Uma Arquitetura Orientada a Componentes para o Ginga

Marcio Ferreira Moreno, Luiz Fernando Gomes Soares, Renato Cerqueira

Departamento de Informática – PUC-Rio
Rua Marquês de São Vicente, 225 – Rio de Janeiro/RJ – 22453-900 – Brasil
{mfmoreno, lfgs, rcerq}@inf.puc-rio.br

Abstract. *This paper discusses how component-based software development can be applied on the design of digital TV declarative middleware architectures. The paper describes how this approach was considered in the design of the declarative environment of Ginga, the ITU-R Recommendation for terrestrial DTV and ITU-T Recommendation for IPTV. As proof of concept, the solutions discussed in this paper have been incorporated into the reference implementation of the Ginga-NCL environment. The measures presented in this paper illustrate the benefits component-based architecture can bring to digital TV middleware systems, such as the reduction in the amount of used resources and the improvement in its dynamic evolution capabilities.*

Resumo. *Este artigo discute a aplicação dos conceitos do desenvolvimento orientado a componentes de software na arquitetura dos middlewares declarativos para TV digital. O artigo descreve como tais conceitos foram considerados no projeto do ambiente declarativo do middleware Ginga, Recomendação ITU-R para TV digital terrestre e Recomendação ITU-T para IPTV. Como prova de conceito, as soluções discutidas foram incorporadas à implementação de referência do ambiente Ginga-NCL. As medições apresentadas neste artigo indicam os benefícios que uma arquitetura baseada em componentes pode trazer na concepção de middlewares voltados para TV digital, tais como a redução do consumo de recursos computacionais e o aumento da capacidade de sua evolução dinâmica.*

1. Introdução

Um sistema de TV digital (TVD) pode ser definido, resumidamente, como um conjunto de especificações que determinam as tecnologias envolvidas na transmissão de conteúdo pelas emissoras (ou provedores de conteúdo) e no tratamento de conteúdo nos ambientes de recepção dos telespectadores. Nesse cenário, o suporte às aplicações TVD é realizado por uma camada intermediária de software, ou middleware.

O projeto e implementação de um middleware para receptores de sistemas de TVD trazem desafios expressivos. Geralmente, um dispositivo receptor apresenta forte limitação na quantidade de recursos computacionais (processador de baixo custo para execução das tarefas das aplicações interativas e do sistema operacional (SO), bem como pouca memória para uso dessas aplicações e do SO) e possui hardware especializado para a exibição de conteúdo televisivo.

O desenvolvimento orientado a componentes se mostra atraente nesse cenário de escassos recursos, em que conjuntos de funcionalidades devem estar disponíveis apenas enquanto forem necessários. As bibliotecas que possuem as funcionalidades do middleware deixam assim de ser associadas em tempo de compilação e passam a ser organizadas em componentes [1] que podem ser carregados sob demanda, em tempo de execução.

Um ponto crítico no desenvolvimento baseado em componentes, no entanto, deve ser ressaltado: o retardo inserido no instante em que cada componente é carregado na memória pode comprometer o sincronismo da apresentação. Assim, a questão chave consiste na definição de uma estratégia para que se tenha o mínimo de componentes carregados na memória sem que o retardo inserido ao carregar esses componentes na memória comprometa os relacionamentos espaço-temporais

especificados pelo autor das aplicações.

Outro aspecto desafiador no desenvolvimento de um middleware para sistemas de TVD, e que também se mostra atraente para o emprego de uma arquitetura baseada em componentes, é o suporte à sua atualização em tempo de execução. O suporte à evolução dinâmica permite a integração de novas funcionalidades, substituição de funcionalidades existentes e redefinições arquiteturais, originadas das potenciais mudanças de requisitos não previstas no projeto inicial [2].

Este trabalho discute a aplicação dos conceitos do desenvolvimento orientado a componentes de software na arquitetura do ambiente declarativo do middleware Ginga, padrão do sistema ISDB-T (*International Standard for Digital Broadcasting*) [3] e recomendação ITU-T para serviços IPTV [4]. Como prova de conceito, as considerações discutidas no artigo foram incorporadas à implementação de referência Ginga-NCL. Algumas medições comparativas entre a arquitetura monolítica e a arquitetura baseada em componentes de software da implementação de referência são apresentadas.

O restante do artigo está organizado da seguinte forma. A Seção 2 discute alguns trabalhos relacionados. A Seção 3 apresenta a arquitetura do ambiente declarativo do middleware Ginga, considerando o comportamento esperado de cada módulo da arquitetura. A Seção 4 discute a aplicação dos conceitos do desenvolvimento orientado a componentes na implementação do Ginga-NCL. A Seção 5 apresenta medições comparativas em que a implementação de referência reflete ora uma arquitetura monolítica, ora uma arquitetura orientada a componentes. Finalmente, a Seção 6 é reservada para algumas considerações finais.

2. Trabalhos Relacionados

Diversos trabalhos na literatura têm apresentado soluções baseadas em componentes de software para prover um alto grau de adaptabilidade e um maior controle do uso de recursos computacionais.

Coulson et al. [5] apresentam o OpenCom como um modelo de componentes de software para o desenvolvimento de sistemas de mais baixo nível como, por exemplo, sistemas de middleware, procurando prover as mesmas facilidades de reconfiguração que outras infraestruturas de componentes provêem no nível das aplicações finais. Esse modelo de componentes já foi utilizado no desenvolvimento de sistemas de middleware para diferentes domínios de aplicação, tais como processadores de redes programáveis, sistemas de middleware de comunicação reflexivos e sistemas de roteamento em redes móveis ad-hoc [6] [7].

Similarmente, o modelo de componentes Fractal [8] também tem sido utilizado na construção de sistemas de middleware. Por exemplo, Layaida et al. [9] apresentam um arcabouço baseado no modelo Fractal para a construção de aplicações multimídia, denominado PLASMA. Os autores mostram que sua arquitetura baseada em componentes pode prover adaptabilidade em tempo de execução com um baixo impacto de desempenho, mesmo em dispositivos móveis com recursos computacionais limitados.

Finalmente, Filho et al. [10] especificam o modelo FlexCM, que possui como principal objetivo a composição automática de arquiteturas baseadas em componentes, através da representação explícita das conexões entre cada componente, utilizando um arquivo XML para a descrição arquitetural. Nessa descrição, as interfaces de componentes são identificadas através de GUID (*Globally Unique Identifiers*), que consistem em identificadores de 128 bits gerados de forma a garantir uma unicidade global. Cada componente pode declarar os GUID de suas interfaces providas e requeridas. Assim, um ambiente de execução conforme o modelo FlexCM poderá compor a arquitetura através da descrição XML. A arquitetura do middleware FlexTV [10] segue uma arquitetura conforme o modelo FlexCM.

