Guilherme Daher Ferreira

Um Middleware Declarativo na Plataforma Android^{TM}para o Sistema Brasileiro de Televisão Digital (SBTVD)

Vitória - ES, Brasil

28 de agosto de 2010

Guilherme Daher Ferreira

Um Middleware Declarativo na Plataforma Android^{TM}para o Sistema Brasileiro de Televisão Digital (SBTVD)

Dissertação apresentada para obtenção do Grau de Mestre em Informática pela Universidade Federal do Espírito Santo.

Orientador:
Dr. Magnos Martinello

Co-orientador:
Dra. Roberta Lima Gomes

Departamento de Informática
Centro Tecnolóogico
Universidade Federal do Espírito Santo

Vitória - ES, Brasil

28 de agosto de 2010

Dissertação de Projeto Final de Mestrado sob o título *"Um Middleware Declarativo na Plataforma Android^{TM}para o Sistema Brasileiro de Televisão Digital (SBTVD)"*, defendida por Guilherme Daher Ferreira e aprovada em 28 de agosto de 2010, em Vitória, Estado do Espírito Santo, pela banca examinadora constituída pelos professores:

Prof. Dr. Magnos Martinello
Orientador

Profa. Dra. Roberta Gomes Lima
Co-orientador

Prof. Dr. Luiz Fernando Gomes Soares
Pontifícia Universidade Católica do Rio de Janeiro

Prof. Dr. José Gonçalves Pereira
Universidade Federal do Espírito Santo

Resumo

Assim como em todos os principais sistemas de TV digital terrestre, o *middleware* brasileiro Ginga suporta tanto aplicações declarativas (através da sua apresentação via ambiente declarativo Ginga-NCL) quanto aplicações procedurais (através da sua execução, via ambiente procedural Ginga-J). Diferentemente dos dispositivos fixos, a normatização brasileira estabelece que para os dispositivos portáteis, somente o ambiente Ginga-NCL é obrigatório.

Hoje existem no Brasil alguns dispositivos portáteis que permitem a recepção do sinal de TV digital. No entanto, uma minoria destes aparelhos estão equipados com o *middleware* adotado pelo Sistema Brasileiro de Televisão Digital - SBTVD. Sabidamente, ainda não existe nenhum dispositivo capaz de executar aplicações para televisão digital utilizando a plataforma Android^{TM}como sistema operacional. Não existe também nenhuma implementação aberta disponível para a comunidade científica, de um *middleware* capaz de reproduzir aplicações, no padrão brasileiro, para televisão digital em dispositivos portáteis.

Este trabalho descreve o processo utilizado para implementação do Ginga-NCL, bem como detalhes da codificação, para dispositivos portáteis baseados no sistema operacional Android^{TM}. 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.

Abstract

As with all major digital terrestrial TV systems, the Brazilian middleware called Ginga, supports both declarative applications (through its presentation declarative environment Ginga-NCL) and procedural applications (through its execution procedural environment Ginga-J). Unlike the fixtures, the Brazilian standardization provides that for the portable devices, only the environment Ginga-NCL is required.

Today in Brazil there are some portable devices that allow the reception of digital TV signal. However, a minority of these devices are equipped with the middleware adopted by the Brazilian System of Digital Television - SBTVD. Known, although there is no device capable of running applications to digital television, using the Android™platform operating system, like, there is also no open-source middleware implementation, available to the scientific community, capable of playing, in Brazilian standard, digital TV on portable devices.

This paper describes the process used for implementation of Ginga-NCL, as well as details of the encoding for portable devices based on Android™operating system. As a means to validate the implementation, experiments were conducted to analyze the performance of applications NCL and resource usage of the handheld device.

Dedicatória

Dedico este trabalho

aos meus pais, à minha irmã, à minha namorada, e aos meus amigos.

Agradecimentos

Primeiramente ao meu orientador, professor Dr. Magnos Martinello, pela dedicação ao me orientar, pela paciência e pelo esforço em fazer com que eu superasse meus próprios limites.

Aos meus queridos pais Nazira e Raymundo, pela vida e por tanta dedicação sempre, e à minha irmã Cristina, por tudo.

Agradeço também à minha namorada, Larissa, pelo companheirismo e compreensão nas horas mais difíceis.

Ao Fábio e ao Guilherme, companheiros no desenvolvimento ao longo das muitas noites em que estivemos juntos estudando e programando.

Aos membros da banca, pelos comentários e sugestões.

À escola São Domingos por ter me ensinado a superar os meus desafios com inteligência e sabedoria.

A todo o time de colegas e colaboradores da UFES e da PUC-RJ pelas dicas e sugestões.

Por fim, agradeço aos colaboradores da TOTVS e da RR, assim como da RNP através do Centro de Pesquisa e Desenvolvimento em Tecnologias Digitais para Informação e Comunicação (CTIC), e todos aqueles que não foram referenciados acima, mas que de alguma forma foram envolvidos no projeto ou dele participaram.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

“ *Hómines Sunt Voluntates - Os Homens são seus atos de vontade.*”
Santo Agostinho

1.1 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 alguns fabricantes de dispositivos portáteis, cujos aparelhos permitem a recepção do sinal de TV digital. No entanto, poucos destes dispositivos já estão equipados com o Ginga, 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 [2], e outra imperativa, o Ginga-J [3], fornece uma série de facilidades para a construção e a execução de aplicações interativas em ambiente de TV digital aberta. Desta forma, dos dispositivos portáteis disponíveis no mercado brasileiro, embora se aproveitem da qualidade superior de áudio e vídeo inerente às tecnologias digitais, poucos ainda 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 implementação do Ginga-NCL para dispositivos móveis, foi realizada na PUC-RJ [4], tendo como plataforma base o sistema operacional Symbian.

A plataforma Android™foi lançada em 2008 pelo Google, através do consórcio Open Handset Alliance. Trata-se de uma plataforma de código aberto e que não está vinculada a apenas um fabricante. Esta característica tem permitido a abertura da restrita área de sistemas operacionais para 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™[5]. Um estudo experimental foi realizado para avaliar a sensibilidade de aplicações NCL [6] e o desempenho do *middleware* desenvolvido, incluindo o atraso decorrente da preparação inicial dos *players* em aplicações que contém um número alto de mídias simultâneas e a utilização de CPU e memória durante aplicações NCL com características distintas.

1.2 Motivação e Objetivos

A constituição da cadeia de valor da televisão aberta integra diversos agentes que visam produzir um produto final que é sempre voltado ao telespectador, que em sua totalidade é formado por cidadãos de diferentes classes [7]. Com o advento da Tecnologia Digital, agentes como dispositivos portáteis, até então ausentes nesta cadeia, passam a ter papel importante neste processo, trazendo principalmente, novas oportunidades de estudos e negócios.

Conforme dados da Agência Nacional de Telecomunicações (Anatel), em fevereiro de 2010, o Brasil possuía uma densidade de 91,33 aparelhos celulares para cada 100 habitantes, totalizando 175 milhões de celulares habilitados. Em se tratando de televisão digital portátil no Brasil, somente em 2010 os primeiros dispositivos homologados pela Anatel começaram a ser comercializados.

Porém, até o momento nenhum dispositivo capaz de executar aplicações para televisão digital, possui a plataforma Android™como sistema operacional, assim como não existe nenhuma implementação aberta, disponível para a comunidade científica, de um *middleware* capaz de reproduzir aplicações, no padrão brasileiro, para televisão digital em dispositivos portáteis. Além disso, até a presente data nenhum estudo havia sido publicado, que tenha sido validado em ambiente real, de análise de desempenho do *middleware* Ginga em dispositivos portáteis.

Os fatores acima, unidos à vontade de contribuir com a evolução e difusão do padrão brasileiro de televisão digital, fizeram o autor decidir por elaborar um estudo que fosse capaz de apresentar uma implementação de um *middleware* declarativo na plataforma Android™para o Sistema Brasileiro de Televisão Digital (SBTVD) e também apresentar experimentações nesta plataforma de forma que fosse possível analisar detalhadamente aspectos importantes de funcionamento de um *middleware*, tais como: *i)* o tempo de preparo de mídias simultâneas *ii)* uso de memória e CPU durante a execução de aplicações NCL e *iii)* e viabilidade da utilização do canal de retorno.

1.3 Metodologia

A fim de se realizar a implementação do *middleware* foi necessário elaborar um planejamento detalhado das etapas, de forma que o trabalho fosse conduzido da forma mais controlada possível, evitando desvios que pudessem afetar de alguma forma o desenvolvimento das atividades. Foi estabelecido que seria necessário realizar um estudo dos padrões atualmente existentes, bem como dos sistemas operacionais e de suas plataformas de desenvolvimento. Após isso, foi escolhida a plataforma que mais atendia aos requisitos quanto ao suporte às aplicações multimídia, à licença do código, ao ambiente de programação disponível e ao custo para o desenvolvedor. Posteriormente, iniciou-se a etapa de codificação com base nos levantamentos realizados nas etapas anteriores. Para tanto, partiu-se da implementação pública disponível para dispositivos fixos, sendo que foram necessárias várias alterações e novas implementações que serão apresentadas no decorrer do trabalho. Por fim, foram desenvolvidos testes e avaliações de desempenho de forma a buscar a validação funcional da implementação proposta.

1.4 Estrutura da dissertação

Os demais capítulos do trabalho estão organizados da seguinte forma: Capítulo 2 discute os trabalhos relacionados à esta dissertação. O Capítulo 3 descreve a plataforma Android^{TM}utilizada para a implementação do *middleware*. Capítulo 4 detalha o projeto de desenvolvimento com foco na implementação. O Capítulo 5 apresenta os testes e avaliações de desempenho desenvolvidas. Por fim, o Cápítulo 6 conclui o trabalho apresentando as considerações finais e as perspectivas de trabalhos futuros.

2 Revisão da Literatura

"*A Matemática pura é, a sua maneira, a poesia das ideias lógicas.*"
Albert Einstein

Este capítulo descreve o Estado da Arte em transmissão e recepção de Televisão Digital (TVD) para dispositivos portáteis. Na seção 2.1, uma introdução à TV Digital e um histórico dos principais trabalhos relacionados são apresentados. Na seção 2.2 foram demonstradas as principais organizações responsáveis por estabelecer padrões e normatizações para televisão digital portátil. Por fim, na seção 2.3 os principais sistemas operacionais para dispositivos portáteis são discutidos.

2.1 Televisão Digital

A televisão digital pode ser provida de duas maneiras. Através do primeiro formato, levam-se as tecnologias até então somente utilizadas em computadores para o cenário da televisão. Neste caso, enquadram-se as transmissões de televisão digital por radiodifusão terrestre, via cabo ou satélite. Através do segundo, levam-se as tecnologias, até então somente utilizadas na televisão, para um ambiente computacional. Esta convergência digital deve ser apoiada não só em camadas de baixo nível, mas também na camada de aplicação [8].

No primeiro modelo é possível disponibilizar nos televisores conteúdo de alta definição incluindo iteratividade, personalização de conteúdo, oferecimento de serviços sob demanda, dentre outras funcionalidades. O segundo modelo normalmente é realizado em redes fechadas, como TV's por assinatura e se encontra mais próximo do modelo da internet convencional.

Assim como o primeiro modelo, através da IPTV, também é possível se disponibilizar iteratividade, personalização de conteúdo e oferecimento de serviços sob demanda, entretanto, é necessária a utilização de técnicas de *multcasting*, a fim de viabilizar este mecanismo. Em contrapartida, a viabilização da transmissão pode ser prejudicada devido à complexidade da rede envolvida [9].

Um outro problema da IPTV é o consumo de banda nos serviços sob demanda, como o *Video on Demand* (VoD)[1]. Para tal, mecanismos devem ser criados para que esse consumo seja reduzido e o serviço se torne viável.

Nos EUA a realidade da IPTV está mais próxima, dado que a difusão da televisão já foi concebida através do uso de cabos. Sendo assim, o fator banda é minimizado consideravelmente. No Brasil a TV Digital é oferecida em âmbito nacional por radiodifusão;

[1]Neste serviço o consumidor tem acesso a um acervo de vídeos que podem ser escolhidos e apresentados sob sua demanda, na hora em que lhe for mais conveniente.

sendo assim, a IPTV ficará mais atrativa a medida que mais recursos estiverem disponíveis por um custo mais baixo. Dessa forma, acredita-se que em um futuro próximo, a TV também estará prontamente disponível pela Internet [10].

2.1.1 Histórico e Evolução da TV Digital no Brasil

A década de 90 foi marcada por esforços de Norte-Americanos, Japoneses, Europeus e Asiáticos em pesquisas e padronizações que procuravam sempre um melhor modelo para conceber uma tecnologia de transmissão de televisão em alta resolução.

No Brasil, assim como nos demais países, as pesquisas em grande escala em Televisão Digital também iniciaram-se na década de 90. Em 1994, através da união da Sociedade Brasileira de Engenharia de Televisão (SET) [11] e a Associação Brasileira de Emissoras de Rádio e Televisão (ABERT) [12], foi criado o grupo SET/ABERT [13] que naquele momento foi responsável pelo estudo dos modelos internacionais existentes até então e a sua viabilidade para o Brasil. Nesta mesma época, diversas pesquisas tais como: [14], [15], [16] e [17] também vinham sendo conduzidas por pesquisadores brasileiros e que posteriormente viriam a contribuir consideravelmente na definição do padrão Brasileiro de Televisão Digital.

Após alguns anos de análise dos padrões internacionais, e em face à restrições diversas em cada um deles, os modelos já existentes (alguns deles serão apresentados na próxima seção) se mostraram inadequados às necessidades brasileiras. Por fim, através do Decreto N^{o} 4.901, de 26 de novembro de 2003, foi oficializado o Sistema Brasileiro de Televisão Digital Terrestre (SBTVD-T).

A partir daquele momento diversas pesquisas foram conduzidas, no Brasil, por universidades e por empresas privadas. Posteriormente, com a criação do Fórum do Sistema Brasileiro de TV Digital Terrestre em 2006, padrões foram definidos, os quais vieram a ser aprovados pela Associação Brasileira de Normas Técnicas (ABNT) [18].

As normas estão inseridas em 13 documentos, os quais são responsáveis por formalizar os aspectos relacionados aos sistemas de transmissão, codificação de vídeo e áudio, codificação de dados e interatividade.

Nesses documentos é prevista a criação de um mediador entre o *hardware* e as aplicações, denominada *middleware*. Conforme [19], *middleware* ou mediador é um "componente que faz a mediação entre softwares, o qual é utilizado para compartilhar informações entre programas ocultando do usuário diferenças de protocolos de comunicação, plataformas e dependências do sistema operacional. É geralmente constituído por módulos dotados de APIs de alto nível que proporcionam a sua integração com aplicações desenvolvidas em diversas linguagens de programação e interfaces de baixo nível. Seu objetivo é mascarar a heterogeneidade e fornecer um modelo de programação mais produtivo para os programadores de aplicativos".

