Ginga-NCL em Dispositivos Portáteis: Uma Implementação para a Plataforma Android *

Guilherme Daher Ferreira
daher@inf.ufes.br

Guilherme Nogueira
guilherme@inf.ufes.br

Giovanni Comarela
gcomarela@inf.ufes.br

Fábio Fabris
ffabris@inf.ufes.br

Magnos Martinello
magnos@inf.ufes.br

José Gonçalves P. Filho
zegonc@inf.ufes.br

Departamento de Informática - LPRM
Universidade Federal do Espírito Santo - UFES
Av. Fernando Ferrari, 514, 29060-970 - Vitória/ES, Brasil

ABSTRACT

Recently, there has been in Brazil a few number of devices capable of receiving the digital television signal. However, these devices lack the Brazilian Digital Television System's *middleware*, named Ginga, and therefore they do not allow the execution of interactive multimedia content. This paper reports an implementation of Ginga-NCL for Google's Android platform. In order to validate the implementation, experiments were conducted to analyze the execution of NCL applications, as well as the usage of devices resources. Moreover, the developed environment allows to evaluate the transmission and reception of NCL applications through the standardized DSM-CC over IP protocol.

RESUMO

Hoje já existem no Brasil aparelhos que permitem a recepção do sinal de TV digital. No entanto, estes dispositivos não estão equipados com o Ginga, *middleware* adotado pelo Sistema Brasileiro de Televisão Digital - SBTVD, o que inviabiliza a execução de conteúdo multimídia interativo. Este trabalho descreve uma implementação do Ginga-NCL para dispositivos portáteis baseados no sistema operacional Android. Como meio de validar a implementação, foram conduzidos experimentos para analisar a execução de aplicações NCL, bem como o uso de recursos do dispositivo portátil. Além disso, o ambiente desenvolvido permite avaliar a transmissão e recepção de aplicações NCL por meio do protocolo padronizado DSM-CC sobre IP.

Categories and Subject Descriptors

I.7.2 [**Document Preparation**]: Languages and systems, Hypertext/Hypermedia, Markup languages, Standards.

*Ginga-NCL for Portable Devices: An Implementation for the Android Platform

General Terms

Design, Standardization, Languages

Keywords

Ginga, Middleware, SBTVD, Android OS, TV Digital, DTV

1. INTRODUÇÃO

O acesso ao conteúdo televisivo interativo a partir de dispositivos portáteis multifuncionais, como celulares e smartphones, é uma das grandes apostas para o crescimento da TV digital aberta no Brasil. Hoje já existem no país diversos fabricantes de dispositivos portáteis, tais como a Samsung e LG, cujos aparelhos permitem a recepção do sinal de TV digital. No entanto, estes dispositivos não estão equipados com o Ginga [16, 6], software nacional adotado como padrão de *middleware* pelo Sistema Brasileiro de Televisão Digital - SBTVD. O Ginga, em suas duas modalidades, uma declarativa, o Ginga-NCL [16], e outra imperativa, o Ginga-J [6], fornece uma série de facilidades para a construção e a execução de aplicações interativas em ambiente de TV digital aberta. Desta forma, os dispositivos portáteis disponíveis no mercado brasileiro, embora se aproveitem da qualidade superior de áudio e vídeo inerente às tecnologias digitais, não suportam a interatividade. Logo, existe uma demanda natural por implementações do *middleware* Ginga voltadas para terminais portáteis.

No cenário de mobilidade, a escolha do sistema operacional embarcado no dispositivo portátil, assim como de suas plataformas de desenvolvimento associadas, tornam-se questões proeminentes no processo de desenvolvimento de novas implementações Ginga. Uma primeira iniciativa acadêmica de desenvolvimento do Ginga-NCL para dispositivos móveis, foi realizada na PUC-RJ [7], tendo como plataforma base o sistema operacional Symbian [14].

A plataforma Android foi lançada em 2008 pelo Google, através do consórcio *Open Handset Alliance* [3]. Trata-se de uma plataforma de código aberto, não vinculada a apenas um fabricante, fato este que tem permitido a ampliação da restrita área de telefonia móvel. O Android consiste em um sistema operacional baseado no kernel Linux 2.6 e provê uma *Application Programming Interface* (API) na linguagem Java cujas aplicações desenvolvidas são compiladas para a máquina virtual Dalvik. Esta máquina virtual foi projetada especialmente para a plataforma Android, sendo otimizada para rodar em dispositivos portáteis. Para os desenvolvedores, há um *Software Development Kit* (SDK) disponível gratuitamente e já existe uma ampla gama de aparelhos com Android embarcado.

Este trabalho apresenta uma implementação do *middleware* Ginga-

NCL para dispositivos portáteis, tendo como sistema operacional embarcado o Google Android. Um estudo experimental foi realizado para avaliar a sensibilidade de aplicações NCL [15] e o desempenho do *middleware* desenvolvido, incluindo o atraso decorrente da preparação dos *players* considerando aplicações contendo um número elevado de mídias simultâneas, e a utilização de CPU e memória durante a execução de aplicações NCL com características distintas. O ambiente de testes empregado permite experimentar aplicações multimídia escritas na linguagem NCL, extraindo medidas de interesse. No caso da transmissão e recepção de conteúdo, foi implementado o protocolo DSM-CC (*Digital Storage Media Command and Control*) [11] sobre IP, permitindo efetuar testes preliminares da capacidade de recepção do dispositivo portátil empregado.

As demais seções do artigo estão organizadas da seguinte forma: Seção 2 discute os trabalhos relacionados. A seção 3 descreve a arquitetura do *middleware* Ginga-NCL e os detalhes da implementação realizada no sistema Android. Seção 4 contempla o estudo experimental realizado com vistas à avaliação sistêmica do protótipo desenvolvido. Por fim, a seção 5 conclui o artigo apresentando as considerações finais e as perspectivas de trabalhos futuros.

2. TRABALHOS RELACIONADOS

Prover televisão digital para dispositivos móveis é um objetivo comum em qualquer sistema de televisão digital. Para atingir tal objetivo alguns padrões foram propostos e implementados. O foco desta seção é voltado para descrever brevemente alguns destes padrões.

O *Digital Video Broadcasting* (DVB) é o padrão europeu para televisão digital e define o *Digital Video Broadcasting - Handheld* (DVB-H) [9], como padrão de transmissão da TV Digital para os dispositivos portáteis. O DVB-H é uma extensão do padrão para transmissão terrestre DVB-T, voltado apenas para transmissão. No entanto, os diferentes provedores de conteúdo nos países que o adotam, disponibilizam middleware para aparelhos compatíveis.

