre "nclEditingCommand" para el valor de *eventId* (valor "3" de acuerdo con la Figura 7).

Un descriptor de evento de flujo también se deberá transmitir con el valor de *eventId* apropiado, en el ejemplo "3", y el valor "0x05" de *commandTag,* que indica un comando addDocument (ver Sección 9). El parámetro *uri* debe contener el esquema "x-sbtvd" y el camino absoluto del documento NCL ("C:\nclRepository\weather" de acuerdo con la Figura 7). Finalmente, el IOR del documento NCL en el carrusel de objetos se transporta en el parámetro *xmlDocument (carouselId* = 1, *moduleId* = 1, *objectKey* = 2 de acuerdo con la Figura 7).

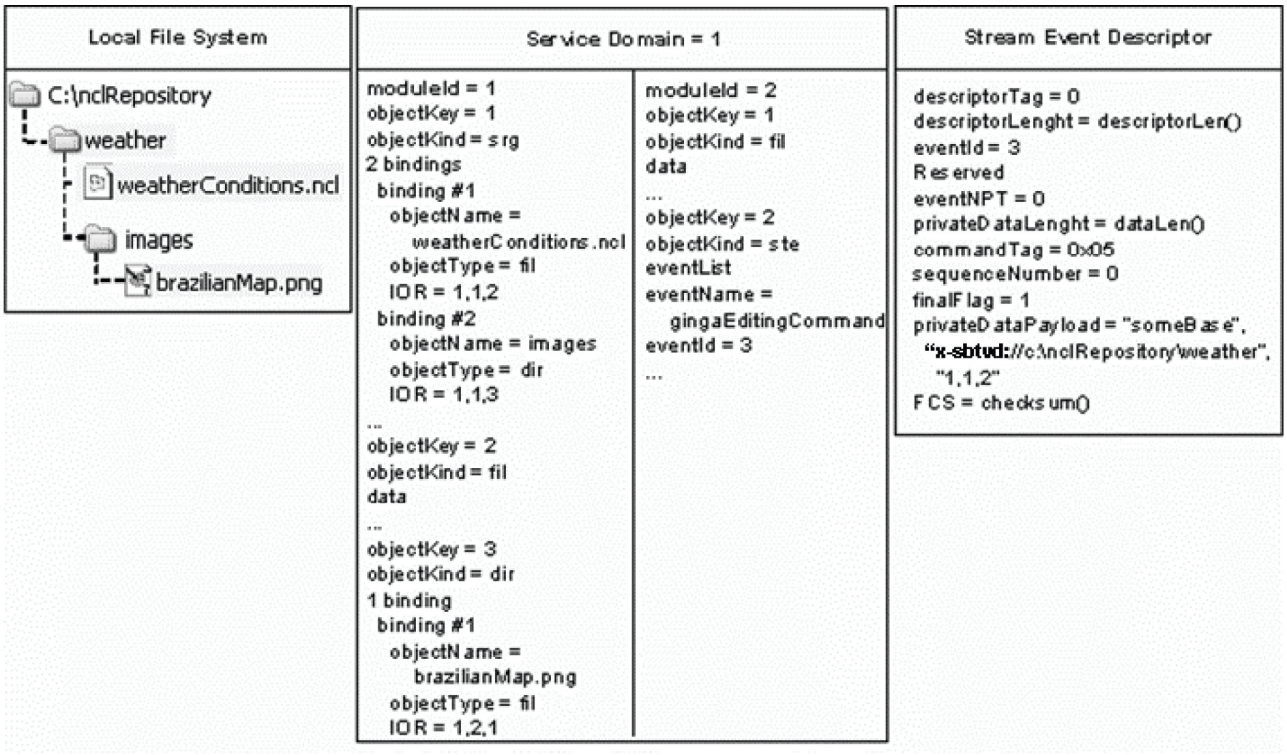

Figura 7 — Ejemplo de una transmisión de documento NCL

12.7.2.2 Transporte de comandos de edición usando estructuras específicas

12.7.2.2.1 Definiciones de las estructuras de datos

Los descriptores de evento (definidos en la sección 9) pueden enviarse en flujo elemental MPEG-2 TS usando eventos de flujo DSM-CC, como discutido en 12.7.1.1, o usando cualquier protocolo para transmisión de datos sin solicitación (“pushed data”).

Tres tipos de estructura de datos pueden ser definidos para dar soporte a la transmisión deparámetros de los comandos de edición NCL: mapa, metadatos y archivos de datos.

Para estructuras de *mapa,* el campo *mappingType* identifica el tipo del mapa. Si el valor de mappingType es igual a “0x01” (“events”), un mapa de eventos es caracterizado. En ese caso, después del campo mappingType, viene una lista de identificadores de eventos, como definido en la Tabla 60. Pueden definirse otros valores para el campo mappingType, pero no son relevantes para esta Norma.

Tabla 60 – Lista de identificadores de eventos definidos por la estructura de mapa

Sintaxis	Número de bits
mappingStructure ( ) {	
mappingType	8
for (i=1; i<N; i++){	
eventId	8
eventNameLength	8
eventName	8 a 255
}	
}	

Los mapas del tipo “event” (mapas de eventos) se usan para mapear nombres de eventos en eventos de los descriptores de evento (ver Figura 5). Los mapas de eventos son usados para indicar qué eventos deben ser obligatoriamente recibidos. Los nombres de eventos permiten especificar tipos de eventos, ofreciendomayor nivel de abstracción a las aplicaciones del *middleware.* En ese caso, conviene que el Gestor de Base Privada, así como los objetos de ejecución procedural NCL (ejemplo, NCLua, NCLet) sean registrados como observadores de los eventos de flujo con los cuales operan, utilizando nombres de eventos, en ese caso: *“nclEditingCommand”*.

Cuando un comando de edición NCL necesita ser enviado, debe crearse obligatoriamente un mapa de eventos, mapeando la string *“nclEditingCommand”* en un *eventId* de un descriptor de evento seleccionado (ver Figura 5). Uno o más descriptores de evento con el *eventId* previamente seleccionado son entonces creados y enviados. Esos descriptores de evento pueden tener sus tiempos de referencia como cero, o pueden tener su ejecución postergada para un tiempo especificado. El Gestor de Bases Privadas debe registrarse obligatoriamente como oyente de un evento *“nclEditingCommand”* para ser notificado de la llegada de ese tipo de evento.

Cada estructura de *archivos de datos* es, de hecho, un contenido de archivo que compone una aplicación NCL o un documento XML definiendo una entidad NCL: un archivo conteniendo la especificación XML o un archivo con contenido de media de la aplicación o del nudo a ser adicionado (video, audio, texto, imagen, ncl, lua etc.).

Una estructura de *metadatos* es un documento XML, como se define en el esquema a seguir. El esquema define, para cada dato entregado sin solicitación (pushed file), una asociación entre su localización en el sistema de transporte (identificación del sistema de transporte (atributo *component_tag)* y la identificación del archivo en el sistema de transporte (atributo *structureId*)) y su identificador de recurso universal (atributo *uri*).

```
<!--
XML Schema for NCL Section Metadata File

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

Public URI: http://www.ncl.org.br/NCLSectionMetadataFile.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Section Metadata File namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:NCLSectionMetadataFile="http://www.ncl.org.br/NCLSectionMetadataFile"
  targetNamespace="http:// www.ncl.org.br/NCL3.0/NCLSectionMetadataFile"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <complexType name="NCLSectionMetadataType">
    <sequence>
         <sequence>
           <element ref="NCLSectionMetadataFile:baseData" minOccurs="0"
                                 maxOccurs="unbounded"/>
         </sequence>
         <element ref="NCLSectionMetadataFile:pushedRoot" minOccurs="0"
                                    maxOccurs="1"/>
         <sequence>
           <element ref="NCLSectionMetadataFile:pushedData" minOccurs="0"
                                 maxOccurs="unbounded"/>
         </sequence>
    </sequence>
    <attribute name="name" type="string" use="optional"/>
    <attribute name="size" type="positiveInteger" use="optional"/>
  </complexType>

  <complexType name="baseDataType">
```

```
    <sequence>
         <element ref="NCLSectionMetadataFile:pushedRoot" minOccurs="0"
                                    maxOccurs="1"/>
         <sequence>
           <element ref="NCLSectionMetadataFile:pushedData"
                          minOccurs="0" maxOccurs="unbounded"/>
         </sequence>
    </sequence>
    <attribute name="uri" type="anyURI" use="required"/>
  </complexType>

  <complexType name="pushedRootType">
    <attribute name="component_tag" type="positiveInteger"
                                    use="optional"/>
    <attribute name="structureId" type="string" use="required"/>
    <attribute name="uri" type="anyURI" use="required"/>
    <attribute name="size" type="positiveInteger" use="optional"/>
  </complexType>

  <complexType name="pushedDataType">
    <attribute name="component_tag" type="positiveInteger"
                                     use="optional"/>
    <attribute name="structureId" type="string" use="required"/>
    <attribute name="uri" type="anyURI" use="required"/>
    <attribute name="size" type="positiveInteger" use="optional"/>
  </complexType

  <!-- declare global elements in this module -->
  <element name="metadata" type="NCLSectionMetadataFile:NCLSectionMetadataType"/>
  <element name="baseData" type="NCLSectionMetadataFile:baseDataType"/>
  <element name="pushedRoot" type="NCLSectionMetadataFile:pushedRootType"/>
  <element name="pushedData" type="NCLSectionMetadataFile:pushedDataType"/>

</schema>
```

Para cada archivo de documento NCL u otros archivos de documento XML, usados en los comandos de edición addDocument o addNode, debe ser definida, por lo menos, una estructura de metadatos. Solamente un archivo de aplicación NCL o un archivo de documento XML representando un nudo NCL a ser insertado puede ser definido por estructura de metadatos. Sin embargo, una aplicación NCL (y sus archivos de contenido) o un documento NCL (y sus archivos de contenido) pueden extenderse por más de una estructura de metadatos. Más aun, pueden existir estructuras de metadatos sin ninguna aplicación NCL o documento XML descritos en sus elementos <pushedRoot> y <pushedData>.

Las tres estructuras anteriormente definidas pueden ser transmitidas usando sistemas de transporte diferentes, como se explica a continuación.

12.7.2.2.2 Transporte en un tipo específico de sección MPEG-2

El uso de un tipo específico de sección MPEG-2 (identificado por un valor específico del campo table_id de una sección privada MPEG-2), a partir de ahora denominada Sección NCL, permite la transmisión de las tres estructuras de datos anteriormente definidas: mapas, metadatos y archivos de datos.

Cada Sección NCL contiene los datos de solamente un estructura. Sin embargo, una estructura puede extenderse por varias Secciones. Las estructuras de datos pueden ser transmitidas en cualquier orden y cuantas veces sea necesario. El inicio de una estructura de datos es limitado por el campo payload_unit_start_indicator de un paquete TS. Después de los cuatro bytes del encabezamiento TS, la carga (payload) del paquete TS comienza, con un campo puntero de un byte indicando el comienzo de una Sección NCL (ver ISO/IEC 13818-1). El encabezamiento de la Sección NCL es definido entonces como una sección MPEG-2 (ver ISO/IEC 13818-1). El primer byte de carga (payload) de la Sección NCL identifica el tipo de la estructura transportada (0x01 para

metadatos; 0x02 para archivos de datos y 0x03 para mapas de eventos). El segundo byte carga un identificador único de la estructura (*structureId*) en el flujo elemental de transporte.

NOTA El flujo elemental y el identicador de la estructura son aquellos que son asociados, por la estructura de metadatos, a través de los atributos *component_ta* y *structureId* de los elementos <pushedRoot> y <pushedData>, a localizadores de archivos (URL).

Después del segundo byte viene una estructura de datos seriada que puede ser mappingStructure (como ilustra la Tabla 60), o la estructura de metadatos (un documento XML), o una estructura de archivos de datos (un contenido de archivo seriado). El demultiplexador de Secciones NCL es responsable de montar la estructura de la aplicación en el dispositivo receptor.

NOTA Es importante resaltar que las Secciones NCL pueden también transportar las estructuras de datos definidas encapsuladas en otras estructuras de datos. Por ejemplo, MPE (Multi-protocol Encapsulation) puede usarse. En ese caso, Secciones NCL serían Secciones MPEG-2 de datagrama. Más aun, todas las estructuras de datos mencionadas pueden empaquetarse en otro formato de datos de protocolo, como, por ejemplo, paquetes FLUTE.

En el mismo flujo elemental que carga la especificación XML (el archivo del documento NCL o de otro documento XML usados en los comandos de edición NCL) es recomendable que se transmita un archivo mapa de eventos, para que sea mapeado el nombre *"nclEditingCommand"* en el *eventId* del descriptor de eventos que deberá obligatoriamente transportar los comandos de edición NCL, como se describe en la Sección 9. El campo *privateDataPayload* del descriptor de eventos debe obligatoriamente cargar el par de referencias {uri, id}. Los parámetros *uri* tienen siempre el valor "null". En el caso de los documentos *addDocument* y *addNode,* el parámetro *id* del primer par debe obligatoriamente identificar el flujo elemental ("component_tag") y la estructura de metadatos que éste transporta ("structureId"), que, a su vez, contiene el camino absoluto del documento NCL o de la especificación del nudo NCL (el camino en el servidor de datos) y la estructura relacionada correspondiente ("structureId") transportada en las Seccions NCL del mismo flujo elemental. Si otras estructuras de metadatos adicionales fueran necesarias para completar los comandos de edición addDocument o addNode command, otros pares {uri, id} deben obligatoriamente hacerse presentes en el comando. En ese caso, los parámetros *uri* también deben tener el valor "null" y los parámetros *id* correspondientes deben referirse obligatoriamente al component_tag y al metadato structureId correspondiente.

La Figura 8 ilustra un ejemplo de transmisión de un documento NCL a través de Secciones NCL. En ese ejemplo, un proveedor de contenido quiere transmitir un programa interactivo llamado "weatherConditions.ncl", almacenado en uno de sus servidores de datos (sistema de archivo local en la Figura 8). Un flujo elemental MPEG-2 (component_tag="0x09") debe, entonces, ser generado, cargando todo el contenido del programa interactivo (el archivo ncl y todos los archivos de contenido de media). Tambiés es generado un mapa de eventos (structureType="0x03", structureId="0x0C", en la Figura 8), mapeando el nombre *"nclEditingCommand"* al valor de eventId (valor "3", en la Figura 8). Un descriptor de evento también debe ser transmitido con el valor apropiado para eventId, en el ejemplo el valor "3", y el valor de commandTag igual a "0x05", que indica un comando addDocument (ver Sección 9). El parámetro uri debe tener obligatoriamente el valor "null" y el parámetro id debe tener obligatoriamente el valor (component_tag= "0x09", structureId= "0x0B"), como en la Figura 8.

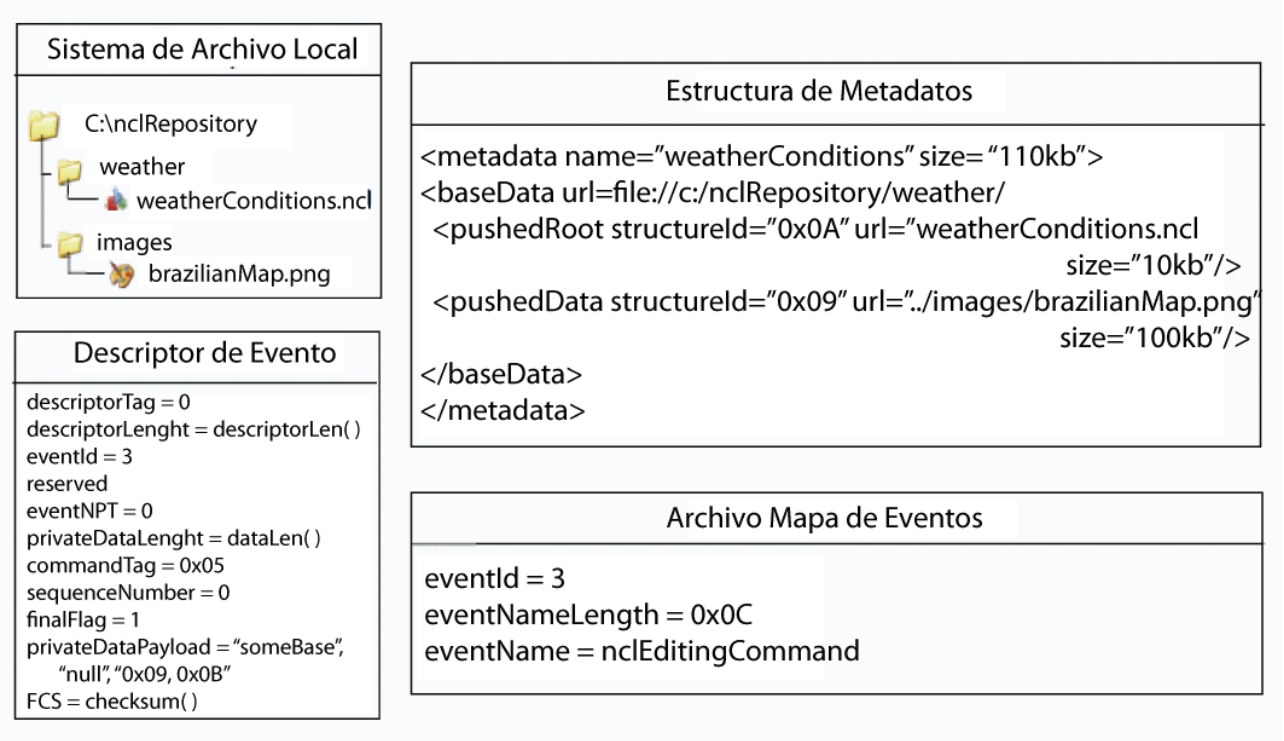

Figura 8 – Ejemplo de transmisión de documento NCL usando Secciones NCL MPEG-2

12.7.2.2.3 Transporte de las estructuras de metadatos como parámetro de comandos de edición

Una alternativa al transporte de estructuras de metadatos directamente en Secciones NCL es tratar estas estructuras como parámetros de los comandos addDocument y addNode, transportados en el campo *privateDataPayload* de los descriptores de evnto.

En ese caso, el conjunto de pares {uri, id} de los comandos addDocument y addNode es sustituido por parámetros de la estructura de metadatos, que definen un conjunto de pares {"uri", "component_tag, structureId"} para cada archivo transmitido sin solicitación *(pushed file)*.

En el ejemplo de la Figura 8, el nuevo transporte sería exactamente el mismo, con excepción del descriptor de evento. Esos descriptores en vez de tener el par {uri; id} igual al valor {"null"; "0x09, 0x0B"} como parámetro, tendrían la estructura de metadatos seriada. En la estructura de metadatos, los atributos *component-tag* de los elementos <pushedRoot> y <pushedData> deben obligatoriamente, en este caso, ser definidos, toda vez que la estructura no es transportada más en el mismo flujo elemental que transporta los archivos de la aplicación NCL.

12.7.2.2.4 Transporte de estructuras de metadatos en secciones de metadatos MPEG-2

Otra alternativa es transportar las estructuras de metadatos en secciones de metadatos MPEG-2, transportadas en flujos MPEG-2 del tipo "0x16". Como siempre, cada sección de metadatos MPEG-2 puede contener datos de solamente una estructura de metadatos. Sin embargo, una estructura de metadatos puede extenderse por varias secciones de metadatos.

La Tabla 61 ilustra la sintaxis de la sección de metadatos para el transporte de estructuras de metadatos, que deben estar de acuerdo con ISO/IEC 13818-1:2007.

Tabla 61 – Sintaxis de la sección para el transporte de estructuras de metadatos

Sintaxis	Número de bits	Valor
Metadata section() {		
table_id	8	0x06
section_syntax_indicator	1	1
prívate_indicator	1	1
random_access_indicator	1	1
decoder_config_flag	1	0
metadata_section_length	12	entero
metadata_service_id	8	entero a ser estandarizado
reserved	8	
section_fragment_indication	2	de acuerdo con la Tabla 62
versión_number	5	entero
current_next_indicator	1	1
section_number	8	entero
last_section_number	8	entero
structureId	8	entero
For (i=1; i< N; i++) {		
serialized_metadata_structure_byte	8	
}		
CRC_32	32	
}		

Tabla 62 – Indicación de fragmento de sección

Valor	Descripción
11	Una única sección de metadatos cargando una estructura de metadatos completa
10	Primera sección de metadatos de una serie, con datos de una misma estructura de metadatos
01	Última sección de metadatos de una serie, con datos de una misma estructura de metadatos
00	Una sección de metadatos de una serie, con datos de una misma estructura de metadatos, pero que no es la primera ni la última sección

Como anteriormente, en el mismo flujo elemental que transporta la especificación XML (el archivo del documento NCL u otro archivo XML usados en los comandos de edicón NCL), es recomendable que se transmita un archivo mapa de eventos con el fin de mapear el nombre *"nclEditingCommand"* al *eventId* del descriptor de evento que carga el comando de edición NCL, comose describe en la Sección 9. El campo *privateDataPayload* del descriptor de evento debe obligatoriamente transportar un conjunto de pares de referencia {uri, id}. Los parámetros uri deben obligatoriamente el valor "null". En el caso de los comandos *addDocument* y *addNode,* el parámetro id del primer par debe obligatoriamente identificar el flujo elemental ("component_tag") del tipo= "0x16" y la estructura de

metadatos (“structureId”) que carga el camino absoluto del documento NCL o de la especificación del nudo NCL (el camino en el servidor de datos). Si otras estructuras de metadatos es usada para relacionar archivos presentes en el documento NCL o en la especificación del nudo NCL, con el fin de completar los comandos addDocument o addNode con contenidos de media, otros pares de referencia {uri, id} deben definirse en el comando. En ese caso, el parámetro uri debe obligatoriamente tener el valor “null” y el parámetro id correspondiente en el par debe obligatoriamente referir al component_tag y el structureId del metadato correspondiente.

En el ejemplo de la Figura 8, el nuevo escenario sería similar. Sólo se deben hacer pequeños cambios de forma que el descriptor de evento refiera al flujo elemental y su sección que carga la estructura de metadatos (component_tag= “0x08” y structureId= “0x0B”), y que la estructura de metadatos también refiera al flujo elemental dónde los archivos del documento serán transportados. La Figura 9 ilustra la nueva situación.

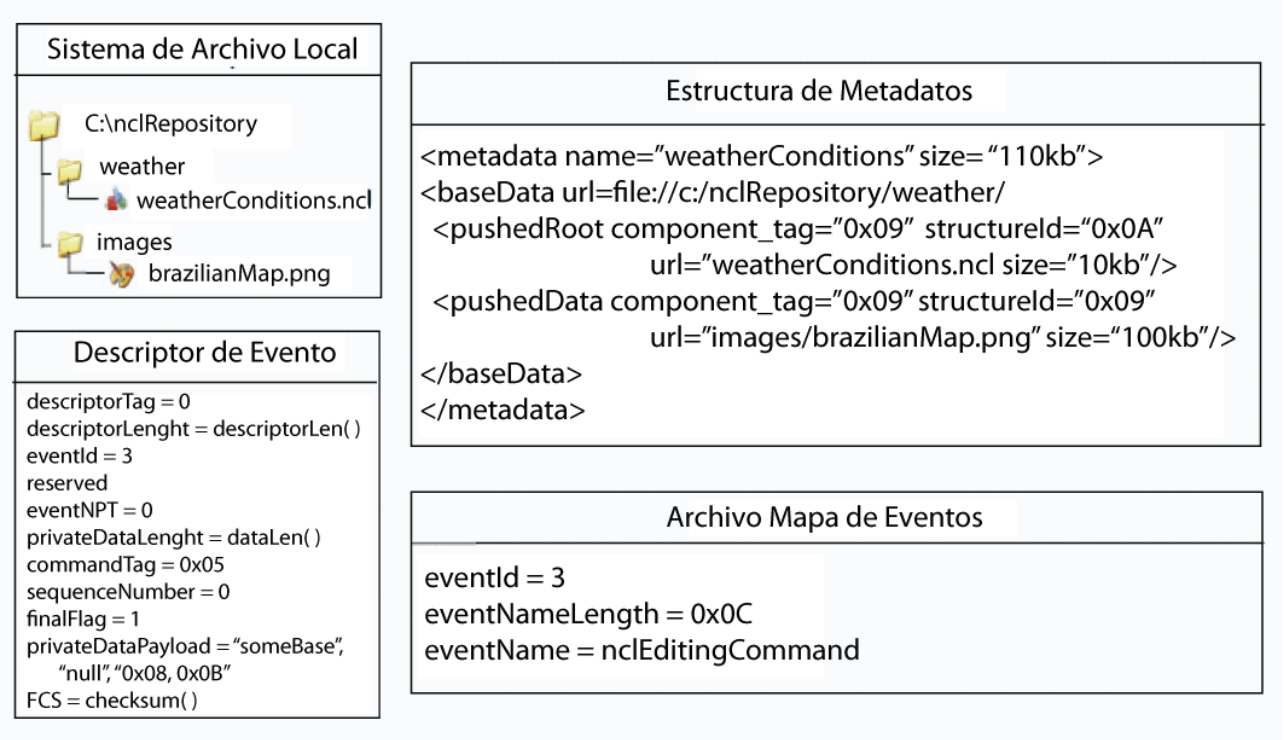

Figura 9 – Ejemplo de transmisión de documento NCL usando Secciones de Metadatos MPEG-2

12.7.3 Transmisión de documentos XML externos

Los documentos XML externos referidos por los elementos <media>, como, por ejemplo, un objeto de media XHTML, se debe transmitir obligatoriamente a través de secciones MPEG-2 específicas (ver la atribució de tipos d flujo para secciones MPEG-2 en ISO/IEC 13818-1).

13 Seguridad

El modelo de seguridad Ginga es totalmente compatible con el modelo de seguridad SBTVD. Éste trata con las mismas áreas de seguridad; o sea, autenticación de aplicaciones de difusión, políticas de seguridad para aplicaciones, seguridad sobre el canal de interacción y gestión de certificados.

La autenticación de aplicaciones Ginga-NCL se debe realizar obligatoriamente del mismo modo para aplicaciones Ginga-J. Si está firmada, la aplicación debe seguir obligatoriamente la estructura de firma como especificado para el Ginga-J. Las aplicaciones Ginga-NCL no autenticadas van a operar dentro de un ambiente de caja de arena (sand box). Las aplicaciones Ginga-NCL autenticadas, asociadas a un archivo de solicitud de autorización, pueden tener autorizaciones concedidas fuera de la caja de arena.

Anexo A
(normativo)

Esquemas de los módulos NCL 3.0 que se utilizan en los perfiles TVD Básico y TVD Avanzado

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="device" type="string" use="optional"/>
   <attribute name="region" 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="beginText" type="string" use="optional"/>
<attribute name="beginPosition" type="unsignedLong" use="optional"/>
<attribute name="endText" type="string" use="optional"/>
<attribute name="endPosition" type="unsignedLong" use="optional"/>

</attributeGroup>

<!-- define the sampleAnchorAttrs attribute group -->
<attributeGroup name="sampleAnchorAttrs">
        <attribute name="first" type="string" use="optional"/>
    <attribute name="last" type="string" 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>

<!-- define the clipAttrs attribute group -->
<attributeGroup name="clipAttrs">
   <attribute name="clip" 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" />
  <attributeGroup ref="mediaAnchor:clipAttrs" />
  </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" />
  <attribute name="externable" type="boolean" use="optional" />
</complexType>

<!--

The following reserved words are used for properties' names.

* For audio media-obejcts: soundLevel; balanceLevel; trebleLevel; bassLevel
* For text media-objects: style, which refers to a style sheet with information for text presentation; textAlign;
fontColor; fontFamily; fontStyle; fontSize; fontVariant; fontWeight.
* For visual media-objects: 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 shall be between 0 and 1, or a real value in the range [0,100]
ending with the character "%" (e.g. 30%)); visible, indicating if the presentation is to be seen or hidden;
rgbChromakey; the object positioning parameters: top, left, bottom, right, width, height, zIndex, plan, location, size
and bounds; the focus movement parameters: moveLeft, moveRight, moveUp, moveDown, focusIndex; the other
related focus parameters: focusBorderColor, selBorderColor, focusBorderWidth, focusBorderTransparency,
focusSrc, and focusSelSrc; the transition parameters: transIn and transOut; the timing parameters: explicitDur and
freeze; and the multiple device parameters: baseDeviceRegion and deviceClass.
* For media-objects in general: player; reusePlayer, which determines if a new player shall be instantiated or if a
player already instantiated shall be used; and playerLife, which specifies what will happen to the player instance at
the end of the presentation.