A padrão SBTVD define o Ginga como *middleware*, que tem como objetivo garantir a interoperabilidade das aplicações em diferentes implementações de plataformas que o suportam [20]. O *middleware* Ginga contém dois módulos responsáveis pela interatividade na Televisão Digital: o Ginga-J [3] e o Ginga-NCL [2]. O primeiro é responsável pelo processamento de aplicações procedurais e o segundo pelo processamento de aplicações declarativas. Os dois módulos são obrigatórios para receptores fixos, entretanto, [21] define Ginga-NCL como ambiente obrigatório e Ginga-J como ambiente opcional para receptores portáteis.

Os esforços no sentido de se desenvolver *middlewares* para dispositivos portáteis iniciaram-se há pouco tempo e ainda não são muito numerosos. Um exemplo é o caso do Japão, país avançado em termos de TV Digital, que começou a migração desse serviço para os dispositivos portáteis apenas em meados de 2006.

No Brasil, o primeiro trabalho científico neste sentido foi apresentado em [4], sendo proposta a implementação de um *middleware* Ginga para dispositivos portáteis baseados no sistema operacional Symbian. Trata-se da primeira implementação do Ginga-NCL para dispositivos portáteis e, como tal, possui uma descrição estendida sobre o padrão e trabalhos relacionados.

O trabalho tem seu foco na descrição da arquitetura do Ginga-NCL para dispositivos portáteis e na apresentação de decisões tomadas durante a implementação do *middleware*. Porém, não apresenta o detalhamento dos módulos desenvolvidos, bem como não faz avaliações experimentais detalhadas sobre a plataforma. Vale dizer que, o código fonte também não foi disponibilizado para domínio público. Na época de desenvolvimento do trabalho não existiam plataformas abertas amplamente disponíveis, como é o caso do sistema Android^{TM}(que será discutido no próximo capítulo), motivo pelo qual, apesar de ressaltar a vantagem de se utilizar uma plataforma aberta e livre, escolheu-se na época o sistema fechado Symbian para desenvolvimento.

O presente trabalho distingue-se dos anteriores ao apresentar não apenas os módulos do Ginga-NCL desenvolvidos em sistema Android^{TM}, mas também por realizar um conjunto de experimentos realizados sobre a plataforma, complementando as discussões iniciadas em [4], particularmente por meio de uma análise aprofundada da execução de aplicações NCL em dispositivos portáteis.

2.2 Padrões para Dispositivos Portáteis

Esta seção tem por objetivo realizar uma apresentação das principais organizações, hoje existentes, responsáveis por estabelecer padrões e normatizações para televisão digital portátil. Diferentemente da abordagem utilizada por [10], esta seção irá apresentar de forma resumida as principais organizações e seus respectivos padrões.

2.2.1 DVB

O *Digital Video Broadcasting Project* (DVB) é um consórcio formado por indústrias, operadoras, desenvolvedores de sistemas e organizações reguladoras que atua em 35 países com objetivo de desenvolver um padrão aberto para televisão e serviços digitais.

Oficialmente em 2004 foi adotado como padrão do grupo o *Digital Video Broadcasting Home* (DVB-H), a fim permitir a transmissão da TV Digital para os dispositivos portáteis. O desenvolvimento deste padrão objetivou atender aos requisitos dos *handhelds* [2], entretanto, ele também pode ser utilizado em outros dispositivos com características semelhantes, como os *smartphones* [3].

O *Multimedia Home Platform* (MHP) é o *middleware* de televisão digital do DVB, sendo que o principal objetivo desta especificação é permitir a utilização de serviços criados para TV Digital, independente da plataforma para a qual tenha sido desenvolvido. O MHP define o *Digital Video Broadcast HyperText Markup Language* (DVB-HTML) e o *Digital Video Broadcast Java* (DVB-J) como seus ambientes declarativos e procedurais, respectivamente. A implementação do primeiro é opcional, dado que ele está embutido no segundo [22].

2.2.2 ARIB

A *Association of Radio Industries and Businesses* (ARIB) foi designada como centro para a promoção do uso eficiente do espectro de rádio pelo Ministério dos Assuntos

[2]Personal digital assistants (PDAs ou Handhelds), ou Assistente Pessoal Digital, pode ser classificado como um computador de dimensões reduzidas.

[3]Smartphone é um telefone celular com funcionalidades avançadas que podem ser estendidas por meio de programas executados no seu Sistema Operacional.

Internos e Comunicações do Japão (MIC), sendo responsável pela realização de estudos de rádio difusão, estabelecimento de normas, fornecimento de serviços de consultoria, cooperação com outras organizações no exterior e fornecimento de serviços de apoio a assuntos relacionados à televisão digital. Todas estas atividades são realizadas em colaboração e com a participação de operadores de telecomunicações, radiodifusão, fabricantes de equipamentos de rádio e organizações relacionadas.

O *Integrated Services Digital Broadcasting Terrestrial* (ISDB-T) é o padrão para TV Digital Terrestre definido por essa organização. Este padrão já dispõe dos requisitos necessários para implementação de *middlewares* para ambientes portáteis, como é observado em [23].

Já existem alguns dispositivos portáteis, chamados de *One Seg Devices*, que oferecem suporte ao ISDB-T. Exemplos são: D903iTV, P903iTV, e o SH903iTV. 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 [24].

2.2.3 ATSC

O *Advanced Television Systems Committee* (ATSC) definiu o *Advanced Common Application Platform* (ACAP) como padrão norte-americano de difusão de serviços de televisão digital [25].

Uma aplicação ACAP pode ser entendida como uma coleção de informações que são processadas por um ambiente de execução, a fim de interagir com um usuário final ou alterar o estado do ambiente do aplicativo. Aplicações ACAP são classificadas de acordo com a forma que o conteúdo é processado, sendo estas categorias divididas em procedurais (ACAP-J) e declarativas (ACAP-X).

Este padrão - *Advanced Television Systems Committee Mobile/Handheld* (ATSC M/H) - não é considerado, a princípio, adequado ao ambiente portátil, de acordo com [23], tendo em vista a utilização do 8-VSB [26] como técnica de modulação. Entretanto, o comitê está em vias de aprovar uma extensão desse padrão para suportar dispositivos móveis. Em dezembro de 2008, o status dele era de 'padrão candidato' e em julho de 2009, após algumas modificações, foi elevado para 'padrão recomendado', estando em vias de aprovação.

2.2.4 SBTVD

O Sistema Brasileiro de Televisão Digital (SBTVD) é também chamado de ISDB-Tb (padrão japonês ISDB-T, versão brasileira) e difere do ISDB-T, principalmente, pelo emprego da compressão de vídeo [27] (MPEG-4) enquanto o ISDB-T utiliza [28] (MPEG-2), e também pelo suporte para interatividade utilizando o *middleware* Ginga.

As instituições Pontifícia Universidade Católica do Rio de Janeiro (PUC-RJ) e a Universidade Federal da Paraíba (UFPB) foram duas das principais responsáveis pela especificação do *middleware* Ginga para o Sistema Brasileiro de TV Digital, sendo a primeira responsável pela especificação do ambiente Declarativo (Ginga-NCL) [20] e a segunda propós a primeira proposta para o ambiente Procedural (Ginga-J) [29].

A utilização de ambos os ambientes é obrigatória nos terminais fixos; porém, apenas o ambiente declarativo Ginga-NCL é obrigatório para dispositivos portáteis. A fim de se permitir o desenvolvimento de aplicações híbridas em terminais portáteis, o padrão determina os meios pelos quais entidades declarativas e procedurais podem se comunicar através de uma ponte entre os dois ambientes.

Já existem implementações comerciais do *middleware* Ginga para dispositivos portáteis do Sistema Brasileiro de Televisão Digital. A normatização atualmente em vigor, [21], especifica os requisitos necessários para implementação do ambiente declarativo neste tipo de dispositivo. O Ginga-NCL determina *profiles* que contêm subconjuntos dos módulos da linguagem, usados para atender a diferentes requisitos, sendo o *Basic DTV*, perfil mínimo para dispositivos portáteis, especificado em [21].

2.2.5 3GPP

A *3rd Generation Partnership Project* (3GPP), é uma organização resultante de um acordo de colaboração firmado entre empresas de padronização em telecomunicações. Fundada em 1998 pela união entre *European Telecommunications Standards Institute* (ETSI), *Association of Radio Industries and Businesses* (ARIB), *China Communications Standards Association* (CCSA), *Alliance for Telecommunications Industry Solutions* (ATIS) e *Telecommunications Technology Association* (TTA), foi responsável pelo padrão *Mobile Broadcast/Multicast Service* (MBMS) de transmissão de TV Digital para dispositivos portáteis.

MBMS é uma tecnologia para distribuição de TV portátil por redes celulares para pequenos terminais (aparelhos de mão). MBMS usa os recursos de rede mais eficientemente, através de conexão ponto a multiponto, comparada com difusão única, no qual cada sessão tem uma conexão separada do servidor de TV portátil para o terminal.

Segundo [23] o MBMS é adequado para transmissão de conteúdos básicos, possibilitando alguns serviços de streaming em tempo real. Segundo o mesmo documento, para *streamings* [4] pesados em redes complexas e densas, outras soluções se mostram mais adequadas.

2.2.6 DMB

Digital Multimedia Broadcasting (DMB) é uma tecnologia de transmissão rádio-digital desenvolvida pela Coréia do Sul como parte do projeto nacional de tecnologia de informação para transmissões tais como: TV, rádio e dados para dispositivos portáteis.

Foi com o DMB que o primeiro serviço de televisão portátil digital no mundo deu-se na Coréia do Sul em Maio de 2005, embora testes já estivessem disponíveis muito mais cedo. Este sistema oferece suporte para operação via satélite (S-DMB) ou terrestre (T-DMB).

O DMB também possui algumas semelhanças com os principais concorrentes da televisão portátil como o DVB-H. Basicamente sua arquitetura é composta de três camadas: compressão, sincronização e transporte. A linguagem utilizada como descritora de cenas e a sincronização são feitas com o uso do MPEG4 BIFS e MPEG4 SL [30], respectivamente, e o transporte foi implementado com o uso do MPEG2 TS [31].

2.3 Sistemas Operacionais para Dispositivos Portáteis

Nesta seção é realizada uma apresentação dos principais sistemas operacionais encontrados atualmente nos dispositivos portáteis comercializados. Suas características quanto às licenças de uso, possibilidade de desenvolvimento, adequação ao SBTVD e posicionamento no cenário mundial também serão discutidas.

[4] Streaming (fluxo, ou fluxo de mídia em português) é uma forma de distribuir informação multimídia numa rede através de pacotes.

2.3.1 iPhone OS

O iPhone OS é um sistema operacional (SO) desenvolvido pela empresa *Apple* utilizado exclusivamente nos dispositivos *iPhone* e *iPod*, produzidos também por essa empresa. A base do sistema deriva do sistema operacional *Mac OS* e a arquitetura é composta de quatro camadas: núcleo do sistema operacional, a camada de serviços, a camada de mídia e a camada 'Cocoa touch'[5]. Sua licença de distribuição é *General Public License versão 3* (GPLv3) [32] que determina que qualquer aplicação desenvolvida precisa ser submetida à *Apple Store*[6] que efetuará uma avaliação para distribuí-la, sendo que a distribuição ocorrerá somente após aprovação assinada com chaves de propriedade da Companhia.

Para o seu desenvolvimento é utilizada a linguagem *Objective-C* que é um pacote derivado do *ANSI C* [33] que provê orientação a objetos. Um Kit de Desenvolvimento (SDK) contendo um emulador é disponibilizado; porém, é necessário o pagamento de uma taxa de desenvolvedor para testar as novas aplicações no dispositivo real. Estes fatores tornam o desenvolvimento para o iPhone relativamente mais caro e complexo que outras plataformas.

No que se refere ao SBTVD, o iPhone não possui suporte nativo; porém, uma empresa japonesa que distribui o *iPhone 3G*, a *Softbank*, desenvolveu um receptor do sistema japonês. Este transmissor recebe o sinal e o repassa ao aparelho através de uma interface sem fio embutida.

Em fevereiro de 2010, o sistema da Apple dominava o mercado global de tráfego de informações via *smartphones*. Durante o período de Fevereiro de 2009 a Fevereiro de 2010, o iPhone aumentou a sua quota de 33% para 50% [34]. Na América latina essa quantidade é ainda maior, atingindo um total de 56% em dezembro de 2009 [35].

2.3.2 Symbian

Symbian é um sistema operacional aberto, cuja maior parte dos seus recursos está sob a licença *Eclipse Public Licence* (EPL) [36] e alguns demais componentes estão disponibilizados sob outras licenças.

Assim como todo sistema portátil, ele também possui alguns recursos para gerenciar e reduzir a utilização de bateria e de memória. Um ponto forte desta plataforma é a ver-

[5]Cocoa Touch é uma API para a criação de programas de software para funcionar no iPhone, iPod Touch e do IPAD da Apple Inc.

[6]A Apple Store é uma cadeia de lojas de varejo de propriedade e operação pela Apple Inc., que vendem e tratam de computadores e equipamentos eletrônicos da Apple.

satilidade no desenvolvimento de aplicações. Atualmente, ele fornece suporte à *Symbian C/C++*, *JavaME*, *FlashLite*, *Perl*, *Python*, *Ruby* e *Lua* .

No que tange a SBTV Digital, Symbian possui uma versão, a 9.5, que oferece suporte a esse tipo de aplicação, aceitando, atualmente, dois padrões de modulação/transmissão, o DVB-H e o ISDB-T.

No último ano, a plataforma Symbiam perdeu uma fatia considerável de mercado global de transmissão de dados em smartphones, saindo de 43% em Fevereiro de 2009 para 18% em Fevereiro de 2010 conforme [34] e [37]. Entretanto, a plataforma Symbiam ainda aparece com quotas de mercado consideráveis (77%) a medida que regiões específicas são analisadas, tais como: Índia, Indonésia, Austrália, Filipinas, Tailândia, Malásia, Singapura e Hong Kong [38]. Na América Latina, por exemplo ao final do ano de 2009, esta plataforma detinha 28% do mercado [35].

2.3.3 Windows Mobile

O Windows Mobile é um sistema operacional, desenvolvido para ser executado em dispositivos portáteis em geral. Devido à semelhança com o sistema operacional Windows amplamente utilizado em PC's este sistema operacional é em geral muito bem aceito pelos usuários. Além disso, as ferramentas utilizadas para o desenvolvimento de aplicações são as mesmas da sua versão *desktop*, como o Visual Studio ou o .NET.

Recentemente, foi disponibilizada a ferramenta gratuita *Microsoft eMbedded Visual C++*, concebida especificamente para o desenvolvimento de programas visando dispositivos embarcados. Porém, o pacote ainda recomendado pela *Microsoft* para o Windows Mobile é o recente *Windows Mobile 6.5 Developer Resource Kit*. Este pacote contém o Visual Studio, ferramenta não gratuita de desenvolvimento da *Microsoft*.