Digital Multimedia Broadcasting (DMB) [8] é uma tecnologia de transmissão desenvolvida pela Coréia do Sul como parte do projeto nacional de tecnologia de informação para transmissões de TV, rádio e dados para dispositivos portáteis. Este sistema oferece suporte para operação via satélite (S-DMB) ou terrestre (T-DMB). O transporte foi implementado com o uso do MPEG-2 TS [10] e seu *middleware* é de carácter procedural baseado no *Java Micro Edition* (JME).

A *Association of Radio Industries and Businesses* (ARIB) é responsável pela padronização do sistema de televisão digital japonês, o *Integrated Services Digital Broadcasting Terrestrial* (ISDB-T). Este padrão permite a transmissão de conteúdo para dispositivos portáteis através do *one-seg*. Na transmissão digital do ISDB-T, cada canal é dividido em 13 segmentos, dos quais 12 são utilizados para transmissão aos dispositivos fixos, capazes de exibir conteúdo de alta qualidade, e 1 segmento é utilizado para transmissão aos dispositivos portáteis, o que nomeia esta forma de transmissão como *one-seg*. O *middleware* de TV Digital do ISDB-T é baseado na linguagem *Broadcast Markup Language* (BML), que é de natureza declarativa e baseada em XHTML1.0 [4].

O Sistema Brasileiro de Televisão Digital (SBTVD) tem como base o ISDB-T e é conhecido mundialmente como *ISDB-T International*. Esta nomenclatura deve-se a um processo de harmonização de padrões definido pelo *Digital Broadcasting Experts Group* (DiBEG), órgão do ARIB que promove o padrão ISDB-T.

A principal diferença entre os padrões japonês (ISDB-T) e o brasileiro (ISDB-T International) é a especificação do *middleware* brasileiro, denominado Ginga, tanto para set-top-boxes quanto para dispositivos portáteis. O Ginga-NCL, parte declarativa do *middleware*, possui o profile *Basic DTV*, utilizado para dispositivos portáteis, especificado em [2]. Além de utilizado no padrão brasileiro de TV digital, o Ginga-NCL é padronizado através da recomendação H.761 do ITU-T [12] como *middleware* para aplicações multimídia em IPTV, provendo interoperabilidade e harmonização entre os distintos *frameworks* para desenvolvimento destas aplicações.

No trabalho [7], foi apresentada uma implementação do *middleware* Ginga para dispositivos portáteis baseados no sistema operacional Symbian. Trata-se da primeira implementação de referência do Ginga-NCL para dispositivos portáteis e, como tal, possui uma descrição estendida sobre o padrão e trabalhos relacionados. O artigo tem seu foco na descrição da arquitetura e na implementação do *middleware*, não apresentando testes experimentais sobre a plataforma desenvolvida. Na época de desenvolvimento do trabalho não existiam plataformas abertas amplamente disponíveis, como é o caso do sistema Android, motivo pelo qual, apesar de ressaltar a vantagem de se utilizar uma plataforma aberta e livre, escolheu-se o sistema fechado Symbian[1] para desenvolvimento.

O presente trabalho distingue-se dos anteriores ao apresentar não apenas uma implementação do Ginga-NCL em sistema Android, mas também efetuando um conjunto de experimentos realizados sobre a plataforma, complementando as discussões iniciadas em [7], particularmente por meio de uma análise aprofundada da execução de aplicações NCL em dispositivos portáteis.

3. MIDDLEWARE DECLARATIVO GINGA-NCL EM ANDROID: ARQUITETURA E IMPLEMENTAÇÃO

O universo das aplicações Ginga pode ser classificado como um conjunto composto por aplicações declarativas e aplicações procedurais. Aplicações declarativas são tratadas pelo subsistema Ginga-NCL e aplicações procedurais pelo subsistema Ginga-J. De acordo com as normas NBR-15606-4 e NBR-15606-5 [2, 1] definidas para o Sistema Brasileiro de Televisão Digital (SBTVD), dentre os dois subsistemas, apenas o declarativo é obrigatório para dispositivos portáteis.

Este trabalho concentra-se no desenvolvimento do middleware declarativo Ginga-NCL, o que habilita a execução de aplicações NCL em dispositivos portáteis equipados especificamente com o sistema operacional Android. Inicialmente uma breve introdução à arquitetura conceitual do Ginga é apresentada, seguida de uma descrição mais detalhada dos componentes arquiteturais que foram implementados ou alterados nesta primeira versão do Ginga-NCL para Android.

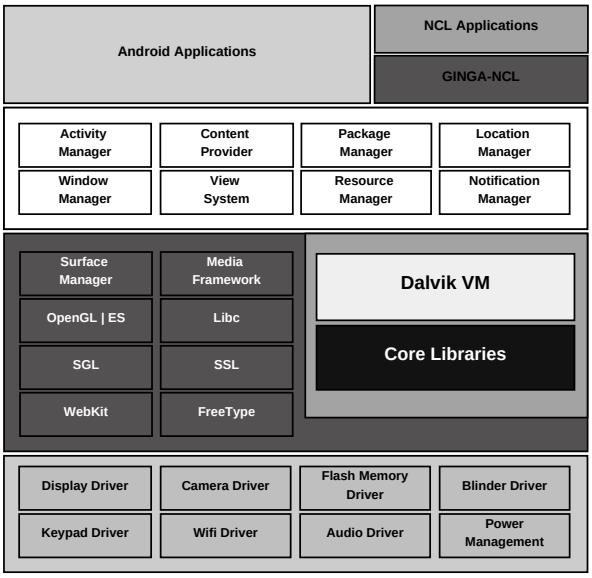
3.1 Arquitetura

Do ponto de vista conceitual, o *middleware* é uma camada na arquitetura que provê serviços para a camada de aplicação. No caso do Ginga-NCL, este *middleware* é responsável pela execução das aplicações NCL e, considerando o sistema Android, conforme mostrado na Figura 1(a), está posicionado sobre o arcabouço (*framework*) de desenvolvimento.