Middlewares declarativos para TVD não são discutidos nesta seção, pois, até onde sabemos, este é o primeiro trabalho que reporta a definição e aplicação de uma arquitetura baseada em componentes de software na construção de um middleware declarativo para TVD, embora

existam relatos de resultados no ambiente imperativo apresentado no parágrafo anterior [10].

Os resultados experimentais apresentados neste artigo reforçam as conclusões obtidas em outros domínios de aplicação e apresentados nos trabalhos citados acima sobre as vantagens da adoção de uma arquitetura baseada em componentes, além de apresentar uma arquitetura específica para o domínio de sistemas de middleware declarativo para TVD interativa. Entretanto, a implementação baseada em componentes de software do Ginga-NCL foi realizada com base diretamente nos serviços providos pelo sistema operacional, sem utilizar uma infraestrutura de software para a construção de sistemas baseados em componentes, como as utilizadas nos trabalhos citados anteriormente. Essa característica pode contribuir para a redução dos requisitos de memória e processamento da solução apresentada. Uma representação comparativa entre os trabalhos discutidos nesta seção e a solução proposta neste artigo é apresentada na Tabela 1.

Tabela 1. Representação Comparativa entre Trabalhos que Apresentam Soluções Baseadas em Componentes de Software

	Evolução Dinâmica	Gerencia Recursos	Elimina overhead do uso de Infraestrutura de Software	Foco no Sincronismo de Mídias
OpenCom	✓	✓		
Fractal	✓	✓		
FlexCM	✓	✓		
Ginga-NCL	✓	✓	✓	✓

Como apresentado na Tabela 1, entre os trabalhos discutidos, a solução apresentada neste artigo é a única que possui como requisito a manutenção das relações de sincronismo especificadas pelo autor da aplicação. Ou seja, a solução apresentada é a única que leva em conta os problemas que podem ser causados pelo retardo na carga dos componentes no sincronismo da execução de um programa, definindo uma estratégia para que se tenha o mínimo de componentes carregados na memória sem que o retardo inserido ao carregar esses componentes na memória comprometa os relacionamentos espaço-temporais especificados pelo autor de aplicações NCL.

3. Arquitetura

O middleware Ginga oferece obrigatoriamente suporte à execução de aplicações desenvolvidas na linguagem NCL [3] [4] e sua linguagem de script Lua [3] [4]. A Figura 1 apresenta a arquitetura modular do middleware, dividida em seus dois subsistemas lógicos: o Ambiente de Apresentação, denominado Ginga-NCL, e o Núcleo Ginga, denominado Ginga-CC. Ginga-CC é responsável por oferecer os serviços básicos ligados à plataforma do dispositivo receptor ao Ambiente de Apresentação, às aplicações residentes e a um possível ambiente imperativo adicional. Ginga-NCL é um subsistema lógico capaz de controlar o ciclo de vida das aplicações NCL e suas execuções.

Figura 1. Arquitetura Ginga

Neste artigo, um módulo consiste em um conjunto formado por componentes de software que agregam funcionalidades para um mesmo fim. Já os componentes de software são definidos como em Szyperski [1]: unidades de composição com interfaces contratualmente especificadas e dependências de contexto explícitas, além de poderem ser independentemente implantados e estar

sujeitos a composição por terceiros.

Os módulos que compõem os dois subsistemas lógicos, Ginga-NCL e Ginga-CC, são discutidos nesta seção considerando o desenvolvimento orientado a componentes do middleware. Nessa discussão, os componentes serão classificados em dois tipos: permanentes ou temporários.

Os componentes permanentes são aqueles que possuem funcionalidades utilizadas de forma ininterrupta enquanto o dispositivo receptor estiver em operação. Como conseqüência, eles devem ser mantidos na memória enquanto o dispositivo receptor estiver nesse modo. Quando atualizações em componentes desse tipo forem necessárias, é desejável que elas sejam realizadas quando o dispositivo receptor estiver em modo *stand-by* [1]. Já os componentes temporários são aqueles em que suas funcionalidades são necessárias apenas durante determinados instantes do tempo em que o dispositivo receptor estiver em operação. Assim, esse tipo de componente pode ser mantido na memória apenas enquanto estiverem em uso pelo middleware.

3.1. Componentização do Ginga-CC

No Ginga-CC, o módulo Sintonizador é responsável por receber conteúdo de TVD transmitido por provedores de conteúdo. Ou seja, o módulo deve ser capaz de identificar um conjunto de serviços que compõem um canal de TV e receber os conteúdos provenientes desses serviços. Os componentes que constituem o módulo Sintonizador devem ser permanentes, uma vez que, normalmente, a apresentação do conteúdo de TV deve ser interrompida apenas pelo usuário telespectador.

As aplicações interativas podem chegar ao dispositivo receptor multiplexadas nos fluxos recebidos pelo módulo Sintonizador, ou por outra interface de rede diferente das de recepção desses fluxos. O módulo Processador de Dados é responsável por monitorar informações que sinalizam a existência e a origem de aplicações interativas. Os componentes que possuem a funcionalidade de monitoramento devem ser permanentes, uma vez que não é possível determinar os instantes em que as informações de sinalização serão enviadas pelo provedor de conteúdo. Caso uma aplicação esteja multiplexada em fluxos recebidos pelo Sintonizador, ela será obtida através do processamento realizado por componentes temporários do módulo Processador de Dados sobre o fluxo recebido. Ao concluir o processamento da aplicação, esses componentes podem ser retirados da memória.

No caso do recebimento da aplicação através de uma interface de rede, um módulo de Transporte é responsável por controlar protocolos e interfaces de rede e atender a demanda do Ambiente de Apresentação e dos Exibidores, que serão discutidos ainda nesta seção, por aplicações e seus conteúdos de mídia. Os componentes que constituem o módulo de Transporte podem ser temporários e mantidos na memória apenas enquanto realizam a recepção dos dados necessários para compor a aplicação, ou enquanto realizam a recepção de componentes para atualizações do middleware, que serão discutidas ainda neste capítulo.

O módulo Gerente de E/S foi definido para gerenciar o armazenamento temporário local das aplicações de TV, incluindo os conteúdos que elas referenciam. Os componentes desse módulo podem ser temporários e mantidos na memória durante todo o processo de recepção e apresentação de uma aplicação.