Podemos destacar como principais diferenciais desta plataforma, o *Microsoft Office* que acompanha o Windows Mobile e que inclui o *Pocket Word*, o *Pocket Excel*, e o *Pocket PowerPoint*. Nestas versões há vários recursos das versões *desktop*, entretanto, ainda com algumas limitações. O programa de conexão *ActiveSync*, que acompanha os dispositivos, tem recursos que permitem converter os arquivos da versão *desktop* para a versão *Pocket PC*. O *Windows Media Player 9* para o Windows Mobile suporta grande parte dos formatos de multimídia existentes, tais como .WMA, .WMV, .MP3, e .AVI. Em algumas versões é possível reproduzir arquivos *MPEG4 Audio* (.M4A).

Como ponto negativo, o custo de uma licença do *Visual Studio* é um fator determinante na utilização desta plataforma. O custo da licença do Windows Móbile varia de $8,00 a $15,00 por dispositivo, de acordo com a *Strategy Analytics*[7].

No cenário global, o sistema operacional da Microsoft tem mantido a estabilidade variando de 2% a 4% nos anos de 2009 a 2010 conforme [34] e [37]. Na América Latina este cenário se mantém, considerando que em dezembro de 2009 este SO possuia 6% do mercado [35].

2.3.4 BlackBerry OS

BlackBerry é o produto vendido pela empresa *Research In Motion* (RIM). A solução BlackBerry é composta por *smartphones* com um software integrado que possibilita acesso a uma série de serviços de dados e comunicação. O seu sistema operacional é proprietário e é feito dedicadamente para os *handhelds* e outros dispositivos portáteis produzidos por esta empresa.

A RIM não disponibiliza o seu SO para que outras empresas utilizem, sendo assim, apenas os *hardwares* fabricados pela própria RIM o possuem. O desenvolvimento de aplicativos para esta plataforma pode ser realizado em Java através de uma plataforma disponibilizada pelo fabricante ou através de *BlackBerry Mobile Data System* (BMDS).

O *BlackBerry Mobile Data System v4.1* é um pacote de desenvolvimento de aplicativos para a BlackBerry. A sua arquitetura pode ser subdividida em três componentes principais:

- BlackBerry MDS Services - Como parte do BlackBerry Enterprise Server, é responsável pelo gerenciamento de interações e solicitações entre *smartphones* BlackBerry e aplicativos empresariais protegidos por *firewall.*
- BlackBerry MDS Developer Tools - Ferramentas para desenvolvedores possibilitando a criação de aplicativos sem fio para *smartphones* BlackBerry.
- BlackBerry MDS Device Software - O BlackBerry MDS Device Software permite que aplicativos criados com *Blackberry MDS Developer Tools* sejam executados em smartphones BlackBerry.

[7] Strategy Analytics, Inc., é uma empresa global de pesquisa e consultoria que concentra-se em oportunidades de mercado e nos desafios das áreas da Eletrônica Automotiva, Consumidor Digital, Mundos Virtuais, Estratégias Wireless, Tarifas e em Tecnologias Embarcadas.

As aplicações de desenvolvimento em ambas as linguagens citadas anteriormente são gratuitas. Os aparelhos da BlackBerry não possuem boa interação com outros tipos de dispositivos. Algumas aplicações, como e-mail, por exemplo, funcionam apenas com um conjunto específico de dispositivos de outras marcas.

Até o momento da publicação deste trabalho não existia nenhuma implementação na plataforma BlackBerry adequada ao SBTVD.

Assim como o Symbian, o sistema operacional da RIM, tem perdido mercado nos últimos anos. De agosto de 2009 à Fevereiro de 2010 a fatia de mercado da plataforma passou de 8% para 4% conforme [34] e [37].

2.3.5 Android^{TM}

Uma das mais recentes e talvez a mais promissora das plataformas é a plataforma Android^{TM}. O Android^{TM}é um sistema aberto constituído em uma pilha de software para dispositivos portáteis incluindo o sistema operacional Linux 2.6 e algumas aplicações já incorporadas chamadas de 'aplicações críticas'.

Qualquer aplicação escrita para este SO deve utilizar a linguagem de programação Java, e posteriormente ser compilada para execução em uma máquina virtual *Dalvik* (Dalvik Virtual Machine) projetada especificamente para utilização em ambientes embarcados.

O Kit de Desenvolvimento para Android^{TM}(Android SDK) fornece as ferramentas e bibliotecas necessárias para desenvolvimento de aplicações na plataforma Android^{TM}usando a linguagem de programação Java, sendo que os desenvolvedores possuem acesso irrestrito à maior parte da codificação destas bibliotecas, o que de fato facilita em muito o desenvolvimento de aplicativos para esta plataforma.

O Android^{TM}inclui um conjunto de bibliotecas C/C++ utilizadas por diversos componentes do sistema. Estas funcionalidades são expostas para os desenvolvedores através do *framework* Android^{TM}. Algumas das principais bibliotecas estão listadas abaixo:

- System C library
- Media Libraries
- Surface Manager
- LibWebCore

- SGL
- 3D libraries
- FreeType
- SQLite.

Até o momento da publicação deste trabalho não existia nenhuma implementação na plataforma Android^{TM}adequada ao SBTVD.

No que se refere ao mercado de tráfego global, Android^{TM}é o sistema operacional que mais rapidamente cresce ano após ano. Avaliações indicam que a plataforma Android aumentou fatia de mercado de 2% em fevereiro de 2009 para 24% em fevereiro de 2010. Os cinco dispositivos Android^{TM}, que mais se destacam em todo o mundo são os Motorola Droid, HTC Dream, HTC Hero, HTC Magic e o Motorola CLIQ [34]. Na América Latina estes dispositivos começaram a ser comercializados recentemente o que justifica o fato de a plataforma Android^{TM}somente possuir 1% do mercado na região [35].

2.3.6 PalmOS

PalmOs é um sistema operacional inicialmente desenvolvido pela empresa *U.S Robotics* subsidiária da *Palm Computing, Inc.* para *personal digital assistants* (PDAs) em 1996. O seu desenvolvimento é baseado na utilização de tela sensível ao toque incluindo interfaces gráficas simplificadas. Ele é fornecido em conjunto a um pacote de aplicações básicas para utilização nos PDA's.

As últimas versões desse sistema foram a *PalmOS Garnet* e a *PalmOS Cobalt*, sendo que a última nem chegou a ser lançada comercialmente. Em 2009, a detentora comercial do PalmOS, a *Palm Inc*, informou a descontinuação do Palm OS para focar o desenvolvimento do WebOS para os próximos dispositivos.

De fato, o PalmOS tem tentado se manter no mercado, muito embora não tenha conseguido se sobressair em relação aos seus concorrentes. Certamente, devido a isto a detentora dos direitos do PalmOS esteja focando o desenvolvimento de uma nova plataforma. Ainda para ilustrar a realidade da *Palm, Inc* até mesmo alguns aparelhos desta empresa não utilizam a variante do seu sistema operacional, utilizando a variante do Windows anteriormente citado, como é o caso do Treo 700w e do Treo 750.

Até o momento da publicação deste trabalho não existia nenhuma implementação na plataforma PalmOS adequada ao SBTVD.

O sistema PalmOs tem sido desconsiderado nos últimos levantamentos mercadológicos, tendo em vista a sua mínima participação no mercado. Em contrapartida o sistema WebOs se mostrou com um pequeno crescimento na América do Norte, passando de 3% naquela região em dezembro de 2009 [35].

2.4 Considerações Finais

Neste capítulo foi possível contextualizar o trabalho no cenário da televisão digital em dispositivos portáteis apresentando um histórico da televisão digital no Brasil, bem como referenciando organizações assim como sistemas operacionais cujo foco é em dispositivos portáteis.

Pela análise da literatura existente e pela percepção do mercado de televisão digital no Brasil, pode-se observar a existência de uma lacuna para desenvolvimento de *middeware* para dispositivos portáteis no padrão SBTVD e de novas tecnologias de televisão digital no padrão SBTVD. As referências para pesquisas ainda são poucas, e também nota-se a existência de poucos produtos no mercado cujo foco seja este. Sem dúvida, deve haver um maior incentivo governamental tanto para as indústrias quanto para os pesquisadores, de forma a fomentar as pesquisas, bem como o desenvolvimento de novas linhas de produtos voltados a este mercado.

Nos próximos capítulos serão enfatizados o SBTVD e a plataforma Android apresentadas neste capítulo, de forma que no capítulo 3 detalhes da plataforma Android™serão discutidos e no capítulo 4 uma implementação do Ginga-NCL no padrão SBTVD na plataforma Android™será proposta.

3 Plataforma Android^{TM}

"*O imperador do futuro será um imperador de ideias.*"

Wiston Churchill

Este capítulo tem como objetivo posicionar o leitor na plataforma que foi utilizada para implementação do *middleware* para dispositivos portáteis. Será apresentado um resumo das principais funcionalidades da plataforma Android^{TM}, bem como uma breve descrição e algumas referências que podem ser utilizadas no aprofundamento do estudo do respectivo sistema.

3.1 O que é Android^{TM}?

A plataforma Android^{TM}foi lançada em 2008 pelo Google, por meio do consórcio *Open Handset Alliance* [39]. Trata-se de uma plataforma de código aberto, não vinculada apenas a um fabricante, fato este que tem permitido a ampliação da restrita área de telefonia móvel.

O Android^{TM}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^{TM}, sendo otimizada para rodar em dispositivos portáteis. Para os desenvolvedores, há um *Software Development Kit* (SDK) disponível em licença GPLv2 e já existe uma ampla gama de aparelhos com Android^{TM}embarcado.

Apesar de ter sido construído com base no Linux, a plataforma não possui todas as funções de uma plataforma Linux padrão, por exemplo, não possui *windowing system* nativo (componente GUI) e não suporta *glibc*.

3.2 Arquitetura

A arquitetura do sistema operacional Android^{TM}, recomendada pela *Open Handset Alliance*, possui um bom grau de liberdade para o desenvolvimento de aplicações no ambiente de telefonia móvel [40].

Grande parte das aplicações são escritas usando a linguagem de programação Java e compiladas em *byte-code* específico para uma máquina virtual projetada especialmente para uso em ambiente embarcados. Essa máquina virtual, chamada de *Dalvik* VM, tem um modelo no qual cada aplicação é executada sob uma instância da máquina virtual (VM). Dessa forma, cada aplicação associada a uma VM tem o seu espaço de endereçamento separado.

Apesar de a maioria das aplicações Android™serem escritas na linguagem Java, a *Dalvik* não é uma máquina virtual Java, já que não executa *bytecode* JVM. Além disso, todo o gerenciamento da memória dos processos (instâncias *Dalvik*) e o controle de dispositivos, como câmera e GPS, é feito pelo Linux compilado em processador arm, mips, ppc ou x86 e sua imagem binária pode rodar tanto nos celulares como no emulador provido com o kit de desenvolvimento.

O diagrama abaixo apresenta os principais componentes do Android™.

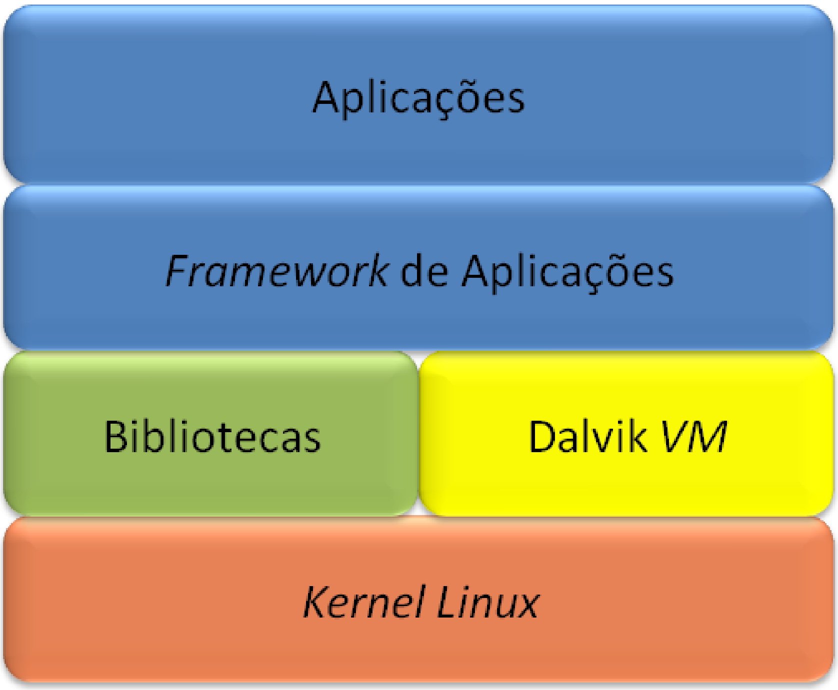


Figura 3.1: Componentes do Android™

Na parte mais acima da Figura 3.1 estão incluídas todas as *Aplicações* que o cliente pode executar. Logo em seguida no diagrama aparece o *Framework* das Aplicações, que é o conjunto de classes que permite o desenvolvimento destas aplicações. As principais classes deste conjunto serão detalhadas na Seção 3.2.1.

Na camada inferior ao *Framework* das Aplicações tem-se as bibliotecas escritas em C/C++ usadas para prover acesso ao hardware aos vários componentes do Android™.

Estas bibliotecas foram desenvolvidas e otimizadas de uma forma que a execução de suas funções não é dependente da máquina virtual *Dalvik* e por essa razão possuem acesso privilegiado ao *hardware*, comparável com *device drivers* [41] em sistemas operacionais. A plataforma provê para isso o *Android NDK*, que é um conjunto de ferramentas que acompanha o SDK nativo que permite o desenvolvimento de aplicações, utilizando a linguagem

C/C++, cuja execução é independente da máquina virtual *Dalvik*.

Entre essas bibliotecas tem-se toda a parte de multimídia usando vários codificadores como MPEG4, H.264, MP3, AAC, AMR, JPG, e PNG, uma biblioteca especifica para renderização de 3D em caso de acelerador gráfico no smartphone e um banco de dados relacional.

Por fim, no nível de sistema operacional, a versão 2.6 do Linux foi escolhida para prover os serviços críticos como gerencia da memória e processos, além de prover a camada de abstração entre o dispositivos de hardware e o resto da pilha de software.

3.2.1 Conceitos básicos para desenvolvimento em Android™

A maior parte dos aplicativos para Android™devem ser escritos na linguagem de programação Java, apesar de também ser possível o desenvolvimento usando c/c++, ainda assim qualquer desenvolvedor que pretenda escrever aplicações nesta plataforma deverá necessariamente conhecer os fundamentos desta linguagem de programação [42]. O código Java implementado pelo desenvolvedor deve ser compilado juntamente com todos os arquivos e recursos necessários para o aplicativo, e posteriormente através de uma ferramenta disponibilizada é gerado um arquivo com sufixo .apk, a fim de possibilitar a distribuição da aplicação em dispositivos portáteis.

Por padrão, cada aplicação é executada em seu próprio processo e cada processo possui sua própria Máquina Virtual (VM). A cada aplicação é atribuído um ID de usuário Linux exclusivo e as permissões são definidas para que os arquivos do aplicativo sejam visíveis apenas para o usuário e apenas para a aplicação em si. Deve ser destacado que também existem formas de tornar uma aplicação visível para as demais [43]. Também é possível estruturar uma aplicação de forma que duas aplicações compartilhem o mesmo ID. Neste caso, para conservar os recursos do sistema, as aplicações com a mesma identificação podem ser executadas em um mesmo processo Linux, neste caso compartilhando a mesma VM.