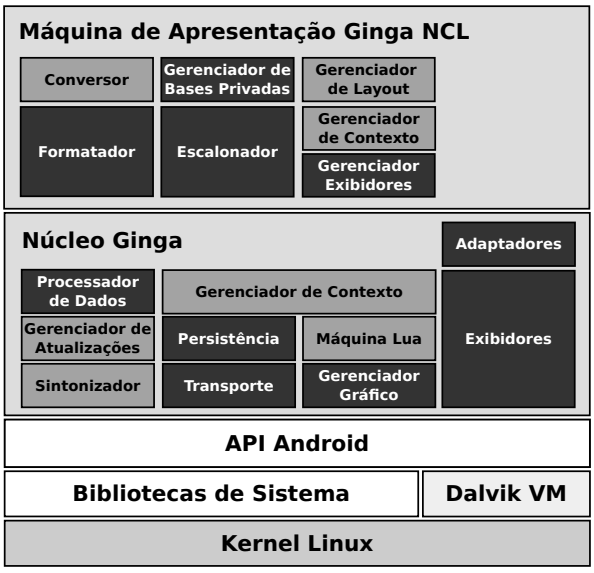
A Figura 1(b) ilustra a sua componentização, na qual pode-se ver a divisão em dois subsistemas [7]: a máquina de apresentação NCL e o núcleo Ginga.

Na arquitetura do Ginga-NCL para Android, assim como para

[1]A versão utilizada foi a Symbian 9.2, parte do Symbian Series1. Esta é a última série de código fechado do sistema Symbian. As séries seguintes, lançadas a partir de 2009, possuem código livre aberto em licença *EPL - Eclipse Public License*



(a) Camada Ginga-NCL sobre Android



(b) Visão dos Componentes Ginga-NCL

Figure 1: Arquitetura Ginga NCL para dispositivos portáteis

set-top-boxes, o núcleo Ginga é responsável por prover recursos necessários à máquina de apresentação NCL. Seu componente **Sintonizador** tem o papel de receber conteúdo de TVD para dispositivos portáteis (1-seg) transmitido por provedores de conteúdo. As aplicações interativas podem chegar ao dispositivo de dois modos: multiplexadas no conteúdo recebido pelo Sintonizador, ou por outra interface de rede presente no dispositivo. Quando a aplicação estiver multiplexada, ela é obtida através de um processamento efetuado pelo módulo **Processador de Dados** sobre o conteúdo recebido. O componente **Transporte** é definido para gerenciar protocolos e interfaces de rede, para o caso em que a aplicação for recebida por este meio. Após o recebimento destas aplicações e do conteúdo referenciado por elas, o módulo **Persistência** atua para gerenciar o seu armazenamento.

O componente **Exibidores** é responsável pelos decodificadores específicos para cada tipo de mídia. Por outro lado, o componente **Adaptadores** é responsável pela padronização da comunicação entre a Máquina de Apresentação e a API de decodificação de conteúdo dos exibidores, sendo que para cada exibidor existe um adaptador associado. Ainda como os **Exibidores**, os **Adaptadores** seguem uma padronização de interface.

O **Gerenciador Gráfico** realiza o controle da renderização dos objetos especificados na aplicação NCL, definindo as dimensões e posicionamento destes objetos. O **Gerenciador de Atualizações**, possui como principal característica receber e instalar atualizações de maneira independente sem interromper o funcionamento do *middleware*, e o **Gerenciador de Contexto**, é responsável por gerenciar o perfil de usuário e demais informações contextuais.

No subsistema da máquina de apresentação, o principal componente é o **Formatador** e seu papel consiste em direcionar todas as ações a serem executadas. Inicialmente o **Formatador** solicita ao **Conversor** que processe a aplicação NCL. O resultado desta conversão é uma árvore de execução denominada Base Privada. A gerência de todas as bases presentes no *middleware* é feita pelo componente **Gerenciador de Bases Privadas**.

Após receber esta estrutura de dados, o **Formatador** dispara a execução do **Escalonador**, cuja responsabilidade é gerir a execução e temporização das mídias presentes na aplicação NCL. O **Escalonador** atribui ao **Gerenciador de Exibidores** o processamento dos recursos necessários para exibição de cada mídia e ao componente **Gerenciador de Layout** a responsabilidade de definir os parâmetros de exibição NCL no dispositivo.

3.2 Implementação

Para a implementação do Ginga-NCL em Android, partiu-se da implementação pública em Java para *set-top box*. No entanto, uma série de modificações foram efetuadas, principalmente nos métodos relacionados ao gerenciamento e manipulação dos elementos gráficos (vídeo, páginas HTML e imagens) e áudio.

Os componentes que tiveram de ser implementados ou alterados encontram-se com fundo escuro e fonte clara na Figura 1(b) e os que permaneceram inalterados em fundo claro com fonte preta.

Não foi possível utilizar diretamente a implementação pública, pois ela faz uso de recursos de bibliotecas gráficas não suportadas no dispositivo portátil (tais como *AWT* e *Swing*). A parte do código que não utiliza estas bibliotecas precisou de poucas modificações devido à natureza portável da plataforma de desenvolvimento Java/Dalvik. A implementação resultante é capaz de executar aplicações NCL declarativas em dispositivos portáteis de maneira satisfatória conforme apresentado na seção 4.

É importante salientar que os dispositivos disponíveis baseados no sistema Android não possuem ainda o hardware necessário para receber conteúdo transmitido via radiodifusão. Por estes motivos, a implementação atual ainda não possui um módulo *Sintonizador* funcional. Além disso, o módulo *Processador de Dados* encontra-se num estado inicial em que foi implementado um cliente para o protocolo DSM-CC, cuja descrição encontra-se na seção 4.3.

As próximas seções apresentam detalhes das classes que foram implementadas nesta primeira versão do Ginga-NCL em Android.

3.2.1 Formatador

A classe *Formatter* atua como um gerenciador em alto nível, sendo responsável pelo controle das aplicações NCL presentes no *middleware*. É este gerenciador o responsável por receber uma aplicação NCL, convertê-la para uma Base Privada, armazená-la e iniciar sua execução, além de repassar comandos para pausar e reiniciar uma aplicação.

A instanciação inicial da classe ocorre na *Activity* principal do *Middleware*, a classe *GingaMobile*. A classe mantém uma referência ao Formatador e, ao receber uma aplicação NCL, realiza as

chamadas para adicionar a aplicação e para iniciar a exibição. Os botões de controle para pausar e parar a aplicação possuem eventos que realizam as chamadas aos respectivos métodos do Formatador para a ação escolhida.

O processo de adição de uma aplicação ao Formatador é feito através da classe *NclDocumentManager*. O Formatador faz uma chamada ao método *addDocument* desta classe, que realiza as seguintes ações: compilar a aplicação NCL para gerar uma Base Privada, adicionar a Base Privada a um mapa de bases e retornar a Base Privada para o Formatador.

Para iniciar a exibição de uma aplicação NCL, o Formatador realiza uma navegação pela Base Privada da aplicação para criar uma lista de eventos de entrada, ou seja, os eventos iniciais da aplicação. Esta lista é enviada ao Escalonador que, a partir destes eventos iniciais, poderá iniciar a execução da aplicação.

