4
Arquitetura e Implementação

O capítulo anterior apresentou dois padrões de metadados que podem ser utilizados por aplicações de Guias Eletrônicos. Para compreender como os metadados são tratados neste trabalho, este capítulo tem a finalidade de descrever a implementação da arquitetura da aplicação adaptativa de Guia Eletrônico, proposta em [Moreno *et al.*, 2009] e revista nesta dissertação. Como tal arquitetura é a base deste trabalho, preferiu-se apresentá-la neste capítulo, na Seção 4.1 já com as melhorias realizadas.

O capítulo está organizado da seguinte forma. A Seção 4.1 apresenta a arquitetura da aplicação adaptativa de Guias Eletrônicos, enquanto as demais seções descrevem a implementação realizada.

4.1.
Arquitetura de Aplicação Adaptativa de Guia Eletrônico

É importante salientar que, para esta dissertação, a aplicação do Guia Eletrônico, baseada na arquitetura descrita em [Moreno *et al.*, 2009], é chamada de *adaptativa*, já que oferece suporte a adaptações, incluindo atualizações, de seus algoritmos em tempo de exibição. A arquitetura é ilustrada na Figura 17.

PUC-Rio - Certificação Digital Nº 0812554/CA

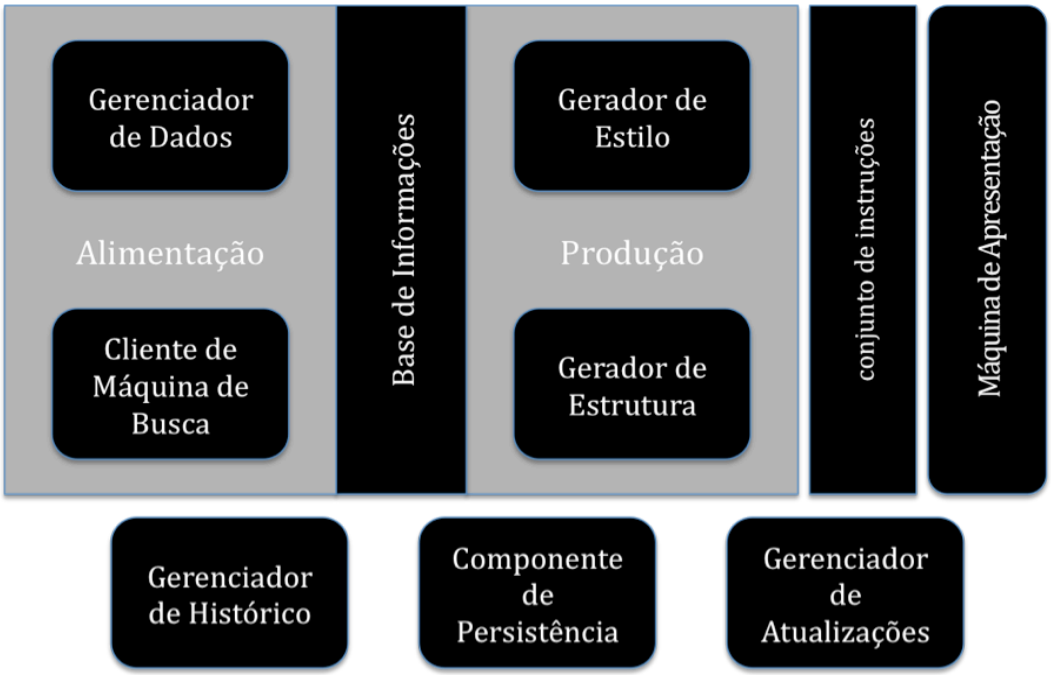

Figura 17 – Arquitetura de aplicação adaptativa de Guia Eletrônico

Como por diversas vezes mencionado, o objetivo da arquitetura é gerar um Guia Eletrônico permitindo que adaptações possam ser feitas na própria arquitetura, sem a necessidade da substituição integral da aplicação, em tempo de exibição do Guia. Para tanto, a arquitetura é separada em componentes e a geração do Guia é dividida em duas fases.

A primeira fase, chamada de Alimentação, tem como objetivo capturar e processar os metadados, que podem ser obtidos tanto por difusão quanto sob demanda. Uma vez adquiridos e processados, o Cliente de Máquina de Busca (controlado pelo Gerenciador de Dados) pode usar os metadados para buscar, em servidores de busca na Internet, por exemplo, mais informações sobre os serviços oferecidos. O Gerenciador de Dados pode filtrar os metadados obtidos, de acordo com as preferências do telespectador, se um sistema de recomendação for adotado. A fase de alimentação termina quando o Gerenciador de Dados salva os metadados em uma estrutura de dados interna da arquitetura chamada de Base de Informações.

A Base de Informações desempenha um papel fundamental na arquitetura, que é desacoplar a fase de Alimentação da etapa seguinte, chamada de fase de Produção. É justamente essa separação que possibilita as atualizações dos componentes de cada fase sem acarretar problemas na exibição do Guia, já que os

PUC-Rio - Certificação Digital Nº 0812554/CA

metadados processados antes de uma atualização estão salvos na Base de Informações.

A fase de Produção utiliza os metadados salvos para gerar um conjunto de instruções, através de dois componentes: Gerador de Estilo e Gerador de Estrutura. O Gerador de Estilo é responsável por definir o estilo visual do Guia Eletrônico. Por exemplo, o Guia pode ser uma tabela, onde a primeira coluna contém os horários de início dos eventos, enquanto a segunda coluna apresenta o nome dos eventos. Essa tabela também pode ser organizada de outra maneira: a primeira linha de cada coluna mostra um determinado horário, enquanto a segunda linha apresenta o nome do evento que será exibido no respectivo horário. Em outras palavras, a função do Gerador de Estilo é justamente gerar as instruções que definem as formas de apresentação das informações do Guia. Já o Gerador de Estrutura é responsável por gerar as instruções que definem o comportamento do Guia Eletrônico. Por exemplo, quando o telespectador selecionar um determinado evento, o Guia pode exibir uma breve sinopse. O Gerador de Estrutura pode também ser implementado para que, à medida que o telespectador navegue pelos eventos, breves sinopses sejam exibidas e, apenas quando o telespectador selecionar um evento específico, informações detalhadas desse evento sejam apresentadas.

É necessário existir um Apresentador ou Máquina de Apresentação para executar o conjunto de instruções gerado pela fase de Produção. O resultado dessa execução é a exibição do Guia Eletrônico. Assim, o conjunto de instruções e sua Máquina de Apresentação são componentes distintos da arquitetura. A Máquina de Apresentação é um dos componentes da arquitetura que usualmente não pode ser atualizado em tempo de exibição, pelo menos em sua totalidade [Moreno, 2010]. Cabe observar também que a Máquina de Apresentação é específica para um determinado tipo de conjuntos de instruções. Ou seja, caso a fase de Produção gere um conjunto de instruções não suportado pela Máquina de Apresentação, o Guia Eletrônico não será exibido.

O Gerenciador de Atualizações é o meta-serviço responsável por receber os pedidos de atualização de cada componente e por manter versões-padrão de todos os componentes. Para cada componente atualizado, o Gerenciador de Atualizações verifica se as interfaces padronizadas foram respeitadas. Ele é, também,

PUC-Rio - Certificação Digital Nº 0812554/CA

responsável por garantir que a consistência do sistema seja mantida após uma atualização.

Outro componente da arquitetura é o Gerenciador de Histórico, responsável por manter um histórico dos metadados capturados e processados pela fase de Alimentação. O objetivo desse componente é permitir que diferentes aplicações, como sistemas de recomendação, consultem tal histórico. Diferente dos trabalhos de Guias Eletrônicos personalizados discutidos no Capítulo 2, a arquitetura não é focada na criação de um sistema de recomendação, mas oferece suporte à adoção de um sistema de recomendação.

Por fim, o Componente de Persistência é utilizado para manter dados salvos após o termino da execução da aplicação. Por exemplo, as preferências do telespectador podem ser salvas por esse componente. Então, caso um sistema de recomendação seja adotado, esse componente pode ser o gerador de Perfil do telespectador, possibilitando a sua substituição isolada, isto é, sem a necessidade de substituir as técnicas de recomendação empregadas, nem mesmo os demais componentes da aplicação.

PUC-Rio - Certificação Digital Nº 0812554/CA

4.2. Aplicação Adaptativa de Guia Eletrônico utilizando o Ginga-NCL

Em um sistema de TV digital por radiodifusão, quando um canal é sintonizado, o receptor demultiplexa o fluxo de transporte para decodificar os fluxos elementares de áudio e vídeo, apresentando o conteúdo ao telespectador. A qualquer instante, o telespectador pode solicitar o Guia Eletrônico, que é gerado por uma aplicação.

Geralmente, em um sistema de TV digital por difusão terrestre, os receptores já possuem uma aplicação de Guia Eletrônico embutida e desenvolvida pelo próprio fabricante do receptor. Como não há uma padronização acerca de Guia Eletrônico, um fabricante desenvolve uma aplicação específica para o seu receptor, da maneira que desejar. Com isso, apesar dos Guias Eletrônicos serem usados para promover as opções de conteúdos oferecidos, os provedores de conteúdo não possuem nenhum controle sobre as aplicações geradoras de Guia Eletrônico embutidas no receptor.

Além do fluxo de áudio e vídeo, aplicações interativas desenvolvidas pelo provedor de conteúdo podem estar multiplexadas no fluxo de transporte. Diferente da aplicação de Guia Eletrônico desenvolvida para um receptor específico, essas aplicações interativas utilizam o conjunto de APIs fornecido pelo *middleware*. Como as APIs são padronizadas pela norma do sistema de TV, elas obrigatoriamente precisam ser respeitadas pelo *middleware*, independente do fabricante do receptor. Então, se uma aplicação interativa faz o uso das APIs conforme a norma, é esperado que ela execute com sucesso em todas as implementações de *middleware* que seguem essa mesma norma. Com isso, a aplicação desenvolvida é portável para todos os receptores que utilizam o *middleware* para o qual ela foi desenvolvida.

Uma diferença entre uma aplicação de Guia Eletrônico e uma outra aplicação interativa é a utilização dos dados das tabelas SI. Nesse aspecto, as APIs fornecidas pelo *middleware* normalmente definem como as aplicações podem requisitar esses dados. Da mesma maneira, a forma como o *middleware* responde às solicitações também é definida pela norma do sistema de TV. Assim, nada restringe que o Guia Eletrônico seja gerado por uma aplicação de TV interativa que utiliza os recursos de um *middleware* específico.

Independente se a aplicação geradora do Guia é residente ou não, a abordagem de utilizar os recursos de um *middleware* para gerar o EPG pode trazer alguns benefícios para o seu desenvolvedor. Primeiro, como mencionado, pode ser desenvolvida uma única aplicação de Guia para todos os sistemas de TV que utilizam o mesmo *middleware*. Segundo, o desenvolvimento de uma aplicação de Guia que utiliza um *middleware* pode ser considerado mais simples, já que o desenvolvedor não precisa possuir conhecimentos específicos sobre o receptor. Finalmente, destaca-se que uma aplicação interativa desenvolvida para um *middleware* pode estar residente no receptor, mas também pode ser enviada por difusão ou ainda ser adquirida sob-demanda, o que torna o Guia Eletrônico uma aplicação mais flexível.

Conforme mencionado ao longo desta dissertação, as aplicações de Guias Eletrônicos conhecidas podem ser substituídas esporadicamente, através de uma atualização da aplicação como um todo, mas não podem ser adaptadas em tempo de exibição.

PUC-Rio - Certificação Digital Nº 0812554/CA

O suporte à adaptabilidade em tempo de exibição permite um maior controle do Guia Eletrônico pelos provedores de conteúdo. A adaptabilidade em tempo de exibição do Guia, como proposto na arquitetura, permite que os provedores de conteúdo modifiquem a aplicação, adicionando, removendo ou alterando funcionalidades de acordo com seus interesses, sem interromper a execução da aplicação e, assim, sem parar a apresentação do EPG. Já a modularização da aplicação, como também é proposto na arquitetura, possibilita, por exemplo, que diferentes componentes sejam compartilhados entre as próprias emissoras e desenvolvedores. Além disso, a modularização da aplicação e o seu suporte à adaptabilidade possibilitam adaptações em componentes específicos sem a necessidade de substituir a aplicação inteira. Como consequência de tais vantagens, os telespectadores podem se beneficiar, uma vez que uma aplicação de Guia com mais recursos possibilita a geração de um Guia Eletrônico mais atrativo.