Activities

O bloco de construção das interfaces de usuário no Android™é uma *Activity*. Pode-se fazer uma analogia entre uma *Activity* no Android™e uma *Window* ou um *Dialog* em uma aplicação para um *desktop*. Certamente esta é uma das classes mais importantes da plataforma. Cada *Activity* é responsável por controlar os eventos da tela e definir qual

View será responsável por desenhar a interface gráfica do usuário [44].

Content Providers

O Android^{TM}permite armazenar informações de diversas formas diferentes utilizando banco de dados, arquivos e o sistema de preferências. Contudo, geralmente essas informações ficam salvas dentro do pacote da aplicação, e somente a aplicação que criou o banco de dados ou o arquivo pode ter acesso às informações.

Os *Content Providers* provém um nível de abstração para que qualquer dado guardado no dispositivo seja acessível por mais de uma aplicação. O desenvolvimento na plataforma Android^{TM}favorece os recursos de uma aplicação, de modo que sejam acessíveis por outras aplicações. *Content Providers* possibilita que se faça isso, e que se mantenha total controle de como esses recursos são utilizados.

Intents

Intents são mensagens de sistema que são executadas sempre que um evento ocorre a fim de notificar as aplicações embarcadas de sua existência, por exemplo, na inserção de um cartão SD ou o recebimento de uma mensagem SMS. Os desenvolvedores não só podem responder aos *Intents* como podem executar suas aplicações quando um destes eventos ocorre.

Services

A classe *Services* é utilizada para executar um serviço em segundo plano, geralmente vinculado a algum processo que deve ser executado por tempo indeterminado e possui um alto consumo de recursos, memória e CPU.

Intents, *Activities* e *Content Providers* são temporárias e podem desaparecer a qualquer momento. *Services*, por outro lado, devem ser desenvolvidos para se manterem em execução a qualquer momento, de preferência, independentes de qualquer *Activity*.

Handler

Um Handler possibilita o envio de mensagens *(android.os.Message)* e o processamento de objetos do tipo *Runnable* associados a *threads*, ou seja, cada handler é associado a uma única *thread* e a sua fila de mensagens respectivamente. Quando se cria um Handler, ele se vincula a thread que o criou, e a partir daquele momento ele irá enviar mensagens e executar objetos do tipo *Runnable* na seqüência em que eles são enviados a sua fila.

3.2.2 Interface com usuário

Em uma aplicação desenvolvida para o sistema Android$^{\mathrm{TM}}$, a interface com o usuário é feita através de objetos do tipo *View* e *View Group*, que são as unidades básicas de interface nesta plataforma.

A classe *View* serve como base para as subclasses *widgets*, que são implementações completas de objetos comuns, tais quais campos de texto e botões. Já a classe *ViewGroup* serve como base para as subclasses *layout*, que definem diferentes tipos de arquitetura visual para a aplicação, como linear, tabular e relativa.

Um objeto do tipo *View* é uma estrutura de dados cujas propriedades guardam os parâmetros de leiaute e conteúdo para uma área retangular específica na tela. Ele gerencia as medidas, leiaute, desenhos, mudança de foco, rolagem e interações por tecla e toque da área retangular em que reside. Como um objeto na interface com usuário, a *View* é um ponto de interação e um receptor dos eventos de interação.

A partir destes elementos é possível construir a interface de uma *Activity*, agrupando os em uma árvore n-ária (Figura 3.2) cuja raiz é um nó do tipo *ViewGroup*, que define o leiaute mais abrangente desta *Activity*. Esta árvore pode ser tão complexa quanto o necessário, podendo utilizar apenas os objetos definidos pela API ou objetos criados pelo usuário, que estendem a classe *View*.

Para a renderização desta árvore na tela, uma referência ao nó raiz é passada a um método da API que irá validar, medir e desenhar a árvore. O processamento da árvore de leiaute é feito de uma forma *top-to-bottom*, de modo que vai instanciando *Views* e adicionando-as aos seus elementos pais, fazendo com que, caso haja sobreposição de elementos, o último a ser desenhado seja colocado na frente.

A maneira mais comum de definir a árvore de Interface com Usuário é por meio de um arquivo XML de leiaute, em que cada elemento do XML é um objeto *View* ou *ViewGroup*. Esta maneira é a mais indicada para construir aplicações cujo leiaute é estático, ou seja, apenas o conteúdo apresentado será modificado em tempo de execução, permanecendo inalterada a disposição dos elementos da interface. Para aplicações que precisem modificar a estrutura do leiaute em tempo de execução, é possível construir a árvore em código Java, utilizando métodos da API.

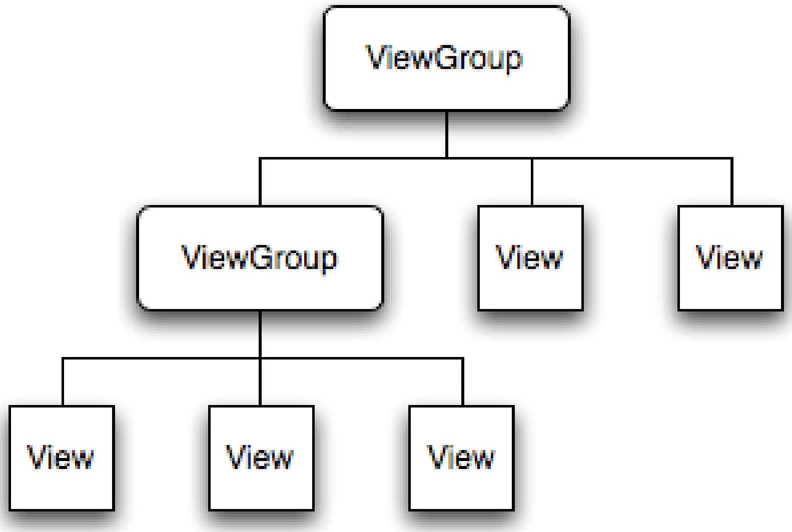


Figura 3.2: Estrutura em árvore da interface Activity retirada de [1]

Um *widget* é um objeto *View* que serve como uma interface de interação com o usuário. A API Android^{TM}oferece um conjunto de elementos *widget* totalmente implementados que de fato ajudam muito no desenvolvimento de uma aplicação.

Uma vez adicionadas algumas *widgets* para interfacear com usuário, o próximo passo é controlar a interação do usuário com eles, para que se possam realizar ações. Para uma aplicação ser informada de eventos interface do usuário, existem duas formas:

A primeira é definir um *listener* de evento e registrá-lo como uma *View*. A outra, é implementar um método *Callback* para a *View*.

A primeira é a forma mais conhecida e usual da linguagem JAVA. A classe de *View* contém uma coleção de interfaces aninhadas de nomes On<alguma coisa>Listener, e sempre que for necessário capturar o evento estas classes devem ser implementadas. A segunda forma é o que se deve fazer quando se implementa uma classe *View* própria e é necessário escutar eventos específicos que ocorrem dentro dela.

3.2.3 Recursos

Resources e *Assets* são uma parte integral de uma aplicação para a plataforma Android^{TM}. Em geral, são elementos externos incluídos e/ou referenciados na aplicação, tais como imagens, áudio, vídeo, strings, leiautes, temas, etc. Toda aplicação Android^{TM}, portanto, contém um diretório para *resources* (res/) e um para *assets* (assets/).

A diferença entre *resources* e *assets* está na forma com que são acessados pela aplicação. O conteúdo do tipo *resource* é acessível via a *classe R*, que é compilada pelo Android™. Já o conteúdo do tipo asset é acessível apenas através do *AssetManager*, que lê o arquivo como um *byte stream*.

Existem diferentes tipos de elementos que podem ser adicionados aos *resources* de uma aplicação, tais como simples valores de elementos comumente usados e imagens necessárias para a interface. Uma descrição destes diferentes tipos será feita a seguir.

Valores de elementos podem ser expressos como *strings*, utilizando formatos não ambíguos para indicar os diferentes tipos de valores. Valores de cor sempre começam com #, seguido pelo valor da cor no formato A-RGB (*alpha red green blue*). Valores de texto são *strings* não começadas por # e podem conter *tags* html para formatação.

Drawables, ou desenháveis, são elementos utilizados para a interface com usuário. Documentos *bitmap* suportados são png, o tipo preferível, jpg e gif. Os bitmaps são compilados e disponibilizados à aplicação sem a extensão, através da *classe R*, como, por exemplo, *res/drawable/imagem1.png* é acessível por *R.drawable.imagem1*. *Color drawables* são retângulos preenchidos com uma cor RGB. Por fim, imagens *Nine-Patch* são um tipo de imagem expansível, em que um conjunto de 9 seções da imagem é utilizado para desenhá-la, sendo 4 bordas em quina, 4 bordas retas e 1 imagem de centro. Esta imagem deve ser do tipo png e sua extensão deve ser de 9.png sempre.

Por fim, elementos importantíssimos, como os leiautes, Menus e Animações utilizados em mais de uma *activity* da aplicação podem ser *resources* comuns a aplicações, para melhor reuso. Estes elementos são definidos através de um arquivo XML que segue uma formatação definida pela plataforma Android™e são acessíveis através da *classe R*.

3.2.4 Elementos Gráficos

Os elementos gráficos da plataforma Android™são gerenciados por uma biblioteca própria para gráficos em duas dimensões (2D) e OpenGL ES 1.0 com alta performance para elementos em três dimensões (3D). Os recursos mais comuns para gráficos em 2D podem ser encontrados no pacote *drawable*. As APIs OpenGL estão disponíveis a partir do pacote *Khronos ES OpenGL* [45].

Basicamente existem duas maneiras de se utilizar gráficos 2D em uma aplicação, que são:

1. Utilizando um método da classe *Draw()*

2. Utilizando um objeto de uma *View*

Na primeira forma, desenham-se os gráficos diretamente na tela por linha de códigos. Esta é a melhor opção quando a performance da manipulação de gráficos de uma aplicação é algo essencial, como em animações procedurais. Na outra forma, utilizando um objeto da classe *View*, o gráfico é subordinado à hierarquia desta *View* no sistema, sendo assim, o programador simplesmente define qual gráfico irá aparecer. Esta é a melhor opção quando há necessidade da utilização de gráficos estáticos e animações pré-definidas [46].

O desenvolvimento de aplicações em 3D é através do *OpenGL ES* que é uma especificação do OpenGL destinada a dispositivos embarcados. Android™atualmente suporta *OpenGL ES 1.0*, que corresponde ao *OpenGL 1.3*. Assim, um aplicativo desenvolvido com *OpenGL 1.3* em um sistema *desktop*, deverá ser portável em Android™. A API específica fornecida pelo Android™é similar ao *OpenGL ES JSR239 J2ME API*.

3.2.5 Audio, Vídeo e Imagens

Segundo informações do fórum de desenvolvedores bem como informações contidas na página oficial [47], as tabelas abaixo descrevem os formatos de mídia de áudio (Tabela 3.1), vídeo (Tabela 3.2) e imagem (Tabela 3.3) suportados pela plataforma Android™. É importante destacar, porém, que qualquer fabricante de dispositivo móvel também pode fornecer suporte para formatos adicionais ou tipos de arquivo não constantes na tabela abaixo.

Como não é o objetivo deste trabalho discutir formatos de mídia, destacaram-se na coluna 'Referência' os locais onde se podem obter maiores informações referentes a cada formato. Nas colunas 'Codifica' e 'Decodifica', têm-se a informação se a plataforma realiza tais operações e na última coluna são apresentados os tipos de arquivo de mídia suportados para cada padrão.

Tipo	Formato	Codifica	Decodifica	Referência	Tipo do arquivo suportado
Audio	AAC LC/LTP	Não	Sim	ISO/IEC 13818-7:2006	3GPP (.3gp) e MPEG-4 (.mp4, .m4a). Sem suporte para AAC (.aac)
	HE-AACv1 (AAC+)	Não	Sim	ISO/IEC 14496-3:2009	
	HE-AACv2 (reforçada AAC+)	Não	Sim	ISO/IEC 14496-3:2009	
	AMR-NB	Sim	Sim	[48]	3GPP (.3gp)
	AMR-WB	Não	Sim	[48]	3GPP (.3gp)
	MP3	Não	Sim	ISO/IEC 11172-3:1993	MP3 (.mp3)
	MIDI	Não	Sim	[49]	Tipo 0 e 1 (.mid, .xmf, .mxmf). Também RTTTL/RTX (.rtttl, .rtx), OTA (.ota), e iMelody (.imy)
	Ogg Vorbis	Não	Sim	[50]	Ogg (.ogg)
	PCM/WAVE	Não	Sim	[51]	WAVE (.wav)

Tabela 3.1: Padrões de áudio suportados

Tipo	Formato	Codifica	Decodifica	Referência	Tipo do arquivo suportado
Vídeo	H.263	Sim	Sim	[52]	3GPP (.3gp) e MPEG-4 (.mp4)
	H.264 AVC	Não	Sim	[53]	3GPP (.3gp) e MPEG-4 (.mp4)
	MPEG-4 SP	Não	Sim	ISO/IEC 14496-2:2004	3GPP (.3gp)

Tabela 3.2: Padrões de vídeo suportados

Tipo	Formato	Codifica	Decodifica	Referência	Tipo do arquivo suportado
Imagem	JPEG	Sim	Sim	[54]	JPEG (.jpg)
	GIF	Não	Sim	[55]	GIF (.gif)
	PNG	Não	Sim	[56]	PNG (.png)
	BMP	Não	Sim	[57]	BMP (.bmp)

Tabela 3.3: Padrões de imagens suportados

3.2.6 Segurança e Permissões

A plataforma Android™consiste de um sistema multi-processos em que cada aplicação, e partes do sistema rodam em seu próprio processo. Com isso, grande parte da segurança entre diferentes aplicações é feita por meio do nível de processo no sistema Linux, utilizando *User ID* e *Group ID* para cada aplicação e diretivas de segurança mais robustas são realizadas através de restrições a operações específicas que um processo possa realizar.

Um ponto central da plataforma é que nenhuma aplicação, por padrão, tem permissão de realizar operações que irão impactar outras aplicações, o sistema, ou o usuário. Isto inclui escrever ou ler dados privados do usuário (contatos, e-mail, etc.), escrever ou ler dados de outra aplicação, acessar a rede, controlar o hardware, etc. Caso uma aplicação necessite realizar tais operações, deve declará-las estaticamente, de modo que é possível saber quais operações uma aplicação fará antes de instalá-la.

Além disso, cada desenvolvedor deve criar um par de chaves criptográficas e utilizá-las para estabelecer uma relação de segurança entre ele e o repositório de aplicações Android™. Desta maneira, é possível controlar quem terá acesso às aplicações no servidor e quais aplicações poderão compartilhar dados através de um mesmo ID de usuário.