Ainda no Ginga-CC, o módulo Exibidores é responsável por prover os decodificadores adequados para a apresentação de conteúdos específicos. Cada exibidor consiste em um componente temporário. O intervalo de tempo em que esses componentes são mantidos na memória é definido pelo Gerente de Exibidores, que será discutido ainda nesta seção. Com o intuito de padronizar a comunicação entre a Máquina de Apresentação e a API de decodificação de conteúdo dos exibidores, um módulo, denominado Adaptadores, foi definido na arquitetura. Quando necessário, um componente exibidor é associado ao componente adaptador para manter o padrão de

[1] Estado em que o dispositivo receptor fica ligado sem operação, em estado de espera.

comunicação. O conteúdo a ser exibido pode ser disponibilizado ao exibidor pelo módulo Processador de Dados, caso esse conteúdo esteja multiplexado no fluxo recebido. Caso contrário, ao ser instanciado, um exibidor pode acionar o módulo de Transporte para permitir o acesso ao conteúdo.

Em particular faz parte do Ginga-CC um interpretador Lua, associado ao componente exibidor de código Lua que, assim como os outros componentes exibidores, é mantido na memória de acordo com as definições do Gerente de Exibidores. O interpretador Lua é um componente temporário, que pode ser mantido na memória apenas durante a apresentação de objetos de mídia com código Lua.

O módulo Gerente Gráfico realiza o controle espacial da renderização de conteúdo, incluindo o conteúdo audiovisual principal de um programa de TV. Uma vez que as funcionalidades de apresentação de conteúdo são constantemente utilizadas, os componentes desse módulo devem ser permanentes.

Os componentes que constituem os módulos definidos na arquitetura podem ser atualizados de forma independente. As atualizações podem ser enviadas pelo provedor de conteúdo por difusão ou obtidas de repositórios de componentes, através do módulo de Transporte. No primeiro caso, cabe a um componente permanente do módulo Gerente de Evolução Dinâmica realizar o monitoramento de informações sobre a existência de possíveis atualizações multiplexadas no conteúdo de TVD recebido pelo módulo Sintonizador. No segundo caso, um componente temporário do módulo Gerente de Evolução é responsável por realizar consultas, através dos componentes do módulo de Transporte, sobre a disponibilidade de atualizações do middleware, podendo ser mantido na memória apenas durante a realização da consulta. Essas consultas podem ser realizadas de acordo com a política de atualizações especificada pelo usuário telespectador. Outros componentes que constituem o módulo Gerente de Evolução são responsáveis por realizar o procedimento de atualização, sem interromper o funcionamento do middleware. Esses componentes podem ser temporários e mantidos na memória apenas enquanto o procedimento de atualização é realizado.

O módulo Gerente de Dispositivos oferece serviços que visam atender aos diversos requisitos relacionados a apresentações distribuídas, incluindo o registro de dispositivos, a comunicação entre os dispositivos, a construção de domínios e a preservação do sincronismo das aplicações [3] [4]. Os componentes desse módulo podem ser temporários e mantidos na memória apenas enquanto houver apresentações distribuídas em múltiplos dispositivos.

Finalmente, um módulo Gerente de Contexto foi definido para gerenciar as informações sobre o perfil do sistema embarcado no dispositivo e sobre o perfil do usuário telespectador. Uma vez que essas informações são persistentes, os componentes desse módulo podem ser temporários e mantidos na memória apenas enquanto consultas sobre as informações que o módulo gerencia sejam atendidas.

3.2. Componentização da Máquina de Apresentação

Ainda na Figura 1, no Ambiente de Apresentação Ginga-NCL, o módulo Formatador é responsável por receber e controlar as aplicações NCL entregues por aplicações residentes ou pelo Ginga-CC. Os componentes que constituem o módulo Formatador podem ser temporários e mantidos na memória apenas enquanto houver aplicações NCL sendo apresentadas.

Ao receber uma aplicação NCL, o Formatador solicita ao módulo *Parser* XML que as especificações NCL sejam interpretadas e traduzidas em estruturas de dados que representam as entidades do modelo conceitual da linguagem NCL, denominado NCM [11]. Os componentes do módulo *Parser* XML podem ser temporários e mantidos na memória até que as entidades NCM sejam geradas. Além disso, esses componentes serão necessários para processar os comandos de edição NCL, que serão discutidos ainda nesta seção. As entidades NCM resultantes da interpretação da aplicação NCL são agrupadas em uma estrutura de dados denominada Base

Privada. Ginga associa pelo menos uma base privada a cada canal (conjunto de serviços) de televisão. Outras bases privadas podem ser definidas, mas no máximo uma por serviço de um canal. Os documentos NCL em uma base privada podem ser iniciados, pausados, retomados, parados e podem referir-se uns aos outros.

Os procedimentos do módulo Escalonador de Apresentação são solicitados pelo Formatador, para a orquestração da apresentação da aplicação NCL, assim que o processo de interpretação realizado pelo *Parser* XML termina.

À medida que a apresentação de uma aplicação é realizada, o Escalonador de Apresentação solicita ao módulo Conversor a realização de um segundo passo de conversão para que as entidades NCM sejam traduzidas em estruturas de dados adequadas à apresentação de aplicações NCL. Os componentes do módulo Conversor podem ser temporários e mantidos na memória durante toda a apresentação da aplicação NCL. Essa é uma das razões da realização da conversão em dois passos.

Para permitir que cada conteúdo seja exibido de forma adequada, o Escalonador de Apresentação solicita que o Gerente de Exibidores crie um exibidor apropriado para cada tipo de mídia, ou seja, de acordo com o tipo de conteúdo a ser exibido em um dado instante. Ao concluir a exibição do conteúdo, o Escalonador de Apresentação é notificado pelo exibidor. Caso não seja mais necessária a exibição desse conteúdo, o Escalonador de Apresentação solicita ao Gerente de Exibidores que o exibidor seja destruído. O mesmo pode ocorrer quando o Escalonador de Apresentação precisa interromper uma exibição que esteja ocorrendo. Assim, o Gerente de Exibidores pode ser responsável pela carga e liberação na memória do componente exibidor.

A exibição de um conteúdo deve ser realizada em regiões do display de um dispositivo, conforme especificado pelo autor da aplicação NCL. O módulo Gerente de Leiaute foi definido para fazer a associação do conteúdo, tratado por um exibidor específico, a essas regiões. As regiões podem ser especificadas na aplicação NCL em dispositivos distintos. Nesse caso, o Gerente de Leiaute utiliza os serviços do Gerente de Dispositivos, discutido na seção anterior, para controle da apresentação e transmissão dos conteúdos a serem apresentados. Os componentes do módulo Gerente de Leiaute podem ser temporários e mantidos na memória durante toda a apresentação da aplicação NCL.

