3
Gerência de Recursos e Evolução Dinâmica

O projeto orientado a componentes de software da arquitetura do middleware declarativo Ginga é discutido neste capítulo. Como prova de conceito, as soluções propostas para a arquitetura, discutidas na Seção 3.1, foram incorporadas na implementação de referência do middleware, como apresentado na Seção 3.2. Validando as propostas apresentadas, as medições apresentadas na Seção 3.3 indicam os benefícios que uma arquitetura baseada em componentes pode trazer a um middleware para sistemas de TVD, tais como a redução do consumo de recursos computacionais e o aumento da capacidade de evolução dinâmica.

3.1.
Arquitetura

O middleware Ginga oferece obrigatoriamente suporte à execução de aplicações desenvolvidas na linguagem NCL (Soares, 2006) e sua linguagem de script Lua (Ierusalimschy, 2006). 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.

Nesta tese, 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 (Szyperski, 2002): 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 neste capítulo 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*[6]. 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.

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

3.1.1.
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.

Para gerenciar o armazenamento temporário local das aplicações de TV, incluindo os conteúdos que referenciam, o módulo Gerente de E/S foi definido. 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 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 (Soares, 2009b). 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.

3.1.2. 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 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 (Soares, 2005). 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.

O módulo Gerente de Recuperação é responsável pela criação de um plano de recuperação a falhas (discutido no Capítulo 4).

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.

3.2. Implementação Ginga e Modelo de Componentes

A implementação de referência do Ginga foi realizada considerando o sistema operacional Linux, a biblioteca *DirectFB* e outras bibliotecas livres e de código aberto que serão apresentadas nesta seção. Essas bibliotecas foram selecionadas para que fosse construída uma implementação de middleware que pudesse executar tanto em terminais de baixo custo como em receptores com maiores recursos.

DirectFB é uma biblioteca que permite a manipulação de interfaces gráficas sem a necessidade da existência de um gerente de estruturas pesadas de um servidor X (*X Window System Server*[7]).

[7] http://linux.die.net/man/1/xserver

Para realizar o processamento de documentos XML é utilizada a biblioteca *libxerces-c*. Essa biblioteca é a base para a construção das entidades do modelo NCM que são dispostas em uma base privada.

Além dessas duas bibliotecas, para a resolução de URIs dos conteúdos dos objetos de mídia a serem exibidos, assim como dos arquivos contendo as descrições dos programas interativos (documentos NCL, por exemplo), foi desenvolvida uma biblioteca própria e incorporada aos serviços básicos do middleware (mais detalhes sobre a implementação do suporte à localização de recursos são discutidos no Capítulo 6). O uso dessa implementação permite que o receptor opere em um modelo multi-redes, podendo localizar recursos não apenas no meio de armazenamento local como também em servidores acessíveis via canal de interatividade.

A implementação do Ginga para a TVD terrestre supõe o modelo de carrossel de objetos conforme o padrão DSM-CC e o suporte oferecido por esse padrão para o envio de eventos de sincronismo DSM-CC, para receber as descrições dos programas interativos e os comandos de exibição e de edição (mais detalhes sobre a implementação do suporte aos comandos de edição são discutidos no Capítulo 5).

O desenvolvimento de exibidores baseou-se nas seguintes bibliotecas, também livres e de código aberto: *libpng*, *libjpeg*, *libmbrowser*, *libxine* e *liblua.* Além disso, para desenvolver o exibidor de áudio e vídeo decodificado por hardware foi utilizada a API video4linux (Cox, 2000).

As bibliotecas *libpng* e *libjpeg* são utilizadas para decodificar imagens estáticas nos formatos PNG (*Portable Network Graphics*) e JPEG (*Joint Photographic Experts Group*), respectivamente. Ambas foram implementadas na linguagem C e, geralmente, são oferecidas nos pacotes de instalação dos sistemas Linux. Os exibidores de imagens estáticas foram, portanto construídos utilizando essas bibliotecas.