Como prova de conceito, este trabalho implementa a arquitetura de aplicação adaptativa de Guias Eletrônicos, apresentada na Seção 4.1. Como a arquitetura de aplicação adaptativa de Guias Eletrônicos é genérica, ou seja, não está restrita à linguagem de programação, é possível implementá-la através de uma aplicação interativa junto a um *middleware*, tirando proveito das vantagens mencionadas anteriormente. Para atingir os objetivos desta dissertação, foi implementada uma aplicação para o ambiente declarativo do *middleware* Ginga [Moreno, 2010].

Quando o Guia Eletrônico é gerado por uma aplicação interativa, essa aplicação e o *middleware* que oferece os recursos necessários à sua execução formam juntos a aplicação do Guia. Dessa forma, a aplicação do Guia Eletrônico adaptativa em tempo de exibição, resultado da implementação da arquitetura, é composta por uma aplicação interativa desenvolvida na linguagem NCL (*Nested Context Language*) [Soares & Rodrigues, 2006] e pelo *middleware* Ginga, que é responsável por executar a aplicação NCL, gerando a apresentação do EPG. Por isso, conforme será detalhado neste capítulo, além de todo o desenvolvimento da aplicação NCL, também faz parte da implementação da arquitetura a adição de funcionalidades ao Ginga.

A NCL é uma linguagem declarativa que possui o foco no sincronismo entre objetos de mídias, tratando a interatividade como um caso particular do sincronismo. Por essa razão, diferente das linguagens baseadas em XHTML

PUC-Rio - Certificação Digital Nº 0812554/CA

[W3C, 2002], os relacionamentos espaciais e temporais são especificados de forma declarativa, não sendo necessária a adição de códigos imperativos. Na implementação da arquitetura, essa característica da linguagem possibilita, por exemplo, a fase de Produção gerar um conjunto de instruções totalmente declarativo.

Caso o conjunto de instruções gerado seja totalmente declarativo em NCL, as regiões e descritores NCL compõem o leiaute, de forma separada da estrutura, que é definida pelos objetos de mídia, contextos e seus relacionamentos. Isso permite, como a arquitetura propõe, a implementação separada dos componentes Gerador de Estilo e Gerador de Estrutura.

Além do suporte ao sincronismo, NCL oferece suporte declarativo à adaptabilidade de conteúdo e apresentação. Essa característica da linguagem pode ser utilizada no cenário onde se deseja que a apresentação do Guia seja adaptada em tempo de exibição. Por exemplo, pode ser interessante que o Guia seja apresentado com diferentes leiautes pré-definidos de acordo com o horário. É justamente através de regras, *switch* de objetos de mídia/contextos e *switch* de descritor da NCL que o conjunto de instruções gerado pela fase de Produção pode ter tal comportamento. Ressalta-se que essa vantagem da linguagem permite apenas adaptações no Guia e não na aplicação geradora do Guia.

Outra vantagem da NCL para um Guia Eletrônico é o suporte a múltiplos dispositivos [Costa *et al.*, 2009], possibilitando a exibição de um Guia Eletrônico gerado por uma única aplicação em diferentes dispositivos exibidores.

Os parágrafos anteriores apresentam vantagens da linguagem NCL, com a finalidade de mostrar que o amplo suporte declarativo oferecido pela linguagem NCL traz facilidades ao desenvolvedor de uma aplicação de Guia Eletrônico, evitando muitas vezes a utilização de códigos imperativos. Como consequência, a implementação da arquitetura também torna-se mais fácil. No entanto, a natureza da implementação da arquitetura exige também uma programação algorítmica. Como exemplos, tanto na fase de Alimentação, onde os metadados são adquiridos e processados, quanto na Produção, onde os metadados são preparados para a apresentação, uma programação imperativa é necessária.

Nos casos em que uma aplicação puramente declarativa não é suficiente, como na implementação da arquitetura proposta, a NCL, como uma linguagem de cola, permite a adição de códigos imperativos como objetos de mídia embutidos.

PUC-Rio - Certificação Digital Nº 0812554/CA

Como Lua [Ierusalimschy, 2006] é a linguagem de script adotada pelo Ginga-NCL, a aplicação NCL desenvolvida possui diferentes objetos Lua, também chamados de objetos NCLua [Sant'Anna *et al.*, 2008], que implementam alguns dos componentes da arquitetura. Para os componentes Gerenciador de Dados, Cliente de Máquina de Busca, Base de Informações, Gerador de Estilo, Gerador de Estrutura e Gerenciador de Histórico há um objeto NCLua embutido na aplicação NCL.

O Guia Eletrônico, como proposto na arquitetura, é gerado pela execução da aplicação NCL. A função de executar aplicações NCL, gerenciando toda a apresentação, pertence à Máquina de Apresentação Ginga-NCL, que faz parte do ambiente declarativo do *middleware* Ginga. O *middleware* Ginga, como ilustrado na Figura 18, possui uma arquitetura modular formada por dois subsistemas lógicos obrigatórios: a Máquina de Apresentação Ginga-NCL e o Núcleo Comum [Moreno, 2010].

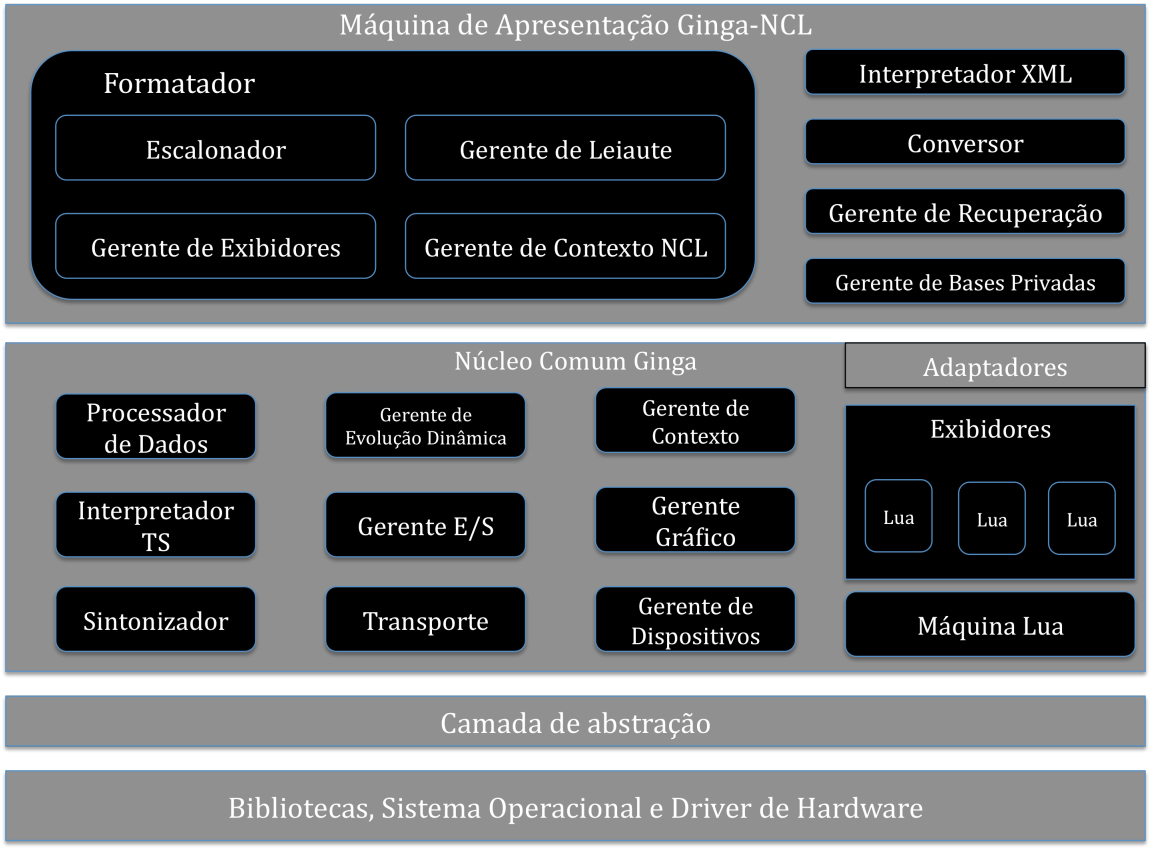

Figura 18 – Arquitetura Ginga. Adaptada de [Moreno, 2010].

Como todas as aplicações interativas desenvolvidas em NCL, a aplicação NCL da implementação da arquitetura é iniciada e controlada pela Máquina de Apresentação Ginga-NCL, cujo núcleo é o Formatador. De posse da aplicação

PUC-Rio - Certificação Digital Nº 0812554/CA

NCL, o Formatador solicita aos módulos Interpretador XML e Conversor para que as entidades declarativas da aplicação sejam interpretadas e convertidas em estruturas de apresentação, que são armazenadas, até o término da execução da aplicação, numa estrutura chamada Base Privada. Essa estrutura é controlada pelo Gerente de Bases Privadas.

Posteriormente, o Formatador, através do módulo Escalonador, orquestra a apresentação da aplicação NCL. Como a aplicação NCL da arquitetura possui objetos NCLua embutidos, é necessário instanciar um exibidor NCLua para cada um dos objetos. Já que o módulo Exibidores, que contém os decodificadores dos diferentes tipos de mídias (incluindo códigos imperativos em NCLua), faz parte do Núcleo Comum, essa etapa é realizada pelo Gerente de Exibidores mediante solicitações do Escalonador. Nessa mesma etapa, o módulo Adaptadores do Núcleo Comum também é importante, já que é responsável por definir uma API genérica, que tem a finalidade de padronizar a comunicação entre a Máquina de Apresentação e os decodificadores [Moreno, 2010]. É justamente essa API que permite a integração entre os dois subsistemas lógicos, Máquina de Apresentação Ginga-NCL e Núcleo Comum, preservando o baixo acoplamento entre eles.

É possível que a aplicação NCL da arquitetura esteja residente no receptor, assim como seja transmitida integral ou parcialmente por radiodifusão pelos provedores de conteúdo, ou ainda ser adquirida sob-demanda, como já mencionado. Qualquer que seja a opção, é necessário um componente que seja responsável por entregar a aplicação NCL para o Formatador. Essa é uma das funções do Núcleo Comum.

O Núcleo Comum foi definido para prover serviços comuns de TV digital tanto ao ambiente declarativo quanto ao ambiente imperativo do *middleware* Ginga. Entre suas funções está a sintonia do canal, realizada pelo módulo Sintonizador, permitindo o recebimento do fluxo de transporte. Após a recepção, o fluxo de transporte é demultiplexado pelo módulo Interpretador TS, enquanto os dados do fluxo são processados pelo módulo Processador de Dados. Cabe salientar a importância dos módulos Sintonizador, Intepretador TS e Processador de Dados na aplicação do Guia Eletrônico, já que são os responsáveis por obter os metadados multiplexados no fluxo de transporte.

Como mencionado na Seção 4.1, a comunicação entre os componentes da arquitetura é indispensável para a geração do Guia. Por exemplo, quando todos os

PUC-Rio - Certificação Digital Nº 0812554/CA

metadados já foram adquiridos e filtrados, o Gerenciador de Dados precisa armazená-los na Base de Informações. Posteriormente, a fase de Produção precisa obter esses metadados armazenados para gerar o conjunto de instruções. Dessa forma, é necessário que os objetos NCLua, que implementam alguns componentes da arquitetura, comuniquem-se, passando valores de um para o outro.