O módulo Gerente de Bases Privadas é responsável por gerenciar todas as Bases Privadas criadas pelo Formatador. Os componentes desse módulo podem ser temporários e mantidos na memória enquanto houver ao menos uma base privada ativa. Além de gerenciar as Bases Privadas, esse módulo é responsável por processar os comandos de edição ao vivo. Comandos de edição foram definidos na norma Ginga-NCL para o controle do ciclo de vida das aplicações NCL e para permitir que atualizações possam ser realizadas sobre essas aplicações enquanto são apresentadas. Como as atualizações podem ser especificadas em documentos XML, quando for o caso, o Gerente de Bases Privadas solicita aos componentes do módulo *Parser* XML para que as atualizações sejam traduzidas em entidades NCM. Ao processar um comando de edição, mudanças sobre as estruturas presentes nas Bases Privadas podem ser realizadas pelo Gerente de Bases Privadas.

Finalmente, um Gerente de Contexto NCL foi definido para realizar adaptações na apresentação das aplicações NCL, de acordo com as informações providas pelo Ginga-CC e especificadas pela própria aplicação NCL.

4. Implementação Ginga e Modelo de Componentes

A partir da versão 0.10.1 da implementação de referência[2] um módulo opcional, denominado Gerente de Componentes, foi criado. A instalação opcional do Gerente de Componentes define,

[2] A atual versão da implementação de referência Ginga-NCL é a 0.12.1. Todas as versões da implementação de referência estão disponíveis como código aberto em www.softwarepublico.gov.br.

em tempo de compilação, a forma com que as bibliotecas que compõem o middleware serão geradas, tendo como resultado ou uma arquitetura "monolítica" ou orientada a componentes. A arquitetura "monolítica", em que todas as bibliotecas são associadas em tempo de compilação, é gerada caso o Gerente de Componentes não seja encontrado na plataforma de recepção. Caso contrário, a arquitetura orientada a componentes é gerada. As duas alternativas são oferecidas, facilitando o porte da implementação do middleware Ginga para plataformas que não ofereçam suporte ao desenvolvimento orientado a componentes. As duas alternativas são também importantes em medições comparativas, como as apresentadas na Seção 5.

A implementação do módulo Gerente de Componentes, apresentado na Figura 2, embute as funcionalidades necessárias para carga e liberação de cada componente que constitui a arquitetura do Ginga. Essas operações são realizadas internamente, pelo componente permanente *ComponentManager*, através da biblioteca *libdl*, que consiste na única dependência inserida para que o suporte a carregamento e liberação dos componentes na memória seja realizado. O *ComponentManager* provê uma interface, denominada *IComponentManager*, para que qualquer parte do código do middleware consiga solicitar a carga, ou liberação, de um componente na memória. A centralização do código responsável pela manipulação de componentes facilita o porte e embarque do middleware em plataformas de recepção que necessitem de uma alternativa à biblioteca *libdl*.

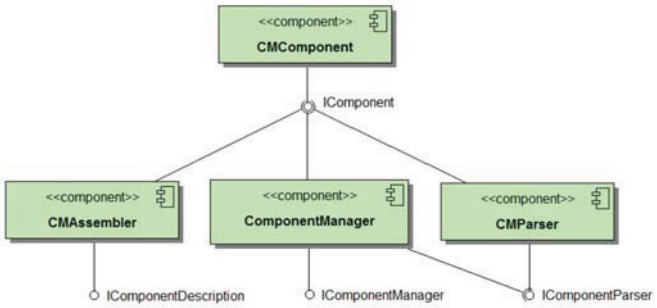

Figura 2. Gerente de Componentes

A arquitetura do Ginga a ser embarcado é especificada por um documento XML, que será discutido ainda nesta seção, gerado por um script de compilação e instalação dos componentes. Quando o dispositivo receptor entra em operação, o *ComponentManager* é imediatamente iniciado, quando então solicita ao componente temporário *CMParser*, através da interface *IComponentParser*, a interpretação do documento XML. O resultado dessa interpretação é um conjunto de informações, representadas pela interface *IComponent*, sobre os detalhes necessários para carga e liberação de todos componentes do Ginga. Como mostra a Figura 2, a interface *IComponent* é provida pelo componente temporário *CMComponent*.

Caso seja necessária uma atualização, o módulo Gerente de Evolução Dinâmica, discutido na seção anterior, realiza o procedimento de atualização e altera o documento XML da arquitetura através da interface *IComponentDescription*, provida pelo componente temporário *CMAssembler*. O Gerente de Evolução Dinâmica é também responsável por notificar o *ComponentManager*, através da interface *IComponentManager*, que uma atualização foi realizada.

A definição de tipo de documento (DTD) para o documento XML que especifica a arquitetura Ginga-NCL é apresentada na Figura 3. Na DTD XML, o elemento *middleware* é definido como elemento pai de um ou mais elementos *component*. Com o objetivo de auxiliar o Gerente de Evolução Dinâmica na tarefa de atualização de componentes, os atributos obrigatórios do elemento *middleware* definem as características da plataforma de recepção que o Ginga será embarcado. Assim, os atributos *platform*, *system* e *version* foram definidos para informar, respectivamente, os seguintes dados da plataforma de recepção: identificador, sistema operacional e versão do *kernel* do sistema operacional.

O elemento *component* foi definido para especificar os detalhes necessários para carga e liberação