3.3 Considerações Finais

Logo que a primeira versão do Android™foi lançada, houve uma nova perspectiva tanto no meio corporativo quanto no acadêmico, pelo fato de estar nascendo uma plataforma aberta para desenvolvimento de aplicações em dispositivos portáteis. Esta estratégia de construir um sistema operacional livre e de código aberto teve seus impactos, visto que levou a plataforma hoje dominante, *Symbian*, a anunciar no primeiro semestre de 2010 que os fontes também estariam disponíveis para desenvolvedores como uma plataforma de código aberto em sua totalidade como uma maneira de não perder espaço de seus concorrentes.

Alguns fatores importantes favoreceram o rápido crescimento da plataforma Android tais como: o suporte a mídias distintas; o suporte à multi camadas gráficas; a utilização de conceitos como processos a nível de kernel com suporte de comunicação entre processos (IPC); a máquina virtual *Dalvik* que permite rodar o código compilado .apk de modo transparente em diferentes aparelhos e processadores e também o custo do licenciamento.

Alguns requisitos, como Sintonizador (*Tuner*) e Filtro de Seção (*Section Filter*), contidos na normatização não foram avaliados dado que o *hardware* utilizado não oferece suporte a estes recursos, que são responsáveis por: *i)* fornecer uma interface que permite selecionar uma determinada freqüência ou realizar uma varredura em uma faixa de freqüência específica e *ii)* oferecer uma interface para a filtragem de seções MPEG-2 TS, respectivamente. Entretanto, a partir da análise dos demais requisitos, não foi possível observar um ponto crítico que impedisse o desenvolvimento do *middleware* no padrão SBTVD que será apresentado no próximo capítulo.

4 *Implementação Ginga-NCL para Android^{TM}*

"*Quem pensa pouco erra muito.*"

Leonardo da Vinci

O projeto de desenvolvimento de um software é muito abrangente e envolve diversas etapas. Não está no escopo desta dissertação discutir cada uma das etapas do projeto para desenvolvimento do *middleware* Ginga para Android^{TM}. Neste capítulo serão apresentadas as principais fases de todo o projeto. Inicialmente serão listados os aspectos que foram considerados para a escolha da plataforma, assim como as etapas seguintes até a fase de codificação e testes, incluindo detalhes da codificação, bem como os diagramas de classes desenvolvidos.

4.1 Análise e Modelagem

No projeto de desenvolvimento do *middleware* Ginga para a plataforma Android^{TM}, [5], pode-se destacar a etapa em que foi realizada uma análise preliminar das plataformas existentes. Através dela foi realizada uma verificação das plataformas móveis quanto ao suporte à aplicações multimídia, à licença do código, ao ambiente de programação disponível (SDK), à API proposta e ao custo para desenvolvedor. Foram avaliadas as plataformas Symbian, Windows, Apple, RIM e Android^{TM}.

A primeira delas é bem robusta e sem dúvida possui um excelente ambiente para desenvolvimento de aplicações, entretanto, no início dos trabalhos a plataforma ainda não estava totalmente aberta, fator este que foi determinante para a sua não utilização no projeto. As plataformas Apple, Windows e RIM assim como a primeira, são bem robustas, entretanto, o ambiente de desenvolvimento é proprietário e ainda existem algumas dificuldades em se obter suporte a algumas funcionalidades dessas plataformas. A questão do licenciamento também foi relevante dado que estas plataformas não possuem licenciamento gratuito.

Após essa análise, foi escolhida a plataforma Android^{TM}devido ao sistema operacional ser de código aberto construído sobre o Kernel Linux 2.6, possuir disponibilidade de SDK sem custos, possuir API relativamente simples baseada em Java, oferecer suporte a diferentes mídias, oferecer um desenvolvimento contínuo do sistema e devido à experiência dos executores que já haviam trabalhado nesta plataforma.

Após a escolha definida, nosso foco voltou-se para uma análise detalhada na plataforma Android^{TM}com objetivo de elaborar o projeto conceitual da arquitetura Ginga especificamente sobre tal plataforma. Foi realizado também um levantamento das bibliotecas que posteriormente deveriam ser estudadas com mais profundidade e que serão discutidas ao longo do capítulo.

Para orientar o projeto conceitual, partiu-se de um estudo detalhado da normatização Ginga [58, 20, 59, 29, 21] a fim de entender os componentes essenciais da arquitetura. Com base neste estudo, foram elaborados os diagramas representados na Figura 4.1 e posteriormente os diagramas de classes que representam a estrutura base na qual o projeto está ancorado.

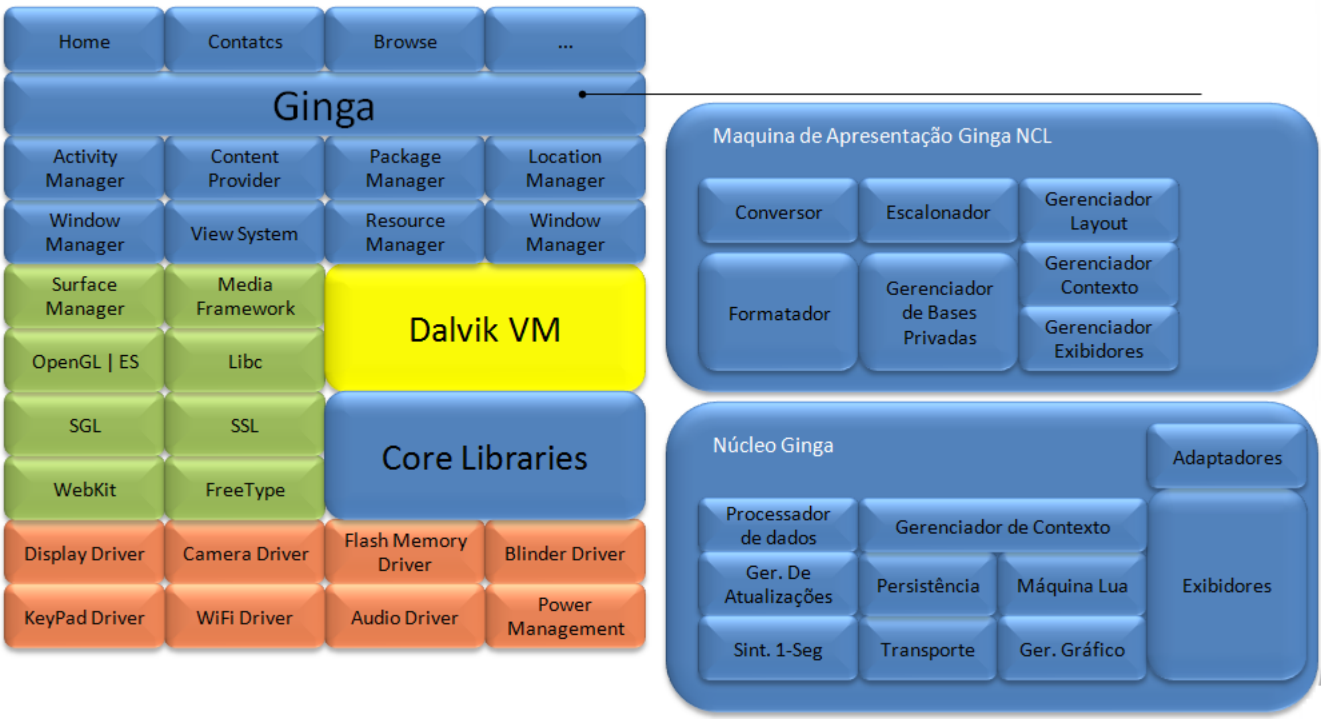


Figura 4.1: *Modelagem*

Quanto à arquitetura do *middleware*, a Figura 4.1 ilustra sua componentização, na qual se pode observar a divisão em dois subsistemas [4]: a máquina de apresentação NCL e o núcleo Ginga. Considerando o Ginga-NCL para o sistema Android$^{\text{TM}}$, conforme demonstrado na mesma figura, a máquina de apresentação do *middleware* encontra-se inserido entre a camada de aplicações e o *framework* de desenvolvimento e, desta maneira, apoia-se na API do sistema para realizar operações específicas como a manipulação do ambiente gráfico. O núcleo encontra-se inserido no *Core Libraries* utilizando recursos providos pela plataforma, como *Media Framework*, *WebKit*, dentre outros.

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, i) multiplexadas no conteúdo recebido pelo Sintonizador, ou ii) por outra interface de rede (por exemplo, canal de dados oferecido pela operadora, conexão bluetooth, wifi, etc). Quando a aplicação estiver multiplexada, ela é obtida através de um processamento efetuado pelo módulo **Processador de Dados** sobre o conteúdo recebido. No caso em que a aplicação for recebida por uma interface de rede, o componente **Transporte** foi definido para gerenciar protocolos e interfaces de rede. Após o recebimento destas aplicações e o conteúdo referenciado por elas, o módulo **Persistência** atua para gerenciar seu armazenamento.

O componente **Exibidores** é responsável pelos decodificadores específicos para cada tipo de mídia. Apoiando-se na API Android^{TM}, foi utilizada para o desenvolvimento a biblioteca *Media Libraries*, a qual provê suporte à reprodução e gravação de áudio (MP3) e a alguns formatos de vídeo baseados em MPEG-4, bem como arquivos de imagem estática BMP, GIF, JPG e PNG. Fortemente ligado a este, o componente **Adaptadores** é responsável pela padronização da comunicação entre a Máquina de Apresentação e a API de decodificação de conteúdo dos exibidores. Ainda como os **Exibidores**, os **Adaptadores** seguem uma padronização de interface.

O componente **Gerenciador Gráfico** foi definido para realizar o controle espacial da renderização de objetos de acordo com o especificado pelas aplicações de TV. A biblioteca *Surface Manager* do Android^{TM}foi utilizada no gerenciamento do subsistema de exibição e camadas gráficas.

No subsistema da máquina de apresentação, o principal componente é o **Formatador** e seu papel consiste em direcionar todas as ações a serem executadas por ela. Inicialmente, o **Formatador** solicita ao **Conversor** que processe a aplicação NCL, previamente recebida e processada por uma estrutura 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.

4.2 Implementação

Como ponto de partida para a implementação, foi necessário realizar uma profunda análise das implementações públicas do *middleware* Ginga para dispositivos fixos, visto que era evidentemente sensato partir de alguma implementação base para que esta fosse adequada à realidade de um dispositivo móvel.

As implementações públicas Ginga estudadas foram a versão em C++[1] e uma versão em JAVA[2]. Ao contrário da versão em C++, a versão pública em JAVA havia sido descontinuada; apesar disso, a decisão foi partir da versão em JAVA devido essencialmente a reutilização de código dos componentes que já estavam desenvolvidos.

Pode-se afirmar que esta escolha trouxe alguns entraves que tiveram que ser solucionados durante a etapa de implementação. A versão JAVA do código não era modularizada, sendo assim, as classes não estavam dispostas de acordo com a arquitetura mostrada no diagrama 4.1. Existiam algumas falhas de programação que impediam a execução de aplicações NCL simples, principalmente no Gerenciador Gráfico e Escalonador. A versão JAVA, foi desenvolvida utilizando bibliotecas gráficas que não estão disponíveis na plataforma Android™, como por exemplo a java.awt.

Devido a todos estes fatores, os módulos **Exibidores**, **Adaptadores**, **Gerenciador Gráfico**, **Conversor**, **Formatador**, **Escalonador**, **Gerenciador de Bases Privadas**, **Gerenciador de Exibidores e Transporte**, tiveram que ser alterados ou totalmente reescritos.

Na próxima seção serão apresentados os módulos implementados, destacados na Figura 4.1, bem como os primeiros testes realizados na plataforma.

4.2.1 Codificação

Formatador

A classe *Formatter* atua como um gerenciador em alto nível, sendo responsável pelo controle das aplicações NCL presentes no *middleware*.

A instanciação inicial da classe ocorre na *Intent* 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 ao gerenciador e para iniciar a exibi-

[1] http://svn.softwarepublico.gov.br/trac/ginga/wiki/Building-Wiki-GingaNCL
[2] http://svn.softwarepublico.gov.br/svn/ginga/ncl30-java/trunk

ção. Os botões de controle para pausar e parar a aplicação possuem eventos que realizam as chamadas ao gerenciador para a ação escolhida.

O processo de adição de uma aplicação ao gerenciador, é feito através da classe *NclDocumentManager*.

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.

A estrutura de classes do Formatador foi mantida em relação à implementação utilizada como referência. Foram necessários alguns ajustes no momento do recebimento do identificador de documento (*Document ID*) e de interface (*Interface ID*) e na sua posterior compilação, assim como na criação do evento de entrada.

Conversor

O módulo conversor é responsável pelo recebimento de uma estrutura NCL qualquer e posterior manipulação desta.

A **DocumentParser**, apresentada na Figura 4.2, faz parte de um parser genérico para qualquer linguagem baseada no padrão XML. Essa classe é responsável pelo gerenciamento do parser, e pode ser representada como ponto de entrada de compiladores de documentos baseados em linguagem XML. Todas as sub-classes executam o construtor dessa classe para garantir que as variáveis e referências sejam propriamente tratadas. A classe **ModuleParser** basicamente é responsável por armazenar objetos definidos na classe anterior.

As demais classes deste módulo são responsáveis por executar funções específicas no processamento da estrutura hierárquica de um documento NCL.