-->
<!-- declare global elements in this module -->

-->
  <!-- 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; textAlign; fontColor; FontFamily;
fontStyle; fontSize; fontVariant; fontWeight.
* 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 shall be between 0 and 1, or a real value in the range [0,100] ending with the character “%” (e.g. 30%)); visible,
indicating if the presentation is to be seen or hidden; rgbChromakey; the object positioning parameters: top, left, bottom, right,
width, height, zIndex, plan, location, size and bounds; the focus movement parameters: moveLeft, moveRight, moveUp,
moveDown, focusIndex; the other related focus parameters: focusBorderColor, selBorderColor, focusBorderWidth,
focusBorderTransparency, focusSrc, and focusSelSrc; the transition parameters: transIn and transOut; the timing parameters:
explicitDur and freeze; and the multiple device parameters: baseDeviceRegion and deviceClass

* For players in general: player; reusePlayer, which determines if a new player shall be instantiated or if a player already
instantiated shall 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="stateProtoype">
  <restriction base="string">
    <enumeration value="sleeping" />
    <enumeration value="occurring" />
    <enumeration value="paused" />
  </restriction>
</simpleType>

<simpleType name="valueUnion">
<union memberTypes="string connectorAssessmentExpression:stateProtoype"/>
</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="optional"/>
    <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:rulePrototype"/>
  <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"/>
     <attribute name="baseId" type="IDREF" use="optional"/>
</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="instance" type="string" 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="positiveInteger" use="optional"/>
<attribute name="moveRight" type="positiveInteger" use="optional"/>
<attribute name="moveUp" type="positiveInteger" use="optional"/>
<attribute name="moveDown" type="positiveInteger" use="optional"/>
<attribute name="focusIndex" type="positiveInteger" 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>
```

A.28 Transition module: NCL30Transition.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/NCL30Transition.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Transition module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:transition="http://www.ncl.org.br/NCL3.0/Transition"
  targetNamespace="http://www.ncl.org.br/NCL3.0/Transition"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >


  <!-- declare global attributes in this module -->

  <!-- define the type attribute prototype -->
  <simpleType name="typePrototype">
    <restriction base="string">
      <enumeration value="in"/>
      <enumeration value="barWipe"/>
```

```
      <enumeration value="boxWipe"/>
      <enumeration value="fourBoxWipe"/>
      <enumeration value="barnDoorWipe"/>
      <enumeration value="diagonalWipe"/>
      <enumeration value="bowTieWipe"/>
      <enumeration value="miscDiagonalWipe"/>
      <enumeration value="veeWipe"/>
      <enumeration value="barnVeeWipe"/>
      <enumeration value="zigZagWipe"/>
      <enumeration value="barnZigZagWipe"/>
      <enumeration value="irisWipe"/>
      <enumeration value="triangleWipe"/>
      <enumeration value="arrowHeadWipe"/>
      <enumeration value="pentagonWipe"/>
      <enumeration value="hexagonWipe"/>
      <enumeration value="ellipseWipe"/>
      <enumeration value="eyeWipe"/>
      <enumeration value="roundRectWipe"/>
      <enumeration value="starWipe"/>
      <enumeration value="miscShapeWipe"/>
      <enumeration value="clockWipe"/>
      <enumeration value="pinWheelWipe"/>
      <enumeration value="singleSweepWipe"/>
      <enumeration value="fanWipe"/>
      <enumeration value="doubleFanWipe"/>
      <enumeration value="doubleSweepWipe"/>
      <enumeration value="saloonDoorWipe"/>
      <enumeration value="windshieldWipe"/>
      <enumeration value="snakeWipe"/>
      <enumeration value="spiralWipe"/>
      <enumeration value="parallelSnakesWipe"/>
      <enumeration value="boxSnakesWipe"/>
      <enumeration value="waterfallWipe"/>
      <enumeration value="pushWipe"/>
      <enumeration value="slideWipe"/>
      <enumeration value="fade"/>
      <enumeration value="audioFade"/>
      <enumeration value="audioVisualFade"/>
    </restriction>
  </simpleType>

  <!-- define subType attribute prototype-->
  <simpleType name="subTypePrototype">
    <restriction base="string">
      <enumeration value="bottom"/>
      <enumeration value="bottomCenter"/>
      <enumeration value="bottomLeft"/>
      <enumeration value="bottomLeftClockwise"/>
      <enumeration value="bottomLeftCounterClockwise"/>
      <enumeration value="bottomLeftDiagonal"/>
      <enumeration value="bottomRight"/>
      <enumeration value="bottomRightClockwise"/>
      <enumeration value="bottomRightCounterClockwise"/>
      <enumeration value="bottomRightDiagonal"/>
      <enumeration value="centerRight"/>
      <enumeration value="centerTop"/>
      <enumeration value="circle"/>
      <enumeration value="clockwiseBottom"/>
      <enumeration value="clockwiseBottomRight"/>
```

```
<enumeration value="clockwiseLeft"/>
<enumeration value="clockwiseNine"/>
<enumeration value="clockwiseRight"/>
<enumeration value="clockwiseSix"/>
<enumeration value="clockwiseThree"/>
<enumeration value="clockwiseTop"/>
<enumeration value="clockwiseTopLeft"/>
<enumeration value="clockwiseTwelve"/>
<enumeration value="cornersIn"/>
<enumeration value="cornersOut"/>
<enumeration value="counterClockwiseBottomLeft"/>
<enumeration value="counterClockwiseTopRight"/>
<enumeration value="crossfade"/>
<enumeration value="diagonalBottomLeft"/>
<enumeration value="diagonalBottomLeftOpposite"/>
<enumeration value="diagonalTopLeft"/>
<enumeration value="diagonalTopLeftOpposite"/>
<enumeration value="diamond"/>
<enumeration value="doubleBarnDoor"/>
<enumeration value="doubleDiamond"/>
<enumeration value="down"/>
<enumeration value="fadeFromColor"/>
<enumeration value="fadeToColor"/>
<enumeration value="fanInHorizontal"/>
<enumeration value="fanInVertical"/>
<enumeration value="fanOutHorizontal"/>
<enumeration value="fanOutVertical"/>
<enumeration value="fivePoint"/>
<enumeration value="fourBlade"/>
<enumeration value="fourBoxHorizontal"/>
<enumeration value="fourBoxVertical"/>
<enumeration value="fourPoint"/>
<enumeration value="fromBottom"/>
<enumeration value="fromLeft"/>
<enumeration value="fromRight"/>
<enumeration value="fromTop"/>
<enumeration value="heart"/>
<enumeration value="horizontal"/>
<enumeration value="horizontalLeft"/>
<enumeration value="horizontalLeftSame"/>
<enumeration value="horizontalRight"/>
<enumeration value="horizontalRightSame"/>
<enumeration value="horizontalTopLeftOpposite"/>
<enumeration value="horizontalTopRightOpposite"/>
<enumeration value="keyhole"/>
<enumeration value="left"/>
<enumeration value="leftCenter"/>
<enumeration value="leftToRight"/>
<enumeration value="oppositeHorizontal"/>
<enumeration value="oppositeVertical"/>
<enumeration value="parallelDiagonal"/>
<enumeration value="parallelDiagonalBottomLeft"/>
<enumeration value="parallelDiagonalTopLeft"/>
<enumeration value="parallelVertical"/>
<enumeration value="rectangle"/>
<enumeration value="right"/>
<enumeration value="rightCenter"/>
<enumeration value="sixPoint"/>
<enumeration value="top"/>
```

```
      <enumeration value="topCenter"/>
      <enumeration value="topLeft"/>
      <enumeration value="topLeftClockwise"/>
      <enumeration value="topLeftCounterClockwise"/>
      <enumeration value="topLeftDiagonal"/>
      <enumeration value="topLeftHorizontal"/>
      <enumeration value="topLeftVertical"/>
      <enumeration value="topRight"/>
      <enumeration value="topRightClockwise"/>
      <enumeration value="topRightCounterClockwise"/>
      <enumeration value="topRightDiagonal"/>
      <enumeration value="topToBottom"/>
      <enumeration value="twoBladeHorizontal"/>
      <enumeration value="twoBladeVertical"/>
      <enumeration value="twoBoxBottom"/>
      <enumeration value="twoBoxLeft"/>
      <enumeration value="twoBoxRight"/>
      <enumeration value="twoBoxTop"/>
      <enumeration value="up"/>
      <enumeration value="vertical"/>
      <enumeration value="verticalBottomLeftOpposite"/>
      <enumeration value="verticalBottomSame"/>
      <enumeration value="verticalLeft"/>
      <enumeration value="verticalRight"/>
      <enumeration value="verticalTopLeftOpposite"/>
      <enumeration value="verticalTopSame"/>
    </restriction>
  </simpleType>

  <attributeGroup name="transAttrs">
    <attribute name="transIn" type="string" use="optional"/>
    <attribute name="transOut" type="string" use="optional"/>
  </attributeGroup>

  <!-- define the transition attribute group -->
  <attributeGroup name="transitionAttrs">
    <attribute name="type" type="transition:typePrototype" use="required"/>
    <attribute name="subtype" type="transition:subTypePrototype" use="optional"/>
    <attribute name="fadecolor" type="string" use="optional" default="black"/>
    <attribute name="dur" type="string" use="optional"/>
    <attribute name="startProgress" use="optional" default="0.0">
      <simpleType>
        <restriction base="decimal">
          <minInclusive value="0.0"/>
          <maxInclusive value="1.0"/>
        </restriction>
      </simpleType>
    </attribute>
    <attribute name="endProgress" use="optional" default="1.0">
      <simpleType>
        <restriction base="decimal">
          <minInclusive value="0.0"/>
          <maxInclusive value="1.0"/>
        </restriction>
      </simpleType>
    </attribute>
    <attribute name="direction" use="optional" default="forward">
      <simpleType>
        <restriction base="string">
```

```
          <enumeration value="forward"/>
          <enumeration value="reverse"/>
        </restriction>
      </simpleType>
    </attribute>
  </attributeGroup>

  <!-- define the transition-modifier attribute group -->
  <attributeGroup name="transitionModifierAttrs">
    <attribute name="horzRepeat" type="decimal" use="optional"  default="1.0"/>
    <attribute name="vertRepeat" type="decimal" use="optional"  default="1.0"/>
    <attribute name="borderWidth" type="nonNegativeInteger" use="optional"  default="0"/>
    <attribute name="borderColor" type="string" use="optional"  default="black"/>
  </attributeGroup>

  <complexType name="transitionPrototype">
     <attributeGroup ref="transition:transitionAttrs"/>
     <attributeGroup ref="transition:transitionModifierAttrs"/>
  </complexType>

  <!-- declare global element in this module -->
  <element name="transition" type="transition:transitionPrototype"/>

</schema>
```

A.29 Metainformation module: NCL30Metainformation.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/NCL30Metainformation.xsd
Author: TeleMidia Laboratory
Revision: 19/09/2006

Schema for the NCL Metainformation module namespace.
-->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:metainformation="http://www.ncl.org.br/NCL3.0/Metainformation"
  targetNamespace="http://www.ncl.org.br/NCL3.0/Metainformation"
  elementFormDefault="qualified" attributeFormDefault="unqualified" >

  <complexType name="metaPrototype">
    <attribute name="name" type="string" use="required"/>
    <attribute name="content" type="string" use="required"/>
  </complexType>

  <complexType name="metadataPrototype">
     <sequence>
          <any minOccurs="0"/>
     </sequence>
  </complexType>
```

```
  <!-- declare global elements in this module -->
  <element name="meta" type="metainformation:metaPrototype"/>

  <!-- declare global elements in this module -->
  <element name="metadata" type="metainformation:metadataPrototype"/>

</schema>
```

Anexo B
(informativo)

Manual de referencia de Lua 5.1

B.1 Introducción

NOTA Este Anexo presenta la especificación del lenguaje de programación Lua, versión 5.1. Este contenido es una traducción del libro de Roberto Ierusalimschy, Luiz Henrique de Figueiredo y Waldemar Celes, Lua 5.1 Reference Manual (Lua.org, agosto de 2006. ISBN 85-903798-3-3) y se imprime nuevamente aquí con aprobación de los autores. El libro también está disponible en línea, en inglés, en <http://www.lua.org/manual/5.1/>.

Lua es un lenguaje de programación de extensión proyectada para dar soporte a la programación procedural en general y que ofrece facilidades para la descripción de datos. El lenguaje también ofrece un buen soporte para programación orientada a objetos, programación funcional y programación orientada a datos. Lua fue planeada para ser utilizada por cualquier aplicación que necesite un lenguaje de script ligero y poderoso. Lua es implementada como una biblioteca, grabación en C limpio (es decir, en el subconjunto común de ANSI C y C++).

Por ser un lenguaje de extensión, Lua no posee la noción de un programa principal: éste solamente funciona incorporado en un programa cliente anfitrión, llamado programa hospedero o simplemente hospedero. Ese programa hospedero puede invocar funciones para ejecutar un pedazo de código Lua, puede escribir y leer variables Lua y puede registrar funciones C para ser llamadas por el código Lua. A través del uso de funciones C, Lua puede ser extendida para tratar de manera apropiada una amplia variedad de dominios, permitiendo así la creación de lenguajes de programación personalizados que comparten un núcleo sintáctico. La distribución Lua incluye un ejemplo de un programa hospedero llamado Lua, el cual usa la biblioteca de Lua para ofrecer un interpretador de línea de comando Lua completo.

Lua es un software libre y, como es usual, se suministra sin garantías, como se indica en su licencia. La implementación descrita en esta Norma, así como artículos técnicos sobre Lua, están disponibles en el sitio web oficial de Lua, www.lua.org.

B.2 El Lenguaje

B.2.1 Notación utilizada

Las construcciones del lenguaje serán explicadas usando la notación BNF extendida usual, en la cual *{a}* significa 0 o más *aes* y *[a]* significa una *a* opcional. No terminales se muestran como non-terminal, palabras clave se muestran como **kword** y otros símbolos terminales se muestran como **`=´**. La sintaxis completa de Lua está descrita en B.8.

B.2.2 Convenciones léxicas

En Lua, *Nomes* (también llamados identificadores) pueden ser cualquier cadena de letras, dígitos y subrayados que no empiecen con un dígito. Esta definición está de acuerdo con la definición de nombres en la mayoría de los lenguajes (la definición de letras depende de cuál es el idioma *(locale):* Cualquier carácter considerado alfabético por el idioma corriente se puede usar como un identificador). Identificadores se usan para nombrar variables y campos de tablas.

Las siguientes palabras clave son reservadas y no se pueden utilizar como nombres:

```
and       break     do        else      elseif
end       false     for       function  if
in        local     nil       not       or
repeat    return    then      true      until     while
```

Lua es un lenguaje que diferencia minúsculas de mayúsculas: and es una palabra reservada, pero And y AND son dos nombres válidos diferentes. Como convención, nombres que empiezan con un subrayado seguido por letras mayúsculas (tales como _VERSION) son reservados para variables globales internas usadas por Lua.

Las siguientes cadenas denotan otros ítems léxicos:

```
+     -     *     /     %     ^
==    ~=    <=    >=    <     >
(     )     {     }     [     ]
;     :     ,     .     ..    ...
```

Cadenas de caracteres literales pueden ser delimitadas a través del uso de comillas simples o comillas dobles, y pueden contener las siguientes secuencias de escape en el estilo de C: '\a' (campanilla), '\b' (backspace), \f' (alimentación de formulario), '\n' (quiebra de línea), '\r' (retorno de carro), '\t' (tabulación horizontal), '\v' (tabulación vertical), '\\' (barra invertida), '\"' (citación [comilla doble]) y '\'' (apóstrofo [comilla simple]). Además de ello, una barra invertida seguida por una quiebra de línea real da como resultado una quiebra de línea en la cadena de caracteres. Un carácter en una cadena de caracteres también se puede especificar por su valor numérico usando la secuencia de escape *ddd,* donde *ddd* es una secuencia de hasta tres dígitos decimales. Si un carácter numérico representado como una secuencia de escape es seguido por un dígito, la secuencia de escape debe poseer exactamente tres dígitos. Cadenas de caracteres en Lua pueden contener cualquier valor de 8 bits, incluyendo ceros dentro de las mismas, los cuales se pueden especificar como '\0'.

Para colocar una comilla doble (simple), una quiebra de línea, una barra invertida o insertar un cero dentro de una cadena de caracteres literal delimitada por comillas dobles (simple) se debe usar una secuencia de escape. Cualquier otro carácter se puede insertar directamente dentro de la cadena literal (algunos caracteres de control pueden causar problemas para el sistema de archivos, pero Lua no tiene ningún problema con relación a ellos).

Cadenas literales largas también pueden ser definidas usando un formato largo delimitado por corchetes largos. Se definió una apertura de corchete largo de nivel n como un abre corchete seguido por n señales de igual seguido por otro abre corchete. De esa forma, una apertura de corchete largo de nivel 0 es escrita como [[, una apertura de corchete largo de nivel 1 es escrita como [=[y así sucesivamente. Un cierre de corchete largo se define de manera análoga; por ejemplo, un cierre de corchete largo de nivel 4 se escribe como]====]. Una cadena de caracteres larga empieza con una apertura de corchete largo de cualquier nivel y termina en el primer cierre de corchete largo del mismo nivel. Literales expresados de esta forma pueden extenderse por varias líneas, no interpretan ninguna secuencia de escape e ignoran corchetes largos de cualquier otro nivel. Estos literales pueden contener cualquier cosa, excepto un cierre de corchete largo de nivel igual al de la apertura.

Por conveniencia, cuando una apertura de corchete largo es inmediatamente seguida por una quiebra de línea, la quiebra de línea no es incluida en la cadena de caracteres. Como ejemplo, en un sistema usando ASCII (en el cual ‘a’ se codifica como 97, quiebra de línea se codifica como 10 y '1' se codifica como 49), los cinco literales siguientes denotan la misma cadena:

```
a = 'alo\n123"'
a = "alo\n123\""
a = '\97lo\10\04923"'
a = [[alo
123"]]
a = [==[
alo
123"]==]
```

Una constante numérica puede ser escrita con una parte decimal opcional y con un exponente decimal opcional. Lua también acepta constantes hexadecimales enteras, a través del uso del código 0x. Ejemplos de constantes numéricas válidas son:

```
3   3.0   3.1416   314.16e-2   0.31416E1   0xff   0x56
```

Un comentario empieza con un guión doble (--) en cualquier lugar, siempre que sea fuera de una cadena de caracteres. Si el texto inmediatamente después de -- no es una apertura de corchete largo, el comentario es un comentario corto, el cual se extiende hasta el fin de la línea. En caso contrario, es un comentario largo, que se extiende hasta el cierre de corchete largo correspondiente. Comentarios largos se usan frecuentemente para desactivar código temporariamente.

B.2.3 Valores y tipos

B.2.3.1 Tipos básicos

Lua es un lenguaje dinámicamente tipado. Esto significa que variables no poseen tipos; solamente valores poseen tipos. No existe definición de tipos en el lenguaje. Todos los valores transportan su propio tipo.

Todos los valores en Lua son valores de primera clase. Esto significa que todos los valores pueden ser almacenados en variables, pasados como argumentos para otras funciones y retornados como resultados.

Existen ocho tipos básicos en Lua: *nil, boolean, number, string, function, userdata, thread* y *table. Nil* es el tipo del valor **nil,** cuya propiedad principal es ser diferente de cualquier otro valor; generalmente representa la ausencia de un valor útil. *Boolean* es el tipo de los valores **false** y **true.** Tanto **nil** como **false** tornan una condición falsa; cualquier otro valor torna la condición verdadera. *Number* representa números reales (punto flotante de precisión doble). Es fácil construir interpretadores Lua que usen otra representación interna para números, tales como precisión simple de punto flotante o enteros largos; ver el archivo luaconf.h. El tipo *string* representa cadenas de caracteres. En Lua, cadenas de caracteres pueden contener cualquier carácter de 8 bits, incluyendo ceros ('\0') dentro de la misma (ver B.2.2).

Lua puede llamar (y manejar) funciones escritas en Lua y funciones escritas en C (ver B.2.6.9).

El tipo *userdata* permite que datos C arbitrarios puedan ser almacenados en variables Lua. Este tipo corresponde a un bloque de memoria y no tiene operaciones predefinidas en Lua, excepto atribución y prueba de identidad. Sin embargo, a través del uso de *metatables,* el programador puede definir operaciones para valores userdata (ver B.2.9). Valores userdata no pueden ser creados o modificados en Lua, solamente a través de la API C. Esto garantiza la integridad de los datos que pertenecen al programa hospedero.

El tipo *thread* representa flujos de ejecución independientes y se utiliza para implementar co-rutinas (ver B.2.12). No confundir el tipo thread de Lua con procesos ligeros del sistema operativo. Lua da soporte a co-rutinas en todos los sistemas, incluso en aquéllos que no dan soporte a procesos ligeros.

El tipo *table* implementa arrays asociativos, es decir, arrays que pueden ser indexados no apenas por números, sino por cualquier valor (excepto **nil).** Tablas pueden ser heterogéneas; es decir, pueden contener valores de todos los tipos (excepto **nil).** Tablas son el único mecanismo de estructuración de datos en Lua; se pueden usar para representar arrays comunes, tablas de símbolos, conjuntos, registros, grafos, árboles, etc. Para representar registros, Lua usa el nombre del campo como un índice. El lenguaje da soporte a esta representación ofreciendo a.name como un azúcar sintáctico para a["name"]. Existen varias formas convenientes de crear tablas en Lua (ver B.2.6.8).

De la misma forma que los índices, el valor de un campo de la tabla puede poseer cualquier tipo (excepto **nil).** En particular, dado que funciones son valores de primera clase, campos de tabla pueden contener funciones. Por lo tanto, tablas pueden también poseer métodos (ver B.2.6.10).

Valores del tipo table, function, thread y userdata (completo) son objetos: Variables no contienen realmente estos valores, solamente referencias a ellos. Atribución, paso de parámetro y retorno de funciones siempre manejan informes para tales valores; estas operaciones no significan ninguna especie de copia.

La función *type* retorna una cadena de caracteres describiendo el tipo de un cierto valor.

B.2.3.2 Coerción

Lua provee conversión automática entre valores del tipo string y del tipo number en tiempo de ejecución. Cualquier operación aritmética aplicada a una cadena de caracteres intenta convertir esta cadena en un número, siguiendo las reglas de conversión usuales. De forma análoga, siempre que un número se utiliza donde una cadena de caracteres es esperada, el número es convertido para una cadena, en un formato razonable. Para un control completo sobre cómo números son convertidos en cadenas, usar la función format de la biblioteca string (ver *string.format).*

B.2.4 Variables

Variables son lugares que se utilizan para almacenar valores.

Existen tres tipos de variables en Lua: Variables globales, variables locales y campos de tablas.

Un nombre simple puede denotar una variable global o una variable local (o un parámetro formal de una función, que es un caso particular de variable local):

```
var ::= Nombre
```

Nombre denota identificadores, como definido en B.2.2.

Se presupone que toda variable es una variable global a menos que sea explícitamente declarada como una variable local (ver B.2.5.8). Variables locales poseen alcance léxico: Variables locales pueden ser libremente accedidas por funciones definidas dentro de su alcance (ver B.2.7).

Antes que la variable reciba su primera atribución, su valor es **nil.**

Corchetes se usan para indexar una tabla:

```
var ::= expprefijo `[´ exp `]´
```

La semántica de accesos a variables globales y a campos de tablas puede ser alterada a través del uso de meta-tablas. Un acceso a una variable indexada t[i] es equivalente a una llamada gettable_event(t,i) (ver B.2.9 para una descripción completa de la función gettable_event. Esta función no es definida ni puede ser llamada en Lua. Se utiliza aquí solamente para fines didácticos).

La sintaxis var.Nome es sólo un azúcar sintáctico para var["Nombre"]:

```
var : := expprefijo `.´ Nombre
```

Todas las variables globales son mantenidas como campos en tablas Lua comunes, llamadas de tablas de ambiente o simplemente de ambientes (ver B.2.10). Cada función tiene su propia referencia para un ambiente, de forma que todas las variables globales dentro de una función se referirán a esta tabla de ambiente. Cuando una función es creada, hereda el ambiente de la función que la creó. Para obtener la tabla de ambiente de una función Lua, se debe llamar a *getfenv.* Para cambiar la tabla de ambiente, se debe llamar a *setfenv* (la única manera de tratar el ambiente de funciones C es a través de la biblioteca de depuración; ver B.5.11).

Un acceso a una variable global x es equivalente a _env.x, que a su vez es equivalente a

```
gettable_event (_env, "x")
```

donde _env es el ambiente de la función corriente (ver B.2.9 para una descripción completa de la función gettable_event. Esta función no es definida ni puede ser llamada en Lua. De modo análogo, la variable _env no es definida en Lua. Fueron usadas aquí solamente para fines didácticos).

B.2.5 Comandos

B.2.5.1 Conceptos básicos

Lua ofrece un conjunto casi convencional de comandos, análogo al conjunto de comandos disponibles en Pascal o C. Este conjunto incluye atribución, estructuras de control, llamadas de funciones y declaraciones de variables.

B.2.5.2 Trechos

La unidad de ejecución de Lua se llama trecho. Un trecho es simplemente una secuencia de comandos, los cuales se ejecutan en forma secuencial. Cada comando puede opcionalmente ser seguido por un punto y coma:

```
trecho ::= {comando [ `;´]}
```

No existen comandos vacíos y por lo tanto la construcción ';;' no es válida.

Lua trata un trecho como el cuerpo de una función anónima con un número variable de argumentos (ver B.2.6.10). De esta forma, trechos pueden definir variables locales, recibir argumentos y retornar valores.

Un trecho puede ser almacenado en un archivo o en una cadena de caracteres dentro del programa hospedero. Cuando un trecho se ejecuta, es primero pre-compilado en instrucciones para una máquina virtual y después el código compilado es ejecutado por un interpretador para la máquina virtual.

Los trechos también pueden ser pre-compilados en una forma binaria; ver el programa luac para más detalles. Programas en forma de código fuente y en forma de un archivo fuente ya compilado son intercambiables; Lua automáticamente determina cual es el tipo del archivo y actúa de conformidad con el mismo.

B.2.5.3 Bloques

Un bloque es una lista de comandos; sintéticamente, un bloque es la misma cosa que un trecho:

```
bloque : := trecho
```

Un bloque puede ser explícitamente delimitado para producir un único comando:

```
comando : := do bloque end
```

Bloques explícitos son útiles para controlar el alcance de declaraciones de variables. Bloques explícitos son también usados a veces para agregar un comando **return** o **break** en medio de otro bloque (ver B.2.5.5).

B.2.5.4 Atribución

Lua permite atribución múltiple. En virtud de ello, la sintaxis para atribución define una lista de variables del lado izquierdo y una lista de expresiones del lado derecho. Los elementos en ambos lados son separados por comas:

```
comando ::= listavar `=´ listaexp

listavar ::= var { `,´ var}

listaexp ::= exp { `,´ exp}
```

Expresiones son discutidas en B.2.6.

Antes de que la atribución sea realizada, la lista de valores es ajustada a la longitud de la lista de variables. Si hay más valores de lo necesario, los valores en exceso se descartan. Si hay menos valores de lo necesario, la lista es extendida con tantos **nil**'s como sean necesarios. Si la lista de expresiones termina con una llamada de función, entonces todos los valores retornados por esta llamada entran en la lista de valores, antes de ser realizado el ajuste (excepto cuando la llamada es delimitada por paréntesis; ver B.2.6).

Un comando de atribución primero evalúa todas sus expresiones y solamente después se realiza la atribución. De esta forma, el código

```
i = 3

i, a[ i] = i+1, 20
```

atribuye 20 a a[3], sin afectar a[4] porque el I en a[i] es evaluado (a 3) antes de recibir el valor 4. De modo análogo, la línea

```
x, y = y, x
```

cambia los valores de x e y.

La semántica de atribuciones para variables globales y campos de tablas puede ser alterada a través del uso de meta-tablas. Una atribución para una variable indexada t[i] = Val es equivalente a settable_event(t,i,val) (ver B.2.9 para una descripción completa de la función settable_event. Esta función no es definida ni puede ser llamada en Lua. Fue usada aquí solamente para fines didácticos).

Una atribución a una variable global x = val es equivalente a la atribución _env.x = val, que a su vez es equivalente a