A biblioteca *libtelemidialinks* foi desenvolvida neste trabalho com o objetivo de integrar um exibidor de objetos HTML ao middleware Ginga-NCL. Essa biblioteca é baseada no navegador links[8], que interpreta páginas HTML 4.0, porém sem suporte a CSS e JavaScript. O código do navegador links precisou ser

[8] http://links.twibright.com/

alterado porque sua implementação não permitia que uma aplicação usuária especificasse a região exata na tela onde a janela do navegador deveria ser aberta. Além disso, o navegador links não foi implementado como uma aplicação do tipo multi-aplicação (i.e. aplicações que permitem que a interface gráfica seja compartilhada por outras aplicações). Para resolver essas limitações, foram realizadas as seguintes modificações: cada instância do navegador passou a consistir em uma linha de execução independente; e a região que o navegador utiliza passou a poder ser parametrizada.

Geralmente, as decodificadoras MPEG dos receptores de TVD não possuem capacidade para decodificar o programa principal e um objeto de vídeo simultaneamente. Dessa forma, para permitir a exibição de objetos de vídeo em paralelo com a exibição do programa principal decodificado por hardware, é utilizada a biblioteca *libxine*. Essa biblioteca permite também que vídeos (inclusive do programa principal) sejam decodificados em plataformas que não possuam hardware de decodificação MPEG. Evidentemente, a resolução dos vídeos decodificados por software irá depender da capacidade de memória e processamento do dispositivo receptor. A biblioteca *libxine* possui capacidade para decodificar objetos de mídia MPEG-1, MPEG-2 e outros[9].

A biblioteca *libxine* é utilizada também para decodificar objetos isolados de áudio (i.e. que não foram multiplexados com objetos de vídeo).

Com o objetivo de interpretar códigos imperativos escritos na linguagem Lua, foi incorporada ao Ginga-NCL a biblioteca *liblua*. Essa integração permite que o paradigma puramente declarativo seja estendido para que o Ginga-NCL funcione com um modelo híbrido declarativo e imperativo. O uso de Lua oferece vantagens: combina programação imperativa com poderosas construções para descrição de dados, baseadas em tabelas associativas e semântica extensível. Lua é uma linguagem tipada dinamicamente, interpretada a partir de *bytecodes*, e tem gerenciamento automático de memória com coleta de lixo.

As bibliotecas de decodificação *libpng, libjpg e libxine* são utilizadas através de estruturas que servem como abstração, oferecidas pela biblioteca DirectFB.

[9] http://www.xine-project.org/home

A gerência de leiaute do Ginga-NCL e o processamento de eventos através de dispositivos interativos (controle remoto, por exemplo) foram construídos utilizando-se da biblioteca DirectFB.

Cada região espacial é associada a uma superfície do DirectFB. Quando a exibição de um objeto de mídia é iniciada, um controlador de eventos do usuário é instanciado. Esse objeto cadastra-se no gerente de eventos do DirectFB como ouvinte de eventos do usuário.

Quando uma superfície é exibida, um processo é criado para esperar eventos do usuário, que possivelmente serão gerados pelos dispositivos de entrada da plataforma, através da superfície que possuir o foco. Esse processo executa apenas enquanto a superfície estiver em exibição. Ao ser gerado um evento, em qualquer superfície que tiver o foco, todos os ouvintes de eventos do usuário são notificados por um gerente de eventos, passando o evento como parâmetro. No tratamento da notificação, verifica-se se o objeto de mídia sendo controlado espera alguma seleção relacionada ao evento recebido. Isso é feito para identificar o evento de seleção específico, a ser tratado pelo módulo Escalonador de Apresentação. Dessa forma, independente de qual superfície tenha o foco, todas as entidades em exibição são sempre notificadas do evento de usuário gerado, garantindo que as ações relacionadas à ocorrência do evento, naquele instante, sejam realizadas de acordo com o especificado na autoria da aplicação.

A implementação de referência foi embarcada em diferentes plataformas. Detalhes sobre a implantação em dispositivos portáteis podem ser encontrados em (Medina, 2008). Com o objetivo de oferecer aos autores de aplicações NCL um ambiente de testes de aplicações independente de plataforma (Moreno, 2009a), o a implementação de referência foi embarcado em uma distribuição Linux. Todo o ambiente encontra-se auto-contido em um CD "bootável".