A comunicação entre um objeto NCLua com uma outra entidade é realizada através da API de eventos (módulo *event*). Entretanto, essa API não permite que um objeto NCLua poste eventos de forma direta aos demais objetos definidos na aplicação. Assim, para possibilitar a comunicação entre objetos NCLua, é também responsabilidade do Formatador fornecer um mecanismo para troca de eventos entre esses objetos. A utilização de tal mecanismo precisa ser explícita no documento NCL, através da definição de pelo menos uma âncora de propriedade (elemento <property>) em cada um dos objetos NCLua, assim como a definição de um elo responsável por permitir a cópia do valor de uma propriedade para outra.

Para facilitar a explicação desse mecanismo, é utilizada como exemplo a comunicação que ocorre entre o NCLua Gerenciador de Dados e o NCLua Base de Informações, com a finalidade de salvar os metadados adquiridos na fase de Alimentação. Como a passagem de valores entre nós NCLua é mediada pelo Formatador, é necessária a utilização de interfaces entre cada objeto NCLua e a aplicação NCL. É a função de uma propriedade de um objeto de mídia fazer a interface do objeto com a aplicação NCL. Com isso, na aplicação NCL da arquitetura, são definidas as propriedades `saveInfo`, do NCLua Gerenciador de Dados, e `getInfo`, do NCLua Base de Informações. A Figura 19 ilustra o código NCL da definição dos objetos NCLua Gerenciador de Dados (*DataManager*) e Base de Informações (*InfoBase*) com a definição das propriedades.

```
<media id="DataManager" src="dataman.lua" >
      <property name="saveInfo "/>
</media>

<media id="InfoBase" src="infobase.lua" >
      <property name="getInfo "/>
</media>
```

Figura 19 – Definição dos objetos NCLua Gerenciador de Dados e Base de Informações

Assim que os metadados estão prontos para serem salvos, o NCLua Gerenciador de Dados posta um evento de atribuição, contendo os metadados a

PUC-Rio - Certificação Digital Nº 0812554/CA

serem passados para o NCLua Base de Informações. Em Lua, um evento é representado por uma tabela com diferentes variáveis ou campos, que são definidos com a finalidade de descrever adequadamente cada evento. Isso possibilita criar formas de tratamento específicas para cada evento. Entre esses campos, o campo *class* descreve a classe do evento, enquanto o campo *type* descreve o tipo. A Figura 20 ilustra o evento de atribuição postado pelo Gerenciador de Dados.

```
evt = {
    class = 'ncl',
    type  = 'attribution',
    name  = 'saveInfo',
    value = metadata
}
evt.action = 'start'; event.post(evt)
evt.action = 'stop';  event.post(evt)
```

Figura 20 – Postagem do evento de atribuição pelo NCLua Gerenciador de Dados

Conforme mostra a Figura 20, o evento postado pelo Gerenciador de Dados é representado pela tabela evt. O campo *class* dessa tabela recebe o valor 'ncl', identificando que o evento pertence à classe ncl e, assim, o evento é passado para o Formatador, que é responsável pelo seu tratamento. A tabela evt ainda conta com os campos *type*, que possui o valor 'attribution', para identificar que o evento é de atribuição; *name*, que identifica a propriedade do Gerenciador de Dados que sofrerá a atribuição (`saveInfo`); e *value*, que possui o valor a ser passado ao NCLua Base de Informações. Nas duas últimas linhas do trecho de código ilustrado na Figura 20, o evento é postado, através da função *event.post()*, indicando o início (`evt.action='start'`) e o término da atribuição (`evt.action='stop'`). Como consequência, o valor do campo *value* da tabela evt é atribuído à propriedade `saveInfo`.

Para ser notificado da ocorrência de um evento, um objeto NCLua precisa registrar um tratador (*handler*), através da função *event.register()*. Assim, na passagem de valores utilizada como exemplo, o NCLua Base de Informações registra um tratador de eventos de atribuição na propriedade `getInfo`. Como pode ser notado na Figura 21, o tratador é responsável pela chamada à função *saveToInfoBase()*, passando o valor do campo *value* do evento recebido como parâmetro. A função *saveToInfoBase()* é responsável por salvar os metadados.

PUC-Rio - Certificação Digital Nº 0812554/CA

```
function metadataHandler(evt)
    if evt.class  ~= 'si'          then return end
    if evt.type   ~= 'attribution' then return end
    if evt.name   ~= 'getInfo'     then return end

    saveToInfoBase(evt.value)
end

event.register(metadataHandler)
```

Figura 21 – Tratador de eventos de atribuição na propriedade getInfo definido no NCLua Base de Informações

A definição das âncoras de propriedade, a postagem de um evento por um NCLua e o registro do tratador de evento por um outro NCLua, como descrito nos parágrafos anteriores, não são suficientes para que a comunicação entre os objetos seja realizada. A troca de informações entre os objetos NCLua é realizada pelo Formatador através da cópia do valor de uma propriedade para outra. O valor a ser passado foi atribuído à propriedade `saveInfo` através da execução do código apresentado na Figura 20. Para que a comunicação entre os objetos NCLua seja efetuada com sucesso, falta a definição de um elo na aplicação NCL que permita a cópia do valor da propriedade `saveInfo` do NCLua Gerenciador de Dados para a propriedade `getInfo` do NCLua Base de Informações. A Figura 22 ilustra o código NCL da definição desse elo.

```
<link id="lsaveMetadata" xconnector="onEndAttributionSet">
    <bind role="onEndAttribution" component="DataManager"
interface="saveInfo"/>
    <bind role="set" component="InfoBase" interface="getInfo">
         <bindParam name="var" value="$get"/>
    </bind>
    <bind role="get" component="DataManager"
interface="saveInfo"/>
</link>
```

Figura 22 – Elo NCL definido para copiar o valor da propriedade saveInfo para a propriedade getInfo

O elo ilustrado na Figura 22 é executado pelo Formatador, assim que uma atribuição na propriedade `saveInfo` do objeto Gerenciador de Dados terminar. No exemplo, como mencionado, isso ocorre justamente quando o NCLua Gerenciador de Dados executa o código de postagem do evento de atribuição nessa propriedade, apresentado na Figura 20. Com isso, a associação do papel "get" com tal propriedade (definida pelo terceiro elemento <bind> do elo) permite consultar e armazenar o seu valor na variável "$get". Posteriormente, o conteúdo da variável "$get" é copiado para a propriedade `getInfo` do NCLua Base de

PUC-Rio - Certificação Digital Nº 0812554/CA

Informações, através da associação com o papel "set" (definida pelo segundo elemento <bind> do elo). Como consequência, o Formatador notifica o decodificador do NCLua Base de Informações, que se encarrega de gerar um novo evento de atribuição na propriedade `getInfo`, estando o novo valor dessa propriedade no campo *value* da tabela do evento. Finalmente, esse novo evento é passado, recebido e processado pelo NCLua Base de Informações, através do tratador que foi criado, salvando os metadados como desejado.

4.3. Gerenciador de Dados

A primeira fase de geração do Guia Eletrônico proposto pela arquitetura é a fase de Alimentação, quando os metadados dos conteúdos oferecidos são adquiridos. Como o Ginga faz parte do Sistema Brasileiro de TV Digital Terrestre, é necessário que o Guia desenvolvido neste trabalho seja capaz de manipular os metadados transmitidos pelas emissoras. Como mencionado, no SBTVD-T, os metadados são transmitidos utilizando as tabelas SI, que foram apresentadas na Seção 3.1.

Na arquitetura de aplicação adaptativa de Guias Eletrônicos, o componente responsável por capturar os metadados das tabelas SI é o Gerenciador de Dados. Dessa forma, são funções do Gerenciador de Dados tanto o recebimento e a demultiplexação do fluxo de transporte, quanto o processamento das seções das tabelas SI.

No *middleware* Ginga, a recepção, decodificação e processamento de seções específicas do fluxo de transporte são funções dos módulos Sintonizador, Interpretador TS e Processador de Dados, respectivamente, do Núcleo Comum. Com isso, cabe destacar que, além do objeto NCLua chamado Gerenciador de Dados embutido na aplicação NCL da arquitetura, tais módulos fazem parte da implementação do componente Gerenciador de Dados proposto pela arquitetura de aplicação adaptativa de Guia Eletrônico.

Também é função do Gerenciador de Dados gerenciar toda a fase de Alimentação, controlando o Cliente da Máquina de Busca, para obtenção de mais metadados, assim como conhecer a estrutura interna utilizada pelo componente Base de Informações, com a finalidade de salvar os metadados obtidos. Os

PUC-Rio - Certificação Digital Nº 0812554/CA

metadados obtidos pelo Gerenciador de Dados são usados pela fase de Produção, para construir o Guia Eletrônico.

Na implementação realizada, mesmo que o módulo Sintonizador esteja recebendo um fluxo de transporte, as tabelas SI só são processadas quando há uma solicitação por tais metadados, que é passada para o módulo Processador de Dados. É função do NCLua Gerenciador de Dados fazer a solicitação aos metadados, assim que sua execução é iniciada. Tal solicitação é feita através da postagem de um evento.

Na API de eventos, a *classe si* foi definida para possibilitar que objetos NCLua acessem os metadados multiplexados no fluxo de transporte. Para cada tabela SI, foi definido um tipo de evento específico: tipo *epg*, para EIT; tipo *services*, para SDT; e o tipo *time*, para TOT. Para receber metadados de uma dessas tabelas, o NCLua deve postar um evento da classe si, com o tipo correspondente à tabela desejada. Para exemplificar, a Figura 23 ilustra um evento de requisição aos metadados da EIT. Ressalta-se que, a cada evento, só podem ser requisitados metadados de uma única tabela.

```
evt = {
     class ='si',
     type  = 'epg',
     stage = 'schedule',
}
event.post(evt)
```

Figura 23 – Evento de requisição de metadados da EIT do tipo *schedule*

A resposta para a solicitação ilustrada na Figura 23 é um evento contendo todos os metadados da EIT do tipo *schedule* do fluxo de transporte, que, como foi apresentado na Seção 3.1.1, podem ser muitos. Para que o NCLua Gerenciador de Dados não receba todos os metadados do fluxo, o evento de requisição pode especificar os metadados que devem ser inseridos no evento de resposta. Como exemplo, a Figura 24 ilustra um evento de requisição para os metadados hora de início (campo start_time), nome (campo name do descritor de evento curto) e descrição curta do evento (campo text_char do descritor de evento curto). Nesse caso, o evento de resposta contém apenas essas três informações, que, como pode ser observado nessa figura, foram solicitadas no campo *fields* do evento de requisição. A maneira como cada metadado deve ser especificado no campo *fields* é definida na norma [ABNT, 2009b].

PUC-Rio - Certificação Digital Nº 0812554/CA

```
evt = {
      class ='si',
      type  = 'epg',
      stage = 'schedule',
      fields = { startTime, name, shortDescription }
}
event.post(evt)
```

Figura 24 – Evento de requisição especificando três metadados da EIT

A classe si de eventos exerce um papel fundamental na implementação da arquitetura, já que ela é responsável por permitir a comunicação do objeto NCLua com os módulos do Núcleo Comum responsáveis pelo processamento dos metadados. Para possibilitar o tratamento de eventos da classe si, isto é, todo o processamento desde a postagem de um evento da classe si até o recebimento do evento de resposta contendo os metadados, novas funcionalidades tiveram de ser implementadas em diferentes módulos do Núcleo Comum[6]. Tais funcionalidades são descritas na próxima seção.

PUC-Rio - Certificação Digital Nº 0812554/CA

4.3.1. Implementações realizadas no Núcleo Comum

Na implementação de referência do Ginga, o módulo Interpretador TS (`gingacc-tsparser`) do Núcleo Comum é responsável por oferecer APIs para a manipulação do fluxo de transporte. Por exemplo, pode-se destacar o demultiplexador, chamado de *Demuxer*, que é responsável por interpretar cada pacote (manipulado através da API *TSPacket*) do fluxo de transporte recebido pelo módulo Sintonizador. O *Demuxer* possibilita que diversos usuários de sua API cadastrem-se como *listeners* de pacotes de fluxos específicos. Ao analisar um determinado o pacote, o *Demuxer* verifica se há *listeners* cadastrados para, por exemplo, o PID daquele pacote. Em caso verdadeiro, ele encaminha o pacote para todos os *listeners* que fizeram tal requisição. Essa funcionalidade é o que possibilita encaminhar um fluxo elementar para o seu decodificador. Especificamente para Guias Eletrônicos, tal funcionalidade é utilizada para filtrar os pacotes que encapsulam seções de tabelas SI, que são manipuladas através da API *TransportSection*.