```
settable_event(_env, "x", val)
```

donde _env es el ambiente de la función siendo ejecutada (la variable _env no es definida en Lua. Fue usada aquí solamente para fines didácticos).

B.2.5.5 Estructuras de control

Las estructuras de control **if, while** y **repeat** poseen el significado usual y la sintaxis familiar:

```
comando : := while exp do bloque end

comando : := repeat bloque until exp

comando ::= if exp then bloque {elseif exp then bloque} [else bloque] end
```

Lua también posee un comando **for,** el cual posee dos variaciones (ver B.2.5.6).

La expresión de la condición de una estructura de control puede retornar cualquier valor. Tanto **false** como **nil** son considerados un valor falso. Todos los valores diferentes de **nil** y **false** son considerados como verdaderos (en particular, el número 0 y la cadena de caracteres vacía también son considerados valores verdaderos).

En el lazo **repeat–until,** el bloque más interno no termina en la palabra clave **until,** sino solamente después de la condición. De esta forma, la condición puede hacer referencia a variables locales declaradas dentro del bloque del lazo.

El comando **return** se utiliza para retornar valores de una función o de un trecho (que sólo es una función). Funciones y trechos pueden retornar más de un valor, de modo que la sintaxis para el comando **return** es

```
comando : := return [ listaexp]
```

El comando **break** se utiliza para terminar la ejecución de un lazo **while, repeat o for,** saltando para el próximo comando después del lazo:

```
comando : := break
```

Un **break** termina la ejecución del lazo más interno.

Los comandos **return** y **break** solamente pueden ser escritos como el *último* comando de un bloque. Si es realmente necesario tener un **return** o **break** en medio de un bloque, entonces se puede usar un bloque interno explícito, como en las expresiones idiomáticas del return end y del break end, pues ahora tanto el **return** como el **break** son los últimos comandos en sus respectivos bloques (internos).

B.2.5.6 Comando for

El comando **for** posee dos variaciones: una numérica y otra genérica.

El lazo **for** numérico repite un bloque de código mientras una variable de control varía de acuerdo con una progresión aritmética. Posee la siguiente sintaxis:

```
comando ::= for nombre `=´ exp `,´ exp [ `,´ exp] do bloque end
```

El bloque es repetido para nombre empezando con el valor de la primera *exp,* hasta que pase el valor de la segunda *exp* a través de pasos seguidos, siendo que a cada paso el valor de la tercera *exp* es sumado a nombre. De forma más precisa, un comando **for** como

```
for v = e1, e2, e3 do block end
```

es equivalente al 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 lo siguiente:

— todas las tres expresiones de control son evaluadas una única vez, antes del inicio del lazo. Deben obligatoriamente producir números;

— *var, limit* y *step* son variables invisibles. Los nombres fueron utilizados aquí solamente para fines didácticos;

— si la tercera expresión (el paso) está ausente, entonces se utiliza un paso de tamaño 1;

— es posible usar **break** para salir de un lazo **for;**

— la variable de lazo v es local al lazo; no es posible usar el valor de esta variable después del fin del **for** o después del **for** haber sido interrumpido por el uso de un **break.** Si es necesario el valor de esta variable, debe atribuirlo a otra variable antes de interrumpir o salir del lazo.

El comando **for** genérico funciona utilizando funciones, llamadas *repetidoras.* A cada repetición, la función repetidora es llamada para producir un nuevo valor, parando cuando este nuevo valor es **nil.** El lazo **for** genérico posee la siguiente sintaxis:

```
comando : := For listadenombres in listaexp do bloque end
listadenombres ::= Nombre { `,´ Nombre}
```

Un comando **for** como

```
for var_1, ···, var_n in explist do block end
```

es equivalente al 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 lo siguiente:

— *explist* es evaluada solamente una vez. Sus resultados son una función *repetidora,* un *estado* y un valor inicial para la primera variable repetidora*;*

— *f, s* y *var* son variables invisibles. Los nombres fueron utilizados aquí solamente para fines didácticos;

— es posible usar **break** para salir de un lazo **for;**

— las variables de lazo *var_i* son locales al lazo; no es posible usar los valores de las mismas después de la terminación de **for.** Si se necesitan estos valores, deben atribuirse a otras variables antes de interrumpir el lazo o salir del mismo.

B.2.5.7 Llamadas de función como comandos

Para permitir posibles efectos colaterales, se pueden ejecutar funciones como comandos:

```
comando ::= llamadadefuncion
```

En este caso, todos los valores retornados por la función se descartan. Llamadas de función son explicadas en B.2.6.9.

B.2.5.8 Declaraciones locales

Variables locales pueden ser declaradas en cualquier lugar dentro de un bloque. La declaración puede incluir una atribución inicial:

```
comando ::= local listadenombres [`=´ listaexp]
```

Caso ocurra una atribución inicial, su semántica es la misma de una atribución múltiple (ver B.2.5.4). En caso contrario, todas las variables son inicializadas con **nil.**

Un trecho también es un bloque (ver B.2.5.2) y por lo tanto variables locales pueden ser declaradas en un trecho fuera de cualquier bloque explícito. El alcance de una variable declarada de esta forma se extiende hasta el fin del trecho.

Las reglas de visibilidad para variables locales son explicadas en B.2.7.

B.2.6 Expresiones

B.2.6.1 Expresiones básicas

Las expresiones básicas en Lua son las siguientes:

```
exp : := expprefijo
exp ::= nil | false | true
exp : := Numero
exp : := Cadena
exp : := funcion
exp ::= constructortabla
exp ::= `...´
exp : := exp opbin exp
exp : := opunaria exp
expprefixo ::= var | llamadadefuncion | `(´ exp `)´
```

Números y cadenas literales se explican en B.2.2; variables se explican en B.2.4; definiciones de funciones se explican en B.6.10; llamadas funciones se explican en B.2.6.9; constructores de tablas se explican en B.2.6.8. Expresiones *vararg,* denotadas por tres puntos ('...'), solamente se pueden usar cuando están inmediatamente dentro de una función que posee un número variable de argumentos; se explican en B.2.6. 10.

Operadores binarios comprenden operadores aritméticos (ver B.2.6.2), operadores relacionales (ver B.2.6.3), operadores lógicos (ver B.2.6.4) y el operador de concatenación (ver B.2.6.5). Operadores unarios comprenden el menos unario (ver B.2.6.2), el **not** unario (ver B.2.6.4) y el operador de tamaño unario (ver B.2.6.6).

Tanto llamadas de funciones como expresiones vararg pueden dar como resultado múltiples valores. Si la expresión se utiliza como un comando (ver B.2.5.7) (lo que solamente es posible para llamadas de funciones), entonces su lista de retorno es ajustada para cero elementos, descartando, por tanto, todos los valores retornados. Si la expresión se utiliza como el último (o el único) elemento de una lista de expresiones, entonces no se realiza ningún ajuste (a menos que la llamada sea delimitada por paréntesis). En todos los demás contextos, Lua ajusta la lista de resultados para un elemento, descartando todos los valores excepto el primero.

Siguen algunos ejemplos:

```
f()                -- adjusted to 0 results
g(f(), x)          -- f() is adjusted to 1 result
g(x, f())          -- g gets x plus all values returned by f()
a,b,c = f(), x     -- f() is adjusted to 1 result (c gets nil)
a,b = ...          -- a gets the first vararg parameter,
                   -- b gets the second (both a and b may get nil if
                   -- there is no corresponding vararg parameter)
a,b,c = x, f()     -- f() is adjusted to 2 results
a,b,c = f()        -- f() is adjusted to 3 results
return f()         -- returns all values returned by f()
return ...         -- returns all received vararg parameters
return x,y,f()     -- returns x, y, and all values returned by f()
{f()}              -- creates a list with all values returned by f()
{...}              -- creates a list with all vararg parameters
{f(), nil}         -- f() is adjusted to 1 result
```

Una expresión delimitada por paréntesis siempre da como resultado un único valor. De esa forma, (f(x,y,z)) es siempre un único valor, aunque f regrese múltiples valores (el valor de (f(x,y,z)) es el primer valor retornado por f, o **nil** si f no retorna ningún valor).

B.2.6.2 Operadores aritméticos

Lua provee los operadores aritméticos usuales: Los operadores binarios + (adición), - (substracción), * (multiplicación), / (división), % (módulo) y ^ (exponenciación); y el operador unario - (negación). Si los operandos son números o cadenas de caracteres que pueden ser convertidas para números (ver B.2.3.2), entonces todas las operaciones poseen su significado usual. La exponenciación funciona para cualquier exponente. Por ejemplo, x^(-0.5) calcula el inverso de la raíz cuadrada de X. Módulo se define como

```
a % b == a - math.floor(a/b)*b
```

Es decir, es el resto de una división redondeada en dirección a menos infinito.

B.2.6.3 Operadores relacionales

Los operadores relacionales en Lua son:

```
==    ~=    <     >     <=     >=
```

Estos operadores siempre tienen como resultado **false** o **true.**

La igualdad (==) primero compara el tipo de sus operandos. Si los tipos son diferentes, entonces el resultado es **false.** En caso contrario, los valores de los operandos son comparados. Números y cadenas de caracteres son comparados de manera usual. Objetos (valores del tipo table, userdata, thread y function) son comparados por referencia: Dos objetos son considerados iguales solamente si son el mismo objeto. Siempre que un nuevo objeto es creado (un valor con tipo table, userdata, thread o function) este nuevo objeto es diferente a cualquier otro objeto que existía anteriormente.

Es posible alterar la manera como Lua compara los tipos table y userdata a través del uso del meta-método "eq" (ver B.2.9).

Las reglas de conversión en B.2.3.2 no se aplican a comparaciones de igualdad. Por lo tanto, "0"==0 es evaluado como **false** y t[0] y t["0"] denotan posiciones diferentes en una tabla.

El operador ~= es exactamente la negación de la igualdad (==).

Los operadores de orden trabajan de la siguiente forma. Si ambos argumentos son números, entonces son comparados como tales. En caso contrario, si ambos argumentos son cadenas de caracteres, entonces sus valores son comparados de acuerdo con la elección de idioma actual. En caso contrario, Lua intenta llamar el meta-método "lt" o el meta-método "le" (ver B.2.9).

B.2.6.4 Operadores lógicos

Los operadores lógicos en Lua son **and, or** y **not.** Así como las estructuras de control (ver B.2.5.5), todos los operadores lógicos consideran **false** y **nil** como falso y cualquier cosa diferente como verdadero.

El operador de negación **not** siempre retorna **false** o **true.** El operador de conjunción **and** retorna su primer argumento si este valor es **false** o **nil;** en caso contrario, **and** retorna su segundo argumento. El operador de disyunción **or** retorna su primer argumento si el valor de este es diferente de **nil** y de **false;** en caso contrario, **or** retorna su segundo argumento. Tanto **and** como **or** usan evaluación de cortocircuito; es decir, el segundo operando es evaluado solamente cuando es necesario. Siguen algunos ejemplos:

```
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
```

En este Norma, --> indica el resultado de la expresión precedente.

B.2.6.5 Concatenación

El operador de concatenación de cadenas de caracteres en Lua es denotado por dos puntos ('..'). Si ambos operandos son cadenas de caracteres o números, entonces son convertidos en cadenas de caracteres de acuerdo con las reglas mencionadas en B.2.3.2. En caso contrario, el meta-método "concat" es llamado (ver B.2.9).

B.2.6.6 El operador de tamaño

El operador de tamaño es denotado por el operador unario #. El tamaño de una cadena de caracteres es su número de bytes (es decir, el significado usual de tamaño de una cadena cuando cada carácter ocupa un byte).

El tamaño de una tabla t es definido como cualquier índice entero n tal que t[n] no es **nil** y t[n+1] es **nil;** además de ello, si t[1] es **nil,** n puede ser cero. Para un array común, con todos los valores diferentes de **nil** yendo de 1 hasta un dato n, su tamaño es exactamente aquél n, el índice de su último valor. Si el array posee "agujeros" (es decir, valores **nil** entre otros dos valores diferentes de **nil),** entonces #t puede ser cualquiera de los índices que inmediatamente preceda un valor **nil** (es decir, puede considerar cualquier valor **nil** como el fin del array).

B.2.6.7 Precedencia

La precedencia de operadores en Lua sigue la tabla abajo, de la menor prioridad a la mayor:

```
or
and
<     >     <=    >=    ~=    ==
..
+     -
*     /     %
not   #     - (unary)
```

Como es usual, se pueden usar paréntesis para alterar las precedencias de una expresión. Los operadores de concatenación ('..') y de exponenciación ('^') son asociativos a la derecha. Todos los demás operadores binarios son asociativos a la izquierda.

B.2.6.8 Constructores de tablas

Constructores de tablas son expresiones que crean tablas. Siempre que un constructor es evaluado, es creada una nueva tabla. Constructores se pueden usar para crear tablas vacías o para crear una tabla e inicializar algunos de sus campos. La sintaxis general de constructores es

```
constructortabla ::= `{´ [listadecampos] `}´
listadecampos : := campo { separadordecampos campo} [ separadordecampos]
campo ::= `[´ exp `]´ `=´ exp | Nombre `=´ exp | exp
separadordecampos ::= `,´ | `;´
```

Cada campo de la forma [exp1] = exp2 agrega a la nueva tabla una entrada cuya clave es exp1 y cuyo valor es exp2. Un campo de la forma Nombre = exp es equivalente a ["Nombre"] = exp. Finalmente, campos de la forma exp son equivalentes a [i] = exp, donde i representa números enteros consecutivos, empezando con 1. Campos en los otros formatos no afectan este conteo. Por ejemplo,

```
a = { [ f(1)] = g; "x", "y"; x = 1, f(x), [30] = 23; 45 }
```

Es equivalente a

```
do
          local t = {}
          t[f(1)] = g
          t[1] = "x"         -- primera exp
          t[2] = "y"         -- segunda exp
          t.x = 1            -- t["x"] = 1
          t[3] = f(x)        -- tercera exp
          t[30] = 23
          t[4] = 45          -- cuarta exp
          a = t
        end
```

Si el último campo en la lista posee la forma exp y la expresión es una llamada de función o una expresión con un número variable de argumentos, entonces todos los valores retornados por la expresión entran en la lista consecutivamente (ver B.2.6.9). Para evitar esto, colocar paréntesis alrededor de la llamada de función (o de la expresión con número variable de argumentos) (ver B.2.6.1).

La lista de campos puede tener un separador más al final, como una conveniencia para código generado automáticamente.

B.2.6.9 Llamadas de función

Una llamada de función en Lua tiene la siguiente sintaxis:

```
llamadadefuncion ::= expprefijo args
```

En una llamada de función, el primer expprefixo y args son evaluados. Si el valor de expprefixo posee tipo *function,* entonces esta función es llamada con los argumentos suministrados. En caso contrario, el meta-método "call" de expprefixo es llamado, teniendo como primer parámetro el valor de expprefixo, seguido por los argumentos originales de la llamada (ver B.2.9).

La forma

```
llamadadefuncion ::= expprefijo `:´ Nombre args
```

se puede usar para llamar "métodos". Una llamada v:nombre(*args*) es un azúcar sintáctico para v.nombre(v,*args*), con la diferencia de que v es evaluado solamente una vez.

Argumentos poseen la siguiente sintaxis:

```
args ::= `(´ [ listaexp] `) ´

args ::= constructordetabla

args ::= Cadena
```

Todas las expresiones suministradas como argumento son evaluadas antes de la llamada. Una llamada de la forma f{*campos}* es un azúcar sintáctico para f({*campos});* es decir, la lista de argumentos consiste solamente en una tabla nueva. Una llamada de la forma f'*cadena'* (o f"*cadena"* o f[[*cadena]])* es un azúcar sintáctico para f('*cadena');* es decir, la lista de argumentos consiste solamente en una cadena de caracteres literal.

Una excepción con relación a la sintaxis de formato libre de Lua es que no es posible colocar una quiebra de línea antes del '(' en una llamada de función. Esta restricción evita algunas ambigüedades en el lenguaje. Si se escribiese

```
a = f

(g) .x(a)
```

Lua podría ver esto como un comando único, a = f(g).x(a). Por lo tanto, si se desean dos comandos, se debe obligatoriamente colocar un punto y coma entre ellos. Si realmente se desea llamar a f, se debe remover la quiebra de línea antes de (g).

Una llamada de la forma return *llamadadefuncion* se denomina llamada final. Lua implementa llamadas finales propias (o recursos finales propios): En una llamada final, la función llamada reutiliza la entrada en la pila de la función que la llamó. Por lo tanto, no hay límite en el número de llamadas finales anidadas que un programa puede ejecutar. Sin embargo, una llamada final apaga cualquier información de depuración sobre la función llamadora. Una llamada final solamente ocurre con una sintaxis particular, donde el **return** posee una única llamada de función como argumento; esta sintaxis hace que la llamada de función retorne exactamente los valores de retorno de la función llamada. De esa forma, ninguno de los ejemplos a continuación son llamadas finales:

```
return (f(_x))  -- el número de resultados es ajustado para 1
return 2 * f(x)
return x, f(x)  -- resultados adicionales
f(x); return        -- resultados desechados
return x or f(x)    -- el número de resultados es ajustado para 1
```

B.2.6.10 Definiciones de funciones

La sintaxis para la definición de una función es

```
función ::= function cuerpodelafuncion

función ::= `(´ [listapar] `)´ bloque end
```

El siguiente azúcar sintáctico simplifica definiciones de funciones:

```
comando ::= Function nombredelafuncion cuerpodelafuncion
comando ::= Local function Nombre cuerpodelafuncion
nombredelafuncion ::= Nombre {`.´ Nombre} [`:´ Nombre]
```

El comando

```
function f () body end
```

es traducido para

```
f = function () body end
```

El comando

```
function t.a.b.c.f () body end
```

es traducido para

```
t.a.b.c.f = function () body end
```

El comando

```
local function f () body end
```

es traducido para

```
local f; f = function () body end
```

y no para

```
local f = function () body end
```

Esto solamente hace diferencia cuando el cuerpo de la función contiene una referencia para f.

Una definición de función es una expresión ejecutable, cuyo valor tiene tipo *function.* Cuando Lua pre-compila un trecho, todos los cuerpos de las funciones del trecho son pre-compilados también. Entonces, siempre que Lua ejecuta la definición de una función, la función es instanciada (o cerrada). Esta instancia de la función (o cierre) es el valor final de la expresión. Instancias diferentes de la misma función pueden referirse a diferentes variables locales externas y pueden tener diferentes tablas de ambiente.

Parámetros se comportan como variables locales que son inicializadas con los valores de los argumentos:

```
listapar ::= listadenombres [`,´ `...´] | `...´
```

Cuando una función es llamada, la lista de argumentos es ajustada para el tamaño de la lista de parámetros, a no ser que la función sea de variedad variable o *vararg,* lo que se indica por tres puntos ('...') al final de su lista de parámetros. Una función vararg no ajusta su lista de argumentos; en vez de eso, recolecta todos los argumentos extras y los suministra a la función a través de una expresión *vararg, la cual* también es representada como tres puntos. El valor de esta expresión es una lista de todos los argumentos extras corrientes, análogo a una función con múltiples valores de retorno. Si una expresión vararg se utiliza dentro de otra expresión o en medio de una lista de expresiones, entonces su lista de valores de retorno es ajustada para un elemento. Si la expresión se utiliza como el último elemento de una lista de expresiones, entonces ningún ajuste se realiza (a menos que la llamada sea delimitada por paréntesis).

Como un ejemplo, considere las siguientes definiciones:

```
function  f(a b)  end
function  g(a b,  ...) end
function  r()   return 1,2,3
```

En este caso, se tiene el siguiente mapeo de argumentos para parámetros y para las expresiones vararg:

```
LLAMADA             PARÁMETROS

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 son retornados usando el comando **return** (ver B.2.5.5). Si el control alcanza el fin de una función sin encontrar un comando **return,** entonces la función retorna sin ningún resultado.

La sintaxis de dos puntos se utiliza para definir métodos, es decir, funciones que poseen un parámetro extra implícito self. De esta forma, el comando

```
function t.a.b.c:f (params) body end
```

Es un azúcar sintáctico para

```
t.a.b.c.f = function (self, params) body end
```

B.2.7 Reglas de visibilidad

Lua es un lenguaje con alcance léxico. El alcance de las variables empieza en el primer comando después de su declaración y va hasta el fin del bloque más interno que incluye la declaración. Considerar el siguiente ejemplo:

```
x = 10          -- variable global
do              -- bloque nuevo
   local x = x         -- nuevo 'x', con valor 10
   print(x)            --> 10
   x = x+1
   do               -- otro bloque
      local x = x+1    -- otro 'x'
      print(x)      --> 12
   end
   print(x)            --> 11
end
print(x)           --> 10 (o x global)
```

En una declaración como local x = x, la nueva x siendo declarada no está en el alcance aún y por lo tanto la segunda x se refiere a una variable externa.

Debido a las reglas de alcance léxico, variables locales pueden ser libremente accedidas por funciones definidas dentro de su alcance. Una variable local usada por una función más interna se denomina *upvalue* o variable local externa, dentro de la función más interna.

Cada ejecución de un comando **local** define noticias variables locales. Considerar el ejemplo a continuación:

```
a = {}
local x = 20
for i=1,10 do
  local y = 0
  a[ i] = function () y=y+1; return x+y end
end
```

El lazo crea diez cierres (es decir, diez instancias de la función anónima). Cada uno de estos cierres usa una variable y diferente, mientras todos ellos comparten la misma variable X.

B.2.8 Tratamiento de errores

Dado que Lua es un lenguaje incorporado de extensión, todas las acciones de Lua empiezan a partir de código C en el programa hospedero que llama una función de la biblioteca de Lua (ver *lua_pcall).* Siempre que un error ocurre durante la compilación o ejecución, el control retorna para C, que puede tomar las medidas apropiadas (tales como imprimir un mensaje de error).

El código Lua puede explícitamente generar un error a través de una llamada a la función *error.* Si es necesario capturar errores en Lua, se puede usar la función *pcall.*

B.2.9 Metatablas

Todo valor en Lua puede tener una meta-tabla. Ésta meta-tabla es una tabla Lua común que define el comportamiento del valor original con relación a ciertas operaciones especiales. Es posible alterar varios aspectos del comportamiento de operaciones sobre un valor especificando campos específicos en la meta-tabla del valor. Por ejemplo, cuando un valor no numérico es el operando de una adición, Lua verifica si existe una función asociada con el campo "__add" en la meta-tabla del valor. Si la función existe, Lua llama esta función para realizar la adición.

Las claves son llamadas en una meta-tabla de eventos y los valores de *meta-métodos.* En el ejemplo anterior, el evento es "add" y el meta-método es la función que realiza la adición.

Es posible obtener la meta-tabla de cualquier valor usando la función *getmetatable.*

Se puede alterar la meta-tabla de tablas a través de la función *setmetatable.* No se puede cambiar la meta-tabla de otros tipos de Lua (a menos que sea utilizada la biblioteca de depuración); para hacer esto se debe usar obligatoriamente la API C.

Tablas y objetos del tipo userdata completos poseen meta-tablas individuales (aunque múltiples tablas y objetos userdata puedan compartir sus meta-tablas); valores de todos los otros tipos comparten una única meta-tabla por tipo. Siendo así, hay solamente una meta-tabla para todos los números, una para todas las cadenas de caracteres, etc.

Una meta-tabla puede controlar cómo un objeto se comporta en operaciones aritméticas, comparaciones con relación a la orden, concatenación, operación de tamaño e indexación. Una meta-tabla también puede definir una función a ser llamada cuando un objeto userdata es recolectado por el colector de basura. Para cada una de estas operaciones Lua asocia una clave específica llamada un evento. Cuando Lua realiza una de estas operaciones sobre un valor, Lua verifica si este valor posee una meta-tabla con el evento correspondiente. Si éste es el caso, el valor asociado a esa clave (el meta-método) controla cómo Lua va a realizar la operación.

Metatablas controlan las operaciones listadas a continuación. Cada operación se identifica por su nombre correspondiente. La clave para cada operación es una cadena de caracteres empezando con el nombre de la operación siendo precedido por dos subrayados, '__'; por ejemplo, la clave para la operación "add" es la cadena "__add". La semántica de estas operaciones se explica mejor por medio de una función Lua que describe cómo el interpretador ejecuta la operación.

El código Lua mostrado en esta sección es meramente ilustrativo; el comportamiento real está codificado en el interpretador y es mucho más eficiente que esta simulación. Todas las funciones usadas en estas descripciones *(rawget, tonumber* etc.) son presentadas en B.5.2. En particular, para recobrar el meta-método de un objeto determinado, se usa la siguiente expresión:

```
metatable (obj)[ event]
```

Esto debe ser leído como

```
rawget(getmetatable(obj) or {} , event)
```

Es decir, el acceso a un meta-método no invoca otros meta-métodos y el acceso a objetos que no poseen metatablas no falla (simplemente da como resultado **nil).**

- **"add":** la operación +.

La función *getbinhandler* abajo define como Lua elige un tratador para una operación binaria. Primero, Lua intenta el primer operando. Si este tipo no define un tratador para la operación, entonces Lua intenta el segundo operando.

```
function getbinhandler (op1, op2, event)
    return metatable(op1)[ event] or metatable(op2)[ event]
end
```

Usando esa función, el comportamiento del op1 + op2 es

```
function add_event (op1, op2)
  local o1, o2 = tonumber(op1), tonumber(op2)
  if o1 and o2 then      -- ¿ambos operandos son numéricos?
    return o1 + o2       -- `+' aquí está el `add' primigenio
  else                -- por lo menos uno de los operandos no es numérico
    local h = getbinhandler(op1, op2, "__add")
    if h then
      -- llama al tratador pasando ambos operandos
      return h(op1, op2)
    else          -- sin tratador disponible: comportamiento estándar
      error("...")
    end
  end
end
```

- **sub":** la operación -. Comportamiento análogo al de la operación “add”.
- **"mul":** la operación *. Comportamiento análogo al de la operación “add”.
- **"div":** la operación /. Comportamiento análogo al de la operación “add”.
- **"mod":** la operación %. Comportamiento análogo a la operación “add”, con la operación o1 - floor(o1/o2)*o2 como operación primitiva.
- **"pow":** la operación ^ (exponenciación). Comportamiento análogo al de la operación “add”, con la función pow (proveniente de la biblioteca matemática del C) como operación primitiva.
- **"unm":** la operación unaria -.

```
function unm_event (op)
  local o = tonumber(op)
  if el then          -- ¿operando es numérico?
    return –o           -- `'aquí es el `unm' primigenio
  else           -- el operando no es numérico
    -- intenta encontrar un tratador para el operando local h = metatable(op).__unm
  if h then
      -- llama el tratador pasando el operando
      return h(op)
  else     -- sin tratador disponible: comportamiento estándar
      error("...")
  end
 end
end
```

- **"concat":** la operación .. (concatenación).

```
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 -- concatenacion de strings primitiva
   else
    local h = getbinhandler(op1, op2, "__concat")
    if h then
      return h(op1, op2)
    else
      error("...")
    end
   end
end
```

- **"len":** la operación #.

```
function len_event (op)
 if type(op) == "string" then
    return strlen(op)         -- tamaño de string primitivo
 elseif type(op) == "table" then
    return #op             -- tamaño de tabla primitivo
 else
    local h = metatable(op).__len
    if h then
      -- llama el tratador pasando el operando
      return h(op)
    else -- sin tratador disponible: Comportamiento
estándar
        error("...")
    end
   end
end
```

Ver B.2.6.6 para obtener una descripción de la longitud de una tabla.

- **"eq":** la operación ==. La función *getcomphandler* define como Lua elige un meta-método para comparación de operadores. Un meta-método solo es seleccionado cuando ambos objetos en comparación tienen el mismo tipo y el mismo meta-método para la operación seleccionada.

```
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
```

El evento “eq” se define 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 es equivalente a not (a = = b).

- **"lt":** la operación <.

```
function lt_event (op1, op2)
 if type(op1) == "number" and type(op2) == "number" then
  return op1 < op2 -- comparación numérica
 elseif type(op1)=="string" and type(op2)=="string" then
  return op1 < op2 -- comparación lexicográfica
 else
  local h = getcomphandler(op1, op2, "__lt")
  if h then
   return h(op1, op2)
  else
   error("...");
  end
 end
end
```

a > b es equivalente a b < a.