A partir da versão 0.10.1 da implementação de referência[10] um módulo opcional, denominado Gerente de Componentes, foi criado. A instalação opcional do Gerente de Componentes define, em tempo de compilação, a forma com que as bibliotecas que compõem o middleware serão geradas, tendo como resultado uma arquitetura "monolítica" ou orientada a componentes. A arquitetura "monolítica",

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

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-NCL para plataformas que não ofereçam suporte ao desenvolvimento orientado a componentes. As duas alternativas serão também importantes em medições comparativas, como as apresentadas na Seção 3.3.

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*[11], 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*.

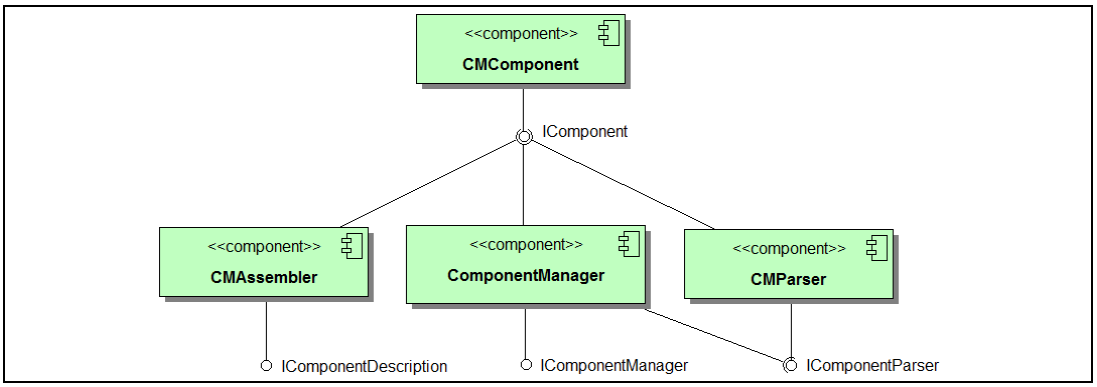


Figura 2 – Diagrama de Componentes do Gerente de Componentes Ginga-NCL.

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

[11] Linux Programmer's Manual: http://www.kernel.org

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 *IComponentDescriptor*, 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 no Quadro 1. 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.

```
<?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>
```

Quadro 1 – DTD XML Descrição da Arquitetura Ginga-NCL

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*.

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 herdam de 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 (Morris, 2005).

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 no Quadro 2, em que um objeto de mídia que referencia um conteúdo *DivX* [12], não conforme às normas Ginga-NCL (ABNT, 2009; ITU-T, 2009), é especificado.

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

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

Quadro 2 – Aplicação NCL que Demanda Evolução Dinâmica.

No Quadro 2, o documento NCL define em seu cabeçalho: uma região (linhas 6 e 7), cujos atributos *left*, *top*, *width* e *height* indicam as dimensões na tela onde será apresentado o objeto *DivX*; e um descritor (linha 10) associado a tal região.

A linha 14 indica que o documento terá sua apresentação iniciada pelo nó de mídia "videoMM" (linha 15), 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 3.1.2), 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*[13] 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 16 a 19), ao Gerente de Evolução Dinâmica.

[13] O elemento *meta* é especificado na linguagem NCL como um único par, propriedade e valor, nos atributos *name* e *content*, respectivamente (ABNT, 2009; ITU-T, 2009).

De posse dessas informações, o Gerente de Evolução Dinâmica infere, através da meta informação definida nas linhas 16 e 17, que uma descrição do componente "ginganclidivxadapter" 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 18 e 19.

Suponha agora que o documento XML adquirido pelo Gerente de Evolução Dinâmica através do módulo de Transporte seja aquele apresentado no Quadro 3. Esse documento, conforme a DTD apresentada no Quadro 1, consiste no conjunto de descrições de componentes, inclusive do componente exibidor para objeto de mídia *DivX*.

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

Quadro 3 – Descrição Remota de Componentes.

Conforme o Quadro 3, o componente "ginganclidivxadapter", em sua versão "1.0.1", pertence ao módulo "gingancl" (linhas 2 e 3). Nas linhas 4 e 5 os componentes "divxplayer", versão "1.0.1", e "gingancladapter", versão "0.12.1", são definidos como dependências do componente "ginganclidivxadapter". Assim, o Gerente de Evolução sabe quais componentes deverão ser atualizados juntamente com o componente "ginganclidivxadapter". Nas linhas 6 e 7, a localização do componente "ginganclidivxadapter" é definida. Os símbolos para criação e destruição do exibidor "DivXPlayerAdapter", que implementa a interface "IPlayerAdapter", são também definidos no XML (linhas 8 a 11). Os componentes definidos como dependências do componente "ginganclidivxadapter" são também especificados nesse mesmo documento XML.