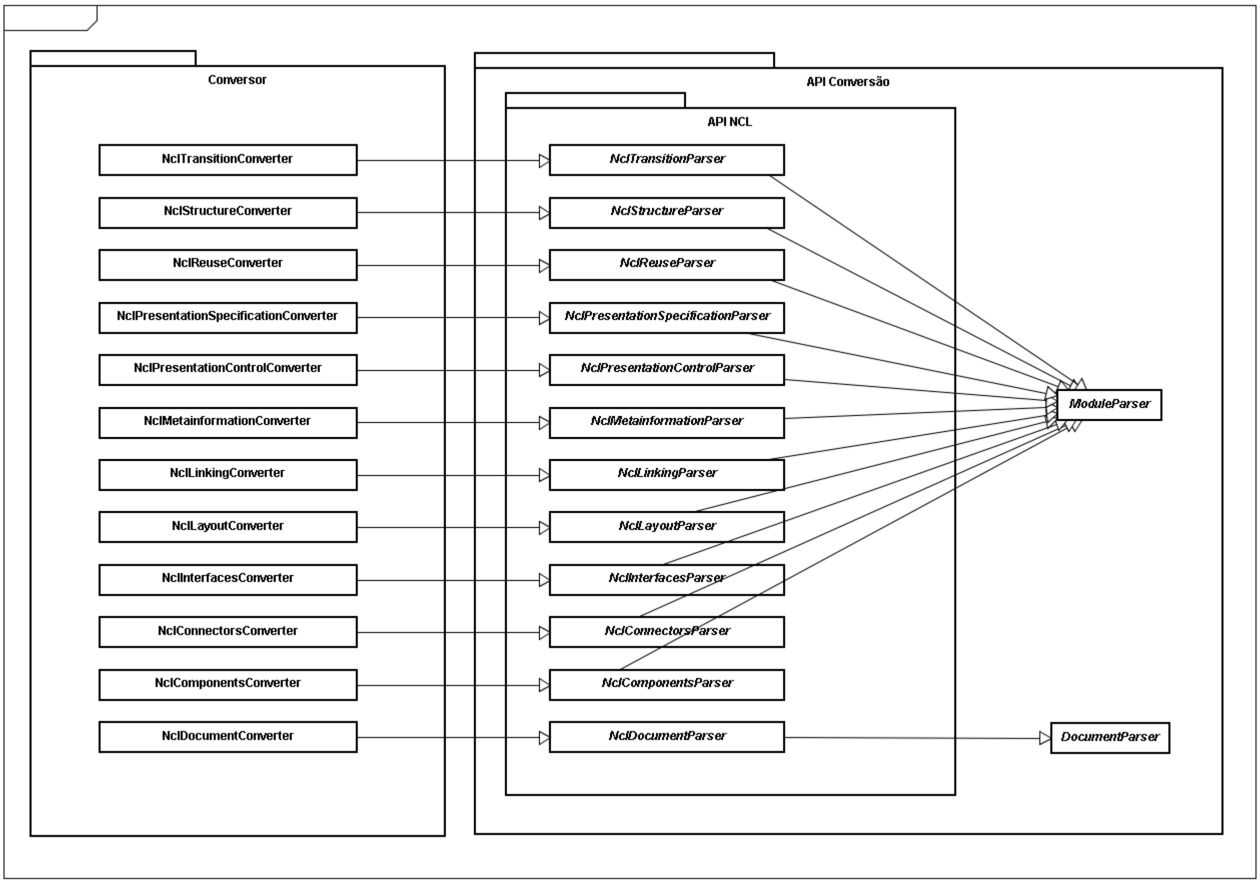


Figura 4.2: *Conversor*

Embora a biblioteca de *parser* Document Object Model (DOM) possua um menor desempenho em relação à Simple API for XML (SAX) [60, 61], foram utilizadas as classes DOM para implementação do módulo Conversor no *middleware* devido a maior experiência dos desenvolvedores com esta biblioteca, mas também pelo fato de que a versão da implementação pública escolhida já estava baseada em DOM.

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. É este gerenciador o responsável por receber uma aplicação NCL e armazenar a estrutura denominada Base Privada.

A Figura 4.3 apresenta o diagrama de classes que define o módulo Bases Privadas. 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 utilizados 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 implementação dessas classes foram seguidos os padrões definidos em [20].

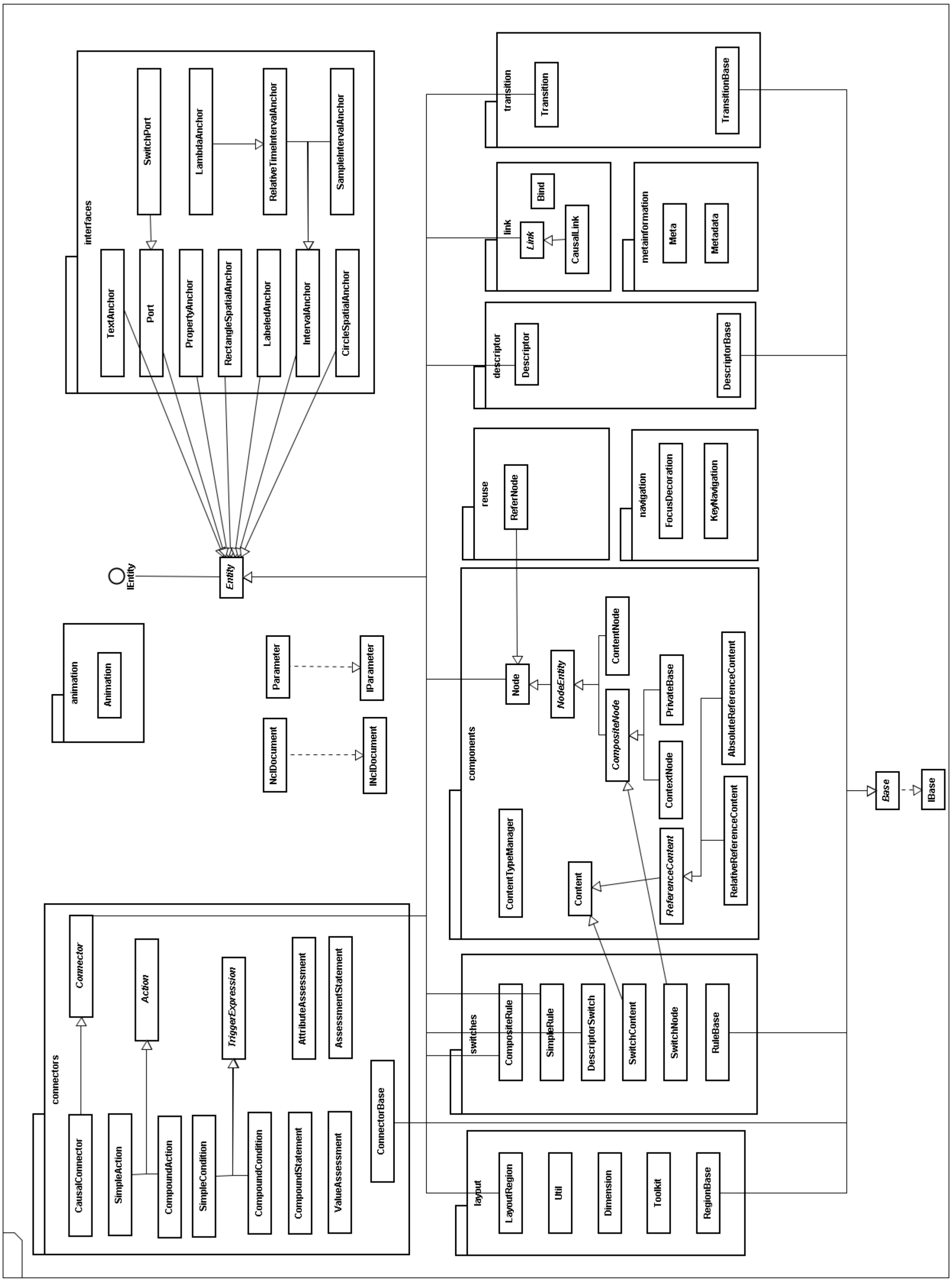


Figura 4.3: *Bases Privadas*

Gerenciador Gráfico

Para realizar o papel do Gerenciador Gráfico do *middleware* Ginga, a classe **InterfaceUpdater** foi implementada. Sua função básica é gerenciar a exibição e remoção de superfícies (**AndroidSurface**) na tela e, portanto, necessita-se alterar o leiaute principal do *middleware*. Entretanto, por uma questão de segurança, a plataforma Android^{TM}não permite que todas as classes pertencentes a uma aplicação alterem o leiaute principal; apenas a *Intent* inicial pode fazê-lo.

De modo a contornar esta limitação da plataforma, a classe foi implementada utilizando um *Handler*, apresentado na seção 3.2.1. Duas classes internas que implementam a interface *Runnable*, *updateUI* e *changeImage*, também devem ser destacadas. 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.

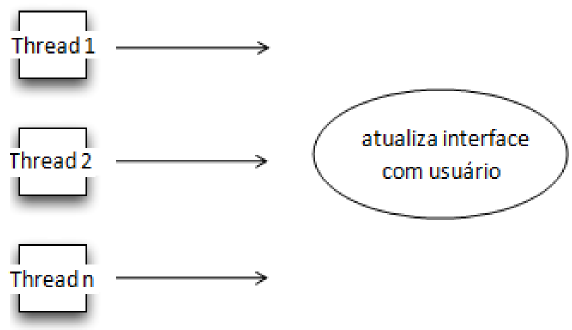
Analogamente, a classe *updateUI* é utilizada para alterar o leiaute principal, adicionando e removendo elementos do tipo **AndroidWindow** adicionados à listas internas através dos métodos *addRemoveTask*, *addDisplayTask*, *addLayout* e *addClearTask*.

É importante destacar que a solução apresentada para contornar uma limitação da plataforma, acrescentou um novo componente não existente até então, ou seja, na versão JAVA, cada *thread* disparada possuia autonomia para manipular o leiaute principal, mas no formato proposto todas as *threads* necessariamente devem passar por uma nova entidade para que esta realize a ação. Esta diferenciação pode ser observada no esquema proposto na Figura 4.4.

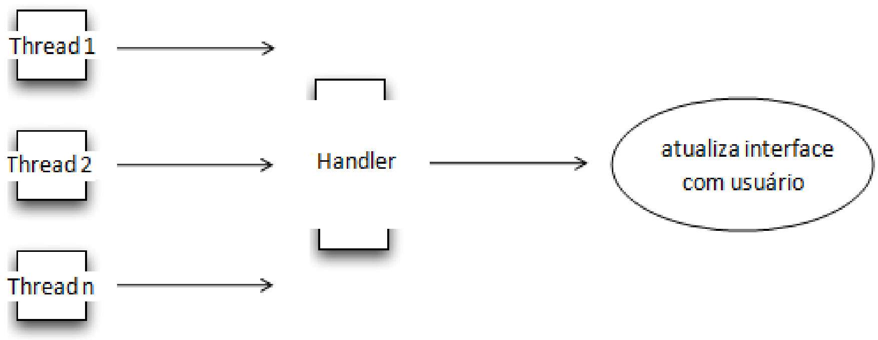
Sem dúvidas, a criação deste novo componente acrescentou um novo custo (principalmente de tempo) no gerenciamento dos recursos do *middleware* e na seção 5.1 será apresentado um estudo, no qual, um dos fatores de atraso no tempo de preparação das mídias é o tempo gasto pelo *handler* do Gerenciador Gráfico.

Gerenciador de Exibidores

O módulo Gerenciador de Exibidores compreende as classes necessárias para gerenciamento da execução dos exibidores de mídias e de interação com usuário. No diagrama de classes indicado pela Figura 4.5, é apresentada a estrutura implementada no *middleware*



(a) Abordagem inicial



(b) Abordagem do middleware para Android

Figura 4.4: Diferença de abordagem para Gerenciador Gráfico

a fim de realizar esse trabalho.

Neste diagrama pode-se destacar a interface **AndroidSurface** que é responsável por definir uma superfície de exibição de mídia. É importante ressaltar que cada mídia executada numa aplicação NCL tem uma superfície de exibição a ela associada. Vê-se, pelo diagrama, que cada tipo de mídia, requer superfícies com características diferentes; para tal, a API Android™provê classes distintas, são elas: *ImageView*, *SurfaceView* e *WebView*. Estas classes são utilizadas como base para as superfícies de apresentação do *middleware*. Com a finalidade 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**, a sua função é prover uma abstração da janela de trabalho do *middleware*, para tal, foi utilizada como base, a classe de leiaute 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 pela recepção dos 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^{TM}, que é responsável por ser chamada sempre que um evento é enviado para uma respectiva *View*.

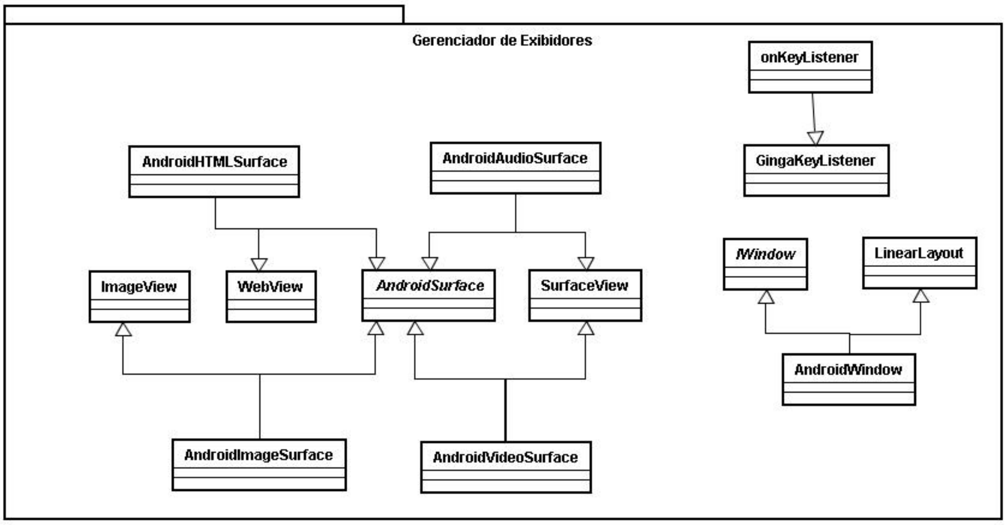


Figura 4.5: *Gerenciador de Exibidores*

Exibidores

No módulo exibidores há 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 se buscou, na implementação deste *middleware*, seguir exatamente as instruções contidas no capítulo 8.2 em [20].

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

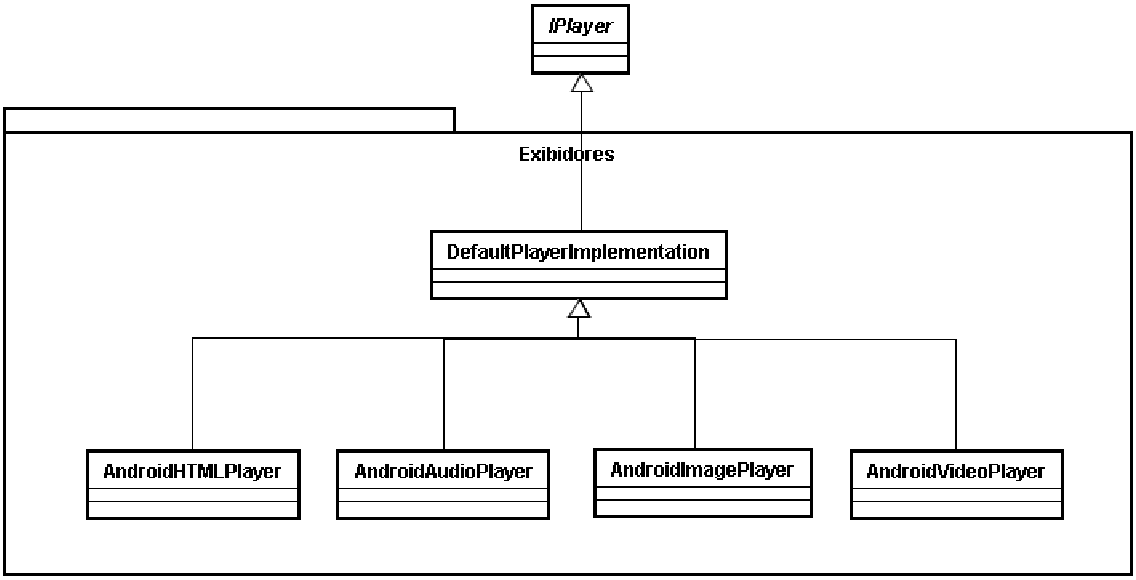


Figura 4.6: *Exibidores*

É importante destacar que na plataforma utilizada como referência, existiam algumas falhas no momento da exibição das mídias, que eram oriundas de defeitos nos exibidores. Grande parte destas falhas foram corrigidas dada a robustez das bibliotecas de mídia da plataforma Android^{TM}.