- **"le":** la operación <=.

```
function le_event (op1, op2)
    if type(op1) == "number" and type(op2) == "number" then
      return op1 <= op2   -- comparación numérica
    elseif type(op1)=="string" and type(op2)=="string" then
      return op1 <= op2   -- comparación 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 es equivalente a b <= a. En la ausencia de un meta-método “le”, Lua intenta el “lt”, asumiendo que a <= b es equivalente a not (b < a).

- **"index":** acceso de indexación *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)         -- llama el tratador
 else return h[key]            -- o repita la operación
 end
end
```

- **"newindex":** atribución 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)             -- llama el tratador
  else h[key] = value                      -- o repita la operación
  end
end
```

- **"call":** llamado cuando Lua llama un valor.

```
function function_event (func, ...)
  if type(func) == "function" then
    return func(...) -- llamada primitiva
  else
    local h = metatable(func).__call
    if h then
      return h(func, ...)
    else
      error("...")
    end
  end
end
```

B.2.10 Ambientes

Además de meta-tablas, objetos del tipo thread, function y userdata poseen otra tabla asociada a ellos, denominada su ambiente. Así como meta-tablas, ambientes son tablas normales y varios objetos pueden compartir el mismo ambiente.

Ambientes asociados con objetos del tipo userdata no poseen significado para Lua. Es sólo una conveniencia para programadores asociar una tabla a un objeto userdata.

Ambientes asociados con flujos de ejecución (threads) son llamados ambientes globales. Se usan como el ambiente estándar por sus flujos de ejecución y funciones no anidadas creadas por el flujo de ejecución (a través de *loadfile, loadstring* o *load)* y pueden ser directamente accedidos por el código C (ver B.3.4).

Ambientes asociados con funciones C pueden ser directamente accedidos por el código C (ver B.3.4). Se usan como el ambiente estándar para otras funciones C creadas por la función.

Ambientes asociados con funciones Lua se usan para resolver todos los accesos a variables globales dentro de la función (ver B.2.4). Se usan como el ambiente estándar para otras funciones Lua creadas por la función.

Es posible alterar el ambiente de una función Lua o del flujo de ejecución que está siendo ejecutado actualmente llamando a *setfenv.* Es posible obtener el ambiente de una función Lua o del flujo de ejecución siendo ejecutado actualmente llamando a *getfenv.* Para tratar el ambiente de otros objetos (userdata, funciones C, otros flujos de ejecución) se debe usar obligatoriamente la API C.

B.2.11 Recolecta de basura

B.2.11.1 Conceptos básicos

Lua realiza gestión automática de la memoria. Esto significa que no es necesario preocuparse con la asignación de memoria para nuevos objetos ni con la liberación de memoria cuando los objetos ya no son necesarios. Lua administra la memoria automáticamente ejecutando un colector de basura de vez en cuando para recolectar todos los objetos muertos (es decir, aquellos objetos que han dejado de ser accesibles desde Lua). Todos los objetos en Lua están sujetos a la gestión automática de memoria: tablas, userdata, funciones, flujos de ejecución y cadenas de caracteres.

Lua implementa un colector de basura “marcar y limpiar” *(Mark-and-sweep)* incremental. El colector usa dos números para controlar su ciclo de Recolecta de basura: La *pausa del colector de basura* y el multiplicador de paso del colector de basura.

La pausa del colector de basura controla cuánto tiempo el colector espera antes de iniciar un nuevo ciclo. Los valores mayores hacen que el colector sea menos agresivo. Los valores menores que 1 significan que el colector no esperará para iniciar un nuevo ciclo. Un valor de 2 significa que el colector esperará hasta que la memoria total en uso se duplique antes de iniciar un nuevo ciclo.

El multiplicador de paso controla la velocidad relativa del colector con relación a la asignación de memoria. Los valores mayores tornan al colector más agresivo pero también aumentan el tamaño de cada paso incremental. Los valores menores que 1 hacen que el colector sea muy lento y puede ocurrir que el colector nunca termine un ciclo. El valor estándar, 2, significa que el colector se ejecuta a una velocidad que es "dos veces" la velocidad de asignación de memoria.

Es posible alterar estos números a través de llamadas a las funciones *lua_gc* en C o *collectgarbage* en Lua. Ambas reciben valores en puntos porcentuales como argumentos (de modo que un argumento cuyo valor es 100 significa un valor real de 1). Con estas funciones también se puede controlar el colector directamente (por ejemplo, pararlo y reiniciarlo).

B.2.11.2 Meta-métodos de Recolecta de basura

Usando la API C, se puede configurar los meta-métodos del colector de basura para objetos userdata (ver B.2.9). Éstos meta-métodos también se denominan finalizadores. Los finalizadores permiten que se coordine la Recolecta de basura de Lua con la gestión de recursos externos (tales como el cierre de archivos, conexiones de red o de bancos de datos o la liberación de su propia memoria).

Los objetos userdata con un campo __gc en su meta-tablas no son colectados inmediatamente por el colector de basura. En vez de eso, Lua los coloca en esa lista. Después que la recolecta se realiza, Lua hace el equivalente de la siguiente función para cada objeto userdata en una lista:

```
function gc_event (userdata)
   local h = metatable(userdata) .__gc
   if h then
      h (userdata)
   end
end
```

Al final del ciclo de recolecta de basura, los finalizadores para los objetos userdata son llamados en el orden inverso al de su creación, entre aquellos colectados en ese ciclo. Es decir, el primer finalizador a ser llamado es asociado con el objeto userdata que fue creado por último en el programa. El userdata solo es efectivamente liberado en el próximo ciclo de Recolecta de basura.

B.2.11.3 Tablas débiles

Una tabla débil es una tabla cuyos elementos son informes débiles. Una referencia débil es ignorada por el colector de basura. En otras palabras, si los únicos informes para un objeto son informes débiles, entonces el colector de basura colectará este objeto.

Una tabla débil puede tener claves débiles, valores débiles o ambos. Una tabla con claves débiles permite la recolecta de sus claves pero impide la recolecta de sus valores. Una tabla con claves débiles y valores débiles permite la recolecta tanto de las claves como de los valores. En cualquier caso, si la clave es colectada o el valor es colectado, el par entero es borrado de la tabla. La fragilidad de una tabla es controlada por el campo __mode de su meta-tabla. Si el campo __mode es una cadena de caracteres conteniendo el carácter 'k', las claves de la tabla son débiles. Si __mode contiene 'v', los valores en la tabla son débiles.

Después de usar una tabla como una meta-tabla, no se debe alterar el valor de su campo __mode. En caso contrario, el comportamiento débil de las tablas controladas por ésta meta-tabla es indefinido.

B.2.12 Co-rutinas

Lua ofrece soporte a co-rutinas, también conocidas como flujos de ejecución *(threads)* colaborativos. Una co-rutina en Lua representa un flujo de ejecución independiente. Al contrario de procesos ligeros en sistemas que dan soporte a múltiples flujos de ejecución, una co-rutina solamente suspende su ejecución a través de una llamada explícita a una función de cesión.

Es posible crear una co-rutina con una llamada a la *coroutine.create.* Su único argumento es una función que es la función principal de la co-rutina. La función create solamente crea una nueva co-rutina y retorna una referencia para la misma (un objeto del tipo *thread);* no inicia la ejecución de la co-rutina.

Cuando la función *coroutine.resume* es llamada por primera vez, recibiendo como su primer argumento el objeto del tipo thread retornado por *coroutine.create, la co-rutina* inicia su ejecución, en la primera línea de su función principal. Después que la co-rutina empieza a ser ejecutada, continúa ejecutando hasta terminar o ceder.

Una función puede terminar su ejecución de dos formas: Normalmente, cuando su función principal retorna (explícita o implícitamente, después de la última instrucción); y de manera anormal, si ocurre un error no protegido en el primer caso, *coroutine.resume* retorna **true** más cualquier valor retornado por la función principal de la co-rutina. En el caso de ocurrir errores, *coroutine.resume* retorna **false** más un mensaje de error.

Una co-rutina cede la ejecución a través de una llamada a la función *coroutine.yield.* Cuando una co-rutina cede, la *coroutine.resume* correspondiente retorna inmediatamente, incluso si la cesión ocurrió dentro de una llamada de función anidada (es decir, no ocurrió dentro de la función principal, sino en una función llamada directa o indirectamente por la función principal). En el caso de una cesión, *coroutine.resume* también retorna **true,** más cualquier valor pasado para *coroutine.yield.* La próxima vez que se reinicie la ejecución de la misma co-rutina, continúa su ejecución desde el punto donde cedió, con la llamada para *coroutine.yield* retornando cualesquiera argumentos extras pasados para *coroutine.resume.*

Como *coroutine.create,* la función *coroutine.wrap* también crea una co-rutina, pero en vez de retornar la propia co-rutina, retorna una función que, cuando es llamada, retoma la ejecución de la co-rutina. Cualesquiera argumentos pasados para esta función van como argumentos extras para *coroutine.resume. Coroutine.wrap* retorna todos los valores retornados por *coroutine.resume,* excepto el primero (el código booleano de error). Diferentemente de *coroutine.resume, coroutine.wrap* no captura errores; cualquier error es propagado para el llamador.

Como un ejemplo, considerar el siguiente 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"))
```

Cuando se ejecuta este código, produce el siguiente 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 Interfaz de programación de la aplicación (API)

B.3.1 Conceptos básicos

Todas las funciones de la API, así como los tipos y constantes relacionados, están declarados en el archivo de encabezamiento lua.h.

Incluso cuando se usa el término "función", cualquier operación en la API puede, de forma alternativa, ser provista como una macro. Tales macros usan cada uno de sus argumentos exactamente una vez (con excepción del primer argumento, que es siempre un estado Lua) y por lo tanto no generan cualquier efecto colateral oculto.

Como en la mayoría de las bibliotecas C, las funciones de la API Lua no verifican la validez o la consistencia de sus argumentos. Sin embargo, es posible alterar este comportamiento compilando Lua con una definición apropiada para la macro luai_apicheck, en el archivo luaconf.h.

B.3.2 Pila

Lua usa una *pila virtual* para pasar y recibir valores de C. Cada elemento en esta pila representa un valor Lua **(nil,** un número, una cadena de caracteres etc.).

Siempre que Lua llama C, la función llamada recibe una nueva pila, que es independiente de pilas anteriores y de pilas de funciones C que aún estén activas. Esta pila contiene inicialmente cualesquiera argumentos para la función C y es donde la función C apila sus resultados para ser retornados al llamador (ver *lua_CFunction).*

Por conveniencia, la mayoría de las operaciones de consulta en la API no sigue una disciplina estricta de pila. En vez de eso, pueden referirse a cualquier elemento en la pila usando un índice. Un índice positivo representa una posición *absoluta* en la pila (empezando por 1); un índice negativo representa una posición relativa a la parte superior de la pila. De manera más específica, si la pila posee *n* elementos, entonces el índice 1 representa el primer elemento (es decir, el elemento que fue apilado en la pila primero) y el índice *n* representa el último elemento; el índice -1 también representa el último elemento (es decir, el elemento en la parte superior) y el índice *-n* representa el primer elemento. Se dice que un índice es *válido* si está entre 1 y la parte superior de la pila (es decir, si 1 ~ abs(índice) ~ tope).

B.3.3 Tamaño de la pila

Cuando se interactúa con la API de Lua, se es responsable por asegurar consistencia. En particular, se es responsable por controlar un reventón de la pila. Se puede usar la función *lua_checkstack* para aumentar el tamaño de la pila.

Siempre que Lua llama a C, asegura que por lo menos LUA_MINSTACK posiciones en la pila están disponibles. LUA_MINSTACK es definida como 20, entonces generalmente no es necesario preocuparse con el espacio de la pila a menos que su código posea lazos apilando elementos en la pila.

La mayoría de las funciones de consulta acepta como índices cualquier valor dentro del espacio de la pila disponible, es decir, índices hasta el tamaño máximo de la pila que se configuró a través de la función *lua_checkstack*. Tales índices son llamados índices aceptables. Más formalmente, definido un índice aceptable de la siguiente forma:

```
(índice < 0 && abs(índice) <= tope) ||

(índice > 0 && índice <= espaciodelapila)
```

Observar que 0 nunca es un índice aceptable.

B.3.4 Pseudo-índices

A menos que se diga lo contrario, cualquier función que acepta índices válidos también puede ser llamada con *pseudo-índices,* que representan algunos valores Lua que son accesibles para el código C pero que no están en la pila. Pseudo-índices se usan para acceder al ambiente del flujo de ejecución, el ambiente de la función, el registro y los upvalues de la función C (ver B.3.5).

El ambiente del flujo de ejecución (donde las variables globales existen) está siempre en el pseudo-índice LUA_GLOBALSINDEX. El ambiente de la función C funcionando está siempre en el pseudo-índice LUA_ENVIRONINDEX.

Para acceder y alterar el valor de variables globales, se puede usar operaciones de tablas usuales sobre una tabla de ambiente. Por ejemplo, para acceder al valor de una variable global, hacer

```
lua_getfield(L, LUA_GLOBALSINDEX, varname);
```

B.3.5 Cierres C

Cuando una función C es creada, es posible asociar algunos valores a la misma, creando entonces un cierre C; estos valores son llamados *up values* y son accesibles para la función siempre que es llamada (ver *lua_pushcclosure).*

Siempre que una función C es llamada, sus upvalues son posicionados en pseudo-índices específicos. Estos pseudo-índices son generados por la macro lua_upvalueindex. El primer valor asociado con una función está en la posición lua_upvalueindex(1), y así sucesivamente. Cualquier acceso a lua_upvalueindex(*n*), donde *n* es mayor que el número de upvalues de la función actual, produce un índice aceptable (aunque inválido).

B.3.6 Registro

Lua provee un registro, una tabla predefinida que puede ser usada por cualquier código C para almacenar cualquier valor Lua que el código C necesite almacenar. Esta tabla está siempre localizada en el pseudo-índice LUA_REGISTRYINDEX. Cualquier biblioteca de C puede almacenar datos en esta tabla, pero debe tener cuidado para elegir claves diferentes de aquellas usadas por otras bibliotecas, para evitar colisiones. Típicamente, se debe usar como clave una cadena de caracteres conteniendo el nombre de su biblioteca o un objeto del tipo userdata ligero con la dirección de un objeto C en su código.

Las claves enteras en el registro son usadas por el mecanismo de referencia, implementado por la biblioteca auxiliar, y por lo tanto no deben ser usadas para otros propósitos.

B.3.7 Tratamiento de errores en C

Internamente, Lua usa el mecanismo de longjmp de C para tratar errores (se puede también utilizar excepciones si se utiliza; ver el archivo luaconf.h.) Cuando Lua encuentra cualquier error (tales como errores de asignación de memoria, errores de tipo, errores de sintaxis y errores de tiempo de ejecución) dispara un error; es decir, hace un desvío largo. Un ambiente protegido usa setjmp para establecer un punto de recuperación; cualquier error desvía el flujo de ejecución para el punto de recuperación activado más reciente.

La mayoría de las funciones en la API puede disparar un error, por ejemplo debido a un error de asignación de memoria. La documentación para cada función indica si puede disparar errores.

Dentro de una función C se puede disparar un error llamando *lua_error.*

B.3.8 Funciones y tipos

Todas las funciones y tipos de la API C se listan a continuación en orden alfabético. Cada función tiene un indicador como éste: [-o, +p, x]

El primer campo, o, representa cuántos elementos la función desapila de la pila. El segundo campo, p, indica cuántos elementos la función apila en la pila (cualquier función siempre apila sus resultados después de desapilar sus argumentos). Un campo en la forma x|y significa que la función puede apilar (o desapilar) x o y elementos, dependiendo de la situación; una marca de interrogación '?' significa que no se puede saber cuántos elementos desapila/apila la función mirando solamente sus argumentos (por ejemplo, el número de elementos puede depender de lo que está en la pila). El tercer campo, x, dice si la función puede disparar errores: '-' Significa que la función nunca dispara cualquier error; 'm' significa que la función puede disparar un error solamente debido a la falta de memoria; 'y' significa que la función puede disparar otros tipos de error; 'v' significa que la función puede disparar un error intencional.

---

lua_Alloc

```
typedef void * (*lua_Alloc) (void *ud, void *ptr, size_t osize, size_t nsize);
```

El tipo de la función de asignación de memoria usada por los estados Lua. La función de asignación debe proveer una funcionalidad análoga a la de realloc, pero no exactamente la misma. Sus argumentos son ud, un indicador opaco pasado para lua_newstate; ptr, un indicador para el bloque siendo asignado/reasignado/liberado; osize, el tamaño original del bloque; y nsize, el nuevo tamaño del bloque. Ptr es NULL si, y solamente si, osize es cero. Cuando nsize es cero, la función de asignación debe retornar NULL; si osize es diferente de cero, el bloque de memoria indicado por ptr debe ser liberado. Cuando nsize no es cero, la función de asignación retorna NULL si, y solamente si, no puede asignar el tamaño del bloque requerido. Cuando nsize no es cero y osize es cero, la función de asignación debe comportarse como malloc. Cuando nsize y osize no son cero, la función de asignación se comporta como realloc. Lua presupone que la función de asignación nunca falla cuando osize >= nsize.

Sigue una implementación simple para la función de asignación. Se utiliza en la 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 presupone que free(NULL) no posee ningún efecto y que realloc(NULL, size) es equivalente a malloc(size). ANSI C garantiza esos dos comportamientos.

lua_atpanic

```
lua_CFunction lua_atpanic (lua_State *L, lua_CFunction panicf);
```

Establece una nueva función de pánico y retorna la función de pánico antigua.

Si un error ocurre fuera de cualquier ambiente protegido, Lua llama una función de pánico y entonces llama exit(EXIT_FAILURE), terminando entonces la aplicación hospedera. Su función de pánico puede evitar esta salida en el caso que nunca retorne (por ejemplo, haciendo un desvío largo).

La función de pánico puede acceder al mensaje de error en la parte superior de la pila.

lua_call

```
void lua_call (lua_State *L, int nargs, int nresults);
```

Llama una función.

Para llamar una función se debe usar el siguiente protocolo: Primero, la función a ser llamada es apilada en la pila; a continuación, los argumentos de la función son apilados en orden directo; es decir, el primer argumento es apilado primero. Por último se llama *lua_call;* nargs es el número de argumentos que se amontonó en la pila. Todos los argumentos y el valor de la función son desapilados de la pila cuando la función es llamada. Los resultados de la función son apilados en la pila cuando la función retorna. El número de resultados es ajustado para nresults, a menos que nresults sea LUA_MULTRET. En este caso, *todos los* resultados de la función son apilados. Lua cuida para que los valores retornados quepan dentro del espacio de la pila. Los resultados de la función son apilados en la pila en orden directo (el primer resultado es apilado primero), de modo que después de la llamada el último resultado está en la parte superior de la pila.

Cualquier error dentro de la función llamada es propagado hacia arriba (con un longjmp).

El siguiente ejemplo muestra como el programa hospedero puede hacer el equivalente a este código Lua:

```
a = f("how", t.x, 14)
```

Sigue el mismo código en C:

```
lua_getfield(L, LUA_GLOBALSINDEX, "f");    /* función a ser llamada   */
lua_pushstring(L, "how");                    /* primer argumento     */
lua_getfield(L, LUA_GLOBALSINDEX, "t");   /* tabla a ser indexada     */
lua_getfield(L, -1, "x");   /* apila el resultado de t.x (2° arg)     */
lua_remove(L, -2);                          /* remueve 't' de la pila */
lua_pushinteger(L, 14);                              /* 3er. argumento  */
lua_call(L, 3, 1);     /* llama 'f' con 3 argumentos y 1 resultado    */
lua_setfield(L, LUA_GLOBALSINDEX, "a");   /* establece ‘a’ global     */
```

El código anterior es "equilibrado": al final, la pila está de regreso a su configuración original. Esto se considera una buena práctica de programación.

lua_C Function

```
typedef int (*lua_CFunction) (lua_State *L);
```

El tipo para funciones C.

Con el objeto de comunicarse apropiadamente con Lua, una función C debe usar el siguiente protocolo, el cual define el modo cómo se pasan los parámetros y resultados: Una función C recibe sus argumentos de Lua en su pila en orden directo (el primer argumento es apilado primero). Por lo tanto, cuando la función empieza, lua_gettop(L) retorna el número de argumentos recibidos por la función. El primer argumento (si hay) está en el índice 1 y su último argumento está en el índice lua_gettop(L). Para retornar valores para Lua, una función C sólo los apila en la pila, en orden directo (el primer resultado se apila primero) y retorna el número de resultados. Cualquier otro valor en la pila debajo de los resultados será debidamente desechado por Lua. Como una función Lua, una función C llamada por Lua también puede retornar muchos resultados.

Como un ejemplo, la siguiente función recibe un número variable de argumentos numéricos y retorna el promedio y la suma de ellos:

```
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);
```

Garantiza que existen por lo menos posiciones extra disponibles en la pila. La función retorna falso si no puede aumentar el tamaño de la pila para el tamaño deseado. Esta función nunca comprime la pila; si la pila ya es mayor que el nuevo tamaño, no tendrá su tamaño modificado.

lua_close

```
void lua_close (lua_State *L);
```

Destruye todos los objetos en el estado Lua suministrado (llamando los meta-métodos de recolecta de basura correspondientes, en caso de haberlos) y libera toda la memoria dinámica usada por ese estado. En varias plataformas, puede no ser necesario llamar esta función, porque todos los recursos son naturalmente liberados cuando el programa hospedero muere. Por otro lado, programas que se quedan funcionando por mucho tiempo, como un *daemon* o un servidor Web, pueden necesitar liberar estados tan pronto no sean necesarios, para evitar un crecimiento excesivo del uso de la memoria.

lua_concat

```
void lua_concat (lua_State *L, int n);
```

Concatena los n valores en la parte superior de la pila, los desapila y deja el resultado en la parte superior de la pila. Si n es 1, el resultado es el único valor en la pila (es decir, la función no hace nada); si n es 0, el resultado es la cadena de caracteres vacía. La concatenación se realiza de acuerdo con la semántica usual de Lua (ver B.2.6.5).

lua_cpcall

```
int lua_cpcall (lua_State *L, lua_CFunction func, void *ud);
```

Llama la función C `func` en modo protegido. `func` empieza solamente con un único elemento en su pila, el objeto userdata ligero conteniendo `ud`. En caso de errores, *`lua_cpcall`* retorna el mismo código de error de *`lua_pcall,`* más el objeto de error en la parte superior de la pila; en caso contrario, retorna cero y no cambia la pila. Todos los valores retornados por `func` se descartan.

lua_createtable

```
void lua_createtable (lua_State *L, int narr, int nrec);
```

Crea una nueva tabla vacía y la apila en la parte superior de la pila. La nueva tabla posee espacio pre-asignado para `narr` elementos array y `nrec` elementos no array. Esta pre-asignación es útil cuando se sabe exactamente cuántos elementos va a tener la tabla. En caso contrario se puede usar la función *`lua_newtable.`*

lua_dump

```
int lua_dump (lua_State *L, lua_Writer writer, void *data);
```

Descarga una función como un trecho de código binario. Recibe una función Lua en la parte superior de la pila y produce un trecho de código binario que, si se carga nuevamente, da como resultado una función equivalente a aquella que fue descargada. Para producir partes del trecho de código, *lua_dump* llama la función writer (ver *lua_Writer)* con el argumento fecha suministrado para escribirlos.

El valor retornado es el código de error retornado por la última llamada a la función writer; 0 significa que no ocurrieron errores.

Esta función no desapila la función Lua de la pila.

lua_equal

```
int lua_equal (lua_State *L, int index1, int index2);
```

Retorna 1 si los dos valores en los índices aceptables index1 e index2 son iguales, siguiendo la semántica del operador == de Lua (es decir, puede llamar meta-métodos). En caso contrario retorna 0. También retorna 0 si cualquiera de los índices no es válido.

lua_error

```
int lua_error (lua_State *L);
```

Genera un error Lua. El mensaje de error (que puede ser de hecho un valor Lua de cualquier tipo) debe estar en la parte superior de la pila. Esta función hace un desvío largo y por lo tanto nunca retorna. (ver *luaL_error).*

lua_gc

```
int lua_gc (lua_State *L, int what, int data);
```

Controla el colector de basura

Esa función ejecuta varias tareas, de acuerdo con el valor del parámetro *what:*

- **LUA_GCSTOP:** interrumpe el colector de basura.
- **LUA_GCRESTART:** reinicia el colector de basura.
- **LUA_GCCOLLECT:** realiza un ciclo completo de Recolecta de basura.
- **LUA_GCCOUNT:** retorna la cantidad de memoria (en Kbytes) que está siendo usada habitualmente por Lua.
- **LUA_GCCOUNTB:** retorna el resto de la división de la cantidad de bytes de memoria usada habitualmente por Lua por 1024.
- **LUA_GCSTEP:** realiza un paso incremental de recolecta de basura. El "tamaño" del paso es controlado por `data` (valores mayores significan más pasos) de manera no especificada. Si se desea controlar el tamaño del paso, se debe ajustar de manera experimental el valor de `data`. La función retorna 1 si el paso finalizó un ciclo de recolecta de basura.
  - **LUA_GCSETPAUSE:** establece `data`/100 como el nuevo valor para la *pausa* del colector (ver B.2.11). La función retorna el valor anterior de la pausa.
  - **LUA_GCSETSTEPMUL:** establece `data`/100 como el nuevo valor para el multiplicador de paso del colector (ver B.2.11). La función retorna el valor anterior del multiplicador de paso.

lua_getallocf

```
lua_Alloc lua_getallocf (lua_State *L, void **ud);
```

Retorna la función de asignación de memoria de un dato estado. Si ud no es NULL, Lua almacena en *ud el indicador opaco pasado para *lua_newstate.*

lua_getfenv

```
void lua_getfenv (lua_State *L, int index);
```

Coloca en la pila la tabla de ambiente del valor en el índice suministrado.

lua_getfield

```
void lua_getfield (lua_State *L, int index, const char *k);
```

Coloca en la pila el valor `t[k]`, donde `t` es el valor en el índice válido suministrado. Como en Lua, esta función puede disparar un meta-método para el evento "index" (ver B.2.9).

lua_getglobal

```
void lua_getglobal (lua_State *L, const char *name);
```

Coloca en la pila el valor de la global name. Esta función es definida como una macro:

```
#define lua_getglobal(L, s) lua_getfield(L, LUA_GLOBALSINDEX, s)
```

lua_getmetatable

```
int lua_getmetatable (lua_State *L, int index);
```

Coloca en la pila la meta-tabla del valor en el índice aceptable suministrado. Si el índice no es válido o si el valor no posee una meta-tabla, la función retorna 0 y no coloca nada en la pila.

lua_gettable

```
void lua_gettable (lua_State *L, int index);
```

Coloca en la pila el valor t[k], donde t es el valor en el índice válido suministrado y k es el valor en la parte superior de la pila.

Esta función desapila la clave 'k' (colocando el resultado en su lugar). Como en Lua, esta función puede disparar un meta-método para el evento "index" (ver B.2.9).

lua_gettop

```
int lua_gettop (lua_State *L);
```

Retorna el índice del elemento en la parte superior de la pila. Visto que los índices empiezan en 1, este resultado es igual al número de elementos en la pila (y por lo tanto 0 significa una pila vacía).

lua_insert

```
void lua_insert (lua_State *L, int index);
```

Mueve el elemento en la parte superior para el índice válido suministrado, desplazando los elementos arriba de este índice para abrir espacio. Esta función no puede ser llamada con un pseudo-índice, porque un pseudo-índice no es una posición real de la pila.

lua_Integer

```
typedef ptrdiff_t lua_Integer;
```

El tipo usado por la API Lua para representar valores enteros.