[6] Neste trabalho, foi utilizada a implementação de referência do Ginga, disponível em www.softwarepublico.gov.br.

Com a finalidade de prover suporte à manipulação dos metadados das tabelas SI, o módulo de Interpretador TS precisou ser estendido, através do oferecimento de novas APIs. Dentre elas, destacam-se principalmente EventInfo e ServiceInfo, responsáveis por oferecer APIs de manipulação dos metadados de um evento da EIT e de um serviço da SDT, respectivamente. Foi adicionada também, para cada um dos descritores apresentados na Seção 3.1, uma API que oferece métodos de manipulação das informações encapsuladas nos campos definidos no descritor. As APIs oferecidas por EventInfo e ServiceInfo oferecem o método *process()*, com a finalidade de processarem, respectivamente, um evento e serviço. Na execução desse método, os descritores, quando encontrados, são devidamente instanciados. Para o processamento dos descritores, um método *process()* foi implementado em cada um deles. A Tabela 3 e a Tabela 4 apresentam, respectivamente, a API oferecida pela *EventInfo* e *ServiceInfo*. Note, nessas tabelas, que os descritores utilizados em cada evento ou serviço podem ser obtidos através do método *getDescriptors()*.

Tabela 3 – API oferecida pela EventInfo

Método	Descrição
getEventId()	retorna o eventId do evento
getRunningStatus()	retorna o estado de exibição
getFreeCAMode()	retorna *true*, se o evento possui acesso condicional ou retorna *false*, caso contrário
getStartTime()	retorna o horário de início do evento
getDuration()	retorna a duração do evento
getEndTime()	retorna o horário de término
getDescriptors()	retorna os descritores do evento
process()	processa o evento: recebe um conjunto de bits e faz a sua interpretação, preenchendo variáveis internas referentes aos campos do evento

PUC-Rio - Certificação Digital Nº 0812554/CA

Tabela 4 – API oferecida pela ServiceInfo

Método	Descrição
getServiceId()	retorna o serviceId do serviço
getEitScheduleFlag()	retorna *true*, se há EIT *schedule* no serviço, ou retorna *false*, caso contrário
getEitPresentFollowingFlag()	retorna *true*, se há EIT *present/following* no serviço, ou retorna *false*, caso contrário
getRunningStatus()	retorna o estado de exibição do serviço
getFreeCAMode()	retorna *true*, se o serviço possui acesso condicional ou retorna *false*, caso contrário
getDescriptors()	retorna os descritores do serviço
process()	processa o serviço: recebe um conjunto de bits e faz a sua interpretação, preenchendo variáveis internas referentes aos campos do evento

Especificamente para aplicação de Guia Eletrônico, a API fornecida pelo módulo Interpretador TS é utilizada diretamente pelo módulo Processador de Dados (`gingacc-dataprocessing`), para processar os metadados e retorná-los aos *listeners* solicitantes. O módulo Processador de Dados define um filtro de seções, chamado *SectionFilter*, responsável por retornar seções específicas do fluxo de transporte. Na aplicação de Guia Eletrônico, o filtro de seções é utilizado para filtrar seções de cada tabela SI solicitada. Para oferecer o filtro de seções de tabelas SI, o principal objeto do módulo Processador de Dados, denominado *DataProcessor*, cadastra-se como *listener* do *Demuxer* para pacotes com PIDs 0x11 (SDT), 0x12 (EIT), 0x29 (CDT) e 0x70 (TOT). Tais pacotes, quando recebidos, são passados para o filtro de seções, que se encarrega de obter e retornar as seções presentes no pacote. De posse de uma seção específica, o tratador dessa seção pode ser chamado.

Com o objetivo de fornecer uma API única para o processamento de seções SDT, EIT, CDT e TOT, o processador SI, denominado *SIProcessor*, foi

PUC-Rio - Certificação Digital Nº 0812554/CA

adicionado ao módulo Processador de Dados. Como pode ser observado na Tabela 5, a API do *SIProcessor* oferece métodos de decodificação específicos para cada um dos tipos de seção de tabelas SI. Por exemplo, o método *decodeEitSection()* recebe uma event_information_section e, para cada evento encontrado nessa seção, instancia uma *EventInfo* e chama o método *process()*, para processar o evento de TV (a partir deste ponto até o final desta seção, será chamado de evento de TV, para diferenciar de eventos Lua).

Tabela 5 – API oferecida pelo SIProcessor

Método	Descrição
getInstance()	Retorna a instância única do SIProcessor.
decodeSdtSection(TransportSection)	decodifica uma service_description_section.
decodeEitSection(TransportSection)	decodifica uma event_information_section.
decodeCdt(TransportSection)	decodifica uma common_data_section.
decodeTot(TransportSection)	decodifica uma time_offset_section.
addSIListener(SIListener)	adiciona um *listener* junto com sua requisição. Quando a requisição for processada, os metadados serão passados ao *listener*.
removeSIListener(SIListener)	remove o *listener* passado como parâmetro.

Para solicitar metadados, usuários da API do *SIProcessor* devem cadastrar-se como *listener* do *SIProcessor*, através do método *addSIListener()*. O *SIProcessor* processa e armazena os metadados multiplexados no fluxo, com a finalidade de gerenciar e responder todas as solicitações de metadados. Dentre os metadados processados, o *SIProcessor* elimina aqueles que não foram solicitados, permitindo responder solicitações específicas de objetos NCLua.

O Exibidor Lua (*LuaPlayer*), definido no módulo Exibidores (`gingacc-player`), foi modificado para se cadastrar como *listener* do *SIProcessor*, assim que um NCLua posta um evento da classe si. Para receber os metadados

PUC-Rio - Certificação Digital Nº 0812554/CA

processados pelo *SiProcessor*, o método *pushSIEvent()* foi implementado e adicionado à API oferecida pelo *LuaPlayer*. Assim, quando o processamento dos metadados solicitados termina, o *SIProcessor* chama o método *pushSIEvent()*, que é responsável pela criação do evento de resposta, isto é, um evento da classe si, do mesmo tipo do evento de requisição, contendo os metadados requisitados.

Cabe destacar que a forma como o evento de resposta deve ser estruturado e, assim, ser recebido por um objeto NCLua, é determinada pela norma [ABNT, 2009b]. Geralmente, o evento é estruturado por um conjunto de tabelas aninhadas.

Finalmente, a última etapa consiste em postar o evento de resposta para o NCLua. No entanto, o método de postagem *ext_postHash()* oferecido pelo *LuaEvent* não permite a postagem de tabelas aninhadas. Para solucionar esse problema, foi feita uma última modificação: a implementação do método *ext_postHashRec()*, que é chamado pelo método *pushSIEvent()* do *LuaPlayer*, postando o evento de resposta para o NCLua.

Os parágrafos anteriores apresentam as modificações que foram feitas nos módulos Interpretador TS, Processador de Dados e Exibidor Lua do Núcleo Comum, para possibilitar que o Guia da arquitetura adaptativa seja gerado através de uma aplicação declarativa em NCL com scripts NCLua. Para maior clareza e didática, as modificações foram apresentadas por módulo. No entanto, tal escolha não descreve a sequência de passos de todo o processamento realizado para que o NCLua Gerenciador de Dados receba os metadados, conforme desejado.

Para apresentar a sequência de passos do processo de alimentação dos metadados da tabela SI, será utilizada como exemplo uma requisição de metadados da EIT. De forma simplificada, os passos são:

1. Evento da classe si e do tipo epg é postado pelo NCLua Gerenciador de dados (Figura 23, página 60);
2. *LuaPlayer* cadastra-se como *listener* do *SIProcessor*, que notifica o *DataProcessor*;
3. *DataProcessor* cadastra-se como *listener* do *Demuxer* para os pacotes com PID 0x12;
4. *Demuxer* retorna os pacotes para o *DataProcessor*;
5. *SectionFilter* é executado para retornar as seções ao *DataProcessor*;
6. Para cada seção EIT, o método *decodeEitSection()*, definido no *SIProcessor*, é executado. Para cada evento de TV da seção, uma

PUC-Rio - Certificação Digital Nº 0812554/CA

classe *EventInfo* é instanciada e o evento de TV é processado através do método *process()*;

7. *SIProcessor* filtra os metadados de acordo com a solicitação;
8. O método *pushSIEvent()* do *LuaPlayer* é chamado pelo *SIProcessor*, passando os metadados como parâmetros. O método gera um novo evento da classe si e do tipo epg, contendo os metadados no campo *data*;
9. Método *ext_postHashRec()* do *LuaPlayer* é chamado para postar o evento;
10. NCLua Gerenciador de Dados recebe o evento de resposta com os metadados.

4.3.2. Utilização das Tabelas SI

Um aspecto importante da implementação da arquitetura realizada nesta dissertação é a utilização de metadados reais, isto é, multiplexados num fluxo de transporte transmitido por um provedor de conteúdo. Tais metadados foram obtidos através da recepção do sinal do Sistema Brasileiro de TV Digital Terrestre (SBTVD-T). Como o conjunto de tabelas SI é a principal forma de alimentação de uma aplicação de Guia desenvolvida para o SBTVD-T, é interessante mostrar como as tabelas estão sendo, no ato da escrita desta dissertação, utilizadas por algumas emissoras brasileiras, como por exemplo, Globo, SBT e Bandeirantes.

Na transmissão da EIT *present/following*, as emissoras sempre transmitem duas seções distintas. A seção que é transmitida primeiro possui número de seção (campo sectionNumber – ver Figura 4, página 29) igual a 0 e encapsula metadados do evento corrente (*present*). Já a outra seção possui o número de seção igual a 1, e transmite metadados do evento seguinte (*following*). Até o término da exibição do evento corrente, as mesmas seções são retransmitidas. Cabe mencionar que no padrão SI [ABNT, 2009a] não há restrições em relação ao número de eventos que podem ser descritos em cada seção. Além disso, uma especificação que provê informações tanto para a implementação, quanto para a utilização do padrão SI adotado pelo SBTVD, não foi encontrada. Entretanto, essa estruturação da EIT *present/following* é a mesma definida pela especificação

PUC-Rio - Certificação Digital Nº 0812554/CA

[ETSI, 2009c] de implementação e utilização do padrão DVB-SI. Apesar da norma brasileira referenciar o padrão DVB-SI [ETSI, 2010] diversas vezes, tais padrões possuem algumas diferenças.

A maior diferença atualmente entre as transmissões dos metadados pelos provedores de conteúdo é a utilização dos descritores. Conforme apresentado na Seção 3.1, descritores diferentes são definidos no padrão SI, provendo muitas informações sobre cada conteúdo. Alguns desses descritores precisam ser transmitidos sempre, enquanto que outros são de transmissão opcional. Dessa forma, os descritores opcionais deveriam ser utilizados apenas para transmitir informações adicionais, ou seja, informações que não foram transmitidas pelos descritores obrigatórios. No entanto, os descritores opcionais encontrados no fluxo de transporte transmitido pela emissora Globo não são utilizados de forma adequada. Já no fluxo de transporte da emissora SBT foram encontrados apenas descritores obrigatórios.

Possivelmente, o pior caso é a utilização do descritor de séries para todos os eventos da EIT *present/following* do fluxo de transporte da TV Globo. Todos os campos desse descritor são sempre preenchidos com o mesmo valor, independente do conteúdo descrito. Esse mesmo problema ocorre também com o descritor de conteúdo.

Outra utilização inadequada feita pela TV Globo é a do descritor de evento estendido, transmitido para todos os eventos sem adicionar informações valiosas, apesar de sua transmissão ser opcional. O descritor de evento estendido oferece campos que podem ser utilizados de diferentes maneiras. O caso típico é a descrição da lista de créditos do conteúdo. Além disso, o descritor possui um campo dedicado para prover uma descrição textual estendida do conteúdo. No entanto, na maioria das vezes, a Globo transmite tal descritor apenas para repetir o nome do evento, que já é transmitido pelo descritor de evento curto.