3.2.2 Gerenciador de Bases Privadas

O módulo Bases Privadas tem como principal função armazenar a estrutura processada pelo Formatador NCL, responsável por tratar as aplicações recebidas pelo núcleo.

A principal classe deste módulo, **Entity**, estende a interface **IEntity**. Todas as classes deste módulo (nós, links, descritores, etc) devem implementar essa interface, considerando que cada entidade NCL/NCM tem como atributo um único Identificador (ID).

A área funcional *Layout* especifica elementos e atributos que definem como os objetos serão inicialmente apresentados dentro de regiões de dispositivos de saída. *Components* define os tipos básicos de objetos de mídia, e também é responsável pela definição de nós de contexto através de elementos <context>. A área funcional *Interfaces* permite a definição de interfaces de nós (objetos de mídia ou nós de composição) que serão utilizadas em relacionamentos com outras interfaces de nós. Por sua vez, *Linking* é responsável por definir os elos, que utilizam conectores. Além dos módulos básicos, a área funcional *Connectors* também define módulos que agrupam conjuntos de módulos básicos para facilitar a definição do perfil de linguagem. O NCL permite uma grande reutilização de seus elementos; para tal, a área funcional *Reuse* também foi implementada. *Animation* é, na verdade, uma combinação de fatores de suporte ao desenho do objeto e de suporte ao movimento do objeto ou, mais propriamente, suporte para a alteração do objeto em função do tempo. Por fim, *Metainformation* contém informações sobre o conteúdo utilizado ou exibido. Para a implementação destas classes foram seguidos os padrões definidos em [13].

3.2.3 Gerenciador de Gráfico

Para realizar o papel do Gerenciador Gráfico do *middleware* Ginga, foi implementada a classe **InterfaceUpdater**. Sua função básica é gerenciar a exibição e remoção de superfícies (*Surfaces*) na tela e, portanto, necessita alterar o layout principal do *middleware*. Entretanto, a plataforma Android não permite que todas as classes pertencentes a uma aplicação alterem o layout principal; apenas a *Activity* inicial pode fazê-lo.

De modo a contornar esta limitação da plataforma, a classe foi implementada utilizando um *Handler*, uma classe da API Android que permite executar uma *thread* com as permissões da *Activity* inicial, e duas classes internas que implementam a interface *Runnable*, *updateUI* e *changeImage*. A classe **changeImage** é utilizada para a troca de uma imagem exibida na tela e, como a exibição é feita através da classe **ImageView**, o método *changeView* da classe **InterfaceUpdater** é utilizado. Neste método, um vetor contendo a *View* a ser removida e a *View* a ser exibida é adicionado a uma lista de vetores a ser lida pela *thread changeImage* quando esta é executada.

Analogamente, a classe *updateUI* é utilizada para alterar o layout principal, adicionando e removendo elementos do tipo **AndroidWindow** adicionados à listas internas através dos métodos *addRemoveTask*, *addDisplayTask*, *addLayout* e *addClearTask*.

3.2.4 Gerenciador de Exibidores

O módulo Gerenciador de Exibidores compreende as classes necessárias ao gerenciamento da execução dos exibidores de mídias e de interação com usuário.

Pode-se destacar nesta implementação a interface **AndroidSurface**, responsável por definir uma superfície de exibição de mídia. É importante destacar que cada mídia executada numa aplicação NCL tem uma superfície de exibição a ela associada. Cada tipo de mídia requer superfícies com características diferentes. Para tal, a API Android provê classes distintas para exibição destas mídias. São elas: *ImageView*, *SurfaceView* e *WebView*. Estas classes são utilizadas como base para as superfícies de apresentação do *middleware*. A fim de realizar a interface entre o *middleware* e as bibliotecas nativas da API, foram implementadas a **AndroidAudioSurface**, **AndroidHTMLSurface**, **AndroidVideoSurface** e a **AndroidImageSurface**.

Uma outra classe importante é a **AndroidWindow**. Sua função é prover uma abstração da janela de trabalho do *middleware*, para tal, foi utilizada como base a classe de layout de exibição da API *LinearLayout* e a interface NCL **IWindow**. Em suma, todos os elementos exibidos pelo *middleware* estão contidos numa **AndroidWindow**.

Por fim, tem-se a classe **GingaKeyListener**, responsável por receber os dados de interação do usuário via teclado. Esta recebe os eventos e repassa-os para as classes necessárias para realizar a ação pretendida. Esta classe implementa a interface *OnKeyListener*, provida pela API Android, que é chamada sempre que um evento é enviado para uma respectiva *View*.

3.2.5 Exibidores

No módulo exibidores temos os apresentadores de mídia definidos para cada tipo suportado pelo *middleware*. Eles são utilizados pelos adaptadores para inserir as mídias numa apresentação e definem as funções básicas de apresentação necessárias, como: *start*, *pause*, *abort*, *resume* e *stop*. O comportamento destas funções é definido por norma, sendo que buscou-se, na implementação deste *middleware*, seguir exatamente as instruções contidas no capítulo 8.2 em [13].

O módulo foi implementado de forma que tem-se uma classe correspondente ao exibidor para cada adaptador, por exemplo, **AndroidAudioPlayer** e **AndroidAudioPlayerAdapter**, estão relacionados.

3.2.6 Adaptadores

Os adaptadores atuam como intermediários entre o formatador e os exibidores respectivos a cada tipo de mídia, de forma que existe uma relação um para um entre eles. Cabe ressaltar que tanto os adaptadores de mídia quanto os de código procedural implementam as interfaces de adaptador e listener de eventos. Desta forma, o adaptador recebe uma chamada de execução de uma mídia e carrega o exibidor necessário e, ao captar um evento, repassa-o a este exibidor para que este realize uma ação baseada neste evento.

Para implementação do módulo Adaptadores foi necessário estender as interfaces NCL **IFormaterPlayerAdapter** e **IProceduralPlayerAdapter**, além da interface **InputEventListener**. Para cada tipo de mídia a ser executado existe um adaptador distinto, sendo eles: **AndroidImagePlayerAdapter**, **AndroidVideoPlayerAdapter**, **AndroidHTMLPlayerAdapter**, **AndroidAudioPlaye-**

rAdapter e **AndroidLuaPlayerAdapter**.

A implementação dos quatro primeiros foi elaborada de forma que cada um deles estenda a classe **AndroidMediaPlayerAdapter**. Esta classe é responsável por agrupar características comuns àquelas que a estendem. Por fim, a classe **FormatterPlayerAdapter** é responsável por executar comandos de gerenciamento das mídias de Áudio, Vídeo, Imagem e HTML.