El tipo estándar es un ptrdiff_t, que es comúnmente el mayor tipo entero con señal que la máquina maneja "cómodamente".

lua_isboolean

```
int lua_isboolean (lua_State *L, int index);
```

Retorna 1 si el valor en el índice aceptable suministrado posee tipo booleano y 0 en caso contrario.

lua_iscfunction

```
int lua_iscfunction (lua_State *L, int index);
```

Retorna 1 si el valor en el índice aceptable suministrado es una función C y 0 en caso contrario.

lua_isfunction

```
int lua_isfunction (lua_State *L, int index);
```

Retorna 1 si el valor en el índice aceptable suministrado es una función (C o Lua) y 0 en caso contrario.

lua_isl ig htuserdata

```
int lua_islightuserdata (lua_State *L, int index);
```

Retorna 1 si el valor en el índice aceptable suministrado es un objeto userdata ligero y 0 en caso contrario.

lua_isnil

```
int lua_isnil (lua_State *L, int index);
```

Retorna 1 si el valor en el índice aceptable suministrado es **nil** y 0 en caso contrario.

lua_isnone

```
int lua_isnone (lua_State *L, int index);
```

Retorna 1 si el índice aceptable suministrado no es válido (es decir, si se refiere a un elemento fuera del espacio de la pila corriente) y 0 en caso contrario.

lua_isnoneornil

```
int lua_isnoneornil (lua_State *L, int index);
```

Retorna 1 si el índice aceptable suministrado no es válido (es decir, si se refiere a un elemento fuera del espacio de la pila corriente) o si el valor en este índice es **nil** y 0 en caso contrario.

lua_isnumber

```
int lua_isnumber (lua_State *L, int index);
```

Retorna 1 si el valor en el índice aceptable suministrado es un número o una cadena de caracteres que puede ser convertida para un número y 0 en caso contrario.

lua_isstring

```
int lua_isstring (lua_State *L, int index);
```

Retorna 1 si el valor en el índice aceptable suministrado es una cadena de caracteres o un número (lo cual siempre puede ser convertido para una cadena) y 0 en caso contrario.

lua_istable

```
int lua_istable (lua_State *L, int index);
```

Retorna 1 si el valor en el índice aceptable suministrado es una tabla y 0 en caso contrario.

lua_isthread

```
int lua_isthread (lua_State *L, int index);
```

Retorna 1 si el valor en el índice aceptable suministrado es del tipo thread y 0 en caso contrario.

lua_isuserdata

```
int lua_isuserdata (lua_State *L, int index);
```

Retorna 1 si el valor en el índice aceptable suministrado es un objeto userdata (completo o ligero) y 0 en caso contrario.

lua_lessthan

```
int lua_lessthan (lua_State *L, int index1, int index2);
```

Retorna 1 si el valor en el índice aceptable index1 es menor que el valor en el índice aceptable index2, siguiendo la semántica del operador < de Lua (es decir, puede llamar meta-métodos). En caso contrario retorna 0. También retorna 0 si cualquiera de los índices no es válido.

lua_load

```
int lua_load (lua_State *L, lua_Reader reader, void *data, const char *chunkname);
```

Carga un trecho de código Lua. Si no ocurre ningún error, *lua_load* apila el trecho compilado como una función Lua en la parte superior de la pila. En caso contrario, apila un mensaje de error. Los valores de retorno de *lua_load* son:

- 0 --- sin errores;
- LUA_ERRSYNTAX --- error de sintaxis durante la pre-compilación.
- LUA_ERRMEM --- error de asignación de memoria.

Esta función solamente carga un trecho; no lo ejecuta.

Lua_load automáticamente detecta si el trecho está en forma de texto o en la forma binaria y lo carga de manera correcta (ver el programa luac).

La función *lua_load* usa una función reader suministrada por el usuario para leer el trecho de código (ver *lua_Reader).* El argumento fecha es un valor opaco pasado para la función de lectura.

El argumento chunkname da un nombre al trecho, el cual se utiliza para mensajes de error y en informaciones de depuración (ver B.3.9).

---

lua_newstate

```
lua_State *lua_newstate (lua_Alloc f, void *ud);
```

Crea un estado nuevo independiente. Retorna NULL si no puede crear el estado (debido a la falta de memoria). El argumento f es la función de asignación; Lua hace toda la asignación de memoria para este estado a través de esta función. El segundo argumento, ud, es un indicador opaco que Lua simplemente pasa para la función de asignación a cada llamada.

---

lua_newtable

```
void lua_newtable (lua_State *L);
```

Crea una nueva tabla vacía y la coloca en la pila. Es equivalente a lua_createtable(L, 0, 0).

---

lua_newthread

```
lua_State *lua_newthread (lua_State *L);
```

Crea un nuevo objeto del tipo thread, lo coloca en la pila y retorna un indicador para un *lua_State* que representa este nuevo flujo de ejecución. El nuevo flujo de ejecución retornado por esta función comparte todos los objetos globales (tales como tablas) con el estado original, pero posee una pila de ejecución independiente.

No hay una función explícita para terminar o destruir un flujo de ejecución. Objetos del tipo thread están sujetos a la Recolecta de basura, así como cualquier otro objeto de Lua.

---

lua_newuserdata

```
void *lua_newuserdata (lua_State *L, size_t size);
```

Esta función asigna un nuevo bloque de memoria con el tamaño suministrado, coloca en la pila un nuevo objeto userdata completo con la dirección del bloque y retorna esta dirección.

Objetos userdata representan valores C en Lua. Un *userdata completo* representa un bloque de memoria. Es un objeto (así como una tabla): Se debe crear, puede tener su propia meta-tabla y se puede detectar cuando él está siendo colectado. Un objeto userdata completo solamente es igual al mismo (usando la igualdad primitiva, sin el uso de meta-métodos).

Cuando Lua colecta un userdata completo con un meta-método gc, Lua llama el meta-método y marca el userdata como finalizado. Cuando este userdata es colectado nuevamente entonces Lua libera su memoria correspondiente.

lua_next

```
int lua_next (lua_State *L, int index);
```

Desapila una clave de la pila y apila un par clave-valor de la tabla en el índice suministrado (el "próximo" par después de la clave suministrada). Si no hay más elementos en la tabla, entonces *lua_next* retorna 0 (y no apila nada).

Un recorrido típico se parece con éste:

```
/* 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 el recorrido de una tabla, no llamar a *lua_tolstring* directamente sobre una clave, a menos que se sepa que la clave es realmente una cadena de caracteres. Recordar que *lua_tolstring* altera el valor en el índice suministrado; esto confunde la próxima llamada para *lua_next.*

lua_Number

```
typedef double lua_Number;
```

El tipo de números en Lua. Por estándar, es doble, pero se puede alterar en luaconf.h.

A través del archivo de configuración es posible alterar Lua para operar con otro tipo para números (por ejemplo, float o long).

lua_objlen

```
size_t lua_objlen (lua_State *L, int index);
```

Retorna el "tamaño" del valor en el índice aceptable suministrado: Para cadenas de caracteres, éste es el tamaño de la cadena; para tablas, éste es el resultado del operador de tamaño ('#'); para objetos del tipo userdata, éste es el tamaño del bloque de memoria asignado para el userdata; para otros valores, el tamaño es 0.

lua_pcall

```
 lua_pcall (lua_State *L, int nargs, int nresults, int errfunc);
```

Llama una función en modo protegido.

Tanto nargs cuanto nresults poseen el mismo significado que poseían en *lua_call.* Si no hay errores durante la llamada, *lua_pcall* se comporta exactamente como *lua_call.* Sin embargo, si hay cualquier error, *lua_pcall* lo captura, coloca un único valor en la pila (el mensaje de error) y retorna un código de error. Como *lua_call, lua_pcall* siempre remueve la función y sus argumentos de la pila.

Si errfunc es 0, entonces el mensaje de error retornado en la pila es exactamente el mensaje de error original. En caso contrario, errfunc es el índice en la pila de una función de tratamiento de errores. (En la implementación actual, este índice no puede ser un pseudo-índice.) En el caso de errores de tiempo de ejecución, esta función será llamada con el mensaje de error y su valor de retorno será el mensaje retornado en la pila por *lua_pcall.*

Típicamente, la función de tratamiento de errores se utiliza para agregar más información de depuración al mensaje de error, como un trazo de la pila. Tal información no se puede obtener después del retorno de *lua_pcall,* pues en este punto la pila ya fue deshecha.

La función *lua_pcall* retorna 0 en caso de éxito o uno de los siguientes códigos de error (definidos en lua.h):

- LUA_ERRRUN: un error en tiempo de ejecución.
- LUA_ERRMEM: error de asignación de memoria. Para tales errores, Lua no llama la función de tratamiento de errores.
- LUA_ERRERR: error durante la ejecución de la función de tratamiento de errores.

lua_pop

```
void lua_pop (lua_State *L, int n);
```

Desapila n elementos de la pila.

lua_pushboolean

```
void lua_pushboolean (lua_State *L, int b);
```

Apila un valor booleano con valor b en la pila.

lua_pushcclosure

```
void lua_pushcclosure (lua_State *L, lua_CFunction fn, int n);
```

Apila un nuevo cierre C en la pila.

Cuando una función C es creada, es posible asociar algunos valores a la misma, creando entonces un cierre C (ver B.3.5)*;* estos valores son entonces accesibles para la función siempre que es llamada. Para asociar valores con una función C, primero estos valores se deben colocar en la pila (cuando hay múltiples valores, el primer valor es apilado primero). Entonces *lua_pushcclosure* es llamada para crear y colocar la función C en la pila, con el argumento n informando cuántos valores deben ser asociados con la función. *Lua_pushcclosure* también desapila estos valores de la pila.

lua_pushcfunction

```
void lua_pushcfunction (lua_State *L, lua_CFunction f);
```

Apila una función C en la pila. Esta función recibe un indicador para una función C y coloca en la pila un valor Lua del tipo function que, cuando llamado, invoca la función C correspondiente.

Cualquier función para ser registrada en Lua debe seguir el protocolo correcto para recibir sus parámetros y retornar sus resultados (ver *lua_CFunction*).

`lua_pushcfunction` es definida como una macro:

```
#define lua_pushcfunction(L,f) lua_pushcclosure(L,f,0)
```

lua_pushfstring

```
const char *lua_pushfstring (lua_State *L, const char *fmt, ...);
```

Coloca en la pila una cadena de caracteres formateada y retorna un indicador para esta cadena. Es análogo a la función C sprintf, pero posee algunas diferencias importantes:

- no es necesario asignar espacio para el resultado: El resultado es una cadena de caracteres y Lua se encarga de la asignación de memoria (y de la desasignación, a través de la Recolecta de basura);
- los especificadores de conversión son bastante limitados. No hay *flags*, tamaños o precisiones. Los especificadores de conversión pueden ser solamente '%%' (inserta un '%' en la cadena), '%s' (inserta una cadena terminada por cero, sin restricciones de tamaño), '%f' (inserta un lua_Number), '%p' (inserta un indicador como un número hexadecimal), '%d' (inserta un int) y '%c' (inserta un int como un carácter).

lua_pushinteger

```
void lua_pushinteger (lua_State *L, lua_Integer n);
```

Coloca un número con valor n en la pila.

lua_push lightuserdata

```
void lua_pushlightuserdata (lua_State *L, void *p);
```

Coloca un objeto del tipo userdata ligero en la pila.

Un userdata representa valores de C en Lua. Un *userdata* ligero representa un indicador. Es un valor (como un número): No se puede crear, no posee una meta-tabla individual y no es colectado (ya que nunca fue creado). Un userdata ligero es igual a "cualquier" userdata ligero con la misma dirección C.

lua_pushlstring

```
void lua_pushlstring (lua_State *L, const char *s, size_t len);
```

Apila la cadena de caracteres indicada por s con tamaño len en la pila. Lua crea (o reutiliza) una copia interna de la cadena suministrada, de tal forma que la memoria indicada por s puede ser liberada o reutilizada inmediatamente después del retorno de la función. La cadena puede contener ceros dentro de la misma.

lua_pushnil

```
void lua_pushnil (lua_State *L);
```

Coloca un valor nil en la pila.

lua_pushnumber

```
void lua_pushnumber (lua_State *L, lua_Number n);
```

Coloca un número con valor n en la pila.

lua_pushstring

```
void lua_pushstring (lua_State *L, const char *s);
```

Apila la cadena terminada por cero indicada por s en la pila. Lua crea (o reutiliza) una copia interna de la cadena suministrada, de tal forma que la memoria indicada por s puede ser liberada o reutilizada inmediatamente después del retorno de la función. La cadena no puede contener ceros dentro de la misma; se presupone que la cadena termina en el primer cero.

lua_pushthread

```
void lua_pushthread (lua_State *L);
```

Apila el flujo de ejecución representado por L en la pila. Retorna 1 si este flujo de ejecución es el flujo de ejecución principal de su estado.

lua_pushvalue

```
void lua_pushvalue (lua_State *L, int index);
```

Apila una copia del elemento en el índice válido suministrado en la pila.

lua_pushvfstring

```
const char *lua_pushvfstring (lua_State *L, const char *fmt,
                              va_list argp);
```

Equivalente a *lua_pushfstring,* excepto que esta función recibe una va_list en vez de un número variable de argumentos.

lua_rawequal

```
int lua_rawequal (lua_State *L, int index1, int index2);
```

Retorna 1 si los dos valores en los índices aceptables index1 e index2 son iguales primitivamente (es decir, sin hacer llamadas meta-métodos). En caso contrario retorna 0. También retorna 0 si cualquiera de los índices no es válido.

lua_rawget

```
void lua_rawget (lua_State *L, int index);
```

Análogo a *lua_gettable,* pero hace un acceso primitivo (o sea, sin usar meta-métodos).

lua_rawgeti

```
void lua_rawgeti (lua_State *L, int index, int n);
```

Coloca en la pila el valor t[n], donde t es el valor en el índice válido suministrado. El acceso es primitivo; es decir, no invoca meta-métodos.

lua_rawset

```
void lua_rawset (lua_State *L, int index);
```

Análogo a *lua_settable,* pero hace una atribución primitiva (es decir, sin usar meta-métodos).

lua_rawseti

```
void lua_rawseti (lua_State *L, int index, int n);
```

Hace el equivalente a t[n] = v, donde t es el valor en el índice válido suministrado y v es el valor en la parte superior de la pila. Esta función desapila el valor de la pila. La atribución es primitiva; es decir, no invoca meta-métodos.

lua_Reader

```
typedef const char * (*lua_Reader)
                        (lua_State *L, void *data, size_t *size);
```

La función de lectura usada por *lua_load.* Siempre que necesita otro pedazo del trecho, *lua_load* llama la función de lectura, pasando junto su parámetro fecha. La función de lectura debe retornar un indicador para un bloque de memoria con un nuevo pedazo del trecho y atribuir a *size el tamaño del bloque. El bloque debe existir hasta que la función de lectura sea llamada nuevamente. Para señalizar el fin del trecho, la función de lectura debe retornar NULL. La función de lectura puede retornar pedazos de cualquier tamaño mayor que cero.

lua_register

```
void lua_register (lua_State *L, const char *name, lua_CFunction f);
```

Establece la función C f como el nuevo valor de la global name. Esta función es definida como una 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);
```

Remueve el elemento en el índice válido suministrado, desplazando hacia abajo los elementos arriba de este índice para rellenar el hueco. Esta función no puede ser llamada con un pseudo-índice, ya que el pseudo-índice no es una posición real de la pila.

lua_replace

```
void lua_replace (lua_State *L, int index);
```

Mueve el elemento de la parte superior para la posición suministrada (y lo desapila), sin desplazar cualquier elemento (reemplazando por lo tanto el valor en la posición suministrada).

lua_resume

```
int lua_resume (lua_State *L, int narg);
```

Inicia y recomienza una co-rutina en un flujo de ejecución.

Para iniciar una co-rutina, se debe primero crear un nuevo flujo de ejecución (ver *lua_newthread);* a continuación se debe colocar en su pila la función principal más cualesquiera argumentos; por último es llamado *lua_resume,* con narg siendo el número de argumentos. Esta llamada retorna cuando la co-rutina suspende o finaliza su ejecución. Cuando retorna, la pila contiene todos los valores pasados para *lua_yield* o todos los valores retornados por el cuerpo de la función. *Lua_resume* retorna *LUA_YIELD* si la co-rutina cede, 0 si la co-rutina termina su ejecución sin errores o un código de error en el caso de ocurrir errores (ver *lua_pcall).* En el caso de errores, la pila no es deshecha, de forma que se puede usar la API de depuración sobre la misma. El mensaje de error está en la parte superior de la pila. Para reiniciar una co-rutina, se debe colocar en la pila de la misma solamente los valores a ser pasados como resultados de yield y entonces llamar a *lua_resume.*

lua_setallocf

```
void lua_setallocf (lua_State *L, lua_Alloc f, void *ud);
```

Cambia la función de asignación de un dato estado para f con objeto userdata ud.

lua_setfenv

```
int lua_setfenv (lua_State *L, int index);
```

Desapila una tabla de la pila y establece esta tabla como siendo el nuevo ambiente para el valor en el índice suministrado. Si el valor en el índice suministrado no es una función, ni un flujo de ejecución ni un objeto userdata, *lua_setfenv* retorna 0. En caso contrario, la función retorna 1.

lua_setfield

```
void lua_setfield (lua_State *L, int index, const char *k);
```

Hace el equivalente a t[k] = v, donde t es el valor en el índice válido suministrado y v es el valor en la parte superior de la pila.

Esta función desapila el valor de la pila. Como en Lua, esta función puede disparar un meta-método para el evento "index" (ver B.2.9).

lua_setg lobal

```
void lua_setglobal (lua_State *L, const char *name);
```

Desapila un valor de la pila y lo establece como el nuevo valor de la global name. Esta función es definida como una macro:

```
#define lua_setglobal(L,s)  lua_setfield(L, LUA_GLOBALSINDEX, s)
```

lua_setmetatable

```
int lua_setmetatable (lua_State *L, int index);
```

Desapila una tabla de la pila y establece esta tabla como la noticia meta-tabla para el valor en el índice aceptable suministrado.

lua_settable

```
void lua_settable (lua_State *L, int index);
```

Hace el equivalente a t[k] = v, donde t es el valor en el índice válido suministrado, v es el valor en la parte superior de la pila y k es el valor inmediato abajo de la parte superior.

Esta función desapila tanto la clave como el valor de la pila. De la misma forma que en Lua, esta función puede disparar un meta-método para el evento "newindex" (ver B.2.9).

lua_settop

```
void lua_settop (lua_State *L, int index);
```

Acepta cualquier índice aceptable, o 0, y establece este índice como la parte superior de la pila. Si el nuevo tope es mayor que el antiguo, entonces los nuevos elementos son rellenados con **nil.** Si index es 0, entonces todos los elementos de la pila son removidos.

lua_State

```
typedef struct lua_State lua_State;
```

Estructura opaca que guarda el estado completo de un interpretador Lua. La biblioteca de Lua es totalmente reentrante: No existen variables globales. Toda la información sobre un estado es mantenida en esta estructura.

Un indicador para éste estado debe ser pasado como el primer argumento para toda función en la biblioteca, excepto para *lua_newstate,* que crea un nuevo estado Lua desde el cero.

lua_status

```
int lua_status (lua_State *L);
```
Retorna el estatus del flujo de ejecución L.

El estatus puede ser 0 para un flujo de ejecución normal, un código de error si el flujo de ejecución termina su ejecución con un error o LUA_YIELD si el flujo de ejecución estuviera suspendido.

lua_toboolean

```
int lua_toboolean (lua_State *L, int index);
```

Convierte un valor Lua en el índice aceptable suministrado para un valor booleano C (0 o 1). Como todos los tests en Lua, *lua_toboolean* retorna 1 para cualquier valor Lua diferente de **false** y de **nil;** en caso contrario la función retorna 0. La función también retorna 0 cuando llamada con un índice no válido (si se desea aceptar solamente valores booleanos de hecho, usar *lua_isboolean* para testar el tipo del valor).

lua_tocfunction

```
lua_CFunction lua_tocfunction (lua_State *L, int index);
```

Convierte un valor en el índice aceptable suministrado para una función C. Tal valor debe ser una función C; en caso contrario, retorna NULL.

lua_tointeger

```
lua_Integer lua_tointeger (lua_State *L, int idx);
```

Convierte el valor Lua en el índice aceptable suministrado para el tipo completo con señal *lua_Integer.* El valor Lua debe ser un número o una cadena que puede ser convertida para un número (ver B.2.3.2); en caso contrario, *lua_tointeger* retorna 0.

Si el número no es un entero, es truncado de alguna manera no especificada.

lua_tolstring

```
const char *lua_tolstring (lua_State *L, int index, size_t *len);
```

Convierte el valor Lua en el índice aceptable suministrado para una cadena C. Si len no es NULL, la función también establece *len como el tamaño de la cadena. El valor Lua debe ser una cadena de caracteres o un número; en caso contrario, la función retorna NULL. Si el valor es un número, entonces *lua_tolstring* también *cambia el valor real en la pila para una cadena.* (Este cambio confunde *lua_next* cuando *lua_tolstring* se aplica a claves durante un recorrido de tabla.)

lua_tolstring retorna un indicador totalmente alineado para una cadena de caracteres dentro del estado Lua. Esta cadena siempre tiene un cero ('\0') después de su último carácter (como en C), pero puede contener otros ceros en su interior. Visto que Lua posee Recolecta de basura, no hay garantía de que el indicador retornado por *lua_tolstring* será válido después de ser removido de la pila el valor correspondiente.

lua_tonumber

```
lua_Number lua_tonumber (lua_State *L, int index);
```

Convierte el valor Lua en el índice aceptable suministrado para el tipo C *lua_Number* (ver *lua_Number).* El valor Lua debe ser un número o una cadena que puede ser convertida para un número (ver B.2.3.2); en caso contrario, *lua_tonumber* retorna 0.

lua_topointer

```
const void *lua_topointer (lua_State *L, int index);
```

Convierte el valor en el índice aceptable suministrado para un indicador C genérico (void*). El valor puede ser un objeto userdata, una tabla, un flujo de ejecución o una función; objetos diferentes van a suministrar indicadores diferentes. No hay manera de convertir el indicador de vuelta a su valor original.

Típicamente esta función se utiliza solamente para informaciones de depuración.

lua_tostring

```
const char *lua_tostring (lua_State *L, int index);
```

Equivalente a *lua_tolstring* con len siendo igual a NULL.

lua_tothread

```
lua_State *lua_tothread (lua_State *L, int index);
```

Convierte el valor en el índice aceptable suministrado para un flujo de ejecución (representado como lua_State*). Este valor debe ser un flujo de ejecución; en caso contrario, la función retorna NULL.

lua_touserdata

```
void *lua_touserdata (lua_State *L, int index);
```

Si el valor en el índice aceptable suministrado es un objeto userdata completo, la función retorna la dirección de su bloque. Si el valor es un userdata ligero, la función retorna su indicador. En caso contrario, retorna NULL.

lua_type

```
int lua_type (lua_State *L, int index);
```

Retorna el tipo del valor en el índice aceptable suministrado o LUA_TNONE para un índice no válido (es decir, un índice para una posición de la pila "vacía"). Los tipos retornados por *lua_type* se codifican por las siguientes constantes definidas en lua.h: LUA_TNIL, LUA_TNUMBER, LUA_TBOOLEAN, LUA_TSTRING, LUA_TTABLE, LUA_TFUNCTION, LUA_TUSERDATA, LUA_TTHREAD y LUA_TLIGHTUSERDATA.

lua_typename

```
const char *lua_typename (lua_State *L, int tp);
```

Retorna el nombre del tipo codificado por el valor tp, que debe ser uno de los valores retornados por *lua_type*.

lua_Writer

```
typedef int (*lua_Writer) (lua_State *L, const void* p, size_t sz, void* ud);
```

El tipo de la función de escritura usada por *lua_dump.* Siempre que la misma produce otro pedazo de trecho, *lua_dump* llama la función de grabación, pasando junto el buffer a ser escrito (p), su tamaño (sz) y el parámetro fecha suministrado para *lua_dump.*

La función de grabación retorna un código de error: 0 significa ningún error; cualquier otro valor significa un error y hace *lua_dump* parar de llamar la función de grabación.

lua_xmove

```
void lua_xmove (lua_State *from, lua_State *to, int n);
```

Cambia valores entre diferentes flujos de ejecución del *mismo* estado global. Esta función desapila n valores de la pila from y los apila en la pila to.

lua_yield

```
int lua_yield (lua_State *L, int nresults);
```

Cede una co-rutina.

Esta función solamente debe ser llamada como la expresión de retorno de una función C, de la siguiente forma:

```
return lua_yield (L, nresults);
```

Cuando una función C llama a *lua_yield* de esta manera, la co-rutina siendo ejecutada suspende su ejecución y la llamada a *lua_resume* que inició esta co-rutina retorna. El parámetro results es el número de valores de la pila que son pasados como resultados para *lua_resume.*

B.3.9 Interfaz de depuración

Lua no posee mecanismos de depuración pre-definidos. En vez de eso, ofrece una interfaz especial por medio de funciones y ganchos. Esta interfaz permite la construcción de diferentes tipos de depuradores, medidores y otras herramientas que necesitan "información interna" del 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;
```

Una estructura usada para guardar diferentes pedazos de información sobre una función activa. *Lua_getstack* rellena solamente la parte privada de esta estructura, para uso posterior. Para rellenar los otros campos de *lua_Debug* con información útil, llamar a *lua_getinfo.*

Los campos de *lua_Debug* poseen el siguiente significado:

- **source:** si la función fue definida en una cadena de caracteres, entonces source es esa cadena. Si la función fue definida en un archivo, entonces source empieza con un '@' seguido por el nombre del archivo;
- **short_src:** una versión "adecuada" para impresión de source, para ser usada en mensajes de error;
- **linedefined:** el número de la línea donde la definición de la función empieza;
- **lastlinedefined:** el número de la línea donde la definición de la función termina;
- **what:** la cadena "Lua" si la función es una función Lua, "C" si es una función C, "main" si es la parte principal de un trecho y "tail" si fue una función que hizo una recursión final. En el último caso, Lua no posee ninguna otra información sobre la función;
- **currentline:** la línea corriente donde la función suministrada está siendo ejecutada. Cuando ninguna información sobre la línea está disponible, se atribuye -1 a currentline;
- **name:** un nombre razonable para la función suministrada. Dado que funciones en Lua son valores de primera clase, las mismas no poseen un nombre fijo: Algunas funciones pueden ser el valor de múltiples variables globales, mientras otras pueden estar almacenadas solamente en un campo de una tabla. La función lua_getinfo verifica cómo la función fue llamada para encontrar un nombre adecuado. Si no es posible encontrar un nombre, entonces se atribuye NULL a name;
- **namewhat:** explica el campo name. El valor de namewhat puede ser "global", "local", "method", "field", "upvalue" o "" (la cadena vacía), de acuerdo con cómo la función fue llamada (Lua usa la cadena vacía cuando ninguna otra opción parece aplicarse);
- **nups:** el número de upvalues de la función.

---

lua_gethook

```
lua_Hook lua_gethook (lua_State *L);
```

Retorna la función gancho actual.

---

lua_gethookcount

```
int lua_gethookcount (lua_State *L);
```

Retorna el conteo de gancho actual.

---

lua_gethookmask

```
Int lua_gethookmask (lua_State *L);
```

Retorna la máscara de gancho actual.

lua_getinfo

```
int lua_getinfo (lua_State *L, const char *what, lua_Debug *ar);
```

Retorna información sobre una función específica o una invocación de función específica.

Para obtener información sobre una invocación de función, el parámetro ar debe ser un registro de activación válido que fue rellenado por una llamada anterior a *lua_getstack* o fue suministrado como argumento para un gancho (ver *lua_Hook).*

Para obtener información sobre una función se debe colocar en la pila e iniciar la cadena what con el carácter '>'. (En este caso, lua_getinfo desapila la función en la parte superior de la pila.) Por ejemplo, para saber en cuál línea una función f fue definida, se puede escribir el siguiente código:

```
lua_Debug ar;
lua_getfield(L, LUA_GLOBALSINDEX, "f"); /* get global `f' */
lua_getinfo(L, ">S", &ar);
printf("%d\n", ar.linedefined);
```

Cada carácter en la cadena what selecciona algunos campos de la estructura ar para ser rellenados o un valor a ser apilado en la pila:

- **'n':** rellena los campos name y namewhat;
- **'S':** rellena los campos source, short_src, linedefined, lastlinedefined y what;
- **'l':** rellena el campo currentline;
- **'u':** rellena el campo nups;
- **'f':** coloca en la pila la función que está ejecutando en el nivel suministrado;
- **'L':** coloca en la pila una tabla cuyos índices son el número de las líneas que son válidas en la función (una línea válida es una línea con algún código asociado, es decir, una línea donde se puede colocar un punto de parada. Líneas no válidas incluyen líneas vacías y comentarios).

Esta función retorna 0 en caso de error (por ejemplo, en el caso de una opción inválida en what).

lua_getlocal