Em relação aos descritores obrigatórios, a emissora TV Globo também poderia utilizá-los de forma adequada. Por exemplo, ao invés de transmitir uma descrição curta no campo específico para tal função (text_char) do descritor de evento curto, como faz o SBT, a TV Globo, na maioria das vezes, transmite mais uma vez o nome do evento. Já a TV Bandeirantes só envia informações nesse campo quando o evento trata-se de um filme. Para o SBT, deve ser mencionado

PUC-Rio - Certificação Digital Nº 0812554/CA

que o descritor de componente e o descritor de componente de áudio, dois descritores obrigatórios, não são transmitidos para nenhum evento.

A estruturação da EIT s*chedule* contém um número maior de eventos, já que ela é responsável por transmitir eventos da grade de programação. O padrão exige que, nessa tabela, seja transmitida uma lista de eventos, incluindo eventos cuja exibição ocorre após a exibição do evento seguinte (transmitido na EIT *present/following*). Diferente da EIT *present/following*, em que as seções são sempre numeradas com 0 e 1, o número das seções da EIT *schedule* é sempre incrementado (excetuando retransmissões) de 8 em 8 até 255. Além disso, os provedores de conteúdo geralmente inserem em uma seção metadados referentes a três horas de programação.

É importante destacar que a emissora SBT transmite seções de uma única EIT do tipo *schedule* (seções identificadas com table_id 0x50), enquanto a emissora TV Globo transmite seções que correspondem a duas tabelas EIT *schedule* distintas, que são identificadas com table_id 0x50 e 0x58. O padrão SI reserva os identificadores 0x50 até 0x5F para EIT *schedule*. Por exemplo, uma emissora pode usar um identificador nesse intervalo para transmitir seções com eventos da grade com menos descritores. Já seções com mais descritores podem ser transmitidas utilizando outro identificador. É exatamente isso que faz a TV Globo, transmitindo alguns descritores nas seções com identificador 0x50, e apenas um descritor em cada seção identificada com 0x58. No entanto, nesse último caso, é transmitido o descritor de evento estendido apenas para informar o nome do evento, não adicionando informações relevantes para a construção do Guia Eletrônico. Com isso, no caso da Globo, bastaria a transmissão de uma única EIT *schedule*. Diferente da TV Globo, que transmite seções com identificadores diferentes de forma alternada, a emissora TV Bandeirantes utiliza dois identificadores da seguinte forma. Quando uma seção identificada com 0x50 atinge o número de seção 255, seções com identificador 0x51 começam a ser transmitidas, continuando a descrever a grade de programação a partir do último evento descrito pela última seção com table_id 0x50. Além disso, as seções table_id 0x51 continuam a encapsular os mesmos descritores utilizados pelas seções com o identificador 0x50. Desse modo, a utilização de dois identificadores pela TV Bandeirantes não caracteriza a transmissão de duas EITs *schedule* distintas.

PUC-Rio - Certificação Digital Nº 0812554/CA

Em relação à tabela SDT, as transmissões entre os provedores de conteúdo são similares. Apesar da SDT definir diversos descritores, as emissoras utilizam apenas o descritor de serviço, fornecendo (corretamente) o nome do serviço e do provedor de serviço. Também pode ser destacado que, na SDT dos fluxos de transporte sintonizados, não foi encontrado o descritor de transmissão de logotipos. Como consequência, nenhuma seção CDT foi encontrada nos fluxos de transporte e o Guia Eletrônico não exibe logo de nenhuma emissora.

Finalmente, a TOT é sempre transmitida pelas emissoras junto com o descritor de mudança de fuso horário.

Concluindo, as tabelas EIT e SDT, incluindo seus descritores, permitem fornecer um grande conjunto de informações sobre eventos e serviços, possibilitando a geração de Guias Eletrônicos ricos. No entanto, no cenário do SBTVD-T, as tabelas SI ainda não são utilizadas pelas emissoras de forma eficiente, tornando difícil ou quase impossível a geração de um EPG atrativo ao telespectador. Por exemplo, pode ser desenvolvida uma aplicação de Guia para o SBTVD-T que processe e apresente as informações do descritor de série e descritor de conteúdo. No entanto, o EPG resultante não será satisfatório, já que tais descritores não só apresentam informações inúteis, como apresentam informações erradas.

Cabe ressaltar que a utilização de aplicações de Guia com um sistema de recomendação baseado em filtragem de conteúdo alimentadas unicamente pelas tabelas SI do SBTVD-T pode ter um resultado insatisfatório, já que as emissoras ainda não estão fornecendo informações ricas sobre os conteúdos, assim como exige tal técnica de recomendação. Uma solução para esse problema pode ser a utilização da filtragem colaborativa ou a obtenção dos metadados sob demanda. No entanto, é provável que com o tempo essas informações sejam enriquecidas e corretamente configuradas uma vez que isso pode trazer benefícios para as próprias emissoras, como a possibilidade de implementação de sistemas de propaganda contextualizados.

PUC-Rio - Certificação Digital Nº 0812554/CA

4.3.3.
Objeto NCLua Gerenciador de Dados

Para iniciar o processo de construção do Guia Eletrônico, assim que iniciado, o Gerenciador de Dados posta eventos da classe si, solicitando metadados da tabelas EIT (*present/following* e *schedule*) e SDT.

Após receber os metadados da EIT, o Gerenciador de Dados aciona o NCLua Cliente da Máquina de Busca (a partir deste ponto, será chamado de CMB) para obter outras informações sobre os conteúdos. Como as máquinas de busca recebem de entrada uma consulta (*query*), cabe ao Gerenciador de Dados gerar e passar a consulta de busca ao CMB. Devido à falta de outros metadados mais informativos, como apresentado na Seção 4.3.2, apenas o nome do evento é utilizado como termo da consulta. Assim como proposto pela arquitetura, o Gerenciador de Dados recebe e processa as informações adquiridas pelo CMB.

Tanto o envio da consulta quanto o recebimento da resposta da busca são realizadas através de eventos de atribuição de propriedade. Para isso, foram definidas as propriedades `query` e `result`.

A solicitação da busca pode ser implementada da seguinte maneira. Para cada programa de TV, um evento de atribuição na propriedade `query`, contendo o nome de um programa no campo *value*, é gerado e postado pelo Gerenciador de Dados. Por sua vez, o CMB, a cada evento de atribuição recebido, processa a *query* e retorna o resultado através de um novo evento. No entanto, essa forma de implementação foi considerada ineficiente, já que muitos eventos são postados, mas poucas informações são inseridas em cada evento.

Para solucionar tal ineficiência, bastaria que o campo *value* dos eventos de atribuição postados fosse uma tabela Lua. Assim, o Gerenciador de Dados poderia enviar diversas consultas em um mesmo evento. Já o CMB, ao receber um evento, acessaria cada consulta da tabela e buscaria mais informações. Os resultados da busca de todas as consultas poderiam ser adicionados nessa tabela, que seria retornada, em um novo evento, para o Gerenciador de Dados. No entanto, como uma propriedade de um objeto de mídia é um elemento NCL, só é possível atribuir a uma propriedade um valor do tipo *string*. Por essa razão, antes de ser inserida no campo *value* dos eventos de atribuição, a tabela é transformada em

PUC-Rio - Certificação Digital Nº 0812554/CA

string. Na recepção do evento, o processo inverso precisa ser realizado, transformando a *string* em tabela.

É importante destacar que esse simples mecanismo também é utilizado para passar de uma só vez os metadados adquiridos ao NCLua Base de Informações. Cabe imaginar a grande quantidade de eventos, cada um contendo apenas um único metadado, que precisariam ser gerados e postados com a finalidade de salvar todos os metadados adquiridos durante a fase de Alimentação.

Ao final da fase de Alimentação, isto é, quando o Gerencidor de Dados recebe o evento com as informações adquiridas sob demanda pelo CMB, um evento de atribuição na propriedade `saveInfo` é postado pelo Gerenciador de Dados, para que todos os metadados sejam armazenados na Base de Informações. Porém, como o CMB utiliza serviços que podem estar indisponíveis, ou que podem possuir um alto tempo de resposta, foi implementado no Gerenciador de Dados um mecanismo de espera para receber as informações obtidas pelo CMB.

O mecanismo de espera foi implementado utilizando eventos da seguinte forma. Assim que o Gerenciador de Dados posta um evento de atribuição contendo consultas, ele também posta, para si mesmo, um evento da classe user, contendo o tempo decorrido desde o início da aplicação, obtido através da função *event.uptime()*. No tratador de eventos da classe user definido no Gerenciador de Dados, é verificado se a resposta do CMB já foi recebida. Se a resposta foi recebida, nada precisa ser feito, uma vez que os metadados já foram salvos. Caso contrário, é calculada a diferença entre o valor corrente retornado pela função *event.uptime()* e a informação de tempo inserida no evento recebido. Se tal diferença for maior ou igual ao tempo de espera estipulado, os metadados obtidos são enviados para a Base de Informações, desconsiderando quaisquer respostas recebidas após o tempo de espera. Se o tempo de espera ainda não esgotou, um novo evento com a hora corrente é postado novamente. Esses passos se repetem até a resposta do CMB ser recebida ou até esgotar o tempo de espera, que é igual a 3 segundos.

Cabe mencionar que esse mecanismo de espera, utilizando eventos da classe user, poderia ter sido implementado no CMB. Preferiu-se implementá-lo no próprio Gerenciador de Dados, com a finalidade de isolar os componentes da fase de Alimentação. Ou seja, na ocorrência de um erro qualquer (não apenas a indisponibilidade do servidor) que impeça o envio ou a recepção do resultado da

PUC-Rio - Certificação Digital Nº 0812554/CA

busca, o Gerenciador de Dados não é afetado e os dados já obtidos são devidamente salvos na Base de Informações.

Finalmente, a última função do Gerenciador de Dados é salvar os metadados na Base de Informações, postando um evento de atribuição na propriedade `saveInfo`. Os metadados de cada evento de TV armazenados na Base de Informações são: nome do provedor do serviço, identificador do serviço, nome do serviço, identificador do evento de TV, nome do evento de TV, hora de início, duração, hora de término, descrição curta e, quando recebido, o resultado da busca realizada pelo CMB.

4.4. Cliente de Máquina de Busca

O Cliente de Máquina de Busca (CMB) é um dos componentes com a implementação mais flexível, já que a arquitetura não faz restrições em relação ao servidor de máquina de busca utilizado pelo CMB. Com isso, o objeto NCLua pode ser implementado de diferentes maneiras, de acordo com o servidor de máquina de busca utilizado. Por exemplo, neste trabalho, foram implementados objetos NCLua que se conectam a servidores de busca na Internet (como Google e Yahoo), com a finalidade de retornar páginas HTML relacionadas ao conteúdo buscado.

Independente da sua implementação, o CMB recebe do Gerenciador de Dados as informações a serem buscadas e comunica-se com um servidor de máquina de busca. Se a busca for realizada com sucesso, o CMB encaminha o resultado ao Gerenciador de Dados. Dessa forma, o Gerenciador de Dados é o único componente da arquitetura que se comunica com o CMB.

Foram definidas duas propriedades no CMB, para possibilitar a comunicação com o Gerenciador de Dados: `query`, utilizada para receber a informação a ser buscada (*consulta*), e `result`, utilizada para retornar o resultado da busca.

Com a finalidade de processar as consultas, foi definido um tratador de eventos de atribuição na propriedade `query`. O tratador espera que o campo *value* de um evento recebido seja uma tabela convertida para *string*. Em cada posição (índice) dessa tabela deve haver um campo denominado *query*, responsável por

PUC-Rio - Certificação Digital Nº 0812554/CA

armazenar uma consulta de busca. A tabela pode conter outros campos definidos pelo Gerenciador de Dados, com a finalidade de gerenciar as requisições e respostas de busca.