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 7 do Quadro 3) 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 mostra-se 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[14], conforme definido por Costa et al. (Costa, 2006a).

3.3. 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. Os resultados de outras medições são apresentados no Anexo III desta tese.

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* (Marsh, 2009)). Inicialmente, a memória RAM foi limitada a 256MB (mem=256M no arquivo de configuração do boot (Marsh, 2009)), com apenas 144MB livres (sistema operacional e os outros processos utilizam 112MB).

Como já apresentado anteriormente, para a manipulação de interfaces gráficas, bem como a decodificação e renderização de textos, imagens, áudios e vídeos, a biblioteca *DirectFB* foi utilizada.

A biblioteca DirectFB realiza, internamente (sem controle da implementação do middleware Ginga-CC), a carga dinâmica do decodificador. Dessa forma, para simularmos o comportamento das implementações MONO e COMP, dois documentos NCL testes tiveram de ser utilizados, para simular o mesmo cenário. O documento A, usado no teste da implementação COMP, ao ser iniciado, dispara a apresentação em seqüência de um objeto de texto, seguido de um objeto de imagem, seguido de um objeto de áudio, seguido de um objeto declarativo com código HTML, seguido de um objeto imperativo com código

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

Lua; seqüencia que é repetida duas vezes. Para simular o efeito do mesmo documento A em uma implementação MONO, um documento B é usado, contendo objetos de mídia de texto, imagem, áudio, HTML e Lua, todos disparados simultaneamente. O documento A tem exatamente a mesma duração do documento B. Note que ao disparar simultaneamente os componentes do documento B, estamos obrigando a biblioteca DirectFB carregar 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 texto, imagem 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 controladas pelo próprio Ginga-NCL, esse problema não acontece.

Deve ser também salientado 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.

A Figura 3 apresenta um gráfico comparativo da quantidade de código alocado na memória entre as duas implementações (MONO, em verde; COMP, em vermelho) enquanto realizam a apresentação das aplicações respectivas. Note, na Figura 3, que no intervalo entre 0s e 0,625s a implementação MONO carrega na memória todas as funcionalidades da biblioteca DirectFB necessárias. Por sua vez, a implementação COMP carrega apenas o necessário para iniciar a

apresentação da aplicação NCL (veja Seção 3.1). No instante 0,625s a diferença entre as duas versões era de 15620 kB de memória.