3.2.7 Escalonador

O escalonador de documentos NCL é um importante módulo da arquitetura do *middleware* implementado, sendo ele o responsável por orquestrar a exibição de todos os elementos presentes em um documento NCL.

A implementação do Escalonador tem como base o recebimento via parâmetro de um evento NCL, que pode ser instantâneo ou ter duração mensurável, e de uma ação a ser realizada sobre este evento. Um evento NCL pode ser do tipo Seleção, em que uma área ou nó do documento é selecionada; Atribuição, em que um valor é atribuído a uma propriedade de um nó NCL; Composição, com o qual um mapa da composição é exibido; ou Apresentação, em que a mídia contida em um nó NCL é exibida. Ações sobre os três primeiros tipos são ações mais simples e de execução rápida; porém, ações em eventos do tipo Apresentação são ações sobre uma mídia pertencente ao documento NCL e necessitam de um tratamento diferente. Este tipo de evento tem associado a ele um Exibidor que executará em uma *thread* separada a mídia em questão e sobre o qual serão executadas as ações.

É importante destacar, no Escalonador, a ação de *start* para um evento do tipo Apresentação. Ao ser feita esta chamada, o player da mídia em questão deve ser instanciado e preparado para exibição. Esta preparação consiste em criar uma superfície de exibição para a mídia e enviá-la ao Gerenciador de Layout para que esta possa ser adicionada ao conjunto de superfícies em exibição. Após realizar estas preparações, o escalonador é adicionado como listener do evento e é chamado o método *start* do *exibidor*.

Para manter a sincronia dos eventos, o Escalonador provê uma função de *Callback* e registra-se como listener dos eventos não instantâneos. Através deste *Callback* o Escalonador recebe atualizações sobre o estado de cada evento iniciado e consegue manter uma estrutura interna para gerência de eventos condicionais. Desta maneira, o escalonador gerencia a instanciação de novas *threads* e a construção das superfícies de exibição associadas. As ações de controle sobre as mídias são recebidas pelo escalonador e repassadas aos Exibidores específicos, e os eventos gerados pelos players, como o término de uma exibição, são passados ao escalonador através do *Callback*.

4. EXPERIMENTOS PARA VALIDAÇÃO

Para avaliar o desempenho do *middleware* foram desenvolvidas aplicações NCL que exploram características importantes desta plataforma. As aplicações foram executadas no *middleware* embarcado em um dispositivo HTC com processador Qualcomm RMSM 7200A, 528 MHz executando o sistema operacional Android, com 512 MB de memória ROM e 288 MB de memória RAM. As dimensões são 113 x 55.56 x 13.65 mm com resolução de 320x480 HVGA.

O primeiro experimento visa analisar o atraso decorrente da preparação inicial dos players de mídia chamados pelo escalonador em aplicações que contém um número elevado de mídias simultâneas. O segundo experimento foi desenvolvido para analisar a utilização de CPU e memória durante a execução de aplicações NCL com características diferentes. Por fim, o terceiro experimento analisa a implementação do protocolo DSM-CC sobre IP e a saturação do buffer de recepção no dispositivo.

4.1 Análise do tempo de preparo para exibição de mídias simultâneas

Conforme o apresentado na seção 3.2.7, o processo de iniciar um evento do tipo Apresentação é custoso e envolve a preparação da mídia. O experimento apresentado pretende analisar o comportamento do tempo de preparação para documentos NCL contendo um grande número de mídias a serem exibidas simultaneamente, indicando o tempo de atraso relativo a este processo.

Considerando que este tempo de preparo do exibidor pode variar significativamente em função do número de mídias simultâneas, foram elaboradas aplicações NCL incrementando-se gradativamente o número de mídias a fim de quantificar a variação do tempo de preparo. Os testes foram realizados em aplicações exibindo simultaneamente de 1 a 50 mídias. No entanto, o *middleware* não foi capaz de exibir arquivos com mais de 40 mídias devido às limitações de seus recursos de memória e processamento, tendo a execução abortada pelo sistema operacional durante o processo de preparação.

Para cada um dos arquivos foi realizado um estudo de tempo de preparação das mídias que seriam exibidas. As Figuras 2(a) e 2(b) ilustram os tempos de preparo para cada mídia presente nas aplicações NCL contendo 14 e 38 mídias de imagem de 2.3 KB cada, respectivamente, que deveriam ser exibidas simultaneamente. Os demais gráficos não foram plotados neste trabalho visto que o comportamento dos mesmos se mantém semelhante aos abaixo apresentados.

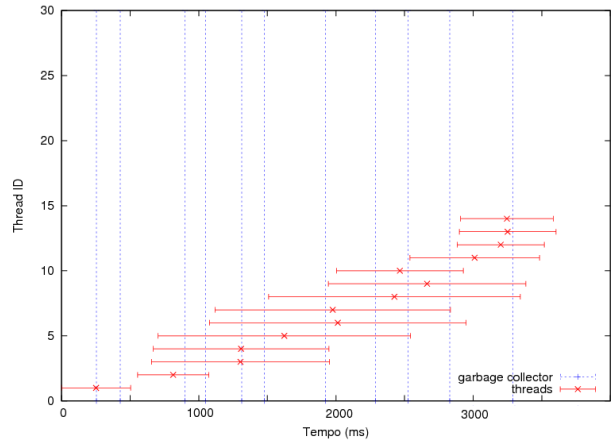
Durante os experimentos foi observado que haviam no máximo 6 *threads* concorrentes do *Media Player* (ocorrido, por exemplo na Figura 2(a)). O fato de haver um limite máximo de *threads* simultâneas era um resultado esperado para dispositivos portáteis. No entanto, essa observação sugere aos desenvolvedores de aplicativos NCL uma orientação quanto ao limite no número de mídias simultâneas que podem ser suportadas em aplicativos NCL nesta categoria de dispositivos, devido ao atraso considerável no tempo de preparo em decorrência deste número máximo de threads de carregamento. Isto é dito considerando que o dispositivo utilizado para testes reflete uma capacidade média-alta de processamento e memória atual para a categoria.

Além disso, pode-se perceber que várias *threads* tendem a terminar em um instante próximo à execução do *garbage collector* (GC), por exemplo, no instante 2350ms da Figura 2(b), onde 4 *threads* encerram sua execução logo após essa chamada. Esta característica peculiar é explicada pela liberação de recursos (memória) necessários para o término da execução dessas *threads* concorrentes.