De posse da tabela, o CMB faz solicitações a um servidor para que efetue a busca por cada consulta, com a finalidade de adquirir mais informações. Já para armazenar os resultados obtidos, é criado um campo chamado *result* em todas as posições da tabela, associando cada consulta com o resultado da sua busca. Outros campos, se existentes, são mantidos na tabela sem modificações. A tabela é, finalmente, convertida para *string* e inserida no campo *value* de um evento de atribuição na propriedade `result`. O evento é postado e, como consequência, as informações adquiridas são enviadas ao Gerenciador de Dados.

O CMB pode obter informações em padrões estruturados (como SI, XML-TVAnytime ou outros) ou pode extrair informações de dados não estruturados. É justamente essa flexibilidade do CMB que permite que a aplicação do Guia não fique dependente da utilização de um único padrão de metadados. Como consequência, não há restrição em relação ao tipo dos dados obtidos pelo CMB. Por exemplo, o CMB pode, como mencionado, ser implementado para se conectar a serviços de busca na Internet e retornar endereços de páginas HTML obtidos por tais serviços. Nada impede, entretanto, que o CMB seja implementado para consultar um banco de dados de programas televisivos, obtendo informações textuais, como sinopses, sobre cada consulta. Em outras palavras, o tipo dos dados obtidos sob demanda é definido pela implementação do CMB.

Apesar de uma aplicação de Guia Eletrônico normalmente manipular diversos metadados, o EPG deve apresentar cada metadado numa região da tela específica. Para isso, a aplicação do Guia define o tratamento específico de cada metadado. Essa diferença de tratamento também é válida para os dados obtidos pelo CMB. Para exemplificar, uma página HTML não deve ser apresentada da mesma maneira que informações textuais e, por isso, uma página HTML e informações textuais não devem possuir o mesmo tratamento.

Independente do tipo dos dados obtidos pelo CMB, os dados devem ser salvos pelo Gerenciador de Dados na Base de Informações. Posteriormente, a fase de Produção faz uma solicitação à Base de Informações, obtendo todos os metadados, com o intuito gerar instruções referentes à apresentação de cada um deles. No entanto, excetuando o resultado da busca, as informações tratadas pela

PUC-Rio - Certificação Digital Nº 0812554/CA

fase de Produção consistem de metadados típicos de Guias Eletrônicos (isto é, hora de início, nome do programa etc.), que têm o mesmo tratamento, independente de como foram obtidos. O mesmo não acontece com as informações adquiridas sob demanda, já que a forma de apresentação de tais informações pode diferir de acordo com a implementação do CMB. Como consequência, a fase de Produção não pode ser implementada de forma totalmente genérica, uma vez que é necessário gerar as instruções referentes ao estilo e à estrutura da apresentação dos dados adquiridos pelo CMB.

Como na implementação da arquitetura os componentes podem sofrer adaptações, a especificidade da fase de Produção em relação aos dados do CMB merece uma atenção ainda maior. Por exemplo, se o CMB for adaptado e o novo componente retornar um tipo de dado não tratado pela fase de Produção, a menos que a fase de Produção também seja adaptada para tratar tal tipo, os dados adquiridos pelo CMB não serão exibidos. Da mesma maneira, se a fase de Produção for adaptada, é necessário que os componentes atualizados continuem apresentando corretamente as informações que foram obtidas através do CMB.

Com a finalidade de identificar o tipo das informações obtidas pelo CMB e, com isso, possibilitar o tratamento diferenciado de cada tipo, um campo denominado *type* é inserido dentro de cada campo *result*. Por exemplo, uma das implementações do componente CMB utiliza o *site* Google como servidor de máquina de busca. Ao receber o resultado, o CMB executa sua função *parser()* para processar o conteúdo da página HTML retornada pelo servidor de busca, extraindo o título e o endereço da página HTML referenciada em cada resultado encontrado. Nesse caso, o campo *result* gerado consiste de uma tabela com os primeiros endereços de páginas HTML (e respectivos títulos) do resultado da busca da consulta. Para identificar tal tipo, o campo *type* possui o valor “links”.

Já um outro CMB implementado utiliza a enciclopédia Wikipédia para adquirir informações textuais. Ao receber a resposta do Wikipédia, o CMB realiza o processamento da página HTML, extraindo os primeiros parágrafos, que são armazenados no campo *result*. Para identificar as informações textuais, o campo *type* recebe o valor “text”. Ressalta-se que os componentes podem ser atualizados e, com isso, o tratamento de cada tipo pode ser modificado, assim como outros tipos podem ser definidos.

PUC-Rio - Certificação Digital Nº 0812554/CA

Além de páginas HTML e informações textuais, metadados típicos de Guia Eletrônico também podem ser obtidos sob demanda. Nessa linha, outro objeto CMB implementado acessa um servidor de metadados estruturados no padrão XML TV-Anytime.

Para possibilitar a apresentação de metadados utilizando o padrão XML TV-Anytime, foi realizada a implementação de um *parser* para esses documentos. Normalmente, cada tabela do padrão é separada em um documento específico. Para obtê-las, o CMB faz requisições ao servidor pelos documentos disponibilizados. De posse dos documentos, o *parser* é efetuado em cada um deles, interpretando as tabelas XML, apresentadas na Seção 3.2. Após o processamento de cada documento, os metadados extraídos de elementos e atributos XML são armazenados numa estrutura de dados interna do CMB. O último passo consiste em passar os metadados obtidos para o Gerenciador de Dados. Por essa razão, é necessário que o CMB realize uma conversão da sua estrutura interna para um formato conhecido pelo Gerenciador de Dados. Como o Gerenciador de Dados já conhece o formato utilizado pelo componente Base de Informações, tal formato foi adotado.

Todas as versões do CMB implementadas utilizam eventos da classe tcp, através da API *tcp.lua*, para a comunicação com o servidor.

PUC-Rio - Certificação Digital Nº 0812554/CA

4.5. Base de Informações

No processo de geração do Guia, o objeto NCLua Base de Informações exerce duas funções: armazenar os metatados adquiridos na fase de Alimentação e responder solicitações, uma vez que a fase de Produção precisa acessar os metadados armazenados.

Ao ser iniciado, o componente Base de Informações fica aguardado os metadados a serem salvos. Para salvá-los, é necessário que a Base de Informações receba um evento de atribuição na propriedade `getInfo`. Uma vez recebidos, os metadados são armazenados (pela função *saveToInfoBase()*) numa tabela chamada *infoBase*.

A tabela *infoBase* é utilizada para manter os metadados durante a execução da aplicação e possui a seguinte estrutura. Como é ilustrado na Figura 25, cada

posição (índice) da tabela *infoBase* refere-se a um determinado serviço. Alguns campos são utilizados para descrever os serviços, como *serviceName*, que fornece o nome do serviço. Como diversos eventos podem estar associados a um serviço, em cada posição da tabela também foi especificado um vetor denominado *events*. Em cada índice do vetor *events,* são armazenadas informações que descrevem um determinado evento de TV. Os principais campos definidos para descrever um evento são: *eventId*, *name*, *startTime*, *duration*, *endTime*, *shortDescription* e *onDemand*. O campo *onDemand* é uma tabela utilizada para armazenar as informações adquiridas sob demanda pelo Cliente da Máquina de Busca. A estrutura da tabela *onDemand*, com exceção do campo *text*, que deve estar sempre presente, é definida pelo Cliente de Máquina de Busca.

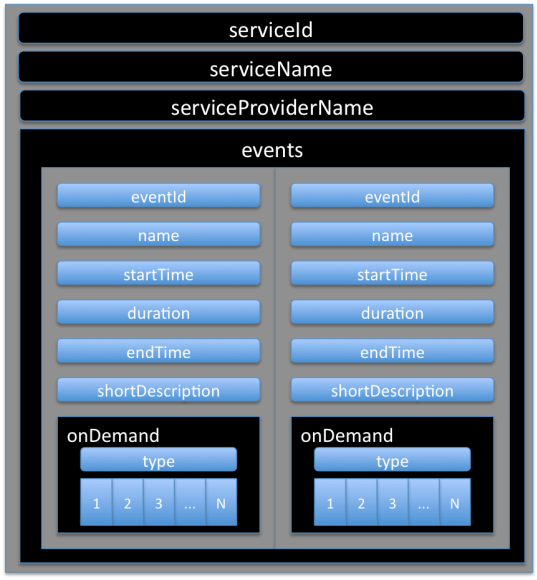

Figura 25 – Estrutura de uma posição da tabela infoBase

A requisição pelos metadados armazenados é implementada através de eventos de atribuição na propriedade `request` do objeto Base de Informações. Para permitir diferentes requisições utilizando a mesma propriedade, o campo *value* é utilizado para caracterizar a requisição. O principal tipo de requisição utilizada é para o conjunto de todos os metadados já armazenados na Base de Informações. Nesse caso, o campo *value* deve conter o valor “all”. Ao receber uma requisição do tipo “all”, o objeto Base de Informações posta um evento de atribuição na propriedade `production`, contendo todos os metadados armazenados (ou seja, uma cópia da tabela *infoBase*). Além disso, à medida que a

PUC-Rio - Certificação Digital Nº 0812554/CA

Base de Informações recebe novos metadados do Gerenciador de Dados, eles são passados à fase de Produção, com o intuito de mantê-la sempre atualizada.

Em outros tipos de requisições definidos, é necessário utilizar mais de uma variável para especificar a requisição. Nesses casos, o evento recebido deve possuir, no seu campo *value*, uma tabela (convertida para *string*) com diferentes campos. Entre eles, o principal campo, *type*, especifica o tipo da requisição. Como exemplo, também é possível requisitar metadados específicos, como metatados de todos os eventos de TV cujos horários de início e término da exibição são informados na própria requisição. Para tais requisições, o campo *type* deve conter o valor "time" e os campos *startTime* e *endTime* informam, respectivamente, a hora de início e a hora de término da exibição. Como consequência, serão passados à fase de Produção apenas os metadados dos eventos que possuem as horas de início e término da exibição dentro dos valores especificados nos campos *startTime* e *endTime* do evento de requisição.

É importante destacar que, para a geração do Guia Eletrônico, as funcionalidades descritas anteriormente poderiam ser implementadas no NCLua Gerenciador de Dados, já que, ao final do processo de Alimentação, tal objeto possui todos os metadados. No entanto, o que impede essa forma de implementação, obrigando o desenvolvimento de outro NCLua, fica por conta das adaptações que podem ser realizadas no Gerenciador de Dados. É justamente a separação do Gerenciador de Dados e Base de Informações em objetos NCLua distintos que possibilita adaptações em ambos os componentes da fase de Alimentação, sem resultar na perda de metadados adquiridos antes da adaptação.

É importante também salientar que o objeto NCLua Base de Informações só deve ser adaptado se o novo objeto possuir os metadados a serem apresentados. Caso contrário, o Guia Eletrônico pode não ser apresentado corretamente, uma vez que metadados armazenados (e que podem estar em apresentação) serão perdidos. Por outro lado, a adaptação em tempo de exibição na Base de Informações permite adaptações nos metadados, além de possibilitar mudanças na estrutura da tabela *infoBase*. A adaptação na estrutura da tabela *infoBase* pode ser interessante no cenário em que o provedor de conteúdo deseja apresentar metadados que antes não eram apresentados, necessitando então adotar uma estrutura diferente da utilizada anteriormente.

PUC-Rio - Certificação Digital Nº 0812554/CA

4.6. Fase de Produção, Conjunto de Instruções e Máquina de Apresentação

O objetivo da fase de Produção é tratar os metadados para gerar um conjunto de instruções que, ao ser executado (ou apresentado) pela Máquina de Apresentação, resulta na exibição do Guia Eletrônico.

O que define os algoritmos implementados em cada componente da fase de Produção é exatamente o conjunto de instruções a ser gerado. Em outras palavras, a implementação dos componentes da fase de Produção é específica para um determinado conjunto de instruções. Por essa razão, caso deseje-se gerar o conjunto de instruções em diferentes linguagens, diferentes componentes da fase de Produção precisam ser implementados.

Também é importante destacar que o conjunto de instruções é específico para a Máquina de Apresentação. Como consequência, a adaptabilidade na fase de Produção deve ser feita com cuidado, já que adaptações no Gerador de Estilo e Gerador de Estrutura podem resultar na geração de um conjunto de instruções numa linguagem diferente daquela suportada pela Máquina de Apresentação. Caso isso ocorra, o Guia não será exibido.