do componente que ele representa. Os atributos *package*, *name* e *version* foram definidos para informar, respectivamente, o módulo da arquitetura Ginga a que o componente pertence, o nome e a versão do componente. O elemento *component* deve possuir como elementos filhos um ou mais elementos *symbol*, zero ou mais elementos *dependency*, um elemento *location*, e zero ou um elemento *repository*.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT middleware (component+)>
<!ATTLIST middleware
          platform CDATA #REQUIRED
          system CDATA #REQUIRED
          version CDATA #REQUIRED>
<!ELEMENT component (
          (symbol, location) |
          (dependency+, location, repository) |
          (dependency+, symbol+, location, repository))>
<!ATTLIST component
          package CDATA #REQUIRED
          name CDATA #REQUIRED
          version CDATA #REQUIRED>
<!ELEMENT location EMPTY>
<!ATTLIST location
          type CDATA #REQUIRED
          uri CDATA #REQUIRED>
<!ELEMENT repository EMPTY>
<!ATTLIST repository
          uri CDATA #REQUIRED>
<!ELEMENT symbol EMPTY>
<!ATTLIST symbol
          object CDATA #REQUIRED
          creator CDATA #REQUIRED
          destroyer CDATA #REQUIRED
          interface CDATA #REQUIRED>
<!ELEMENT dependency EMPTY>
<!ATTLIST dependency
          name CDATA #REQUIRED
          version CDATA #REQUIRED>
```

Figura 3. DTD XML Descrição da Arquitetura Ginga-NCL

O elemento *location* foi definido para que um componente seja localizado corretamente. Essa localização pode ser remota, quando o atributo obrigatório *type* tiver o valor "*remote*", ou local, quando *type* tiver o valor "*local*". O atributo obrigatório *uri* define o endereço em que o componente está localizado. O elemento *repository*, por sua vez, foi definido para que o Gerente de Evolução Dinâmica possua um conjunto de repositórios de componentes. Uma falha ou perda de um componente pode ser resolvida através desse repositório, que também pode ser explorado na busca por atualizações de componentes. O endereço do repositório é especificado no valor do atributo obrigatório *uri* do elemento *repository*.

Para que as funcionalidades de um componente sejam utilizadas após ele ser carregado na memória, é necessário que esse componente defina um símbolo que será encontrado em tempo de execução. O elemento *symbol* possui informações sobre os símbolos definidos no componente. Assim, para que um objeto (*object*) seja encontrado, os seus símbolos de criação (*creator*) e destruição (*destroyer*) são especificados pelos atributos obrigatórios *object*, *creator*, e *destroyer*. O atributo obrigatório *interface* foi definido para permitir ao *ComponentManager* conhecer uma lista de objetos que implementam uma mesma interface. Esse recurso permite, por exemplo, que um componente do middleware acesse o *ComponentManager* para testar todas as interfaces de rede existentes em uma plataforma. Um recurso útil para dispositivos de recepção para sistemas híbridos de TVD [12].

Finalmente, o elemento *dependency* foi definido para especificar quais as dependências de um componente. Assim, o Gerente de Evolução Dinâmica é capaz de atualizar não só um componente como suas dependências. Os atributos obrigatórios *name* e *version* foram definidos para especificar, respectivamente, o nome e a versão do componente de uma dependência.

O Gerente de Componentes e o módulo Gerente de Evolução Dinâmica são utilizados também no

caso de uma aplicação NCL precisar de algum componente não previsto inicialmente. Considere, por exemplo, a aplicação NCL apresentada na Figura 4, em que um objeto de mídia que referencia um conteúdo DivX[3], não conforme às normas Ginga-NCL [3] [4], é especificado.

```
1:  <?xml version="1.0" encoding="UTF-8"?>
2:  <ncl id="evolucaodinamica" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
3:   <head>
4:    <regionBase>
5:      <region id="fullScreen" left="0" top="0" width="100%" height="100%"/>
6:    </regionBase>
7:    <descriptorBase>
8:     <descriptor id="fsDesc" region="fullScreen"/>
9:    </descriptorBase>
10:  </head>
11:  <body id="edBody">
12:   <port id="pDivX" component="videoMM"/>
13:   <media id="videoMM" src="bolinha.avi" descriptor="fsDesc"/>
14:   <meta name="adapter.avi" content="component.description(0).gingancldivxadapter"/>
15:   <meta name="component.description(0)"
16:         content="http://www.gingancl.org.br/releases/descs/divx.xml"/>
17:  </body>
18: </ncl>
```

Figura 4. Aplicação NCL que Demanda Evolução Dinâmica

Na Figura 4, o documento NCL define em seu cabeçalho: uma região (linha 5), cujos atributos *left*, *top*, *width* e *height* indicam as dimensões na tela onde será apresentado o objeto *DivX*; e um descritor (linha 8) associado a tal região.

A linha 12 indica que o documento terá sua apresentação iniciada pelo nó de mídia "videoMM" (linha 13), em que o atributo *src* especifica o conteúdo a ser exibido e o atributo *descriptor* especifica uma associação do nó de mídia "videoMM" ao descritor "fsDesc".

Na apresentação desse documento NCL, o Gerente de Exibidores do Ginga-NCL será solicitado pelo Escalonador de Apresentação (veja Seção 3.2), para que um exibidor de mídia *DivX* seja criado. Caso não encontre o exibidor para esse tipo de conteúdo, o Gerente de Exibidores deve notificar a falha ao Escalonador de Apresentação para que o funcionamento do middleware não seja comprometido.

No entanto, o elemento meta[4] da linguagem NCL permite que informações sobre o conteúdo utilizado ou exibido sejam especificadas no documento. Assim, na arquitetura orientada a componentes da implementação de referência, o Gerente de Exibidores solicita uma análise do objeto de mídia, bem como da meta informação existente no documento NCL (linhas 14 a 16), ao Gerente de Evolução Dinâmica.

De posse dessas informações, o Gerente de Evolução Dinâmica infere, através da meta informação definida na linha 14, que uma descrição do componente "gingancldivxadapter" está disponível, em um conjunto de descrições de componentes, para um exibidor de conteúdo com extensão "avi". Além disso, o Gerente de Evolução Dinâmica reconhece o endereço do conjunto de descrições de componentes, através da meta informação definida nas linhas 15 e 16.

Suponha agora que o documento XML adquirido pelo Gerente de Evolução Dinâmica através do módulo de Transporte seja aquele apresentado na Figura 5. Esse documento, conforme a DTD apresentada na Figura 3, consiste no conjunto de descrições de componentes, inclusive do componente exibidor para objeto de mídia *DivX*.

Conforme a Figura 5, o componente "gingancldivxadapter", em sua versão "1.0.1", pertence ao módulo "gingancl" (linha 2). Nas linhas 3 e 4 os componentes "divxplayer", versão "1.0.1", e "gingancladapter", versão "0.12.1", são definidos como dependências do componente "gingancldivxadapter". Assim, o Gerente de Evolução sabe quais componentes deverão ser

[3] Formato de vídeo digital: www.divx.com

[4] O elemento meta é especificado na linguagem NCL como um único par, propriedade e valor, nos atributos *name* e *content*, respectivamente [3] [4].

atualizados juntamente com o componente “gingancldivxadapter”. Na linha 5, a localização do componente “gingancldivxadapter” é definida. Os símbolos para criação e destruição do exibidor “DivXPlayerAdapter”, que implementa a interface “IPlayerAdapter”, são também definidos no XML (linhas 6 e 7). Os componentes definidos como dependências do componente “gingancldivxadapter” são também especificados nesse mesmo documento XML (elementos omitidos, sem prejuízo, para economia de espaço).