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, como pode ser visto na Figura 4.7. 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 executada existe um adaptador distinto, são eles: **AndroidImagePlayerAdapter**; **AndroidVideoPlayerAdapter**; **AndroidHTMLPlayerAdapter**; **AndroidAudioPlayerAdapter** 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 a fim de gerenciamento das mídias de Áudio, Vídeo, Imagem e HTML.

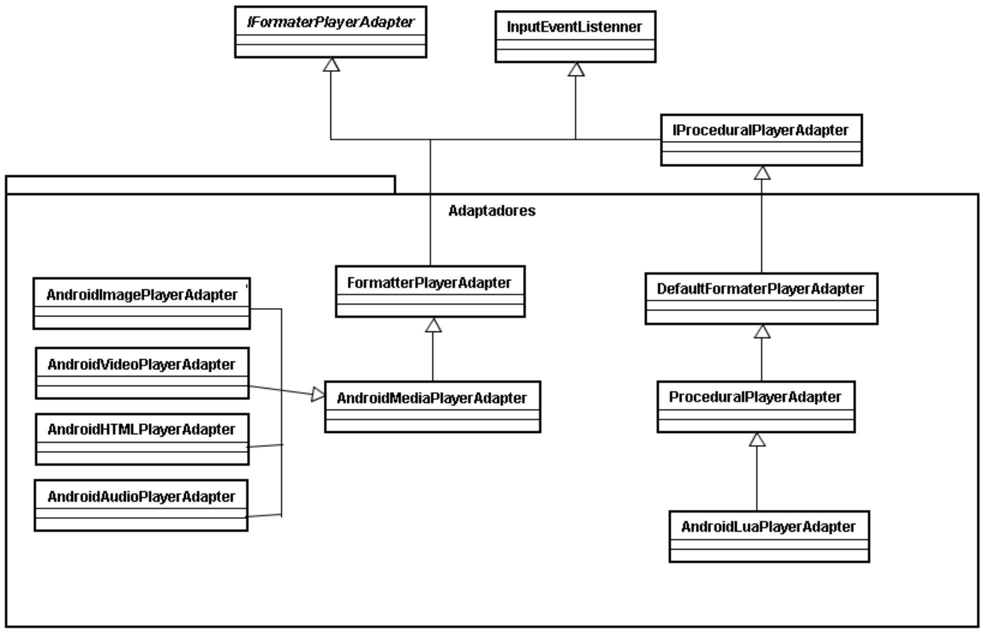


Figura 4.7: *Adaptadores*

As mídias procedurais possuem um tratamento especial, de modo que a classe mais específica responsável pelo tratamento destas mídias é a **AndroidLuaPlayerAdapter**. Esta classe estende **ProceduralPlayerAdapter** responsável por definir estruturas de dados específicas para tratamento de mídias procedurais, que por sua vez, estende **DefaultFormatterPlayerAdapter** responsável por definir funções especificas para tratamento deste tipo mídia.

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 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", que é definido pela atribuição de um valor a uma propriedade de um nó (representado por um elemento <*media*>, <*body*>, <*context*> ou <*switch*>); "Composição", que é definido pela apresentação da estrutura de um nó de composição (representado por um elemento <*body*>, <*context*> ou <*switch*>) ou "Apresentação", em que a mídia contida em um nó NCL é exibida.

Ações sobre os três primeiros tipos são mais simples e de execução rápida, porém ações em eventos do tipo Apresentação necessitam de um tratamento diferente, pois são, sobre uma mídia pertencente ao documento. Esse tipo de evento tem a si associado 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 Leiaute 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*. Este processo pode ser visualizado na Figura 4.8.

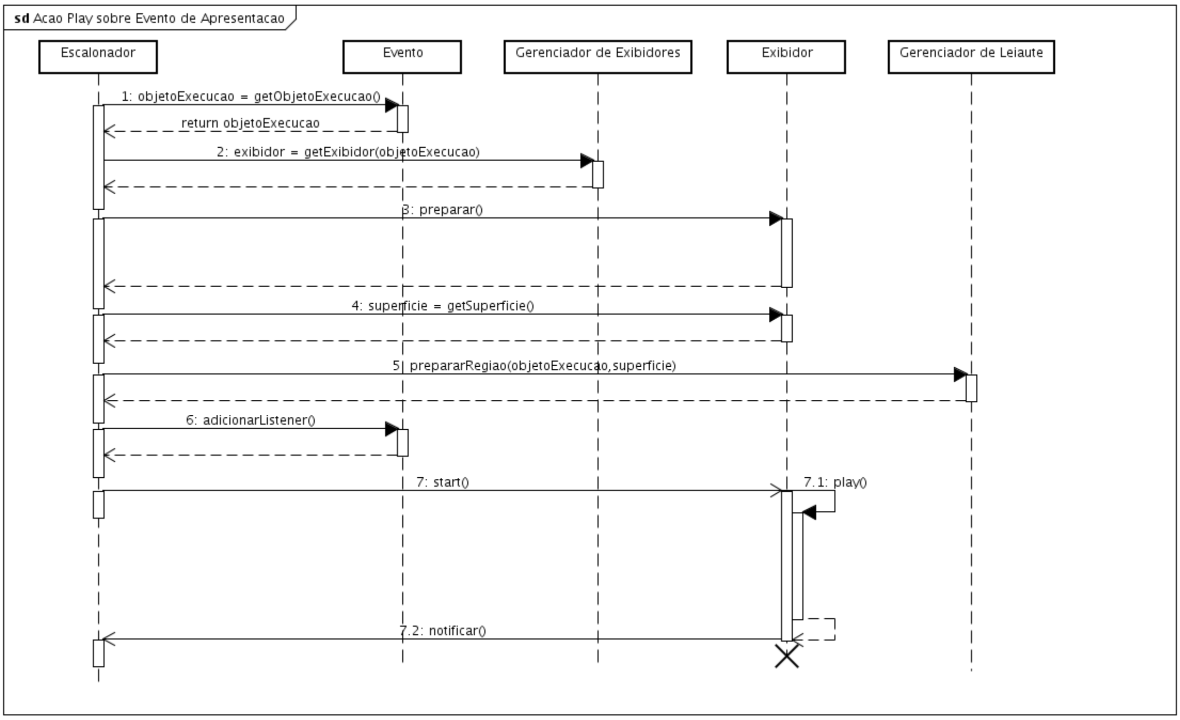


Figura 4.8: *Escalonador*

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 passadas ao Escalonador através de um *Callback*, permitindo a correta exibição de eventos consecutivos.

Transporte

O módulo de transporte foi definido de forma a gerenciar protocolos e interfaces de rede. Nesta primeira implementação somente foi tratado o protocolo HTTP, para tal foi utilizada a biblioteca fornecida pela API Android™ *HttpURLConnection*.

4.2.2 Testes Preliminares

A fim de realizar os testes preliminares na plataforma, foi desenvolvida uma série de documentos NCL, com base em [62] para construção de programas audivisuais interativos. Tais aplicativos validam componentes básicos do *middleware*. Somente após estas validações o *middleware* foi embarcado para que as avaliações de desempenho mais robustas fossem realizadas.

Apresentar imagens na tela sem sincronismo e com sincronismo

Este teste tem por objetivo validar a exibição de mídias de imagens. O primeiro 4.10(a) exibe imagens na tela do dispositivo com base na definição realizada no arquivo NCL. O segundo teste exibe as imagens 4.10(b) e 4.10(c), utilizando critérios de sincronismo entre as exibições. O sincronismo entre as mídias foi definido através da utilização de *link*, *port* e *media*, conforme apresentado na Figura 4.9. No arquivo onde não há sincronismo foram definidos 4 *ports*, sendo que cada mídia é apresentada através de uma destas 4 *ports*.

```
<port id="pInicio" component="img1"/>

<media id="img1" src="images/1.jpg" descriptor="dArea1"/>
<media id="img2" src="images/2.gif" descriptor="dArea2"/>

<link id="lImgonEndstart" xconnector="onEndStart">
   <bind component="img1" role="onEnd" />
   <bind component="img2" role="start" />
</link>
```

Figura 4.9: *Bloco de código responsável pela sincronização de mídias*

Este validador explora principalmente a utilização dos componentes Gerenciador de Exibidores e Exibidores, no que tange a exibição de mídias de imagem. Também explora recursos do Conversor, Escalonador, Gerenciador de Bases Privadas, Formatador e Adaptadores.

(a) Imagens sem sincronismo

(b) Imagens com sincronismo

(c) Imagens com sincronismo

Figura 4.10: Imagens na tela sem sincronismo e com sincronismo

Através destes testes foi possível detectar a restrição da plataforma em que apenas uma *thread* pode modificar o layout da aplicação. Para tal foi necessário desenvolver o controlador apresentado em 4.2.1. Os demais resultados destes testes foram satisfatórios dado que o dispositivo respondeu bem a esta bateria de testes.

Apresentar imagens na tela com sincronismo e reutilização de área

Este teste tem por objetivo validar a exibição de mídias de imagens com sincronismo, entretanto, reutilizando a mesma área de exibição. A reutilização de áreas, conforme imagens 4.12(a), 4.12(b) e 4.12(c), é uma limitação em muitos dispositivos portáteis, sendo assim, este teste visa verificar esta limitação. A reutilização de área foi definida através da utilização de *link*, *port* e *media*, condorme Figura 4.11.

```
<port id="pInicio" component="img1"/>

<media id="img1" src="images/1.jpg" descriptor="dArea1"/>
<media id="img2" src="images/2.gif" descriptor="dArea2"/>
<media id="img3" src="images/3.png" descriptor="dArea1"/>

<link id="lImgonEndstart1" xconnector="onEndStart">
   <bind component="img1" role="onEnd" />
   <bind component="img2" role="start" />
</link>

<link id="lImgonEndstart2" xconnector="onEndStart">
   <bind component="img2" role="onEnd" />
   <bind component="img3" role="start" />
</link>
```

Figura 4.11: *Bloco de código responsável pela reutilização de área*

Assim como o primeiro, este validador explora principalmente a utilização dos componentes Gerenciador de Exibidores e Exibidores, no que tange a exibição de mídias de imagem. Também explora recursos do Conversor, Escalonador, Gerenciador de Bases Privadas, Formatador e Adaptadores.

(a) Imagens com sincronismo (b) Imagens com sincronismo (c) Imagens com sincronismo

Figura 4.12: Imagens na tela com sincronismo e reutilização

Após a realização dos testes foi possível perceber que o *middleware* se comportou bem, e neste momento também foi possível identificar algumas falhas de programação que foram corrigidas, bem como pontos de gargalo que foram otimizados para redução do tempo de resposta à exibição das mídias.

Apresentar áudios no dispositivo sem sincronismo e com sincronismo

Assim como o primeiro teste, este tem por objetivo validar a apresentação de mídias de áudio com e sem sincronismo. Para esta validação foram desenvolvidas situações básicas, como a simples apresentação do áudio, bem como situações mais avançadas com a utilização de recursos como conectores e links de componentes para definição da sincronização entre os áudios. A estrutura utilizada foi semelhante àquela apresentada na Figura 4.9.

É importante destacar que neste teste foi percebido que em determinadas situações uma mídia de áudio não era apresentada, embora estivesse definida no arquivo NCL. Com os avanços dos testes foi percebido que se tratava do tempo de preparo de mídias simultâneas, que será apresentado no próximo capítulo.

Apresentar vídeos no dispositivo sem sincronismo e com sincronismo

A avaliação de mídias de vídeo, incluiu situações de *i)* simples exibição de vídeo, *ii)* apresentação de vídeos e interrupção em um determinado instante, *iii)* apresentação de vídeos com interrupção em um determinado instante e exibição de outro em seguida mantendo o primeiro pausado, *iv)* apresentação de vídeos com interrupção em um determinado instante e exibição de outro em seguida com destruição do primeiro, e por fim, *v)* apresentação de vídeos com interrupção em um determinado instante e a exibição de outro em seguida mantendo o primeiro pausado combinando também a exibição de imagens e apresentação de áudios aleatórios.

Para todos os testes, foram utilizados recursos apresentados em [62], no capítulo "Sincronizando nós de mídia através de elos e conectores"e no capítulo "Sincronizando diversos nós de mídia estática a um nó de mídia contínua". Em todos eles o *middleware* se comportou bem.

Validar eventos de interação com o telespectador

A fim de se validar eventos de interação com o telespectador foram desenvolvidos arquivos de testes para *i)* validar eventos simples, como dar *play* ao se pressionar uma tecla ou se pressionar a tela do dispositivo, *ii)* exibir um vídeo em *loop* permitindo o usuário prosseguir para um segundo vídeo, pressionando a tecla 1, *iii)* alternar a exibição de vídeos com base no toque das teclas 1 e 2 e por fim, *iv)* exibir um vídeo em *loop*, apresentando duas opções de idioma, ao selecionar uma das opções é exibido um vídeo de abertura, e em seguida um vídeo sem som e um áudio de acordo com o idioma selecionado.

Para todos os testes, foram utilizados recursos apresentados em [62], no capítulo "Interação com Usuário". Em todos eles o *middleware* se comportou bem, sendo necessário apenas alguns ajustes pontuais na codificação dos exibidores.

4.3 Considerações Finais

Este capítulo apresentou as principais etapas e dificuldades encontradas para a construção do *middleware*. Foram apresentados os módulos desenvolvidos, bem como detalhes de sua implementação.

É importante destacar porém, que um fator causador das maiores dificuldades foi partir de uma implementação pública do *middleware* já existente, mas que tinha sido concebida para ser executada em um dispositivo fixo. Em contrapartida, teria havido um esforço ainda maior de desenvolvimento, se a codificação do Ginga-NCL tivesse sido todo refeita sem usar uma implementação pública como ponto de partida.

Por outro lado, outra dificuldade encontrada no desenvolvimento foi a necessidade de um mapeamento conceitual, por exemplo, da parte gráfica, que originalmente foi desenvolvida usando bibliotecas *.awt* do Java, e que na sua origem não são suportadas na plataforma Android™. Sendo assim foi necessário um processo de mapeamento destas bibliotecas que permitissem representação equivalente na plataforma alvo.

No próximo capítulo algumas avaliações de desempenho, serão apresentadas, de forma a ter uma perspectiva de como o *middleware* se comporta em situações reais de utilização.

5 Experimentações e Avaliação dos Resultados

"É preciso ter um desprezo saudável pelo impossível. É preciso tentar coisas que a maioria das pessoas não tentaria."

Frase ouvida na faculdade por Larry Page, co-fundador da Google

Para validar a implementação do *middleware*, foram construídas aplicações multimídia NCL que fazem uso da plataforma desenvolvida. Tais aplicações permitem experimentar de modo não exaustivo alguns pontos chaves da arquitetura Ginga-NCL com objetivo de avaliar o desempenho em dispositivos portáteis.

As aplicações são executadas no *middleware* embarcado em um dispositivo HTC com processador Qualcomm RMSM7200A, 528 MHz executando o sistema operacional Android^{TM}, 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.