Neste trabalho, como o Guia adaptativo proposto pela arquitetura é gerado através de uma aplicação NCL, o Ginga-NCL é responsável por controlar a apresentação dessa aplicação, fazendo parte da implementação da arquitetura. Desse modo, os componentes da fase de Produção são implementados por dois objetos NCLua, possibilitando produzir, como será detalhado adiante, um conjunto de instruções totalmente declarativo em NCL. Tal abordagem tira proveito das vantagens da utilização da linguagem NCL para implementação de Guias Eletrônicos, mencionadas na Seção 4.2. Com isso, a Máquina de Apresentação Ginga-NCL é a Máquina de Apresentação do conjunto de instruções na implementação da arquitetura.

O Gerador de Estilo é responsável pelo processamento necessário para a geração das entidades declarativas referentes ao estilo do Guia Eletrônico. Ou seja, com o conjunto de instruções totalmente em NCL, o Gerador de Estilo tem a responsabilidade de criar as regiões e descritores de uma aplicação NCL. Os identificadores dos descritores criados são passados para o Gerador de Estrutura.

PUC-Rio - Certificação Digital Nº 0812554/CA

Posteriormente, o Gerador de Estrutura pode criar contextos e objetos de mídia, referenciando os descritores criados pelo Gerador de Estilo, através dos identificadores recebidos. O Gerador de Estrutura também é responsável por definir o sincronismo espaço-temporal entre os objetos de mídia através da criação de elos NCL.

Os metadados a serem apresentados, isto é, os metadados armazenados na Base de Informações, são recebidos pelo Gerador de Estrutura em um evento de atribuição na propriedade `receiveInfo`. Novos eventos são recebidos, assim que novos metadados forem salvos na Base de Informações. A cada recepção de um evento, o Gerador de Estrutura se encarrega de atualizar o Guia Eletrônico com as novas informações.

Para agrupar os objetos de mídia correspondente aos metadados, o Gerador de Estrutura adiciona um contexto com o identificador 'epgCtx' no elemento <body> da aplicação NCL da arquitetura. Todas as entidades NCL adicionadas pelo Gerador de Estrutura são inseridas nesse contexto.

O Guia Eletrônico gerado apresenta os metadados como uma lista, em que cada linha refere-se a um evento. Para cada evento, o Guia apresenta, a hora de início e o nome. Ao navegar pelos eventos, o Guia exibe, quando disponível, a descrição curta. Já se o telespectador selecionar o nome de um dos eventos, o Guia exibe, quando disponível, as informações adquiridas sob demanda pelo Cliente de Máquina de Busca.

Um dos cuidados tomados na fase de Produção foi o tratamento dos dados obtidos sob demanda. Relembrando, se as informações obtidas sob demanda não forem metadados típicos de EPG, elas estão salvas em uma tabela chamada *onDemand*, presente na estrutura de cada evento armazenado na Base de Informações. A fase de Produção foi implementada para esperar apenas dois valores no campo *type* de *onDemand*: "links" e "text".

No primeiro caso, as informações em *onDemand* devem estar divididas em dois vetores: *titles* e *links*. Enquanto *titles* armazena o título das páginas HTML obtidas, o vetor *links* armazena os endereços de tais páginas. No instante que o telespectador seleciona um evento de TV, a lista de eventos tem sua apresentação encerrada, para que outra lista, com os títulos das páginas, seja exibida. Nessa nova lista, quando a mídia referente ao título de uma das páginas é selecionada pelo telespectador, é disparada a apresentação da página HTML. Já o tipo *text*

PUC-Rio - Certificação Digital Nº 0812554/CA

possui um tratamento mais simples, uma vez que as informações textuais são armazenadas na única posição do campo *text* de *onDemand*. Nesse caso, ao selecionar o nome do evento, o texto é exibido no centro da tela, sobrepondo à lista de eventos.

Tanto o Gerador de Estilo quanto o Gerador de Estrutura criam todas as entidades NCL do conjunto de instruções utilizando comandos de edição ao vivo [Costa *et al.*, 2006].

Os comandos de edição foram especificados com a finalidade de permitir a edição ao vivo de uma aplicação NCL. Os comandos de edição podem ser disparados por objetos imperativos embutidos em uma aplicação NCL. Como toda aplicação NCL em apresentação está dentro da Base Privada, é responsabilidade do Gerente de Bases Privadas receber e processar os comandos de edição, acionando os outros módulos necessários para realizar as edições na aplicação NCL em apresentação.

Para disparar os comandos de edição, os objetos NCLua precisam utilizar a *classe edit* da API de eventos. O evento postado deve conter diferentes campos de acordo com o comando a ser disparado. Os campos obrigatórios para todos os eventos de comandos que adicionam elementos NCL são *command*, que identifica o comando; e *data*, que especifica o código NCL do elemento. Já os opcionais são *baseId* e *documentId*. O primeiro identifica o base privada, enquanto o segundo identifica o documento a ser modificado.

Como exemplo, para adicionar uma região, o Gerador de Estilo necessita postar um evento contendo obrigatoriamente os campos *command*, que identifica o comando *addRegion*; *regionBaseId*, que indica o identificador da base de regiões (atributo *id* do elemento <regionBase>) onde a região deve ser adicionada; e *data*, que fornece o código NCL da região. Por sua vez, o Gerador de Estrutura precisa postar um evento contendo os campos *compositeId* e *data*, com a finalidade de adicionar, no contexto identificado no campo *compositeId*, o objeto de mídia cujo código NCL está definido no campo *data*. Nesse caso, o campo *command* deve receber o valor *addNode*.

Em relação à utilização da classe edit da API de eventos, vale mencionar que, após postar um evento referente ao disparo de um determinado comando de edição, o objeto NCLua não recebe nenhum evento de resposta, informando se o processamento do comando foi realizado com sucesso ou se a ocorrência de

PUC-Rio - Certificação Digital Nº 0812554/CA

algum erro impossibilitou a edição da aplicação NCL. Isso merece destaque, já que, muitas vezes, diversos comandos precisam ser disparados em uma ordem específica e basta um erro em apenas um deles para que o conjunto de instruções seja gerado de forma incompleta, impossibilitando a apresentação do Guia Eletrônico.

Como os objetos NCLua da fase de Produção não recebem notificações de erro, quando comandos de edição são disparados em sequência e um deles falha, os eventos dos comandos seguintes continuam a ser postados, disparando comandos, que são processados desnecessariamente pelo Gerenciador de Base Privada. Como exemplo, suponha a apresentação da hora de um determinado evento de TV. Para isso, é necessário que o Gerador de Estilo crie uma região (comando *addRegion*) e um descritor (comando *addDescriptor*), postando dois eventos. Já o Gerador de Estrutura também posta dois eventos: um para adicionar o objeto de mídia (comando *addNode*) e outro para adicionar um elo (comando *addLink*), responsável por disparar a apresentação do objeto de mídia. Caso um erro impeça a adição da região especificada no comando *addRegion*, os seguintes eventos são postados sem necessidade, uma vez que, o processamento dos comandos seguintes também resultará em erro.

Na tentativa de minimizar o problema mencionado no parágrafo anterior e facilitar o disparo de comandos de edição pelos objetos NCLua da fase de Produção, foi criada uma API de comandos de edição em Lua (*command.lua*). Tal API define, no mínimo, uma função para cada comando de edição utilizado na criação do conjunto de instruções. Por exemplo, para cada comando de edição que adiciona elementos, foram definidas e implementadas tanto uma função que recebe o código NCL do elemento a ser adicionado quanto outra função cujos parâmetros correspondem aos atributos do elemento. Nesse último caso, a própria função é responsável pela geração do código NCL. Além disso, as funções encarregam-se de realizar uma verificação nos parâmetros recebidos e o evento disparador do comando só é postado, caso tais parâmetros passem nos testes verificadores.

Para exemplificar a utilização da API de comandos de edição *command.lua*, suponha a adição de um objeto de mídia, tarefa executada diversas vezes pelo Gerador de Estrutura. A adição de um objeto de mídia pode ser feita através da função *addNode()*, passando o código NCL como parâmetro, ou através da função

PUC-Rio - Certificação Digital Nº 0812554/CA

addMedia(), que recebe como parâmetros valores dos atributos *id*, *src* e *descriptor* do objeto de mídia a ser adicionado. Em ambas as funções, o evento só é postado, se o atributo *id* não foi utilizado antes e se a API já foi solicitada para adicionar o descritor identificado no parâmetro *descriptor*. Já que o disparo de comando de edição com parâmetros inválidos não resulta na edição da aplicação NCL, a API *command.lua* evita a postagem desnecessária de eventos e, consequentemente, o processamento de comandos de edição inválidos. Outro exemplo de utilização da API *command.lua* que pode ser mencionado é realizado no Gerador de Estilo. Muitas vezes, os elementos NCL, referentes às regiões e descritores, possuem diversos atributos. Para esses casos, foram criadas funções (*addRegionTable* e *addDescTable*) que, ao invés de receberem um parâmetro para cada atributo do elemento, numa ordem específica, recebem uma tabela Lua, onde cada campo corresponde a um atributo do elemento. As funções encarregam-se de ler os campos da tabela, gerar o código NCL do elemento, e postar o evento, facilitando a adição de regiões e descritores.

Outro ponto a destacar é que, durante a realização deste trabalho, foi detectado que a API de eventos de Lua especificamente para o disparo de comandos edição (classe edit) não estava conforme a norma [ABNT, 2009b]. Para deixar a API de eventos conforme a norma, modificações foram realizadas. Como consequência, a fase de Produção precisou ser modificada, já que a estrutura dos eventos referentes aos comandos de edição estava seguindo a API antiga. Com a utilização da API *command.lua*, esse problema foi facilmente contornado, bastando realizar pequenas modificações na própria API *command.lua*, uma vez que os componentes da fase de Produção só disparam comandos de edição dessa API. Sem a API *command.lua*, seria necessário modificar cada parte do código do Gerador de Estilo e Gerador de Estrutura responsável pela postagem de um evento referente ao disparo de um comando de edição.

Para gerar o conjunto de instruções, cabe destacar que os comandos de edição podem ser disparados juntos, assim que a fase de Produção receba os metadados a serem apresentados, gerando todo o conjunto de instruções referente aos metadados recebidos. Como se trata de uma edição ao vivo, os comandos de edição podem, contudo, ser disparados sob demanda, isto é, à medida que o telespectador interaja com o Guia, solicitando a apresentação de determinadas informações.

PUC-Rio - Certificação Digital Nº 0812554/CA

A implementação desse último caso foi considerada mais complexa. Como o objeto NCLua Gerador de Estrutura está em um contexto diferente dos objetos de mídia do Guia Eletrônico, é necessário definir, para cada possível seleção do usuário que acarreta a apresentação de novas informações, um elo no elemento <body> e uma porta que referencia o objeto selecionado no contexto 'epgCtx'. Cada elo adicionado deve especificar que, quando um objeto for selecionado, uma propriedade do Gerador de Estrutura deve receber a atribuição de um valor. Para isso, uma outra propriedade precisa ser definida e o valor atribuído a ela deve indicar qual objeto foi selecionado. Ao receber um evento de atribuição, o Gerador de Estrutura pode disparar comandos de edição referentes à adição de objetos de mídia e seus relacionamentos, com a finalidade de apresentar as informações solicitadas. O Gerenciador de Estrutura sabe as informações que devem ser apresentadas através do valor atribuído à sua propriedade. Desse modo, precisa ser definido um tratamento específico para cada valor atribuído.

No cenário em que é preferível apresentar as informações mais rapidamente, realizar o processamento da Produção de instruções junto com a apresentação do Guia pode ser insatisfatório, já que as informações podem demorar para serem apresentadas. Entretanto, na criação do Guia sobdemanda, não há desperdício de recursos, uma vez que as informações apresentadas foram solicitadas pelo telespectador. Neste trabalho, por disparar menos comandos de edição, foi preferido gerar o conjunto de instruções de uma única vez. Porém, fica como sugestão de trabalho futuro uma análise de eficiência entre as duas abordagens.