```
1:  <?xml version="1.0" encoding="UTF-8"?>
2:  <component package="gingancl" name="gingancldivxadapter" version="1.0.1">
3:    <dependency name="divxplayer" version="1.0.1"/>
4:    <dependency name="gingancladapter" version="0.12.1"/>
5:    <location type="remote" uri="http://www.gingancl.org.br/releases/referenceimp/"/>
6:    <symbol object="DivXPlayerAdapter"     creator="createDivXAdapter"
7:            destroyer="destroyDivXAdapter" interface="IPlayerAdapter"/>
8:  </component> ...
```

Figura 5. Descrição Remota de Componentes

Ao adquirir o componente adequado à plataforma, através do módulo de Transporte, o Gerente de Evolução Dinâmica solicita sua persistência em uma URL local, atualiza o documento XML que descreve a arquitetura do middleware e notifica o *ComponentManager* sobre a atualização realizada. Na atualização do XML da arquitetura, os novos componentes são incluídos ou atualizados de acordo com as informações presentes no XML de atualização. A exceção é o elemento *location*, que é definido com a URL local em que o componente tornou-se persistente. Um ponto importante nesse processo de atualização é a definição do elemento *repository*, que será preenchido através do atributo *uri* (linha 5 da Figura 5) do elemento *location* do XML de atualização.

Enquanto o componente não for adquirido, o Gerente de Exibidores mantém o Escalonador de Apresentação informado sobre a ausência desse componente (notificação de falha já mencionada), garantindo o funcionamento do middleware mesmo se o ambiente receptor não conseguir adquirir o componente. Outro ponto interessante que deve ser ressaltado é que a arquitetura do middleware é alterada dinamicamente não só em tempo de execução do middleware, mas também em tempo de apresentação da aplicação NCL.

O módulo Escalonador de Apresentação e o módulo Gerente de Exibidores garantem que o retardo inserido no instante em que cada componente exibidor é carregado na memória não comprometa o sincronismo especificado nas aplicações NCL. Para isso, o Escalonador de Apresentação mantém uma linha de execução independente, que fica monitorando todas as ações executadas na apresentação de uma aplicação. Nessa linha de execução, toda ação para iniciar a apresentação de um objeto de mídia é previamente detectada. Nesse caso, o Gerente de Exibidores é solicitado para que a preparação do exibidor adequado seja realizada. Essa solução se mostra adequada, conforme discutido na próxima seção, para a preparação dos componentes já existentes no ambiente de recepção. Atualmente, nenhum mecanismo de pré-busca foi definido para os componentes que deverão ser tratados pelo Gerente de Evolução Dinâmica.

Outro ponto em que a solução de monitoramento não apresenta um comportamento ideal é na liberação dos componentes. Para garantir que um componente esteja preparado para futuras ações, que possivelmente ainda não foram agendadas no Escalonador de Apresentação, o Gerente de Exibidores mantém os exibidores ociosos na memória por um intervalo de tempo, mesmo após a conclusão da apresentação de cada objeto de mídia. Esse comportamento pode ser observado na próxima seção. Para maior acuidade nos instantes de liberação dos componentes exibidores, está sendo analisada uma integração entre o módulo Escalonador de Apresentação e as estruturas de dados definidas por um HTG[5], conforme definido por Costa et al. [13].

[5] HTG (*Hypermedia Temporal Graph*) é uma estrutura formada por dígrafos capaz de controlar o comportamento temporal das aplicações durante sua execução.

5. Avaliação por Medições

Para avaliar a arquitetura baseada em componentes proposta quanto a eficiência na utilização de recursos, o consumo de CPU (processamento), e retardos introduzidos, diversas medições comparativas entre a configuração monolítica (implementação *Mono*) e a configuração baseada em componentes de software (implementação *Comp*) do Ginga foram realizadas. Nesta seção apresentaremos apenas algumas das avaliações realizadas, justificando a arquitetura proposta.

Para a realização das medidas, a seguinte configuração de hardware foi usada para o receptor. A estação não possui hardware específico para decodificação de nenhum tipo de mídia, ou seja, toda decodificação é realizada por software. A partição de swap foi desabilitada, não permitindo *cache* em disco (comando *swapoff –a*). Inicialmente, a memória RAM foi limitada a 256MB (mem=256M no arquivo de configuração do boot), com apenas 144MB livres (sistema operacional e os outros processos utilizam 112MB).

A implementação de referência utiliza a biblioteca *DirectFB* para a manipulação de interfaces gráficas, bem como a decodificação e renderização de textos, imagens, áudios e vídeos.

A biblioteca *DirectFB* realiza, internamente (sem controle da implementação do middleware Ginga), a carga dinâmica de decodificadores de mídia. Dessa forma, para compararmos as implementações *Mono* e *Comp*, dois documentos NCL testes tiveram de ser utilizados. O primeiro documento (documento A), será usado na comparação entre a implementação *Comp* e a implementação *Mono* com os recursos de carga dinâmica da biblioteca *DirectFB*. O segundo documento (documento B), será usado para simular o efeito do mesmo documento A em uma implementação *Mono* sem os recursos de carga dinâmica.