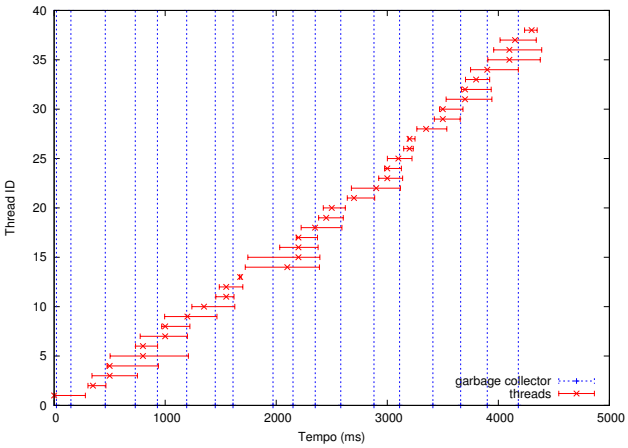
Nota-se ainda que o tempo de preparação das mídias está relacionado ao intervalo entre chamadas ao GC. Para um intervalo grande, há uma maior probabilidade de saturação dos recursos no dispositivo, isso leva a um aumento no tempo de execução da *thread*. Para intervalos menores, por exemplo, entre 1400ms e 1500ms da Figura 2(b), a liberação de recursos pelo GC permitiu uma diminuição no tempo de execução das threads 11, 12 e 13. No entanto, é evidente que há ainda um conjunto de *threads* do sistema que competem por recursos e isso também é um fator que afeta o tempo total de preparo.

4.2 Análise do uso de memória e CPU durante a execução de aplicações NCL

Recursos como memória e capacidade de processamento são usualmente limitados em dispositivos portáteis, fazendo com que o seu gerenciamento seja um ponto chave no desenvolvimento, em particular, de um *middleware* embarcado, cuja gestão de recursos tem



(a) Aplicação contendo 14 mídias NCL



(b) Aplicação contendo 38 mídias NCL

Figure 2: Tempo de preparo para aplicações NCL

grande impacto nas aplicações que o utilizam.

O objetivo deste teste é analisar a utilização de CPU e memória em duas aplicações NCL, desenvolvidas em conjunto à PUC-RJ, com características diferentes. Para obter as medidas referentes a esses recursos durante a execução, foi utilizada a ferramenta *Android Debug Bridge* (ADB)[2]. Tal ferramenta permite um acesso via *shell* ao sistema linux do Android, no qual o *middleware* possui um processo identificável. Baseando-se no */proc* [3] pode-se medir o uso de recursos pelos processos.

Para capturar as telas que serão apresentadas nesta seção, foi utilizada a ferramenta *Device Screen Capture* contida no Android SDK, através da qual é possível capturar uma imagem que está sendo exibido na tela do dispositivo.

4.2.1 Fórmula 1

A primeira aplicação apresenta uma exibição de um grande prêmio de Fórmula 1. A aplicação consiste em apresentar um vídeo principal, com resolução de 320x180, e algumas combinações de mídias de imagem secundárias que são exibidas de acordo com a sequência de toques no visor do dispositivo.

[2]Android Debug Bridge é uma ferramenta contida no Android SDK que permite gerir o estado de uma instância do emulador ou de um dispositivo executando Android

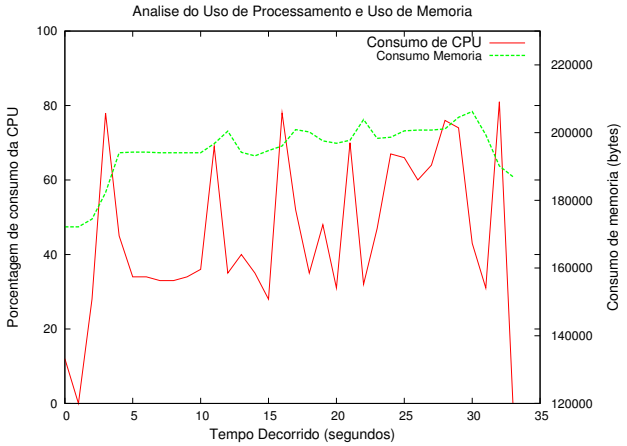
[3]O /proc no Linux é ponto de montagem de um pseudo sistema de arquivos que permite acessar informações de processos no sistema.

Na parte inferior da tela existem três opções, através das quais o telespectador poderá interagir com a aplicação. Caso o telespectador selecione a primeira opção, serão apresentados os pilotos participantes do Grande Prêmio, conforme Figura 3(a). Caso seja selecionada a segunda opção, serão apresentados as escuderias participantes, conforme Figura 3(b). Por fim, caso seja selecionada a ultima opção, serão exibidos os circuitos nos quais as corridas serão disputadas.

É possível ver na Figura 3(a) que a imagem do carro está sobreposta ao vídeo. Essa era uma das limitações da plataforma utilizada no trabalho [7]. Na implementação em Android, isto é possível devido ao gerenciador gráfico da API Android, que permite a sobreposição de *Surfaces*.

(a) Pilotos participantes na temporada

(b) Escuderias inscritas na temporada



(c) Análise de CPU e Memória

Figure 3: Fórmula 1

Observando a execução da aplicação, Figura 3(c), pode-se perceber que apesar de exibir um vídeo juntamente com imagens e interação com usuário, o uso da CPU não passa de 80%, enquanto a utilização de memória não chega a 10% da capacidade do dispositivo (210Kb de 288Mb disponíveis).

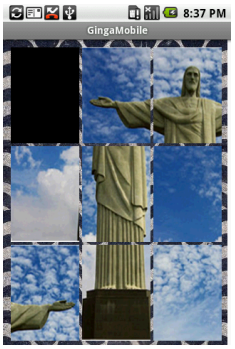
A existência de vales e picos no consumo de CPU é um comportamento que ocorre sempre que há uma interação do usuário selecionando uma das opções disponíveis. Ao fazer isto, um evento é

gerado pelo sistema Android e repassado ao *middleware*, que deve reconhecê-lo e realizar a ação relacionada, processo que resulta no aumento temporário do uso de CPU.

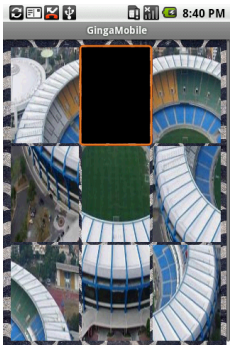
No que diz respeito ao consumo de memória, pode-se perceber um aumento do consumo até o instante 5s, que é explicado pela instanciação em memória dos objetos a serem manipulados pelo *middleware*. Após este instante, percebe-se uma certa linearidade no consumo de memória com pequenas variações de consumo. Estas variações ocorrem devido à troca das imagens exibidas e o preenchimento do buffer de exibição do vídeo, que representam um aumento do consumo, em contrapartida às execuções do GC, que representam um declínio. Ao final, com o consumo do buffer, a curva do uso de memória apresenta uma queda.