Em relação à atualização da fase de Produção, cabe destacar que, se os componentes adaptados aguardarem o recebimento de novos metadados para gerar o conjunto de instruções, a adaptação do leiaute e estrutura do Guia só terá efeito assim que tais metadados forem recebidos. Com isso, para que o leiaute e estrutura do Guia sejam modificados assim que uma adaptação for realizada na fase de Produção, o novo Gerador de Estrutura precisa acessar os metadados já adquiridos e armazenados na Base de Informações. É justamente por isso que foi adicionado um mecanismo de requisição à Base de Informações. Para que tal requisição seja efetuada, é necessário postar um evento de atribuição na propriedade `requestInfo`. Esse evento deve conter no seu campo *value* o valor "all".

PUC-Rio - Certificação Digital Nº 0812554/CA

4.7. Gerenciador de Histórico e Componente de Persistência

O Gerenciador de Histórico é definido na arquitetura com o objetivo de salvar todos os metadados adquiridos na fase de Alimentação, possibilitando que sistemas de recomendação utilizem o histórico da Alimentação para, por exemplo, gerar o perfil do telespectador. Tanto o histórico quanto o perfil do telespectador gerado podem ser armazenados utilizando o Componente de Persistência.

Quando um conjunto de metadados é recebido, ele é salvo em um arquivo em disco pelo Gerenciador de Histórico. Além de salvar os metadados, o Gerenciador de Histórico implementa um mecanismo que calcula o tempo que o telespectador manteve o receptor sintonizado no mesmo canal. Tal mecanismo, de forma similar ao mecanismo de espera implementado no Gerenciador de Dados, utiliza os metadados recebidos, eventos da classe user e a hora corrente. Essa informação do tempo de sintonia também é armazenada no arquivo de histórico, possibilitando auxiliar um sistema de recomendação na geração do perfil implícito do usuário. Como exemplo de utilização, o sistema de recomendação [Weiss *et al.*, 2008] compara o tempo de duração de um evento com o tempo que o telespectador o assistiu, para determinar o quanto o telespectador gostou do evento.

No *middleware* Ginga, é função do módulo Gerente de Contexto (do subsistema lógico Núcleo Comum) armazenar as preferências do telespectador, assim como as características do aparelho receptor. Como o objetivo deste trabalho não é desenvolver um sistema de recomendação, esse módulo não foi utilizado. Pela mesma razão, o histórico gerado pelo Gerenciador de Histórico não é requisitado por nenhum outro componente, durante a execução da aplicação da arquitetura.

PUC-Rio - Certificação Digital Nº 0812554/CA

4.8. Adaptações e Gerenciador de Atualizações

Na arquitetura proposta, as adaptações (ou atualizações) em tempo de exibição dos componentes da aplicação são realizadas pelo Gerenciador de Atualizações. Como na implementação da arquitetura os recursos do Ginga são utilizados na geração do Guia, é vantagem que as adaptações também sejam realizadas através de recursos do *middleware* Ginga.

Cabe destacar que, até o presente momento, só é passível de adaptações componentes implementados em objetos NCLua. Dessa forma, a parte do Gerenciador de Dados implementada pelos módulos Sintonizador, Interpretador TS e Processador de Dados do Núcleo Comum (veja Seção 4.3.1) não podem sofrer adaptações em tempo de exibição, assim como os demais módulos do Ginga. Isso quer dizer que, na implementação da arquitetura de aplicação adaptativa, a Máquina de Apresentação do conjunto de instruções em NCL, que é justamente a Máquina de Apresentação Ginga-NCL, e o Gerenciador de Atualizações também não são passíveis de adaptações em tempo de exibição. Por outro lado, o Ginga já possui uma arquitetura componentizada [Moreno, 2010]. O que falta para possibilitar atualizações em tempo de execução dos componentes do Ginga é a implementação do módulo Gerente de Evolução Dinâmica do Núcleo Comum. A implementação do Gerente de Evolução Dinâmica junto com uma análise da sua eficiência em prover adaptações em módulos dos subsistemas Máquina de Apresentação Ginga-NCL e Núcleo Comum, durante a apresentação da aplicação NCL do Guia Eletrônico, é uma outra sugestão de trabalho futuro.

As adaptações da aplicação NCL são realizadas através de comandos de edição ao vivo. Por isso, o Gerente de Bases Privadas, módulo responsável por receber e processar os comandos de edição, é o núcleo da implementação do componente Gerenciador de Atualizações da arquitetura. Já que os comandos de edição carregam elementos NCL, a cada comando recebido, o Gerente de Bases Privadas aciona o Interpretador XML e Conversor, com a finalidade de interpretar e converter os elementos NCL em estruturas de apresentação. Posteriormente, a Base Privada é atualizada pelo Gerente de Bases Privadas, com o objetivo de efetuar as mudanças requisitadas.

PUC-Rio - Certificação Digital Nº 0812554/CA

Para serem enviados por radiodifusão, os comandos de edição devem ser encapsulados em descritores de eventos de fluxo DSM-CC [ABNT, 2009b]. Como mostra a Figura 26, o descritor de evento define um campo *eventNPT*, que indica em quanto tempo o comando de edição deve ser disparado após a sua recepção. Já o campo *commandTag* identifica cada comando de edição. O campo *privateDataField* é responsável por encapsular os parâmetros dos comandos de edição.

Sintaxe

```
StreamEventDescriptor () {
     descriptor_tag
     descriptor_length
     eventId
     Reserved
     eventNPT
     privateDataLength
     commandTag
     sequenceNumber
     finalFlag
     privateDataPayload
     FCS
}
```

Figura 26 – Estrutura de um descritor de evento de fluxo

Nas adaptações da aplicação NCL do Guia Eletrônico, são usados comandos de edição para adicionar entidades NCL. Tais comandos definem, pelo menos, os parâmetros *baseId*, que é o identificador da base privada; e o *documentId*, que é o identificador do documento a ser modificado. Quando esses comandos são disparados para adicionar um elemento NCL já definido na aplicação, eles possuem semântica de alteração. Em outras palavras, o primeiro passo para adaptar um componente da arquitetura é enviar o comando de edição *addNode* referenciando o novo componente. O Gerente de Bases Privadas se encarrega de remover o objeto antigo e adicionar o novo objeto referenciado no comando.

O comando de edição *addNode* tem os parâmetros *compositeId* e *{uri, id}*, além de *baseId* e *documentId.* O parâmetro *compositeId* é o identificador do contexto onde o objeto deve ser inserido. Como todos os objetos NCLua foram definidos no elemento <body> da aplicação NCL, o parâmetro *compositeId* sempre possui o valor “idBody”, independente do objeto NCLua a ser adaptado.

Em transmissões por radiodifusão, assim como aplicações NCL, o novo objeto NCLua é transmitido no carrossel de objetos [ISO, 1998], que é um protocolo utilizado para a transmissão cíclica de um sistema de arquivos. Quando o comando *addNode* é enviado, é necessário que o novo NCLua já esteja no

PUC-Rio - Certificação Digital Nº 0812554/CA

carrossel. A função do par de parâmetros *{uri,id}* é justamente informar a localização do arquivo da especificação NCL do elemento <media> e do arquivo .lua do novo componente no carrossel de objetos [ABNT, 2009b].

Para cada arquivo, o parâmetro *uri* possui o esquema "x-sbtvd" com o caminho do arquivo no servidor de dados, enquanto o parâmetro *id* faz referência ao IOR (*Interoperable Object Reference*) do arquivo no carrossel de objetos. No protocolo carrossel de objetos, o IOR de um arquivo corresponde aos parâmetros *carroselId*, *moduleId* e *objectKey* [ABNT, 2009b]. Com essas informações, o *middleware* Ginga consegue localizar e adicionar o novo objeto na aplicação NCL.

É importante destacar que o processamento do comando de edição *addNode* não resulta no início da execução do objeto adicionado. Além disso, não existe um comando de edição específico para disparar a apresentação de um determinado nó. Desse modo, é necessário adicionar um elo, através do comando *addLink*, que especifique como ação a apresentação dessa mídia. Como condição, o elo pode especificar o início de uma âncora de conteúdo do vídeo principal definida utilizando uma referência NPT (*Normal Play Time*) [ISO, 1998]. No entanto, não há como o provedor de conteúdo que solicitou a atualização definir em quais instantes de tempo os comandos de edição serão recebidos, processados pelo Gerente de Bases Privadas e com isso, as referências NPT das inserções dos elementos NCL <media> e <link> também não podem ser previstas. Como consequência dessa imprevisibilidade, o elo pode ser inserido com uma condição que já ocorreu, não sendo disparado nunca, ou que pode demorar a ocorrer, atrasando a execução do novo componente.

Para resolver o problema exposto no parágrafo anterior, durante a realização deste trabalho, dois mecanismos foram analisados.

O primeiro mecanismo utiliza dois comandos de edição, *addLink* para adicionar um elo e *setPropertyValue*, para atribuir um valor a uma propriedade, que pode ser qualquer uma, com exceção das propriedades utilizadas para a comunicação dos objetos NCLua e variáveis de ambiente. Por exemplo, foi definido um nó *settings* (objeto de mídia do tipo *application/x-ginga-settings*) com uma propriedade chamada `update` especificamente para as adaptações dos componentes. O comando *setPropertyValue* tem os seguintes parâmetros: *nodeId*,

PUC-Rio - Certificação Digital Nº 0812554/CA

que é o identificador do nó; *propertyId*, que identifica a propriedade (atributo *name* do elemento <property>) e *value*, que é o valor a ser atribuído.

Como ilustrado na Figura 27, o elo adicionado nesse primeiro mecanismo especifica o disparo da execução do novo NCLua Cliente de Máquina de Busca, no momento em que a propriedade `update` do nó *settings* mudar de valor. Destaca-se que o comando *addLink* deve ser o primeiro a ser enviado. Desse modo, após o processamento de ambos os comandos de edição, o novo componente é executado.

```
<link id="lupdateComponent2" xconnector="onEndAttributionStart">
    <bind       role="onEndAttribution"       component="settings"
interface="update"/>
    <bind role="start" component="searchEngineClient"/>
</link>
```

Figura 27 – Elo que especifica a execução do novo NCLua quando a propriedade update for modificada.

O segundo mecanismo consiste na utilização de uma âncora de conteúdo do tipo *now* [ABNT, 2009b]. O atributo *interface* de um elemento <bind> de um elo (adicionado pelo comando *addLink*) pode referenciar uma âncora não existente na aplicação. Nesse caso, a Máquina de Apresentação Ginga-NCL aguarda o recebimento de um comando *addInterface* que adiciona, como filho do elemento <media> do vídeo principal, uma âncora cujo atributo *begin* possui o valor "*now*". O valor "*now*" especifica o NPT corrente do vídeo principal em decodificação. Quando tal âncora for adicionada, a ação especificada pelo elo é disparada.

Como exemplo, suponha a adição em uma aplicação NCL de um elo que associe o início de uma âncora do tipo *now* como condição para iniciar a apresentação de uma imagem (objeto de mídia já definido na aplicação). Assim que o comando *addInterface* que adiciona essa âncora for processado, a imagem é apresentada.

Resumindo, para adaptar um objeto de mídia com o mecanismo de âncora *now*, é necessário utilizar os comandos de edição *addNode*, *addLink* e *addInterface*, nessa ordem. No entanto, destaca-se que, atualmente, esse mecanismo ainda não é suportado pela implementação de referência do Ginga-NCL.

Os comandos de edição também podem ser utilizados para modificar diretamente os elementos NCL adicionados pela fase de Produção (conjunto de

PUC-Rio - Certificação Digital Nº 0812554/CA

instruções em NCL), com a finalidade de modificar o leiaute e a estrutura do Guia. No entanto, esse tipo de adaptação é temporária, já que não tem efeito sobre os novos comandos de edição postados pela fase de Produção, assim que novos metadados forem recebidos. Pela mesma razão, destaca-se que essa adaptação pode comprometer a exibição do Guia Eletrônico. Por exemplo, a região de apresentação de um metadado pode ser modificada por um comando de edição, sobrepondo uma outra informação.

PUC-Rio - Certificação Digital Nº 0812554/CA