```
const char *lua_getlocal (lua_State *L, const lua_Debug *aire, int n);
```

Obtiene información sobre una variable local de un registro de activación suministrado. El parámetro ar debe ser un registro de activación válido que fue rellenado por una llamada anterior a *lua_getstack* o fue suministrado como un argumento para un gancho (ver *lua_Hook).* El índice n selecciona cuál variable local inspeccionar (1 es el primer parámetro o variable local activa y así sucesivamente, hasta la última variable local activa). *lua_getlocal* coloca el valor de la variable en la pila y retorna el nombre de la misma.

Nombres de variables empezando con '(' (abre paréntesis) representan variables internas (variables de control de lazos, temporales y funciones C locales.).

Retorna NULL (y no apila nada) cuando el índice es mayor que el número de variables locales activas.

lua_getstack

```
int lua_getstack (lua_State *L, int level, lua_Debug *aire);
```

Obtiene información sobre la pila de tiempo de ejecución del interpretador.

Esta función rellena partes de una estructura *lua_Debug* con una identificación del registro de activación de la función ejecutando en un dado nivel. El nivel 0 es la función ejecutando actualmente, al tiempo que el nivel *n+1* es la función que llamó el nivel *N.* Cuando no hay errores, *lua_getstack* retorna 1; cuando es llamada con un nivel mayor que la profundidad de la pila, la función retorna 0.

lua_getupvalue

```
const char *lua_getupvalue (lua_State *L, int funcindex, int n);
```

Obtiene información sobre un upvalue de un cierre (para funciones Lua, upvalues son variables locales externas que la función usa y que son conceptualmente incluidas en el cierre de la misma). *lua_getupvalue* obtiene el índice n de un upvalue, coloca el valor del upvalue en la pila y retorna su nombre. Funcindex apunta para el cierre en la pila (upvalues no poseen un orden específico, ya que son activos a lo largo de toda la función. Entonces, son numerados en un orden arbitrario).

Retorna NULL (y no apila nada) cuando el índice es mayor que el número de upvalues. Para funciones C, esta función usa la cadena vacía "" como un nombre para todo el upvalues.

lua_Hook

```
typedef void (*lua_Hook) (lua_State *L, lua_Debug *aire);
```

El tipo para funciones de gancho de depuración.

Siempre que un gancho es llamado, se atribuye al campo event de su argumento ar el evento específico que disparó el gancho. Lua identifica estos eventos con las siguientes constantes: LUA_HOOKCALL, LUA_HOOKRET, LUA_HOOKTAILRET, LUA_HOOKLINE y LUA_HOOKCOUNT. Además de ello, para eventos de línea, el campo currentline también es atribuido. Para obtener el valor de cualquier campo en ar, el gancho debe llamar a *lua_getinfo.* Para eventos de retorno, event puede ser LUA_HOOKRET, el valor normal, o LUA_HOOKTAILRET. En el último caso, Lua está simulando un retorno de una función que hizo una recursión final; en este caso, es inútil llamar a *lua_getinfo.*

Mientras Lua está ejecutando un gancho, deshabilita otras llamadas ganchos. Por lo tanto, si un gancho llama a Lua de vuelta para ejecutar una función o un trecho, esta ejecución ocurre sin cualesquiera llamadas a ganchos.

lua_sethook

```
int lua_sethook (lua_State *L, lua_Hook func, int mask, int count);
```

Establece la función de gancho de depuración.

El argumento f es una función de gancho. Mask especifica sobre para cuáles eventos el gancho será llamado: es formado por una conjunción bit-a-bit de las constantes LUA_MASKCALL, LUA_MASKRET, LUA_MASKLINE y LUA_MASKCOUNT. el argumento count solamente tiene significado cuando la máscara incluye LUA_MASKCOUNT. Para cada evento, el gancho es llamado como explicado a continuación:

- **el gancho de llamada (CALL):** es llamado cuando el interpretador llama una función. El gancho es llamado después de Lua entrar en la nueva función, antes de la función recibir sus argumentos;
- **el gancho de retorno (RET):** es llamado cuando el interpretador retorna de una función. El gancho es llamado después de Lua salir de la función. No se tiene acceso a los valores a ser retornados por la función;
- **el gancho de línea (LINE):** es llamado cuando el interpretador está por iniciar la ejecución de una nueva línea de código o cuando vuelve atrás en el código (aunque sea a la misma línea) (este evento solamente ocurre cuando Lua está ejecutando una función Lua;

- **el gancho del lua (COUNT):** es llamado después del interpretador ejecutar cada una de las instrucciones count (este evento solamente ocurre cuando Lua está ejecutando una función Lua).

Un gancho es desactivado atribuyéndose cero a mask.

---

lua_setlocal

```
const char *lua_setlocal (lua_State *L, const lua_Debug *ar, int n);
```

Establece el valor de una variable local de un registro de activación suministrado. Los parámetros ar y n son como en *lua_getlocal* (ver *lua_getlocal). lua_setlocal* atribuye el valor en la parte superior de la pila a la variable y retorna el nombre de la misma. La función también desapila el valor de la pila.

Retorna NULL (y no desapila nada) cuando el índice es mayor que el número de variables locales activas.

---

lua_setupvalue

```
const char *lua_setupvalue (lua_State *L, int funcindex, int n);
```

Establece el valor de un upvalue de un cierre. La función atribuye el valor en la parte superior de la pila al upvalue y retorna su nombre. La misma también desapila el valor de la pila. Los parámetros funcindex y n son como en la función *lua_getupvalue* (ver *lua_getup value).*

Retorna NULL (y no desapila nada) cuando el índice es mayor que el número de upvalues.

B.4 Biblioteca auxiliar

B.4.1 Conceptos básicos

La biblioteca auxiliar suministra varias funciones convenientes para la interfaz de C con Lua. Mientras la API básica suministra las funciones primitivas para todas las interacciones entre C y Lua, la biblioteca auxiliar suministra funciones del más alto nivel para algunas tareas comunes.

Todas las funciones de la biblioteca auxiliar son definidas en el archivo de encabezamiento lauxlib.h y poseen un código luaL_.

Todas las funciones en la biblioteca auxiliar son construidas sobre la API básica y por lo tanto las mismas no ofrecen nada que no se pueda hacer con la API básica.

Varias funciones en la biblioteca auxiliar son usadas para verificar argumentos de funciones C. El nombre de las mismas siempre es luaL_check* o luaL_opt*. Todas esas funciones disparan un error si la verificación no se realiza. Visto que el mensaje de error es formateado para argumentos (por ejemplo, "bad argument #1"), no se debe usar estas funciones para otros valores de la pila.

B.4.2 Funciones y tipos

Todas las funciones y tipos de la biblioteca auxiliar son listadas a continuación en orden alfabético.

---

luaL_addchar

```
void luaL_addchar (luaL_Buffer B, char c);
```

Agrega el carácter c al buffer B (ver *luaL_Buffer).*

luaL_addlstring

```
void luaL_addlstring (luaL_Buffer *B, const char *s, size_t l);
```

Agrega la cadena de caracteres indicada por s con tamaño l al buffer B (ver *luaL_Buffer).* La cadena puede contener ceros en su interior.

luaL_addsize

```
void luaL_addsize (luaL_Buffer B, size_t n);
```

Agrega al buffer B (ver *luaL_Buffer)* una cadena de largo n copiada anteriormente para el área de buffer (ver *luaL_prepbuffer).*

luaL_addstring

```
void luaL_addstring (luaL_Buffer *B, const char *s);
```

Agrega la cadena terminada por 0 indicada por s al buffer B (ver *luaL_Buffer).* La cadena no puede contener ceros dentro de la misma.

luaL_addvalue

```
void luaL_addvalue (luaL_Buffer *B);
```

Agrega el valor en la parte superior de la pila al buffer B (ver *luaL_Buffer).* Desapila el valor.

Ésta es la única función sobre buffers de cadenas que debe ser llamada con un elemento extra en la pila, que es el valor a ser agregado al buffer.

lua L_arg check

```
void luaL_argcheck (lua_State *L, int cond, int numarg, const char *extramsg);
```

Verifica si cond es verdadera. Si no, dispara un error con el siguiente mensaje, donde func es recobrada desde la pila de llamada:

```
bad argument #<narg> to <func> (<extramsg>)
```

luaL_argerror

```
int luaL_argerror (lua_State *L, int numarg, const char *extramsg);
```

Dispara un error con el siguiente mensaje, donde func es recobrada desde la pila de llamada:

```
bad argument #<narg> to <func> (<extramsg>)
```

Esta función nunca retorna, pero es idiomático usarla en funciones C como return luaL_argerror(*args*).

luaL_Buffer

```
typedef struct luaL_Buffer luaL_Buffer;
```

El tipo para un buffer de cadena de caracteres.

Un buffer de cadena permite al código C construir cadenas Lua poco a poco. Su estándar de uso es el siguiente:

- primero se declara una variable b del tipo *luaL_Buffer;*
- a continuación se inicializa la variable con una llamada luaL_buffinit(L, &b);
- después se agregan pedazos de la cadena al buffer llamando a cualquiera de las funciones luaL_add*;
- se termina haciendo una llamada luaL_pushresult(&b). Esta llamada deja la cadena final en la parte superior de la pila.

Durante esa operación normal, un buffer de cadena usa un número variable de posiciones de la pila. Entonces, cuando se está usando un buffer, no se debe asumir que sabe dónde está la parte superior de la pila. Se puede usar la pila entre llamadas sucesivas a las operaciones de buffer desde que este uso sea equilibrado; es decir, cuando es llamada una operación de buffer, la pila está en el mismo nivel en el que estaba inmediatamente después de la operación de buffer anterior (la única excepción a esta regla es *luaL_addvalue).* Después de llamar a *luaL_pushresult*, la pila está de vuelta a su nivel cuando el buffer fue inicializado, más la cadena final en su tope.

luaL_buffinit

```
void luaL_buffinit (lua_State *L, luaL_Buffer *B);
```

Inicializa un buffer B. Esta función no asigna cualquier espacio; el buffer debe ser declarado como una variable (ver *luaL_Buffer).*

luaL_callmeta

```
int luaL_callmeta (lua_State *L, int obj, const char *met);
```

Llama un meta-método.

Si el objeto en el índice obj posee una meta-tabla y esta meta-tabla posee un campo y, esta función llama ese campo y pasa el objeto como su único argumento. En este caso esta función retorna 1 y coloca en la pila el valor retornado por la llamada. Si no hay meta-tabla o meta-método, esta función retorna 0 (sin apilar cualquier valor en la pila).

luaL_checkany

```
void luaL_checkany (lua_State *L, int narg);
```

Verifica si la función tiene un argumento de cualquier tipo (incluyendo **nil)** en la posición narg.

luaL_checkint

```
int luaL_checkint (lua_State *L, int narg);
```

Verifica si el argumento narg de la función es un número y retorna este número convertido para un int.

luaL_checkinteger

```
lua_Integer luaL_checkinteger (lua_State *L, int narg);
```

Verifica si el argumento narg de la función es un número y retorna este número convertido para un *lua_Integer.*

luaL_checklong

```
long luaL_checklong (lua_State *L, int narg);
```

Verifica si el argumento narg de la función es un número y retorna este número convertido para un long.

luaL_checklstring

```
const char *luaL_checklstring (lua_State *L, int narg, size_t *l);
```

Verifica si el argumento narg de la función es una cadena y retorna esta cadena; si l no es NULL rellena *l con el tamaño de la cadena.

luaL_checknumber

```
lua_Number luaL_checknumber (lua_State *L, int narg);
```

Verifica si el argumento narg de la función es un número y retorna este número.

luaL_checkoption

```
int luaL_checkoption (lua_State *L, int narg, const char *def, const char
*const lst[]);
```

Verifica si el argumento narg de la función es una cadena y busca por esta cadena en el array lst (lo cual debe ser terminado por NULL). Retorna el índice en el array donde la cadena fue encontrada. Dispara un error si el argumento no es una cadena o si la cadena no pudo ser encontrada.

Si def no es NULL, la función usa def como un valor estándar cuando no hay argumento narg o si este argumento es **nil.**

Ésta es una función útil para mapear cadenas para enumeraciones de C (la convención usual en bibliotecas Lua es usar cadenas en vez de números para seleccionar opciones).

luaL_checkstack

```
void luaL_checkstack (lua_State *L, int sz, const char *msg);
```

Aumenta el tamaño de la pila para Top + sz elementos, disparando un error si la pila no puede ser aumentada para aquel tamaño. Msg es un texto adicional a ser colocado en el mensaje de error.

luaL_checkstring

```
const char *luaL_checkstring (lua_State *L, int narg);
```

Verifica si el argumento narg de la función es una cadena y retorna esta cadena.

luaL_checktype

```
void luaL_checktype (lua_State *L, int narg, int t);
```

Verifica si el argumento narg de la función tiene tipo T. Ver *lua_type* para la codificación de tipos para t.

luaL_checkudata

```
void *luaL_checkudata (lua_State *L, int narg, const char *tname);
```

Verifica si el argumento narg de la función es un objeto userdata del tipo tname (ver *luaL_newmetatable*).

luaL_dofile

```
int luaL_dofile (lua_State *L, const char *filename);
```

Carga y ejecuta el archivo suministrado. Es definida como la siguiente macro:

```
(LuaL_loadfile(L, filename) || lua_pcall(L, 0, LUA_MULTRET, _0))
```

La función retorna 0 si no hay errores o 1 en caso de errores.

luaL_dostring

```
void *luaL_checkudata (lua_State *L, int narg, const char *tname);
```

Carga y ejecuta la cadena suministrada. Es definida como la siguiente macro:

```
(LuaL_loadstring(L, str) || lua_pcall(L, 0, LUA_MULTRET, _0))
```

La función retorna 0 si no hay errores o 1 en caso de errores.

luaL_error

```
int luaL_error (lua_State *L, const char *fmt, ...);
```

Dispara un error. El formato del mensaje de error se da por fmt más cualesquiera argumentos extras, siguiendo las mismas reglas de lua_pushfstring. También agrega en el inicio del mensaje el nombre del archivo y el número de la línea donde el error ocurrió, caso esta información esté disponible.

Esta función nunca retorna, pero es idiomático usarla en funciones C como return luaL_error(*args*).

luaL_getmetafield

```
int luaL_getmetafield (lua_State *L, int obj, const char *met);
```

Coloca en la pila el campo e de la meta-tabla del objeto en el índice obj. Si el objeto no posee una meta-tabla o si la meta-tabla no posee este campo, retorna 0 y no apila nada.

luaL_getmetatable

```
void luaL_getmetatable (lua_State *L, const char *tname);
```

Coloca en la pila la meta-tabla asociada con el nombre tname en el registro (ver *luaL_newmetatable*).

luaL_gsub

```
const char *luaL_gsub (lua_State *L, const char *s, const char *p, const char *r);
```

Crea una copia de la cadena s reemplazando cualquier ocurrencia de la cadena p por la cadena R. Coloca la cadena resultante en la pila y a retorna.

luaL_loadbuffer

```
int luaL_loadbuffer (lua_State *L, const char *buff, size_t sz, const char *name);
```

Carga un buffer como un trecho de código Lua. Esta función usa *lua_load* para cargar el trecho en el buffer indicado por buff con tamaño sz.

Esta función retorna los mismos resultados de *lua_load.* Name es el nombre del trecho, usado para informaciones de depuración y mensajes de error.

luaL_loadfile

```
int luaL_loadfile (lua_State *L, const char *filename);
```

Carga un archivo como un trecho de código Lua. Esta función usa *lua_load* para cargar el trecho en el archivo llamado filename. Si filename es NULL, entonces carga desde la entrada estándar. La primera línea en el archivo es ignorada si empieza con #.

Esta función retorna los mismos resultados de *lua_load,* pero posee un código de error extra LUA_ERRFILE si no puede abrir/leer el archivo.

De la misma forma que *lua_load,* esta función solamente carga el trecho; no lo ejecuta.

luaL_loadstring

```
int luaL_loadstring (lua_State *L, const char *s);
```

Carga una cadena como un trecho de código Lua. Esta función usa *lua_load* para cargar el trecho en la cadena (terminada por cero) S.

Esta función retorna los mismos resultados de *lua_load.*

Así como *lua_load,* esta función solamente carga el trecho; no lo ejecuta.

luaL_newmetatable

```
int luaL_newmetatable (lua_State *L, const char *tname);
```

Si el registro ya posee la llave tname, retorna 0. En caso contrario, crea una nueva tabla para ser usada como una meta-tabla para el objeto userdata, agrega esta tabla al registro con clave tname y retorna 1.

En ambos casos coloca en la pila el valor final asociado con tname en el registro.

luaL_newstate

```
lua_State *luaL_newstate (void);
```

Crea un nuevo estado Lua. Llama a *lua_newstate* con una función de asignación con base en la función estándar de C realloc y entonces establece una función de pánico (ver *lua_atpanic)* que imprime un mensaje de error para la salida de error estándar en caso de errores fatales.

Retorna el nuevo estado o NULL si ocurrió un error de asignación de memoria.

luaL_openlibs

```
void luaL_openlibs (lua_State *L);
```

Abre todas las bibliotecas estándares en el estado suministrado.

luaL_optint

```
int luaL_optint (lua_State *L, int narg, int d);
```

Si el argumento narg de la función es un número, retorna este número convertido para un int. Si este argumento está ausente o si o es **nil,** retorna d. En caso contrario, dispara un error.

luaL_optinteger

```
lua_Integer luaL_optinteger (lua_State *L, int narg, lua_Integer d);
```

Si el argumento narg de la función es un número, retorna este número convertido para un *lua_Integer.* Si este argumento está ausente o si o es **nil,** retorna d. En caso contrario, dispara un error.

luaL_optlong

```
long luaL_optlong (lua_State *L, int narg, long d);
```

Si el argumento narg de la función es un número, retorna este número convertido para un long. Si este argumento está ausente o si o es **nil,** retorna d. En caso contrario, dispara un error.

luaL_optlstring

```
const char *luaL_optlstring (lua_State *L, int narg, const char *d, size_t *l);
```

Si el argumento narg de la función es una cadena, retorna esta cadena. Si este argumento está ausente o si o es **nil,** retorna d. En caso contrario, dispara un error.

Si l no es NULL, rellena la posición *l con el tamaño del resultado.

luaL_optnumber

```
lua_Number luaL_optnumber (lua_State *L, int narg, lua_Number d);
```

Si el argumento narg de la función es un número, retorna este número. Si este argumento está ausente o si o es **nil,** retorna d. En caso contrario, dispara un error.

luaL_optstring

```
const char *luaL_optstring (lua_State *L, int narg, const char *d);
```

Si el argumento narg de la función es una cadena, retorna esta cadena. Si este argumento está ausente o si o es **nil,** retorna d. En caso contrario, dispara un error.

luaL_prepbuffer

```
char *luaL_prepbuffer (luaL_Buffer *B);
```

Retorna una dirección para un espacio de tamaño LUAL_BUFFERSIZE donde se puede copiar una cadena para ser agregada al buffer B (ver *luaL_Buffer).* Después de copiar la cadena para este espacio se debe llamar a *luaL_addsize* con el tamaño de la cadena para agregarla realmente al buffer.

luaL_pushresult

```
void luaL_pushresult (luaL_Buffer *B);
```

Finaliza el uso del buffer B dejando la cadena final en la parte superior de la pila.

luaL_ref

```
int luaL_ref (lua_State *L, int t);
```

Crea y retorna una referencia, en la tabla en el índice t, para el objeto en la parte superior de la pila (y desapila el objeto).

Una referencia es una clave entera única. Desde que no se agregue manualmente claves enteras en la tabla t, *luaL_ref* garantiza la unicidad de la clave que retorna. Se puede recobrar un objeto referido por el referencia r llamando a lua_rawgeti(L, t, r). La función *luaL_unref* libera una referencia y el objeto asociado a la misma.

Si el objeto en la parte superior de la pila es **nil,** *luaL_ref* retorna la constante LUA_REFNIL. La constante LUA_NOREF es totalmente diferente de cualquier referencia retornada por *luaL_ref.*

luaL_Reg

```
typedef struct luaL_Reg {
      const char *name;
      lua_CFunction func;
    } luaL_Reg;
```

El tipo para arrays de funciones a ser registrados por *luaL_register.* Name es el nombre de la función y func es un indicador para la función. Cualquier array de *luaL_Reg* debe terminar con una entrada centinela en la cual tanto name como func son NULL.

luaL_register

```
void luaL_register (lua_State *L, const char *libname, const luaL_Reg *l);
```

Abre una biblioteca.

Cuando llamada con libname igual a NULL, simplemente registra todas las funciones en la lista l (ver *luaL_Reg)* en la tabla en la parte superior de la pila.

Cuando llamada con un valor de libname diferente de NULL, luaL_register crea una nueva tabla t, establece la misma como el valor de la variable global libname, la establece como el valor de package.loaded[libname] y registra en la misma todas las funciones en la lista L. si existe una tabla en package.loaded[libname] o en la variable libname, la función reutiliza esta tabla en vez de crear una noticia.

En cualquier caso la función deja la tabla en la parte superior de la pila.

luaL_typename

```
const char *luaL_typename (lua_State *L, int idx);
```

Retorna el nombre del tipo del valor en el índice suministrado.

luaL_typerror

```
int luaL_typerror (lua_State *L, int narg, const char *tname);
```

Genera un error con un mensaje como el siguiente:

```
location: bad argument narg to 'func' (tname expected, got rt)
```

Donde *location* es producida por *luaL_where, func* es el nombre de la función corriente y *rt* es el nombre del tipo del argumento.

luaL_unref

```
void luaL_unref (lua_State *L, int t, int ref);
```

Libera la referencia ref de la tabla en el índice t (ver luaL_ref). La entrada es removida de la tabla, de modo que el objeto mencionado puede ser colectado. La referencia ref también es liberada para ser usada nuevamente.

Si ref es LUA_NOREF o LUA_REFNIL, luaL_unref no hace nada.

luaL_where

```
void luaL_where (lua_State *L, int lvl);
```

Coloca en la pila una cadena identificando la posición actual del control en el nivel lvl en la pila de llamada. Típicamente esta cadena posee el siguiente formato:

```
chunkname: currentline:
```

Nivel 0 es la función ejecutando corrientemente, nivel 1 es la función que llamó la función que está ejecutando actualmente, etc.

Esta función se utiliza para construir un código para mensajes de error.

B.5 Bibliotecas estándar

B.5.1 Visión general

Las bibliotecas estándar de Lua ofrecen funciones útiles que son implementadas directamente a través de la API C. Algunas de esas funciones ofrecen servicios esenciales para el lenguaje (por ejemplo, *type* y *getmetatable);* otras ofrecen acceso a servicios "externos" (por ejemplo, E/S); y otras podrían ser implementadas en Lua concretamente, pero son bastante útiles o poseen requisitos de desempeño críticos que merecen una implementación en C (por ejemplo, *table.sort*).

Todas las bibliotecas son implementadas a través de la API C oficial y son suministradas como módulos C separados. Normalmente, Lua posee las siguientes bibliotecas estándar:

- biblioteca básica;
- biblioteca de paquetes;
- manipulación de cadenas de caracteres;
- manipulación de tablas;
- funciones matemáticas (sen, log, etc.);
- entrada y salida;
- facilidades del sistema operativo;
- facilidades de depuración.

Exceptuándose la biblioteca básica y la biblioteca de paquetes, cada biblioteca provee todas sus funciones como campos de una tabla global o como métodos de sus objetos.

Para tener acceso a esas bibliotecas, el programa hospedero C debe llamar la función *luaL_openlibs,* que abre todas las bibliotecas estándar. De modo alternativo, es posible abrirlas individualmente llamando a luaopen_base (para la biblioteca básica), luaopen_package (para la biblioteca de paquetes), luaopen_string (para la biblioteca de cadenas de caracteres), luaopen_table (para la biblioteca de tablas), luaopen_math (para la biblioteca matemática), luaopen_io (para la biblioteca de E/S), luaopen_os (para la biblioteca del Sistema Operativo), y luaopen_debug (para la biblioteca de depuración). Esas funciones están declaradas en lualib.h y no deben ser llamadas directamente: Se deben llamar como cualquier otra función C de Lua, por ejemplo, usando *lua_call.*

B.5.2 Funciones básicas

La biblioteca de funciones básicas ofrece algunas funciones esenciales la Lua. Si no se incluye esta biblioteca en su aplicación, se debe verificar cuidadosamente si necesita suministrar implementaciones para algunas de sus facilidades.

assert (v [, message])

Producen un error cuando el valor de su argumento v es falso (ejemplo, **nil** o **false);** en caso contrario, retorna todos sus argumentos. Message es un mensaje de error; cuándo ausente, el mensaje estándar es "¡assertion failed!"

collectgarbage (opt [, arg])

Esta función es una interfaz genérica para el colector de basura. Realiza diferentes funciones de acuerdo con su primer argumento, opt:

- **“stop” :** interrumpe el colector de basura.
- **“restart” :** reinicia el colector de basura.
- **“collect” :** realiza un ciclo de Recolecta de basura completa.
- **“count” :** retorna la memoria total que está siendo usada por Lua (en Kbytes).
- **“step” :** realiza un paso de recolecta de basura. El "tamaño" del paso es controlado por `arg` (valores mayores significan más pasos) de manera no especificada. Si se desea controlar el tamaño del paso, se debe ajustar de manera experimental el valor de `arg`. Retorna **true** si el paso terminó un ciclo de Recolecta de basura.
- **“steppause” :** establece `arg`/100 como el nuevo valor para la *pausa* del colector (ver B.2.1 1).
- **“setstepmul” :** establece `arg`/100 como el nuevo valor para el *multiplicador de paso* del colector (ver B.2.1 1).

dofile (filename)

Abre el archivo indicado y ejecuta su contenido como un trecho de código Lua. Cuando es llamada sin argumentos, dofile ejecuta el contenido de la entrada estándar (stdin). Retorna todos los valores retornados por el trecho. En caso de errores, dofile propaga el error para su llamador (es decir, dofile no ejecuta en modo protegido).

error (message [, level])

Termina la última función protegida llamada y retorna message como el mensaje de error. La función error nunca retorna.

Generalmente, error agrega alguna información sobre la posición del error en el inicio del mensaje. El argumento level especifica cómo obtener la posición del error. Cuando éste es igual a 1 (el estándar), la posición del error es donde la función error fue llamada. Cuando éste es 2, la posición del error es donde la función que llamó a error fue llamada; y así sucesivamente. Pasando un valor 0 para level evita la adición de información de la posición del error al mensaje.

_G

Una variable global (no una función) que almacena el ambiente global (es decir, _G._G = _G). Lua por sí solo no usa esta variable; una modificación de su valor no afecta cualquier ambiente y viceversa. (usar *setfenv* para alterar ambientes.)

getfenv (f)

Retorna el ambiente que está siendo usado normalmente por la función. F puede ser una función Lua o un número que especifica la función en aquel nivel de pila: La función que llamó a getfenv posee nivel 1. Si la función suministrada no es una función Lua o se f es 0, getfenv retorna el ambiente global. El valor estándar para f es 1.

getmetatable (object)

Si object no posee una meta-tabla, retorna **nil.** En caso contrario, si la meta-tabla del objeto posee un campo "__metatable", retorna el valor asociado. En caso contrario, retorna la meta-tabla del objeto suministrado.

ipairs (t)

Retorna tres valores: una función repetidora, la tabla t y 0, de modo que la construcción

```
for i,v in ipairs(t) do body end
```

Repetirá sobre los pares (1,t [1]), (2,t[2]), ···, hasta la primera clave entera ausente de la tabla.

load (func [, chunkname])

Carga un trecho usando la función func para obtener sus pedazos. Cada llamada a func debe retornar una cadena de caracteres que concatena con resultados anteriores. Cuando func retorna **nil** (o cuando no retorna ningún valor), eso indica el fin del trecho.

Si no ocurren errores, retorna el trecho compilado como una función; en caso contrario, retorna **nil** más el mensaje de error. El ambiente de la función retornada es el ambiente global.

chunkname se utiliza como el nombre del trecho para mensajes de error e información de depuración. Cuando ausente, el valor estándar es "=(load)".

loadfile ([filename])

Análogo a *load,* pero obtiene el trecho del archivo filename o de la entrada estándar, si ningún nombre de archivo se suministra.

loadstring (string [, chunkname])

Análogo a *load,* pero obtiene el trecho de la cadena suministrada.

Para cargar y rodar una dada cadena, usar la expresión idiomática