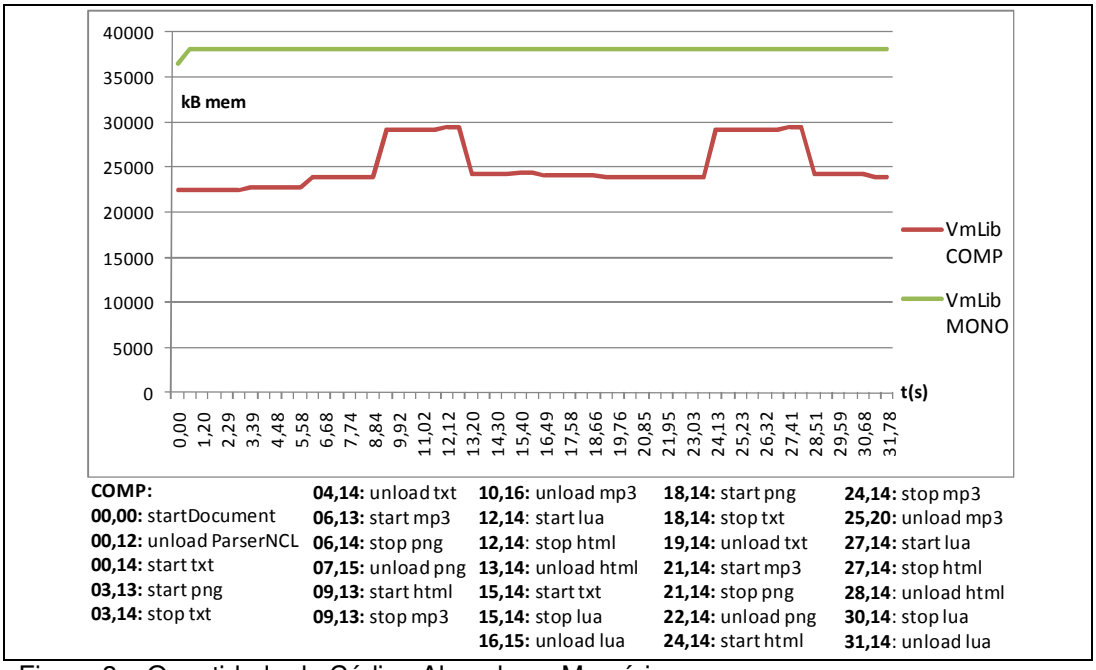
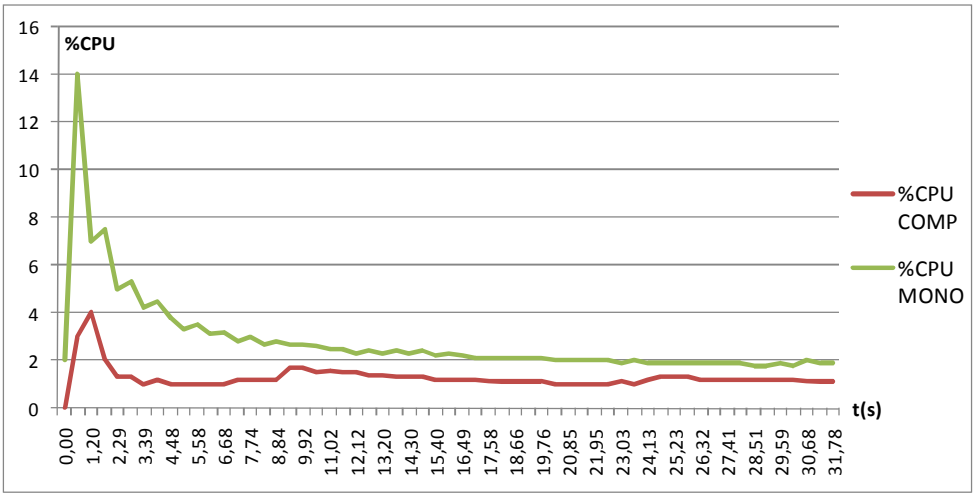


Figura 3 – Quantidade de Código Alocado na Memória.

Ainda na Figura 3, é interessante notar que os instantes t=13,14s, t=16,15s, t=28,14s e t=31,15s são os únicos em que é reduzida a quantidade de código alocado na memória para a implementação COMP. Isso é decorrente do fato mencionado no terceiro parágrafo desta seção: em nenhum instante a biblioteca DirecFB retira seus decodificadores da memória. Ao contrário dos decodificadores de texto, imagem e áudio, que são controlados pela biblioteca DirectFB, os exibidores HTML (liberado em t=13,14s e t=28,14s) e Lua (liberado em t=16,15 e t=31,14s) 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 4 apresenta um gráfico comparativo do uso de CPU entre as duas implementações (MONO, em verde; COMP, em vermelho) enquanto realizam a apresentação das aplicações discutidas no segundo parágrafo desta seção. Note, na Figura 4, a diferença inicial do processamento exigido entre as duas versões, 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 nenhuma sobrecarga relevante de processamento.



COMP:
00,00: startDocument
00,12: unload ParserNCL
00,14: start txt
03,13: start png
03,14: stop txt
04,14: unload txt
06,13: start mp3
06,14: stop png
07,15: unload png
09,13: start html
09,13: stop mp3
10,16: unload mp3
12,14: start lua
12,14: stop html
13,14: unload html
15,14: start txt
15,14: stop lua
16,15: unload lua
18,14: start png
18,14: stop txt
19,14: unload txt
21,14: start mp3
21,14: stop png
22,14: unload png
24,14: start html
24,14: stop mp3
25,20: unload mp3
27,14: start lua
27,14: stop html
28,14: unload html
30,14: stop lua
31,14: unload lua

Figura 4 – Uso CPU.