O documento A, ao ser iniciado, dispara a apresentação em seqüência de um objeto de imagem, seguido de um objeto de texto, seguido de um objeto imperativo com código Lua, seguido de um objeto de áudio, seguido de um objeto declarativo com código HTML; seqüencia que é repetida duas vezes. O documento B contém objetos de mídia de imagem, texto, Lua, áudio e HTML, todos disparados simultaneamente. O documento A tem exatamente a mesma duração do documento B. Note que ao disparar simultaneamente os objetos de mídia do documento B, estamos fazendo com que a biblioteca *DirectFB* carregue todos os componentes decodificadores desde o início, ou seja, sem utilizar sua facilidade de carga dinâmica, simulando assim o comportamento monolítico, não componentizado.

Embora tendo tomado o cuidado de simular corretamente as implementações *Mono* e *Comp*, no que se refere à carga de componentes, nossas medidas não são exatas por outra característica da biblioteca *DirectFB*: em nenhum instante um decodificador carregado é retirado da memória, mesmo após ser solicitado à biblioteca a destruição desse decodificador. Assim, na implementação *Comp* estaremos levando em conta, para os objetos de mídia imagem, texto e áudio, apenas a carga dos decodificadores, não ficando ainda mais evidenciada sua vantagem na descarga do componente. Em outras palavras, o desempenho da arquitetura *Comp* deveria ser ainda melhor, mas, mesmo assim, com as medidas realizadas já se tem dados suficientes para uma análise. Como os objetos Lua e HTML usam bibliotecas e decodificadores controlados pelo próprio Ginga, esse problema não acontece.

Deve ser também salientada a ausência de objetos de vídeo na aplicação teste. A razão é pelo fato que, em plataformas de receptores reais, tais objetos são decodificados por hardware. A decodificação desses objetos por software demanda tanta memória e CPU que prejudicaria a escala dos gráficos de comparação e tornaria difícil observar as mudanças na utilização dos recursos nos outros casos.

Para se obter um melhor resultado e não fazer medições desnecessárias é necessário calcular o número ideal de medições que devem ser feitas. Para esse cálculo, será considerado um fator de confiança, o erro máximo desejado e também o desvio padrão, obtido a partir de uma coleta prévia de medições (dez medições).

A fórmula $n = \left(\frac{Z.s}{E}\right)^2$ foi utilizada para determinar o tamanho amostral n, onde: E é o erro permissível, Z é o valor da variável normal padrão associado ao grau de confiança adotado e s é o desvio padrão da coleta prévia.

Para a avaliação do consumo de memória, a coleta prévia apresentou o desvio padrão máximo de 889,72 kB. Considerando o valor 1,96 (valor da variável normal padrão associado ao grau de confiança 95%) e que o erro não supere 175 kB; obtém-se o valor 99,30 (definido como 100 medições) como o tamanho da amostra *n*.

A coleta prévia para avaliação do consumo de CPU (porcentagem) apresentou o desvio padrão máximo de 1,41. Considerando o valor 1,96 (valor da variável normal padrão associado ao grau de confiança 95%) e que o erro não supere 0,28; obtém-se o valor 97,50 (definido como 100 medições) como o tamanho da amostra *n*.

A Figura 6 apresenta um gráfico comparativo da quantidade de código alocado na memória entre a implementação *Comp* enquanto realiza a apresentação do documento A, implementação *Mono* enquanto realiza a apresentação do documento A (*Mono-A*) e a implementação *Mono* enquanto realiza a apresentação do documento B (*Mono-B*). Note, na Figura 6, que inicialmente, no instante 0s, a quantidade de memória alocada para a implementação *Mono* (independente se *Mono-A* ou *Mono-B*) é 3,7 vezes maior que para a implementação *Comp*. O instante t=0s é individualmente importante por ser um momento em que apenas o middleware Ginga está na memória. Ou seja, as estruturas envolvidas no suporte a apresentação das aplicações ainda não foram alocadas na memória. No intervalo entre t=0,02s e t=0,56s a implementação *Mono-B* carrega na memória todas as funcionalidades da biblioteca *DirectFB* necessárias. Por sua vez, a implementação *Comp* carrega as estruturas necessárias para iniciar a apresentação da aplicação NCL (veja Seção 3). No instante t=0,78s, em que o primeiro objeto de mídia da aplicação é apresentado, a diferença entre a versão *Comp* e o melhor caso da versão *Mono* (*Mono-A*) era de 14567 kB de memória. No instante t=26,05s a diferença entre as duas versões tem seu menor valor, por ser o instante em que o componente responsável pela apresentação do objeto de mídia HTML está na memória, e pelo fato da biblioteca *DirectFB* não permitir a descarga dos decodificadores que foram carregados anteriormente.

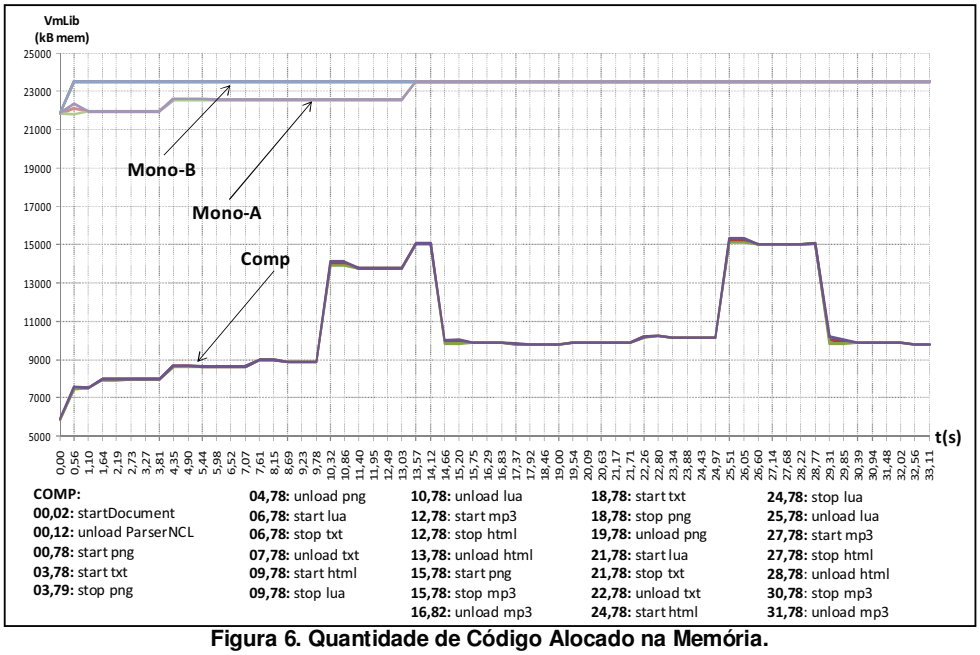

Figura 6. Quantidade de Código Alocado na Memória.

Ainda na Figura 6, é interessante notar que os instantes t=10,78s, t=13,78s, t=25,78s e t=28,78s são os únicos em que tem início a redução de uma quantidade considerável de código alocado na

memória para a implementação *Comp*. Isso também é decorrente do fato mencionado no quarto parágrafo desta seção: em nenhum instante a biblioteca *DirectFB* retira seus decodificadores da memória. Ao contrário dos decodificadores de imagem, texto e áudio, que são controlados pela biblioteca *DirectFB*, os exibidores HTML (liberação iniciada em t=13,78s e t=28,78s) e Lua (liberação iniciada em t=10,78 e t=25,78s) são controlados pelo Gerente de Componentes do Ginga-NCL.