```
assert (loadstring(s)) ()
```

Cuando está ausente, el valor estándar para chunkname es la cadena suministrada.

next (table [, index])

Permite a un programa recorrer todos los campos de una tabla. Su primer argumento es una tabla y su segundo argumento es un índice en esa tabla. Next retorna el próximo índice de la tabla y su valor asociado. Cuando llamada con **nil** como su segundo argumento, next retorna un índice inicial y su valor asociado. Cuando llamada con el último índice o con **nil** en una tabla vacía, next retorna **nil.** Si el segundo argumento está ausente, entonces éste es interpretado como **nil.** En particular, se puede usar next(t) para verificar si la tabla está vacía.

El orden en el cual los índices son enumerados no se especifica, incluso para índices numéricos (para recorrer una tabla en orden numérico, use el **for** numérico o la función *ipairs*).

El comportamiento de next es *indefinido* si, durante el recorrido, se atribuye cualquier valor a un campo no existente en la tabla. Se puede, sin embargo, modificar campos existentes. En particular, se puede limpiar campos existentes.

pairs (t)

Retorna tres valores: la función *next,* la tabla t y **nil,** de modo que la construcción

```
for k,v in pairs(t) do body end
```

Repetirá sobre todos los pares clave–valor de la tabla t.

Ver la función *next* para los cuidados que se deben tener al modificar la tabla durante su recorrido.

pcall (f, arg1, arg2, ...)

Llama la función f con los argumentos suministrados en *modo protegido.* Esto significa que cualquier error dentro de f no es propagado; en vez de eso, pcall captura el error y retorna un código indicando el estatus. Su primer resultado es el código de estatus (un booleano), que es verdadero si la llamada ocurrió sin errores. En este caso, pcall también retorna todos los resultados de la llamada, después de este primer resultado. En el caso de ocurrir un error, pcall retorna **false** más el mensaje de error.

print (e1, e2, ...)

Recibe cualquier número de argumentos e imprime sus valores para stdout, usando la función *tostring* para convertirlos en cadenas de caracteres. Print no es proyectada para salida formateada, sino solamente como una manera rápida de mostrar un valor, típicamente para depuración. Para salida formateada, use *string.format.*

rawequal (v1, v2)

Verifica si v1 es igual a v2, sin invocar ningún meta-método. Retorna un booleano.

rawget (table, index)

Obtiene el valor real de table[index], sin invocar ningún meta-método. Table debe ser una tabla; index puede ser cualquier valor.

rawset (table, index, value)

Atribuye value como el valor real de table[index], sin invocar ningún meta-método. Table debe ser una tabla, index puede ser cualquier valor diferente de **nil** y value puede ser cualquier valor Lua.

Esta función retorna table.

select (index, ...)

Si index es un número, retorna todos los argumentos después del argumento número index. En caso contrario, index debe ser la cadena "#" y select retorna el número total de argumentos extras recibidos.

setfenv (f, table)

Establece el ambiente a ser usado por la función suministrada. F puede ser una función Lua o un número que especifica la función en aquel nivel de pila: La función llamando a setfenv posee nivel 1. Setfenv retorna la función suministrada.

Como un caso especial, cuando f es 0 setfenv cambia el ambiente del flujo de ejecución corriente. En este caso, setfenv no retorna ningún valor.

setmetatable (table, metatable)

Establece la meta-tabla para la tabla suministrada (no se puede alterar la meta-tabla de otros tipos desde Lua, solamente desde C.) Si metatable es **nil,** remueve la meta-tabla de la tabla suministrada. Si la meta-tabla original tiene un campo "__metatable", dispara un error.

Esta función retorna table.

tonumber (obj [, base])

Intenta convertir su argumento para un número. Si el argumento ya es un número o una cadena de caracteres que puede ser convertida para un número, entonces tonumber retorna este número; en caso contrario, retorna **nil.**

Un argumento opcional especifica la base para interpretar el numeral. La base puede ser cualquier entero entre 2 y 36, ambos inclusive. En bases arriba de 10, la letra ‘a’ (mayúscula o minúscula) representa 10, 'B' representa 11 y así sucesivamente, con 'Z' representando 35. En la base 10 (el estándar), el número puede tener una parte decimal, así como una parte exponente opcional (ver B.2.2). En otras bases, solamente son aceptados enteros sin señal.

tostring (obj)

Recibe un argumento de cualquier tipo y lo convierte para una cadena de caracteres en un formato razonable. Para un control completo de como números son convertidos, use *string.format.*

Si la meta-tabla de e posee un campo "__tostring", entonces tostring llama el valor correspondiente con e como argumento y usa el resultado de la llamada como su resultado.

type (v)

Retorna el tipo de su único argumento, codificado como una cadena de caracteres. Los resultados posibles de esta función son "nil" (una cadena de caracteres, no el valor **nil),** "number", "string", "boolean", "table", "function", "thread" y "userdata".

unpack (list [, i [, j]])

Retorna los elementos de la tabla suministrada. Esta función es equivalente a

```
return list[ i], list[ i+1] , ···, list[ j]
```

excepto que el código anterior puede ser escrito solamente para un número fijo de elementos. Por estándar, i es 1 y j es el tamaño de la lista, como definido por el operador de tamaño (ver B.2.6.6).

_VERSION

Una variable global (no una función) que almacena una cadena conteniendo la versión corriente del interpretador. El contenido corriente de esta variable es "Lua 5.1".

xpcall (f, err)

Esta función es análoga a *pcall,* excepto que se puede establecer un nuevo tratador de errores.

xpcall llama la función f en modo protegido, usando err como un tratador de errores. Cualquier error dentro de f no es propagado; en vez de eso, xpcall captura el error, llama la función err con el objeto de error original y retorna un código indicando un estatus. Su primer resultado es el código de status (un booleano), que es verdadero si la llamada ocurrió sin errores. En este caso, xpcall también retorna todos los resultados de la llamada, después de este primer resultado. En caso de error, xpcall retorna **false** más el resultado de err.

B.5.3 Manipulación de co-rutinas

Las operaciones relacionadas con co-rutinas constituyen una sub-biblioteca de la biblioteca básica y están dentro de la tabla coroutine. Ver B.2.12 para una descripción general de co-rutinas.

---

coroutine.create (f)

Crea una nueva co-rutina, con cuerpo f. f debe ser una función Lua. Retorna esta nueva co-rutina, un objeto con tipo "thread".

---

coroutine.resume (co [, val1, ..., valn])

Inicia o continúa la ejecución de la co-rutina co. En la primera vez que se "continúa" una co-rutina, empieza ejecutando su cuerpo. Los valores val1, ··· son pasados como los argumentos para el cuerpo de la función. Si la co-rutina ya cedió la ejecución antes, resume la continúa; los valores val1, ··· son pasados como los resultados de la cesión.

Si la co-rutina ejecuta sin ningún error, resume retorna **true** más cualesquiera valores pasados para yield (si la co-rutina cede) o cualquier valor retornados por el cuerpo de la función (si la co-rutina termina). Si hay cualquier error, resume retorna **false** más el mensaje de error.

---

coroutine.running ()

Retorna la co-rutina siendo ejecutada o **nil** cuando es llamada por el flujo de ejecución principal.

---

coroutine.status (co)

Retorna el estatus de la co-rutina co, como una cadena de caracteres: "running", si la co-rutina está ejecutando (es decir, llamó a estatus); "suspended", si la co-rutina está suspensa en una llamada a yield o se no comenzó su ejecución aún; "normal" si la co-rutina está activa pero no está ejecutando (es decir, continuó otra co-rutina); y "dead" si la co-rutina terminó su función principal o se paró con un error.

---

coroutine.wrap (f)

Crea una nueva co-rutina, con cuerpo f. f debe ser una función Lua. Retorna una función que recomienza la co-rutina cada vez que es llamada. cualesquiera argumentos pasados para la función se comportan como los argumentos extras para resume. Retorna los mismos valores retornados por resume, excepto el primer booleano. En caso de error, propaga el error.

---

coroutine.yield ([val1, ..., valn])

Suspende la ejecución de la co-rutina llamadora. La co-rutina no puede estar ejecutando una función C, un metamétodo o un repetidor. Cualesquiera argumentos para yield son pasados como resultados extras para resume.

B.5.4 Módulos

La biblioteca de paquetes provee facilidades básicas para cargar y construir módulos en Lua. Exporta dos de sus funciones directamente en el ambiente global: *Require* y *module.* Todas las otras funciones son exportadas en una tabla package.

module (name [, ...])

Crea un módulo. Si hay una tabla en package.loaded[name], esta tabla es el módulo. En caso contrario, si existe una tabla global t con el nombre suministrado, esta tabla es el módulo. En caso contrario crea una nueva tabla t y la establece como el valor de la global name y el valor de package.loaded[name]. Esta función también inicializa t._NAME con el nombre suministrado, t._M con el módulo (el propio t) y t._PACKAGE con el nombre del paquete (el nombre del módulo completo menos el último componente). Finalmente, module establece t como el nuevo ambiente de la función corriente y el nuevo valor de package.loaded[name], de modo que *require* retorna t.

Si name es un nombre compuesto (es decir, un nombre con componentes separados por puntos), module crea (o reutiliza, si las mismas ya existen) tablas para cada componente. Por ejemplo, si name es a.b.c, entonces module almacena la tabla del módulo en el campo c del campo b de la global a.

Esta función puede recibir algunas *opciones* después del nombre del módulo, donde cada opción es una función a ser aplicada sobre el módulo.

require (mod name)

Carga el módulo suministrado. Esta función empieza buscando en la tabla *package.loaded* para determinar si modname ya fue cargado. En caso afirmativo, requiere retornar el valor almacenado en package.loaded[modname]. En caso contrario, intenta hallar un *cargador* para el módulo.

Para encontrar un cargador, require es guiada por el array *package.loaders.* Modificando este array, podemos alterar cómo require busca por un módulo. La siguiente explicación se basa en la configuración estándar para *package.loaders.*

El primer require consulta package.preload[modname]. Se existe un valor en ese campo, este valor (que debe ser una función) es el cargador. En caso contrario require busca por un cargador Lua usando el camino almacenado en *package.path.* Si eso también falla, busca por un cargador C usando el camino almacenado en *package.cpath.* Si eso también falla, intenta un cargador *todo en uno* (ver *package.loaders).*

Una vez que un cargador se encuentra, require llama el cargador con un único argumento, modname. Si el cargador retorna cualquier valor, require atribuye el valor retornado a package.loaded[modname]. Si el cargador no retorna ningún valor y no fue atribuido ningún valor a package.loaded[modname], entonces require atribuye **true** a esta posición. En cualquier caso, require retorna el valor final de package.loaded[modname].

Si ocurre un error durante la carga o la ejecución del módulo o si no es posible encontrar un cargador para el módulo, entonces require señaliza un error.

package.cpath

El camino usado por *require* para buscar un cargador C.

Lua inicializa el camino C *package.cpath* de la misma forma que inicializa el camino Lua *package.path,* usando la variable de ambiente LUA_CPATH o un camino estándar definido en luaconf.h.

package.loaded

Una tabla usada por *require* para controlar cuáles módulos ya fueron cargados. Cuando se requisita un módulo modname y package.loaded[modname] no es falso, *require* simplemente retorna el valor almacenado allá.

package.loaders

Una tabla usada por *require* para controlar cómo cargar módulos.

Cada posición en esta tabla es una función buscadora. Cuando está buscando un módulo, *require* llama a cada una de estas funciones buscadoras en orden creciente, con el nombre del módulo (el argumento suministrado a *require)* como su único parámetro. La función puede retornar otra función (el *cargador* del módulo) o una cadena de caracteres explicando por qué no halló aquel módulo (o **nil** si no tiene nada a decir). Lua inicializa esta tabla con cuatro funciones.

La primera función buscadora simplemente buscar un cargador en la tabla *package.preload.*

La segunda función buscadora busca un cargador como una biblioteca Lua, usando el camino almacenado en *package.path.* Un camino es una secuencia de *estándares* separadas por punto y coma. Para cada estándar, la función buscadora va a alterar cada punto de interrogación en el estándar para filename, que es el nombre del módulo con cada punto reemplazado por un "separador de directorio" (como "/" en el Unix); entonces intentará abrir el nombre del archivo resultante. Por ejemplo, si el camino es la cadena de caracteres

```
"./?.lua;./?.lc;/usr/local/?/init.lua"
```

la busca de un archivo Lua para el módulo foo intentará abrir los archivos, ./foo.lc y /usr/local/foo/init.lua, en ese orden.

La tercera función buscadora busca un cargador como una biblioteca C, usando el camino suministrado por la variable *package.cpath.* Por ejemplo, si el camino C es la cadena

```
"./?.so;./?.dll;/usr/local/?/init.so"
```

la función buscadora para el módulo foo intentará abrir los archivos ./foo.so, ./foo.dll y /usr/local/foo/init.so, en ese orden. Una vez que encuentra una biblioteca C, esta función buscadora primero usa una facilidad de enlace dinámica para conectar la aplicación con la biblioteca. Entonces intenta encontrar una función C dentro de la biblioteca para ser usada como cargador. El nombre de esta función C es la cadena "luaopen_" concatenada con una copia del nombre del módulo donde cada punto es reemplazado por un subrayado. Además de ello, si el nombre del módulo posee un guión, su código hasta (e incluyendo) el primer guión es removido. Por ejemplo, si el nombre del módulo es a.v1-b.c, el nombre de la función será luaopen_b_c.

La cuarta función buscadora intenta un cargador todo en uno. Busca en el camino C una biblioteca para la raíz del nombre del módulo suministrado. Por ejemplo, al solicitar a.b.c, buscará por una biblioteca C para a. Si encuentra, busca en esa biblioteca por una función de apertura para el submódulo; en este ejemplo, sería luaopen_a_b_c. Con esta facilidad, un paquete puede empaquetar varios submódulos C dentro de una única biblioteca, con cada submódulo guardando su función de apertura original.

---

package.loadlib (libname, funcname)

Conecta dinámicamente el programa hospedero con la biblioteca C libname. Dentro de esta biblioteca, busca por una función funcname y retorna esa función como una función C (de ese modo, funcname debe seguir el protocolo (ver *lua_CFunction*)).

Ésta es una función de bajo nivel. Contorna completamente el sistema de paquetes y de módulos. Diferentemente de *require,* no realiza cualquier busca de camino y no agrega extensiones automáticamente. Libname debe ser el nombre del archivo completo de la biblioteca C, incluyendo, en caso de ser necesario, un camino y una extensión. Funcname debe ser el nombre exacto expuesto por la biblioteca C (que puede depender de cómo se usan el compilador y el conector C).

Esta función no es provista por ANSI C. De esa forma, está disponible solamente en algunas plataformas (Windows, Linux, MAC LAS X, Solaris, BSD, más otros sistemas Unix que dan soporte al estándar dlfcn).

package.path

El camino usado por *require* para buscar un cargador Lua.

Al iniciar, Lua inicializa esta variable con el valor de la variable de ambiente LUA_PATH o con un camino estándar definido en luaconf.h, si la variable de ambiente no está definida. Cualquier ";;" en el valor de la variable de ambiente será reemplazado por el camino estándar.

package.preload

Una tabla para almacenar cargadores para módulos específicos (ver *require).*

package.seeall (module)

Establece una meta-tabla para module con su campo __index refiriéndose al ambiente global, de modo que ese módulo hereda valores del ambiente global. Para ser usada como una opción a la función *module.*

B.5.5 Manejo de cadenas de caracteres

Esta biblioteca provee funciones genéricas para el manejo de cadenas de caracteres, tales como encontrar y extraer subcadenas y casamiento de estándares. Al indexar una cadena en Lua, el primer carácter está en la posición 1 (no en la posición 0, como en C). Índices pueden tener valores negativos y son interpretados como una indexación de detrás para delante, desde el final de la cadena. Por lo tanto, el último carácter está en la posición -1 y así sucesivamente.

La biblioteca de cadenas provee todas sus funciones dentro de la tabla string. La misma también establece una meta-tabla para cadenas donde el campo __index apunta para la tabla string. En consecuencia de eso, se puede usar las funciones de cadenas en un estilo orientado a objetos. Por ejemplo, string.byte(s, i) puede ser escrito como s:byte(i).

string.byte (s [, i [, j]])

Retorna el código numérico interno de los caracteres s[i], s[i+1], ···, s[j]. El valor estándar para i es 1; el valor estándar para j es I.

Códigos numéricos no son necesariamente portátiles entre plataformas.

string.char (i1, i2, ...)

Recibe cero o más enteros. Retorna una cadena con tamaño igual al número de argumentos, en la cual cada carácter posee un código numérico interno igual a su argumento correspondiente.

Códigos numéricos no son necesariamente portátiles entre plataformas.

string.dump (function)

Retorna una cadena conteniendo la representación binaria de la función suministrada, de modo que un *loadstring* posterior en esta cadena retorna una copia de la función. Function debe ser una función Lua sin upvalues.

string.find (s, pattern [, init [, plain]])

Busca la primera coincidencia del estándar pattern en la cadena S. Si la función encuentra una coincidencia, entonces find retorna los índices de s donde esta ocurrencia comenzó y terminó; en caso contrario, retorna **nil.** El tercer argumento, init, es un valor numérico opcional y especifica dónde iniciar la busca; su valor estándar es 1 y puede ser negativo. Un valor **true** para el cuarto argumento, plain, que es opcional, deshabilita las facilidades de coincidencia de estándares, de modo que la función hace una operación "encuentra subcadena" simple, sin considerar ningún carácter en pattern como "mágico". Si plain se suministra, entonces init debe ser suministrado también.

Si el estándar posee capturas, entonces en una coincidencia exitosa los valores capturados son también retornados, después de los dos índices.

string.format (formatstring, e1, e2, ...)

Retorna la versión formateada de su número variable de argumentos siguiendo la descripción fecha en su primer argumento (que debe ser una cadena). El formato de la cadena sigue las mismas reglas de la familia printf de funciones C estándar. Las únicas diferencias son que las opciones/modificadores *, l, L, n, p y h no son ofrecidas y que hay una opción extra, q. La opción q formatea una cadena en una forma adecuada para ser leída de vuelta de forma segura por el interpretador Lua; la cadena es escrita entre comillas dobles y todas las comillas dobles, quiebras de línea, barras alteradas y ceros dentro de la cadena son correctamente escapados cuando escritos. Por ejemplo, la llamada

```
string.format('%q', 'a string with "quotes" and \n new line')
```

producirá la cadena:

```
"a string with \"quotes\" and \

new line"
```

Las opciones c, d, E, e, f, g, G, i, o, u, X y x esperan un número como argumento, mientras que q y s esperan una cadena.

Esta función no acepta valores de cadenas conteniendo ceros dentro de las mismas, excepto cuando esos valores son argumentos para la opción q.

string.gmatch (s, pattern)

Retorna una función repetidora que, cada vez que es llamada, retorna la próxima captura de `pattern` en la cadena `S`. Si `pattern` no especifica ninguna captura, entonces la coincidencia entera es producida a cada llamada. Como un ejemplo, el siguiente lazo

```
s = "hello world from Lua"
for w in string.gmatch(s, "%a+")do
      print (w)
end
```

repetirá sobre todas las palabras de la cadena `s`, imprimiendo una por línea. El próximo ejemplo recolecta todos los pares `key=value` de la cadena suministrada y los coloca en una tabla:

```
t = {}
s = "from=world, to=Lua"
for k, v in string.gmatch(s, "(%w+)=(%w+)") do
    t[k] = v
end
```

Para esa función, un `'^'` en el inicio de un estándar no funciona como un ancla, visto que eso iría a impedir la repetición.

string.gsub (s, pattern, repl [, n])

Retorna una copia de s en la cual todas las (o las primeras n, si se han suministrado) ocurrencias de pattern son sustituidas por una cadena de sustitución especificada por repl, que puede ser una cadena, una tabla o una función. Gsub también retorna, como su segundo valor, el número total de sustituciones que ocurrieron.

Si repl es una cadena, entonces su valor se utiliza para la sustitución. El carácter % funciona como un carácter de escape: Cualquier secuencia en repl de la forma %*n*, con *n* entre 1 9, representa el valor de la *n*-ésima subcadena capturada. La secuencia %0 representa el casamiento entero. La secuencia %% representa un % simple.

Si repl es una tabla, entonces la tabla es consultada a cada coincidencia, usando la primera captura como la clave; si el estándar no especifica ninguna captura, entonces la coincidencia entera se utiliza como la clave.

Si repl es una función, entonces esta función es llamada a siempre que la coincidencia ocurre, con todas las subcadenas capturadas siendo pasadas como argumentos, en el orden en el que fueron capturadas; si el estándar no especifica ninguna captura, entonces la coincidencia entera es pasada como un único argumento.

Si el valor retornado por la consulta a la tabla o por la llamada de función es una cadena o un número, entonces ese valor se utiliza como la cadena de sustitución; en caso contrario, si es **false** o **nil,** entonces no hay sustitución (es decir, el casamiento original es mantenido en la cadena).

Aquí están algunos ejemplos:

```
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)

Recibe una cadena y retorna su tamaño. La cadena vacía "" tiene tamaño 0. Ceros dentro de la cadena son contados, entonces "a\000bc\000" tiene tamaño 5.

string.lower (s)

Recibe una cadena y retorna una copia de esta cadena con todas las letras mayúsculas convertidas para minúsculas. Todos los demás caracteres permanecen iguales. La definición de lo que es una letra mayúscula depende del idioma (*locale*) corriente.

string.match (s, pattern [, init])

Busca la primera coincidencia de pattern en la cadena S. Si encuentra una, entonces match retorna las capturas del estándar; en caso contrario retorna **nil.** Si pattern no especifica ninguna captura, entonces la coincidencia entera es retornada. Un tercer argumento numérico opcional, init, especifica dónde iniciar la busca; su valor estándar es 1 y puede ser negativo.

string.rep (s, n)

Retorna una cadena que es la concatenación de n copias de la cadena s.

string .reverse (s)

Retorna una cadena que es la cadena s invertida.

string .sub (s, i [, j])

Retorna una subcadena de s que inicia en i y continúa hasta j; i y j pueden ser negativas. Si j está ausente, entonces se presupone que es igual a -1 (que es lo mismo que el tamaño de la cadena). En particular, la llamada string.sub(s,1,j) retorna un código de s con tamaño j y string.sub(s, -i) retorna un sufijo de s con tamaño i.

string.upper (s)

Recibe una cadena y retorna una copia de esta cadena con todas las letras minúsculas convertidas para mayúsculas. Todos los demás caracteres permanecen iguales. La definición de lo que es una letra minúscula depende del idioma *(locale)* corriente.

B.5.6 Estándares

Una clase de caracteres se utiliza para representar un conjunto de caracteres. Las siguientes combinaciones se permiten en descripciones de una clase de caracteres:

- **x:** (donde x no es uno de los caracteres mágicos `^$()%.[]* +-?)` representa el propio carácter x.
- `.`: (un punto) representa todos los caracteres.
- **%a:** representa todas las letras.
- **%c:** representa todos los caracteres de control.
- **%d:** representa todos los dígitos.
- **%l:** representa todas las letras minúsculas.
- **%p:** representa todos los caracteres de puntuación.
- **%s:** representa todos los caracteres de espacio.
- **%u:** representa todas las letras mayúsculas.
- **%w:** representa todos los caracteres alfanuméricos.

- **%*x*:** representa todos los dígitos hexadecimales.

- **%z:** representa el carácter con representación 0.

- %***x***: (donde *x* es cualquier carácter no alfanumérico) representa el carácter *x.* Ésta es la manera estándar de escapar los caracteres mágicos. Cualquier carácter de puntuación (incluso los no mágicos) puede ser precedido por un `'%'` cuando usado para representar a sí mismo en un estándar.

- **`[`*`set`*`]:`** representa la clase que es la unión de todos los caracteres en *set.* Una banda de caracteres puede ser especificada separando los caracteres finales de la banda con un `'-'`. Todas las clases `%`*x* descritas arriba también se pueden usar como componentes en *set.* Todos los otros caracteres en *set* representan ellos mismos. Por ejemplo, [`%w_]` (o `[_%w])` representa todos los caracteres alfanuméricos más el subrayado, [`0-7]` representa los dígitos octales y [`0-7%l%-]` representa los dígitos octales más las letras minúsculas más el carácter `'-'`.

  La interacción entre bandas y clases no es definida. Por lo tanto, estándares como [%a-z] o [a-%%] no tienen significado.

- **`[^`*`set`*`]`** *:* Representa el complemento de *set,* donde *set* es interpretado como arriba.

Para todas las clases representadas por una única letra (%a, %c, etc.), la letra mayúscula correspondiente representa el complemento de la clase. Por ejemplo, %S representa todos los caracteres que no son de espacio.

Las definiciones de letra, espacio y otros grupos de caracteres dependen del idioma *(locale)* corriente. En particular, la clase [a-z] puede no ser equivalente a %l.

Un ítem de estándar puede ser:

- una clase de un único carácter, que empareja cualquier carácter simple que pertenezca a la clase;

- una clase de un único carácter seguida por '*', que empareja 0 o más repeticiones de caracteres de la clase. Estos ítems de repetición siempre emparejarán la mayor secuencia posible;

- una clase de un único carácter seguida por '+', que empareja 1 o más repeticiones de caracteres de la clase. Estos ítems de repetición siempre emparejarán la mayor secuencia posible;

- una clase de un único carácter seguida por '-', que también empareja 0 o más repeticiones de caracteres de la clase. Diferentemente de '*', estos ítems de repetición siempre emparejarán a menor secuencia posible;

- una clase de un único carácter seguida por '?', que empareja 0 ó 1 ocurrencia de un carácter de la clase;

- `%`*n*`,` para *n* entre 1 9; tal ítem empareja una subcadena igual a la *n*-ésima cadena capturada;

- %b*xy*, donde x e y son dos caracteres distintos; tal ítem empareja cadenas que empiezan con *x,* terminan con *y* y donde el número de *x*s y de *y*s es equilibrado. Esto significa que, si alguien lee la cadena de izquierda a derecha, contando *+1* para una *x* y *-1* para una *y,* la *y* final es la primera *y* donde el contador alcanza 0. Por ejemplo, el ítem %b() empareja expresiones con paréntesis equilibrados.

Un estándar es una secuencia de ítems de estándar. Un '^' en el inicio de un estándar ancla la coincidencia en el inicio de la cadena siendo usada. Un '$' al final de un estándar ancla la coincidencia al final de la cadena siendo usada. En otras posiciones, '^' y '$' no poseen significado especial y representan a sí mismos.

Un estándar puede contener subestándares delimitados por paréntesis; describen capturas. Cuando una coincidencia ocurre, las subcadenas de la cadena siendo usada que coincidieron con las capturas son almacenadas *(capturadas)* para uso futuro. Las capturas son numeradas de acuerdo con sus paréntesis izquierdos. Por ejemplo, en el estándar "(a*(.)%w(%s*))", la parte de la cadena casando "a*(.)%w(%s*)" es almacenada como la primera captura (y por lo tanto tiene el número 1); el carácter coincidiendo "." es capturado con el número 2 y la parte coincidiendo "%s*" tiene el número 3.

Como un caso especial, la captura vacía () captura la posición de la cadena corriente (un número). Por ejemplo, si aplicamos el estándar "()aA()" en la cadena "flaaap", habrá dos capturas: 3 y 5.

Un estándar no puede contener ceros dentro de él. Use %z como alternativa.

B.5.7 Manejo de tablas

Esta biblioteca provee funciones genéricas para manejo de tablas. Provee todas sus funciones en la tabla table.

La mayoría de las funciones en la biblioteca de tablas presupone que la tabla representa un array o una lista. Para estas funciones, cuando hablamos sobre el "tamaño" de una tabla estamos hablando sobre el resultado del operador de tamaño.

---

table.concat (table [, sep [, i [, j]]])

Dado un array donde todos los elementos son cadenas o números, retorna table[i]..sep..Table[i+1] ··· sep..table[j]. El valor estándar para sep es la cadena vacía, el estándar para i es 1 y el estándar para j es el tamaño de la tabla. Si i es mayor que j, retorna la cadena vacía.

---

table.insert (table, [pos,] value)

Inserta el elemento value en la posición pos de table, desplazando los otros elementos para abrir espacio, en caso de ser necesario. El valor estándar para pos es n+1, donde n es el tamaño de la tabla (ver B.2.6.6), de modo que una llamada table.insert(t,x) inserta x al final de la tabla t.

---

table.maxn (table)

Retorna el mayor índice numérico positivo de la tabla suministrada o cero si la tabla no posee índices numéricos positivos (para realizar su trabajo esta función hace un recorrido lineal de la tabla entera).