4.2.2 Mosaico

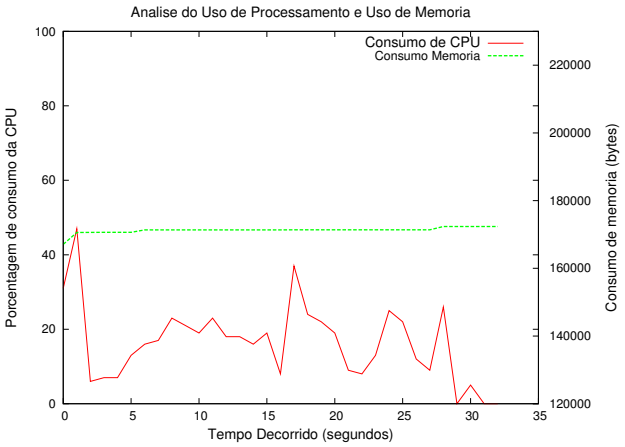
A segunda aplicação testada foi elaborada com base nos cartões postais da cidade do Rio de Janeiro. O seu objetivo é o de testar a integração com o exibidor HTML e o funcionamento do recurso de foco e seleção. Ao ser iniciada, a aplicação em questão divide a tela em nove espaços de igual tamanho. Cada um dos espaços é preenchido com uma imagem embaralhada aleatoriamente, conforme Figuras 4(a) e 4(b), sendo cada um dos nove blocos passível de seleção, bastando que o visor seja tocado para que o foco mude. O objetivo do jogo é que a imagem seja completada, movimentando-se os blocos para o espaço vazio.



(a) Mosaico do Cristo Redentor



(b) Mosaico do Maracanã



(c) Análise de CPU e Memória

Figure 4: Mosaicos

Durante a execução desta aplicação, conforme Figura 4(c), o consumo de memória manteve-se praticamente constante. Isto se deve à instanciação e apresentação de todas as mídias ser feita no início da aplicação. Desta maneira, a interação do usuário apenas altera a disposição das mídias, gerando picos e vales no consumo de CPU, de maneira similar ao exemplo anterior.

Além disto, esta aplicação apresenta um baixo uso de recursos, mantendo o pico mais alto de CPU abaixo de 50% e a média próxima de 25% e com uma utilização de memória inferior a 10% da capacidade total do dispositivo.

4.3 Análise dos Módulos Transporte e Processador de Dados

De modo a iniciar o desenvolvimento do Módulo Processador de Dados, foi implementado o protocolo DSM-CC (*Digital Storage Media Command and Control*)[11], adotado pela norma do SBTVD-T como carrossel de dados. Segundo a norma, o DSM-CC é utilizado sobre o MPEG-2 TS (Transport Stream) e é transmitido como fluxo de dados não sincronizado com os programas. Porém, a implementação escolhida foi DSM-CC sobre o protocolo IP, considerando especificamente as mensagens do carrossel. Embora a codificação e decodificação das seções MPEG-2 em pacotes TS não tenham sido implementadas neste protótipo, isso não restringe a transmissão de dados que compõem os aplicativos NCL.

Para testar a implementação, construiu-se um cenário de experimentação composto pelos seguintes elementos (Figura 5):

- Um servidor cujo papel é transmitir aplicações NCL através de mensagens baseadas no protocolo DSM-CC;
- Uma estação radio base *WIFI* que distribui mensagens recebidas do servidor via canal de broadcast;
- Dispositivos portáteis equipados com o middleware Ginga, contendo um cliente DSM-CC para a recepção de mensagens no módulo Processador de Dados.

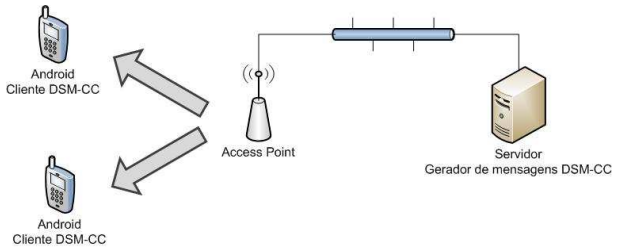


Figure 5: Ambiente de experimentação DSM-CC

Para que a recepção do fluxo transmitido pela emissora seja efetivamente recebido em dispositivos portáteis, é preciso lembrar que há uma notável diferença entre a taxa de transmissão do servidor e a taxa de recepção em dispositivos portáteis. Em outras palavras, é requerido um mecanismo de controle de fluxo de modo que o receptor com recursos limitados (banda, processamento e cpu) não seja saturado.

A ideia deste experimento é investigar essa problemática de saturação dos *buffers* do receptor na simulação de um ambiente de difusão, quando faz-se uso de UDP. Para efeitos de comparação, considera-se o caso no qual o fluxo é transmitido usando o protocolo TCP tradicional.

O experimento consiste em transmitir uma aplicação NCL através do ambiente especificado e analisar o tempo de transferência percebido em um dispositivo portátil rodando o cliente DSM-CC.

Para cada ponto, os experimentos foram repetidos 5 vezes para obtenção de um valor médio. A Figura 6 apresenta os resultados obtidos, com o índice junto a sigla UDP significando o valor do intervalo entre envios consecutivos (*pacing*) em milissegundos.

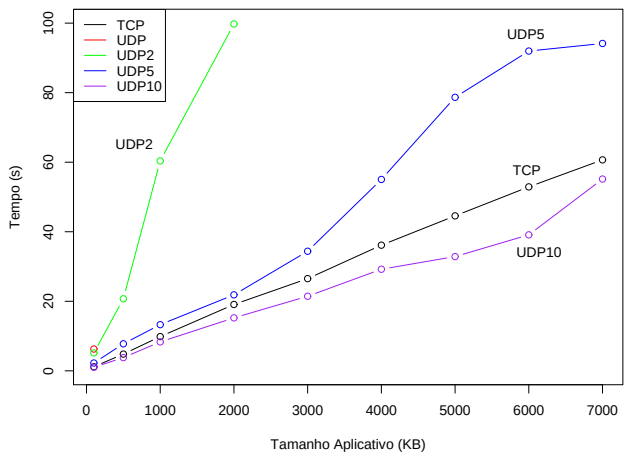


Figure 6: Medições de transferência de aplicativos

A primeira observação é a linearidade do caso TCP. Com relação aos cenários UDP, é possível perceber claramente a influência do tamanho de intervalo sobre o tempo total de transmissão. A medida que aumentou-se o espaçamento entre envios consecutivos, pôde-se perceber uma melhoria no tempo de transmissão, ao ponto que o caso UDP_{10} propiciou melhores resultados que o caso base (TCP). É interessante notar que o gargalo é determinado pela taxa que o dispositivo portátil é capaz de consumir do *buffer* de recepção. Por exemplo, no caso do UDP_2, o tempo de consumo não é suficiente para que os dados sejam recebidos resultando em uma inundação considerável. O resultado sugere que uma melhoria na infra-estrutura de comunicação não implica em uma melhoria da taxa de transferência efetiva. Entretanto, um ponto chave evidenciado pelo resultado é o fato do gargalo estar no dispositivo portátil e este é determinado pela taxa de consumo do *buffer* de recepção.