Como era de se esperar, a eficiência na utilização de recursos de armazenamento é muito grande. Resta saber se isso acontece a um grande custo de processamento. A Figura 7 apresenta um gráfico comparativo do uso de CPU entre as duas implementações *Comp* e *Mono* enquanto realizam a apresentação das aplicações discutidas no quarto e quinto parágrafo desta seção. Note, na Figura 7, a diferença inicial do processamento exigido entre a versão *Comp* e o melhor caso da versão *Mono* (*Mono-A*), devido ao carregamento instantâneo de todas as bibliotecas do Ginga-NCL realizado na implementação *Mono*. Além disso, pode-se notar que todas as operações realizadas para carga e liberação de componentes da implementação *Comp* não acarreta sobrecarga relevante de processamento. Finalmente, na Figura 7, note que o uso de CPU da versão *Comp* está sempre abaixo do uso de CPU da versão *Mono-A*, exceto a partir do instante t=25,51, em que o uso de CPU entre as duas versões se iguala.

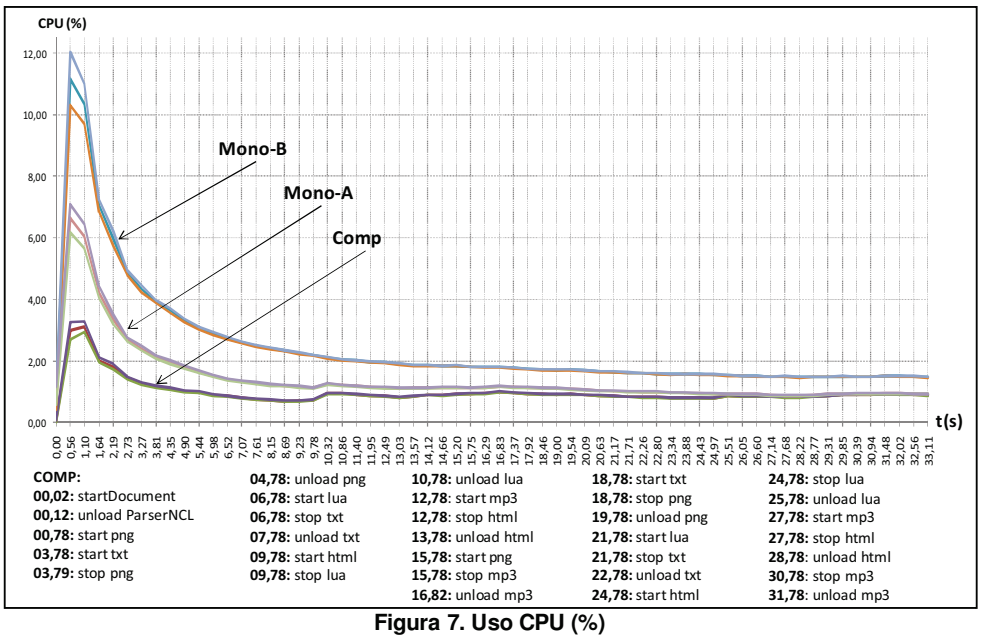

Figura 7. Uso CPU (%)

6. Considerações Finais

Entre as principais questões envolvidas no planejamento, desenvolvimento e implementação de middlewares declarativos para TV digital interativa está a economia de recursos computacionais e o suporte à evolução do middleware em tempo de execução. As soluções baseadas em componentes de software vêm se mostrando atraentes para prover um alto grau de adaptabilidade e um maior controle do uso de recursos computacionais.

Dentre as principais contribuições deste trabalho, pode-se salientar a arquitetura baseada em componentes de software para o middleware Ginga-NCL, com um alto grau de adaptabilidade em tempo de execução, oferecendo um maior poder de extensão da arquitetura e um maior controle sobre os recursos computacionais utilizados. Além disso, foi apresentado um estudo de caso com os resultados de uso dos conceitos do desenvolvimento orientado a componentes no domínio de middleware para sistemas de TV digital interativa.

Como trabalho futuro pretende-se utilizar as estruturas definidas por um HTG, não só para garantir acuidade nos instantes de carga e liberação dos componentes adaptadores, mas também para a definição dos instantes de pré-busca para os componentes que deverão ser tratados pelo Gerente de Evolução Dinâmica. Além disso, pretende-se realizar um estudo para que cada componente adaptador seja tratado como um processo independente, de forma que comportamentos inesperados, como o apresentado pela biblioteca *DirectFB*, não comprometam os recursos do dispositivo receptor após o uso do componente. Finalmente, como trabalho futuro, pretende-se analisar o comportamento das duas configurações (*Mono* e *Comp*) da implementação de referência em situações que existir uma partição de swap habilitada.

Referências

[1] Szyperski, C., Gruntz, D., Murer, S., Component Software – Beyond Object-Oriented Programming. Second edition. ACM Press, 2002.

[2] Barbosa, N. Perkusich, A. Estudo Empírico Comparativo de Modelos de Componentes para o Desenvolvimento de Software com Suporte à Evolução Dinâmica e Não Antecipada. Anais do Workshop de Teses em Engenharia de Software (WTES 2006). 2006.

[3] ABNT NBR 15606-2 Associação Brasileira de Normas Técnicas. Digital Terrestrial Television Standard 06: Data Codification and Transmission Specifications for Digital Broadcasting, Part 2 – GINGA-NCL: XML Application Language for Application Coding (São Paulo, SP, Brazil, November, 2007). http://www.abnt.org.br/imagens/Normalizacao_TV_Digital/ABNTNBR15606-2_2007Ing_2008.pdf.

[4] ITU-T Recommendation H.761, 2009. Nested Context Language (NCL) and Ginga-NCL for IPTV Services. Geneva, April, 2009.

[5] Coulson G. et al. A Generic Component Model for Building Systems Software. ACM Transaction on Computer Systems (TOCS), Volume 26, Issue 1, 2008.

[6] Ramdhany R. et al. MANETKit: supporting the dynamic deployment and reconfiguration of ad-hoc routing protocols. Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware. Urbanna, Illinois. 2009.

[7] Gomes A. T. A. LindaX: Uma Linguagem de Descrição de Sistemas de Comunicação Adaptáveis. Tese de Doutorado. Departamento de Informática. PUC-Rio. Março de 2005.

[8] Bruneton E. et al. The FRACTAL Component Model and Its Support in Java. Software, Practice and Experience. 36(11-12): 1257-1284. 2006.

[9] Layaïda O., Hagimont D., Designing Self-Adaptive Multimedia Applications through Hierarchical Reconfiguration, 5th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS), Athens, Greece, June 2005.

[10] Filho S. M. et al. FLEXCM – A Component Model for Adaptive Embedded Systems. Proceedings of the 31st Annual International Computer Software and Applications Conference - Volume 01. Pages: 119-126. 2007.

[11] Soares L.F.G., Rodrigues R.F. "Nested Context Model 3.0: Part 1 – NCM Core". Relatório Técnico, Laboratório TeleMídia, PUC-Rio, Brasil, 2005.

[12] Weber, J., Newberry, T. "IPTV Crash Course", 2007, The McGraw-Hill Companies.

[13] Costa, R., Moreno M.F., Soares L.F.G. DocEng, ACM Symposium on Document Engineering, "Intermedia Synchronization Management in DTV Systems", São Paulo, Brazil, 2008.