---

table.remove (table [, pos])

Remueve de table el elemento en la posición pos, desplazando los otros elementos para rellenar el espacio, en caso de ser necesario. Retorna el valor del elemento removido. El valor estándar para pos es n, donde n es el tamaño de la tabla, de modo que una llamada table.remove(t) remueve el último elemento de la tabla t.

---

table.sort (table [, comp])

Ordena los elementos de la tabla en un dado orden, *in-place,* de table[1] hasta table[n], donde n es el tamaño de la tabla. Si comp se suministra, entonces él debe ser una función que recibe dos elementos de la tabla y retorna true cuando el primero es menor que el segundo (de modo que not comp(la[i+1,a][i]) será verdadero después de la ordenación). Si comp no se suministra, entonces el operador estándar de Lua < se utiliza en su lugar.

El algoritmo de ordenación no es estable; es decir, elementos considerados iguales por el orden suministrada pueden tener sus posiciones relativas cambiadas por la ordenación.

B.5.8 Funciones matemáticas

Esta biblioteca es una interfaz para la biblioteca matemática de C estándar. Provee todas sus funciones en la tabla math.

math.abs (x)

Retorna el valor absoluto de x.

math. acos (x)

Retorna el arco coseno de x (en radianos).

math.asin (x)

Retorna el arco seno de x (en radianos).

math.atan (x)

Retorna el arco tangente de x (en radianos).

math.atan2 (x)

Retorna el arco tangente de y/x (en radianos), pero usa la señal de los dos parámetros para hallar el cuadrante del resultado (también trata correctamente el caso de x ser cero).

math.aceil (x)

Retorna el menor entero mayor o igual a x.

math.cos (x)

Retorna el coseno de x (presupone que x está en radianos).

math.cosh (x)

Retorna el coseno hiperbólico de x.

math.deg (x)

Retorna el ángulo x (dado en radianos) en grados.

math.exp (x)

Retorna el valor de *ex.*

math.floor (x)

Retorna el mayor entero menor o igual a x.

math.fmod (x, y)

Retorna el resto de la división de x por y.

math.frexp (x)

Retorna m y e tales que $x = m2^e$, e es un entero y el valor absoluto de m está en el intervalo [0.5, 1) (o cero cuando x es cero).

math.huge (x)

El valor de HUGE_VAL, un valor mayor o igual a cualquier otro valor numérico.

math.ldexp (m, e)

Retorna $m2^e$ (e debe ser un entero).

math.log10 (x)

Retorna el logaritmo natural de x.

math.log10 (x)

Retorna el logaritmo base -10 de x.

math.max (x, …)

Retorna el valor máximo entre sus argumentos.

math.min (x, …)

Retorna el valor mínimo entre sus argumentos.

math.modf (x)

Retorna dos números, la parte integral de x y la parte fraccionaria de x.

math. pi

El valor de *pi.*

math.pow (x, y)

Retorna xy (también se puede usar la expresión x^y para computar este valor).

math.rad (x)

Retorna el ángulo x (dado en grados) en radianos.

math.random ([m [,n]])

Esta función es una interfaz para la función generadora pseudo-aleatoria simple rand suministrada por ANSI C. (ninguna garantía puede ser dada para sus propiedades estadísticas).

Cuando llamada sin argumentos, retorna un número real pseudo-aleatorio en el intervalo [0,1). Cuando llamada con un número m, math.random retorna un entero pseudo-aleatorio en el intervalo [1, m). Cuando llamada con dos números m y n, math.random retorna un entero pseudo-aleatorio en el intervalo [m, n).

math.randomseed (x)

Establece x como la "semilla" para el generador pseudo-aleatorio: semillas iguales producen secuencias iguales de números.

math.sin (x)

Retorna el seno de x (presupone que x está en radianos).

math.sinh (x)

Retorna el seno hiperbólico de x.

math.sqrt (x)

Retorna la raíz cuadrada de x (también se puede usar la expresión x^0.5 para computar este valor).

math.tan (x)

Retorna la tangente de x (presupone que x está en radianos).

math.tanh (x)

Retorna la tangente hiperbólica de x.

B.5.9 Facilidades de entrada y salida

La biblioteca de E/S provee dos estilos diferentes para manejo de archivos. El primero usa descriptores de archivo implícitos; es decir, hay operaciones para establecer un archivo de entrada estándar y un archivo de salida estándar y todas las operaciones de entrada/salida son realizadas sobre estos archivos. El segundo estilo usa descriptores de archivo explícitos.

Cuando se usa descriptores de archivo implícitos, todas las operaciones son provistas por la tabla io. Cuando se usa descriptores de archivo explícitos, la operación *io.open* retorna un descriptor de archivo y entonces todas las operaciones son provistas como métodos del descriptor de archivo.

La tabla io también suministra tres descriptores de archivo pre-definidos con sus significados usuales de C: io.stdin, io.stdout e io.stderr.

A menos que dicho de modo contrario, todas las funciones de E/S retornan **nil** en caso de fallo (más un mensaje de error como segundo resultado y un código de error dependiente del sistema como un tercer resultado), o algún valor diferente de **nil** en caso de éxito.

---

io.close ([file])

Equivalente a file:close(). Cuando no recibe file, cierra el archivo de salida estándar.

---

io.flush ()

Equivalente a file:flush en el archivo de salida estándar.

---

io.input ([file])

Cuando es llamada con un nombre de archivo, abre el archivo con aquel nombre (en modo texto) y establece su manipulador como el archivo de entrada estándar. Cuando es llamada con un manipulador de archivo, simplemente establece este manipulador de archivo como el archivo de entrada estándar. Cuando llamada sin parámetros, retorna el archivo de entrada estándar corriente.

En caso de errores esta función dispara el error, en vez de retornar un código de error.

---

io.lines ([filename])

Abre el nombre de archivo suministrado en modo de lectura y retorna una función repetidora que, cada vez que es llamada, retorna una nueva línea del archivo. Por lo tanto, la construcción

```
for line in io.lines(filename) do body end
```

repetirá sobre todas las líneas del archivo. Cuando la función repetidora detecta el fin del archivo, retorna **nil** (para finalizar el lazo) y automáticamente cierra el archivo.

La llamada io.lines() (sin ningún nombre de archivo) es equivalente a io.input():lines(); es decir, repite sobre las líneas del archivo de entrada estándar. En este caso no cierra el archivo cuando el lazo termina.

---

io.open (filename [, mode])

Esta función abre un archivo, en el modo especificado en la cadena mode. Retorna un nuevo manipulador de archivo o, en caso de errores, **nil** más un mensaje de error.

La cadena de caracteres mode puede ser cualquier una de las siguientes:

- **“r”:** modo de lectura (el estándar);
- **“w”:** modo de grabación;
- **“a”:** modo de adición;
- **“r+”:** modo de actualización, todos los datos anteriores son preservados;
- **“w+”:** modo de actualización, todos los datos anteriores son preservados;
- **“a+”:** modo de actualización de adición, datos anteriores son preservados, la grabación solamente se permite al final del archivo.

La cadena mode también puede tener una 'b' al final, que es necesaria en algunos sistemas para abrir el archivo en modo binario. Esta cadena es exactamente la que se utiliza en la función estándar de C fopen.

---

io.output ([file])

Análogo a *io.input,* pero opera sobre el archivo de salida estándar.

---

io.popen ([prog [, mode]])

Inicia el programa prog en un proceso separado y retorna un manipulador de archivo que se puede usar para leer datos de este programa (si mode es "r", el estándar) o escribir datos para este programa (si mode es "w").

Esta función es dependiente del sistema y no está disponible en todas las plataformas.

---

io.read (format1, ...)

Equivalente a io.input():read.

---

io.tmpfile ()

Retorna un manipulador para un archivo temporal. Este archivo es abierto en modo de actualización y es automáticamente removido cuando el programa termina.

---

io.type (obj)

Verifica si obj es un manipulador de archivo válido. Retorna la cadena "file" si obj es un manipulador de archivo abierto, "close file" si obj es un manipulador de archivo cerrado o **nil** si obj no es un manipulador de archivo.

---

io.write (value1, ...)

Equivalente a io.output():write.

---

file:close ()

Cierra file. Archivos son automáticamente cerrados cuando sus manipuladores son colectados por el colector de basura, pero lleva una cantidad indeterminada de tiempo para que eso ocurra.

file:flush ()

Graba cualquier dato escrito para file.

file:lines ()

Retorna una función repetidora que, cada vez que es llamada, retorna una nueva línea del archivo. Por lo tanto, la construcción

```
for line in file:lines() do ... end
```

repetirá sobre todas las líneas del archivo (al contrario de *io.lines,* esa función no cierra el archivo cuando el lazo termina).

file:read (format1, ...)

Lee el archivo file, de acuerdo con los formatos suministrados, los cuales especifican lo que debe ser leído. Para cada formato, la función retorna una cadena (o un número) con los caracteres leídos o **nil** si no puede retornar datos con el formato especificado. Cuando llamada sin formatos, usa el formato estándar que lee la próxima línea toda.

Los formatos disponibles son:

- **"*n":** lee un número; éste es el único formato que retorna un número en vez de una cadena.
- **"*a":** lee el archivo entero, empezando por la posición corriente. Cuando está al final del archivo, retorna la cadena vacía.
- **"*l":** lee la próxima línea (saltando el fin de línea), retornando **nil** al final del archivo. Éste es el formato estándar.
- ***número:*** lee una cadena hasta este número de caracteres, retornando **nil** al final del archivo. Si el número suministrado es cero, la función no lee nada y retorna una cadena vacía o **nil** cuando está al final del archivo.

file:seek ([whence] [, offset])

Establece y obtiene la posición del archivo, medida desde el inicio del archivo, hasta la posición dada por offset más una base especificada por la cadena whence, de la siguiente forma:

- **"set" :** base es la posición 0 (el inicio del archivo);
- **"cur" :** base es la posición corriente;
- **"end" :** base es el fin del archivo;

En caso de éxito, la función seek retorna la posición final del archivo, medida en bytes desde el inicio del archivo. Si esta función falla, retorna **nil,** más una cadena describiendo el error.

El valor estándar para whence es "cur" y para offset es 0. Por lo tanto, la llamada file:seek() retorna la posición del archivo corriente, sin modificarla; la llamada file:seek("set") establece la posición en el inicio del archivo (y retorna 0); y la llamada file:seek("end") establece la posición en el fin del archivo y retorna su tamaño.

file:setvbuf (mode [, size])

Define el modo de buferización para un archivo de salida. Hay tres modos disponibles:

- **“no” :** ninguna buferización; el resultado de cualquier operación de salida aparece inmediatamente;
- **“full” :** buferización completa; la operación de salida se realiza solamente cuando el buffer está lleno (o cuando explícitamente se descarga el archivo (ver *io.flush));*
- **“line” :** buferización de línea; la salida es buferizada hasta que una nueva línea es producida o hay cualquier entrada desde algunos archivos especiales (como un dispositivo de terminal).

Para los últimos dos casos, size especifica el tamaño del buffer, en bytes. El estándar es un tamaño apropiado.

file:write (value1, ...)

Escribe el valor de cada uno de sus argumentos para file. Los argumentos deben ser cadenas de caracteres o números. Para escribir otros valores, use *tostring* o *string.format* antes de write.

B.5.10 Facilidades del sistema operativo

Esta biblioteca es implementada a través de la tabla os.

os.clock ()

Retorna una aproximación de la cantidad de tiempo de CPU, en segundos, usada por el programa.

os.date ([format [, time]])

Retorna una cadena o una tabla conteniendo fecha y horario, formateada de acuerdo con la cadena format suministrada.

Si el argumento time está presente, éste es el tiempo a ser formateado (ver la función *os.time* para una descripción de este valor). En caso contrario, date formatea el horario corriente.

Si format empieza con '!', entonces la fecha es formateada en el Tiempo Universal Coordinado. Después de ese carácter opcional, si format es la cadena "*t", entonces date retorna una tabla con los siguientes campos: Year (cuatro dígitos), month (1--12), day (1--31), hour (0--23), min (0--59), sec (0--61), yday (día del año) e isdst (flag que indica el horario de verano, un booleano).

Si format no es "*t", entonces date retorna la fecha como una cadena de caracteres, formateada de acuerdo con las mismas reglas de la función C strftime.

Cuando llamada sin argumentos, date retorna una representación aceptable de la fecha y del horario que depende del sistema hospedero y del idioma *(locale)* corriente. (Es decir, os.date() es equivalente a os.date("%c")).

os.difftime (t2, t1)

Retorna el número de segundos desde el tiempo t1 hasta el tiempo t2. En POSIX, Windows y algunos otros sistemas, este valor es exactamente t2-t1.

os.execute ([command])

Esta función es equivalente a la función C system. Pasa command para ser ejecutado por un interpretador de comandos del sistema operativo. Retorna un código de estatus, que es dependiente del sistema. Si command está ausente, entonces la función retorna un valor diferente de cero si un interpretador de comandos está disponible y cero en caso contrario.

os.exit ([code])

Llama la función C exit, con un código code opcional, para terminar el programa hospedero. El valor estándar para code es el código de éxito.

os.getenv (varname)

Retorna el valor de la variable de ambiente del proceso varname o **nil** si la variable no está definida.

os.remove (filename)

Remueve un archivo o directorio con el nombre suministrado. Directorios deben estar vacíos para ser removidos. Si esta función falla, retorna **nil,** más una cadena describiendo el error.

os.rename (oldname, newname)

Cambia el nombre de un archivo o directorio llamado oldname para newname. Si esta función falla, retorna **nil,** más una cadena describiendo el error.

os.setlocale (locale [, category])

Establece el idioma *(locale)* corriente del programa. Locale es una cadena de caracteres especificando un idioma; category es una cadena opcional describiendo para cual categoría se debe alterar: "All", "collate", "ctype", "monetary", "numeric" o “time”; la categoría estándar es "all". Esta función retorna el nombre del nuevo idioma o **nil** si la requisición no puede ser honrada.

Si locale es la cadena vacía, se establece el idioma corriente como un idioma nativo definido por la implementación. Si locale es la cadena "C", se establece el idioma corriente como el idioma estándar de C.

Cuando llamada con **nil** como el primer argumento, esta función retorna solamente el nombre del idioma corriente para la categoría suministrada.

os.time ([table])

Retorna el tiempo corriente cuando es llamada sin argumentos o un tiempo representando la fecha y el horario especificados por la tabla suministrada. Esta tabla debe tener campos year, month y day y puede tener campos hour, min, sec e isdst (para una descripción de estos campos, ver la función *os.date*).

El valor retornado es un número, cuyo significado depende de su sistema. En POSIX, Windows y algunos otros sistemas, este número cuenta el número de segundos desde algún tiempo de inicio dado (a "era"). En otros sistemas, el significado no se especifica y el número retornado por equipo se puede usar solamente como un argumento para date y difftime.

os.tmpname ()

Retorna una cadena de caracteres con el nombre de un archivo que se puede usar para un archivo temporal. El archivo debe ser explícitamente abierto antes de ser usado y explícitamente removido cuando no sea más necesario.

B.5.11 Biblioteca de depuración

Esta biblioteca provee las funcionalidades de la interfaz de depuración para programas Lua. Se debe tener cuidado al usar esta biblioteca. Las funciones suministradas aquí deben ser usadas exclusivamente para depuración y tareas análogas, tales como medición *(profiling).* Debe evitarse usarlas como una herramienta de programación usual: pueden ser muy lentas. Además de ello, varias de esas funciones violan algunas suposiciones al respecto del código Lua (por ejemplo, que variables locales a una función no se pueden acceder de fuera de la función o que meta-tablas de objetos userdata no pueden ser modificadas por código Lua) y por lo tanto pueden comprometer un código que, de otro modo, sería seguro.

Todas las funciones en esta biblioteca son suministradas en la tabla debug. Todas las funciones que operan sobre un objeto del tipo thread poseen un primer argumento opcional que es el objeto thread sobre el cual la función debe operar. El estándar es siempre el flujo de ejecución corriente.

debug.debug ()

Entra en un modo interactivo con el usuario, ejecutando cada cadena de caracteres que el usuario entra. Usando comandos simples y otros mecanismos de depuración, el usuario puede inspeccionar variables globales y locales, alterar el valor de las mismas, evaluar expresiones, etc. Una línea conteniendo solamente la palabra cont termina esta función, de modo que la función llamadora continúa su ejecución.

Los comandos para debug.debug no son anidados de modo léxico dentro de ninguna función y, por lo tanto, no poseen acceso directo a variables locales.

debug.getfenv (o)

Retorna el ambiente del objeto o.

debug.gethook ()

Retorna las configuraciones de gancho corrientes del flujo de ejecución como tres valores: La función de gancho corriente, la máscara de gancho corriente y el conteo de gancho corriente (como establecido por la función *debug.sethook*).

debug.getinfo (function [, what])

Retorna una tabla con información sobre una función. Se puede suministrar la función directamente o se puede suministrar un número como el valor de function, que significa la función ejecutando en el nivel function de la pila de llamadas del flujo de ejecución suministrado: Nivel 0 es la función corriente (la propia getinfo); nivel 1 es la función que llamó a getinfo; y así sucesivamente. Si function es un número mayor que el número de funciones activas, entonces getinfo retorna **nil.**

La tabla retornada puede contener todos los campos retornados por *lua_getinfo,* con la cadena what describiendo cuáles campos deben ser rellenados. El estándar para what es obtener todas las informaciones disponibles, excepto la tabla de líneas válidas. En caso de estar presente, la opción 'f' agrega un campo llamado func con la propia función. En caso de estar presente, la opción 'L' agrega un campo llamado activelines con la tabla de líneas válidas.

Por ejemplo, la expresión debug.getinfo(1,"n").name retorna una tabla con un nombre para la función corriente, si un nombre razonable puede ser encontrado, y la expresión debug.getinfo(print) retorna una tabla con todas las informaciones disponibles sobre la función *print.*

---

debug.getlocal (level, local)

Esta función retorna el nombre y el valor de la variable local con índice local de la función en el nivel level de la pila (el primer parámetro o variable local posee índice 1 y así sucesivamente, hasta la última variable local activa). La función retorna **nil** si no existe una variable local con el índice suministrado y dispara un error cuando llamada con un level fuera de la banda de valores válidos (se puede llamar a *debug.getinfo* para verificar si el nivel es válido).

Nombres de variables que empiezan con '(' (abre paréntesis) representan variables internas (variables de control de lazos, temporales y locales de funciones C).

---

debug.getmetatable (object)

Retorna la meta-tabla del object suministrado o **nil** si él no posee una meta-tabla.

---

debug.getregistry ()

Retorna la tabla de registro (ver B.3.6).

---

debug.getupvalue (func, up)

Esta función retorna el nombre y el valor del upvalue con índice up de la función func. La función retorna **nil** si no hay un upvalue con el índice suministrado.

---

debug.setfenv (object, table)

Establece la tabla table como el ambiente del object suministrado. Retorna object.

---

debug.sethook (hook, mask [, count])

Establece la función suministrada como un gancho. La cadena mask y el número count describen cuando el gancho será llamado. La cadena mask puede tener los siguientes caracteres, con el respectivo significado:

- **"c" :** el gancho es llamado siempre que Lua llama una función;
- **"r" :** el gancho es llamado siempre que Lua retorna de una función;
- **"l" :** el gancho es llamado siempre que Lua entra una nueva línea de código.

Con un count diferente de cero, el gancho es llamado después de cada count instrucciones.

Cuando llamada sin argumentos, *debug.gethook* deshabilita el gancho.

Cuando el gancho se llama, su primer parámetro es una cadena de caracteres describiendo el evento que disparó su llamada: "Call", "return" (o "tail return"), "line" y "count". Para eventos de línea, el gancho también obtiene el nuevo número de línea como su segundo parámetro. Dentro del gancho, es posible llamar a getinfo con nivel 2 para obtener más información sobre la función siendo ejecutada (nivel 0 es la función getinfo y nivel 1 es la función de gancho), a menos que el evento sea "tail return". En este caso, Lua está solamente simulando el retorno y una llamada a getinfo retornará datos inválidos.

debug.setlocal (level, local, value)

Esta función atribuye el valor value a la variable local con índice local de la función en el nivel level de la pila. La función retorna **nil** si no hay una variable local con el índice suministrado y dispara un error cuando llamada con un level fuera de la banda de valores válidos (se puede llamar a getinfo para verificar si el nivel es válido). En caso contrario, la función retorna el nombre de la variable local.

debug.setmetatable (object, table)

Establece table como la meta-tabla del object suministrado (table puede ser **nil).**

debug.setupvalue (func, up, value)

Esta función atribuye el valor value al upvalue con índice up de la función func. La función retorna **nil** si no hay un upvalue con el índice suministrado. En caso contrario, la función retorna el nombre del upvalue.

debug.traceback ([message])

Retorna una cadena de caracteres con un trazo de la pila de llamadas. Una cadena opcional message es agregada al inicio del trazo. Un número opcional level dice en cuál nivel iniciar el trazo (el estándar es 1, la función llamando a traceback).

B.6 El interpretador Lua autónomo

Aunque Lua haya sido proyectada como un lenguaje de extensión, para ser incorporada en un programa C hospedero, Lua también es frecuentemente usada como un lenguaje autosuficiente. Un interpretador para Lua como un lenguaje autosuficiente, llamado simplemente lua, se suministra con la distribución estándar. Ese interpretador incluye todas las bibliotecas estándar, incluso la biblioteca de depuración. Su uso es:

```
lua [ options] [ script [ args]]
```

Las opciones son:

- **-e *stat :*** ejecuta la cadena *stat;*
- **-l *mod :*** "requisita" *mod;*
- **-i :** entra en modo interactivo después de ejecutar *script;*
- **-v :** imprime información de versión;
- **-- :** para de tratar opciones;
- **- :** ejecuta stdin como un archivo y para de tratar opciones.

Después de tratar sus opciones, Lua ejecuta el *script* suministrado, pasándole los *args* suministrados como cadenas de argumentos. Cuando llamado sin argumentos, lua se comporta como lua -v -i cuando la entrada estándar (stdin) es un terminal y como lua - en caso contrario.

Antes de ejecutar cualquier argumento, el interpretador verifica si hay una variable de ambiente LUA_INIT. Si su formato es *@filename,* entonces lua ejecuta el archivo. En caso contrario, lua ejecuta la propia cadena de caracteres.

Todas las opciones son manejadas en el orden dado, excepto -i. Por ejemplo, una invocación como

```
$ lua -e'Val=1' -e 'print(Val) ' script.lua
```

irá primero atribuir 1 a a, después imprimirá el valor de a (que es '1') y finalmente ejecutará el archivo script.lua sin argumentos (aquí $ es el *prompt* del interpretador de comandos. Se puede tener un prompt diferente).

Antes de empezar a ejecutar el script, lua guarda todos los argumentos suministrados en la línea de comando en una tabla global llamada arg. El nombre del script es almacenado en el índice 0, el primer argumento después del nombre del script queda en el índice 1 y así sucesivamente. Cualesquiera argumentos antes del nombre del script (es decir, el nombre del interpretador más las opciones) quedan en índices negativos. Por ejemplo, en la llamada

```
$ lua -la b.lua t1 t2
```

el interpretador primero ejecuta el archivo a.lua, después crea la tabla

```
arg = { [-2] = "lua", [-1] = "-la",
            [0] = "b.lua",
            [1] = "t1", [2] = "t2" }
```

y finalmente ejecuta el archivo b.lua. El script es llamado con arg[1], arg[2], ··· como argumentos; él también puede acceder a estos argumentos con la expresión vararg '...'.

En modo interactivo, si se escribe un comando incompleto, el interpretador espera que lo complete e indica esto a través de un prompt diferente.

Si la variable global _PROMPT contiene una cadena de caracteres, entonces su valor se utiliza como el prompt. De manera análoga, si la variable global _PROMPT2 contiene una cadena, su valor se utiliza como el prompt secundario (mostrado durante comandos incompletos). Por lo tanto, los dos prompts pueden ser modificados directamente en la línea de comando o en cualesquiera programas Lua haciendo una atribución a _PROMPT. Ver el ejemplo a continuación:

```
$ lua -e"_PROMPT='myprompt> '" -i
```

El par de comillas más externo es para el interpretador de comandos y el par más interno es para Lua. El uso de -i para entrar en modo interactivo; en caso contrario, el programa iría a terminar silenciosamente después de la atribución a _PROMPT.

Para permitir el uso de Lua como un interpretador de scripts en sistemas Unix, el interpretador de línea de comando salta la primera línea de un trecho de código si él empieza con #. Por lo tanto, scripts Lua se pueden usar como programas ejecutables usando chmod +x y la forma #!, como en

```
#! /usr/local/bin/lua
```

Está claro que la localización del interpretador Lua puede ser diferente en su máquina. Si lua está en su PATH, entonces

```
#! /usr/bin/env lua
```

es una solución más portable.

B.7 Incompatibilidades con la versión 5.0

NOTA Las incompatibilidades que pueden ser encontradas cuando se pasa un programa de Lua 5.0 para Lua 5.1 se listan en esta sección. Se puede evitar la mayoría de las incompatibilidades compilando Lua con opciones apropiadas (ver el archivo luaconf.h). Sin embargo, todas esas opciones de compatibilidad serán removidas en la próxima versión de Lua.

B.7.1 Cambios en el lenguaje

A continuación se listan las alteraciones en el lenguaje introducidos en Lua 5.1:

- el sistema de vararg cambió del pseudo-argumento arg con una tabla con los argumentos extras para la expresión vararg (ver la opción de tiempo de compilación LUA_COMPAT_VARARG en luaconf.h);
- hubo un cambio sutil en el alcance de las variables implícitas del comando **for** y del comando **repeat;**
- la sintaxis de cadena larga/comentario largo *([[string]])* no permite anidado. En esos casos se puede usar la nueva sintaxis *([=[string]=])* (ver la opción de tiempo de compilación LUA_COMPAT_LSTR em luaconf.h).

B.7.2 Cambios en las bibliotecas

A continuación se listan las alteraciones en las bibliotecas estándar introducidas en Lua 5.1:

- la función string.gfind fue rebautizada como *string.gmatch* (ver la opción de tiempo de compilación LUA_COMPAT_GFIND en luaconf.h);
- cuando *string.gsub* es llamada con una función como su tercer argumento, siempre que esta función retorna **nil** o **false** la cadena de reemplazo es la coincidencia entera, en vez de la cadena vacía;
- la función table.setn está ultrapasada y no se debe usar. La función table.getn corresponde al nuevo operador de tamaño (#); usar el operador em vez de la función (ver la opción de tiempo de compilación LUA_COMPAT_GETN em luaconf.h.);
- la función loadlib fue rebautizada como *package.loadlib* (ver la opción de tiempo de compilación LUA_COM PAT_LOADLI B en luaconf.h);
- la función math.mod fue rebautizada como *math.fmod* (ver la opción de tiempo de compilación LUA_COMPAT_MOD en luaconf.h);
- las funciones table.foreach y table.foreachi están ultrapasadas y no deben ser usadas. Se puede usar un lazo **for** con pairs o ipairs en vez de las mismas;
- hubo cambios sustanciales en la función *require* debido al nuevo sistema de módulos. El nuevo comportamiento es básicamente compatible con el antiguo, aunque ahora require obtiene el camino de *package.path* y no más de LUA_PATH;
- la función *collectgarbage* posee argumentos diferentes. La función gcinfo está ultrapasada y no se debe usar; usar collectgarbage("count") en vez de la misma.

B.7.3 Cambios en la API

A continuación son listadas los cambios en la API C introducidas en Lua 5.1:

- las funciones luaopen_* (para abrir bibliotecas) no pueden ser llamadas directamente, como una función C común. Deben ser llamadas a través de Lua, como una función Lua;
- la función lua_open fue reemplazada por *lua_newstate* para permitir que el usuario defina una función de asignación de memoria. Se puede usar *luaL_newstate* de la biblioteca estándar para crear un estado con una función de asignación estándar (con base en realloc);
- las funciones luaL_getn y luaL_setn (de la biblioteca auxiliar) están ultrapasadas y no deben usarse. Usar lua_objlen en vez de lua_getn y nada en lugar de lua_setn;
- la función luaL_openlib fue reemplazada por *luaL_register*;
- la función luaL_checkudata ahora dispara un error cuando el valor suministrado no es un objeto userdata del tipo esperado (en Lua 5.0 retornaba NULL).

B.8 Sintaxis completa de Lua

La sintaxis completa de Lua en la notación BNF extendida es la siguiente (no describe las precedencias de los operadores):

```
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] del 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 puede ser importada por cualquier 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>

</connectorBase>

</head>

</ncl>
```

Bibliografía

[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