5. CONCLUSÕES E TRABALHOS FUTUROS

Com foco nos terminais de recepção portáteis, o presente trabalho descreve uma implementação do middleware Ginga-NCL para a plataforma Android, sistema operacional de código aberto da Google. Esta é, no nosso conhecimento, a primeira implementação do Ginga-NCL para esse novo e promissor sistema operacional para dispositivos embarcados.

A avaliação funcional e de desempenho da implementação está ancorada em um conjunto de experimentos e medições executados a partir de aplicações NCL desenvolvidas. Os resultados obtidos permitem afirmar que a especificação Ginga-NCL e a implementação aqui descrita são capazes de executar com sucesso aplicações NCL em dispositivos portáteis com a plataforma Android. Espera-se que o trabalho possa ser utilizado para estimular e orientar tanto implementações futuras do middleware quanto o desenvolvimento de aplicações NCL voltadas a essa classe de dispositivos. Para isso, pretende-se disponibilizar em repositório público o código fonte da implementação realizada.

O protótipo desenvolvido deverá ser usado como ponto de partida para a execução de novos trabalhos, alguns deles de necessidade imediata, como a realização de baterias de experimentos exaustivos e sistêmicos para garantir a robustez do middleware. Além disto, a máquina de execução Lua deve ser implementada, como requerido pelo padrão de middleware do SBTVD. Com relação ao suporte a múltiplos dispositivos, a implementação deverá suportar o modelo de controle hierárquico do NCL, conforme discutido em [5].

Outro trabalho, já em andamento no LPRM/UFES, é a implementação do módulo Context Manager, previsto na arquitetura conceitual do Ginga. A sua integração no middleware Ginga-NCL Android permitirá a construção de aplicações NCL mais elaboradas, que associam mobilidade, interatividade e sensibilidade ao contexto. Em relação à transmissão de dados, prevê-se a implementação do protocolo MPEG-2 TS para adequar-se à norma. Por fim, em vista da futura disponibilidade de dispositivos baseados em Android com hardware capaz de receber o sinal de radiodifusão, a implementação do módulo Sintonizador será necessária para lidar com esta forma de recepção dos dados.

Agradecimentos

Este trabalho é parcialmente financiado pela Rede Nacional de Ensino e Pesquisa (RNP) através do Centro de Pesquisa e Desenvolvimento em Tecnologias Digitais para Informação e Comunicação (CTIC) sub-projeto Ginga Frevo-GingaRAP, e pela CAPES - Brasil (RH-TVD #225/2008).

6. REFERENCES

[1] ABNT NBR 15606-4. Televisão digital terrestre - Codificação de dados e especificações de transmissão para transmissão digital - Parte 4: Ginga-J - Ambiente para a execução de aplicações procedurais. 2010.

[2] ABNT NBR 15606-5. Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiodifusão digital Parte 5: Ginga-NCL para receptores portáteis - Linguagem de aplicação XML para codificação de aplicações. 2009.

[3] Open Handset Alliance. http://www.openhandsetalliance.com/press_110507.html.

[4] ARIB. *ARIB STD-B24. Data Coding and Transmission Specification for Digital Broadcasting*, Junho 2008.

[5] Romualdo M. de Resende Costa e Marcio Ferreira Moreno e Luiz Fernando Gomes Soares. Ginga-ncl: Suporte a múltiplos dispositivos. In *XV Simpósio Brasileiro de Sistemas Multimídia e Web - WebMedia2009*, Fortaleza, CE, Brasil, 2009.

[6] Guido Lemos de Souza Filho, Luiz Eduardo Cunha Leite, and Carlos Eduardo Coelho Freire Batista. Ginga-j: The procedural middleware for the brazilian digital tv system. *Journal of the Brazilian Computer Society*, 12(4):47–56, march 2007.

[7] Vítor Medina Cruz e Marcio Ferreira Moreno e Luiz Fernando Gomes Soares. Ginga-ncl: Implementação de referência para dispositivos portáteis. In *XIV Simpósio Brasileiro de Sistemas Multimídia e Web - WebMedia2008*, pages 67–74, Vila Velha, ES, Brasil, 2008.

[8] ETSI. ETSI TS 102 428. Digital Audio Broadcasting (DAB); DMB video service; User Application Specification. *European Standard (Telecommunications series)*, Novembro 2005.

[9] Gerard Faria, Jukka A. Henriksson, Erik Stare, and Pekka Talmola. DVB-H: Digital broadcast services to handheld devices. *Proceedings of the IEEE*, 94(1):194–209, Jan. 2006.

[10] D. Hoffman, G. Fernando, and V. Goyal. *RTP Payload Format for MPEG1/MPEG2 Video, RFC 2038*, Outubro 1996.

[11] International Organization for Standardization / International Eletrotecnical Committee ISO/IEC 13818-6. Information technology - coding of audio-visual objects - part 6: Extensions for dsm-cc. 2000.

[12] ITU-T. Nested context language (ncl) and ginga-ncl for iptv services, 2009.

[13] ABNT NBR15606-2. Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiodifusão digital Parte 2: Ginga-NCL para receptores fixos e móveis - Linguagem de aplicação XML para codificação de aplicações. Abril 2008.

[14] Symbian Developer Network. http://developer.symbian.com.

[15] Luiz Fernando Gomes Soares and Simone Diniz Junqueira Barbosa. *Programando em NCL 3.0*. Editora Campus, 2009.

[16] Luiz Fernando Gomes Soares, Rogério Ferreira Rodrigues, and Marcio Ferreira Moreno. Ginga-ncl: the declarative environment of the brazilian digital tv system. *Journal of the Brazilian Computer Society*, 12(4), March 2007.