Os dois primeiros experimentos partem do princípio que a aplicação NCL já foi recebida integralmente no dispositivo. Neste caso, o objetivo é quantificar *i)* o tempo de preparo de mídias simultâneas gasto pelo escalonador de tarefas, e *ii)* o uso de memória e CPU durante a execução de aplicações NCL. Por fim, um terceiro experimento tem como objetivo verificar a viabilidade da utilização do canal de retorno.

5.1 Análise do tempo de preparo de mídias simultâneas

Conforme apresentado na seção 4.2.1, 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 a serem exibidas, foram desenvolvidas 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, exemplificadas através da Figura 5.1, 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.

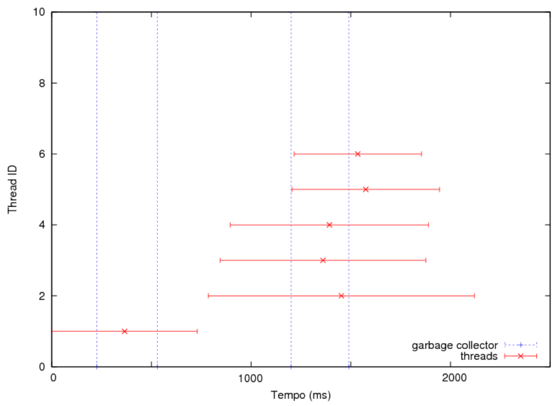
A Figura 5.1 ilustra o código do documento NCL, no qual são exibidas 2 mídias (**img001.jpg** e **img002.jpg**) de modo síncrono com duração explícita, de 7 segundos cada, definida através dos descritores **dArea1** e **dArea2**, nas áreas **area1** e **area2**, sendo o sincronismo definido através do *link* **lImgonBeginstart1**.

```
<xml version="1.0" encoding="ISO-8859-1">
<ncl id="test002" xmlns="http://www.ncl.org.br/NCL3.0/BDTVProfile">
    <head>
      <regionBase>
         <region id="all" width="100%" height="100%">
            <region id="area1"  left="0%"  top="0%"  width="10%" height="25%" />
            <region id="area2"  left="10%" top="0%"  width="10%" height="25%" />
         </region>
      </regionBase>
      <descriptorBase>
         <descriptor id="dArea1" region="area1" explicitDur="7s"/>
         <descriptor id="dArea2" region="area2" explicitDur="7s"/>
     </descriptorBase>
      <connectorBase>
          <causalConnector id="onBeginStart">
             <simpleCondition role="onBegin"/>
             <simpleAction role="start"/>
          </causalConnector>
      </connectorBase>
    </head>
    <body>
        <port id="pInicio" component="img1"/>
        <media id="img1" src="images/img001.jpg" descriptor="dArea1"/>
        <media id="img2" src="images/img002.jpg" descriptor="dArea2" />
        <link id="lImgonBeginstart1" xconnector="onBeginStart">
           <bind component="img1" role="onBegin" />
           <bind component="img2" role="start" />
        </link>
    </body>
</ncl>
```

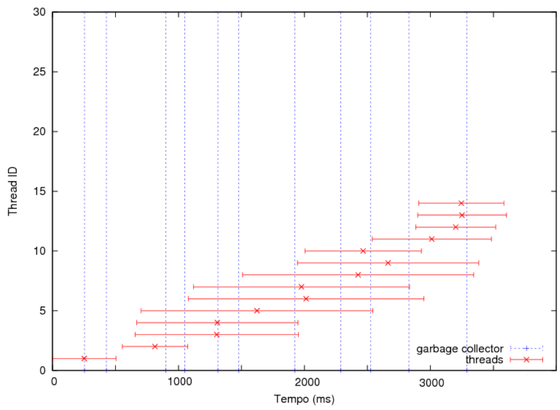
Figura 5.1: *Aplicação NCL para 2 mídias simultâneas*

Para cada um dos documentos NCL foi realizado um estudo de tempo de preparação das mídias que seriam exibidas. As Figuras 5.2(a),5.2(b) e 5.2(c) ilustram os tempos de preparo para cada mídia presente nas aplicações NCL contendo 6, 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 apresentados.

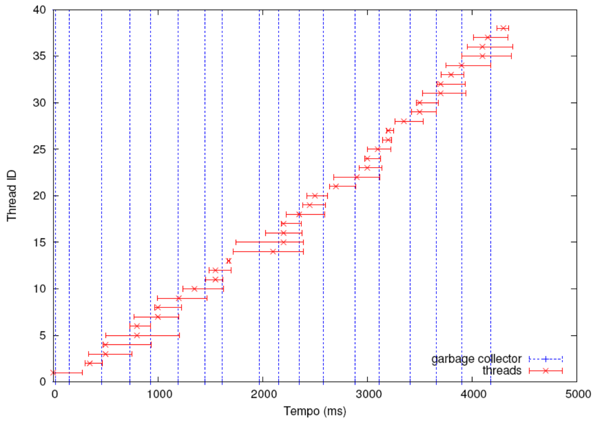
Durante os experimentos foi observado que haviam no máximo 6 *threads* concorrentes do *Media Player* (ocorrido, por exemplo na Figura 5.1). O fato de haver um limite máximo de *threads* simultâneas era um resultado esperado para dispositivos portáteis. No entanto, essa observação sugere aos desenvolvedores 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.



(a) Aplicação contendo 6 mídias NCL



(b) Aplicação contendo 14 mídias NCL



(c) Aplicação contendo 38 mídias NCL

Figura 5.2: Tempo de preparo para aplicações NCL

Além disso, pode-se perceber que várias *threads* tendem a terminar em um instante próximo à execução do *garbage collector*[1] (GC), por exemplo, no instante 2350ms da Figura 5.2(c), 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.

Observa-se também um comportamento similar nas aplicações 5.2(a), 5.2(b) e 5.2(c), no que tange a instanciação da primeira *thread*. Em todas elas inicialmente apenas uma *thread* é instanciada, e posteriormente após o fim, outras são instanciadas em paralelo. Este comportamento se deu, devido a utilização de apenas uma <*port*> no documento NCL proposto. Quando se realizam os mesmos testes utilizando mais de uma <*port*>, percebe-se que o paralelismo já é iniciado no primeiro instante.

[1]Garbage collector pode ser entendido como um processo computacional utilizado para a automatização da gestão de memória em sistemas, através do qual é possível recuperar uma área de memória inutilizada por um programa. Mecanismo este, que pode evitar problemas de vazamento de memória, resultando no esgotamento da memória livre para alocação.

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 5.2(c), a liberação de recursos pelo GC permitiu uma diminuição no tempo de execução das theads 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.

5.2 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 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, por meio da qual é possível capturar uma imagem que está sendo exibida na tela do dispositivo.

5.2.1 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 mantém informações sobre o estado do kernel, permitindo acessar informações de cada processo rodando 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 5.3(a). Caso seja selecionada a segunda opção, serão apresentadas as escuderias participantes, conforme Figura 5.3(b). Por fim, caso seja selecionada a ultima opção, serão exibidos os circuitos nos quais as corridas serão disputadas 5.3(c).

É possível ver na Figura 5.3(a) que a imagem do carro está sobreposta ao vídeo. Essa era uma das limitações da plataforma utilizada no trabalho [4]. Na implementação em Android^{TM}, 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) Circuitos inscritos na temporada

Figura 5.3: Fómula 1

Observando a execução da aplicação (Figura 5.4) 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^{TM}e repassado ao *middleware*, que deve reconhecê-lo e realizar a ação relacionada, processo que resulta no aumento temporário do uso de CPU.

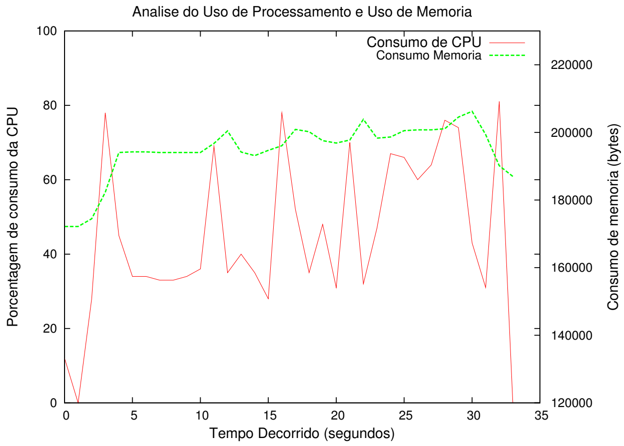


Figura 5.4: Análise de CPU e Memória

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.

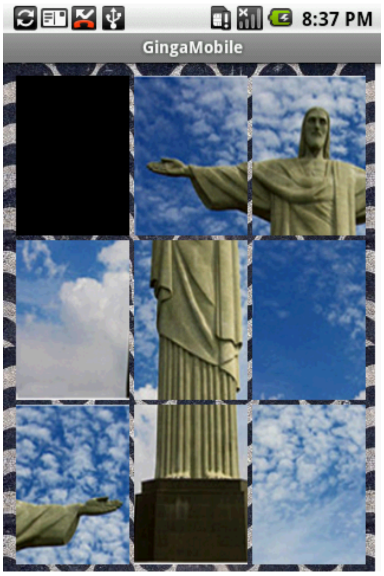
5.2.2 Mosáico

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.

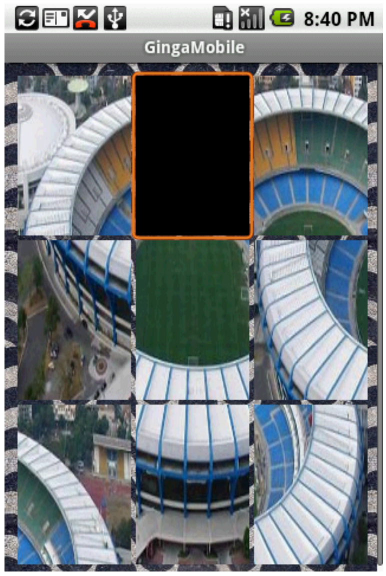
Cada um dos espaços é preenchido com uma imagem embaralhada aleatoriamente, conforme Figuras 5.5(a) e 5.5(b), sendo cada um dos nove blocos passível de seleção, bastando que o visor seja tocado para que o foco mude. O objetivo do jogo é que a imagem seja completada, movimentando-se os blocos para o espaço vazio.

A aplicação NCL possui uma única região, um descritor, uma porta e uma mídia HTML. Esta mídia faz uso de recursos Javascript que são responsáveis pelo algorítmo que faz o tratamento da disposição das imagens.

Durante a execução desta aplicação, conforme Figura 5.6, o consumo de memória manteve-se praticamente constante. Isto se deve à instanciação e apresentação de todas



(a) Mosáico do Cristo Redentor



(b) Mosáico do Maracanã

Figura 5.5: Mosáicos

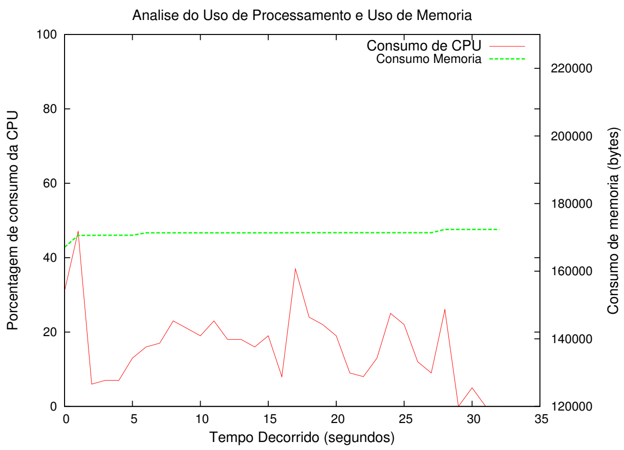


Figura 5.6: Análise de CPU e Memória

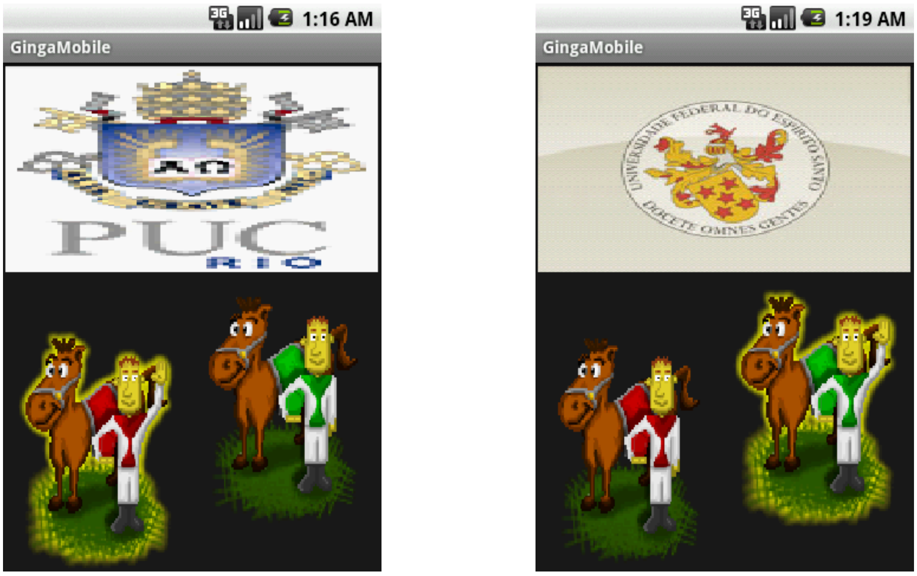
as mídias ser feita no início da aplicação. Desta maneira, a interação do usuário apenas altera a disposição das mídias, gerando picos e vales no consumo de CPU, de maneira similar ao exemplo anterior.

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.

5.3 Apresentação de Mídias Remotas para Teste do Canal de Retorno

Com objetivo de se avaliar a resposta do middleware quanto à utilização do canal de retorno, foi desenvolvida uma aplicação NCL contendo duas imagens que atuam como seletores sensíveis ao toque, através das quais o telespectador pode selecionar uma entre duas imagens a serem apresentadas.

Ao selecionar o primeiro seletor, a plataforma busca uma mídia que está localizada no site da PUC-RJ e a apresenta na parte superior da tela. Caso seja selecionada o segundo seletor, uma imagem que está localizada no site da UFES, é apresentada na mesma região da imagem anterior.



(a) Imagem do brasão da PUC (b) Imagem do brasão da UFES

Figura 5.7: Mídias Remotas

Nesta aplicação somente o exibidor de imagem foi utilizado, embora o protocolo de comunicação tenha sido o HTTP, não foi necessária a utilização do exibidor HTML visto que a imagem estava localizada em um provedor de serviços HTTP. Este exibidor não ofereceu problemas, dado que já havia sido testado previamente.

Ao avaliar-se a aplicação, ficou claro, como esperado, que o tempo de resposta pode ser longo devido ao tempo de download via conexão sem fio (neste caso WIFI), utilizada como canal de retorno para o dispositivo. Percebeu-se também que a usabilidade pode ficar comprometida visto que o usuário da aplicação pode permanecer um longo tempo aguardando a imagem ser carregada, para somente depois conseguir visualizá-la.

5.4 Considerações Finais

Este capítulo apresentou aplicações que foram desenvolvidas com objetivo de realizar experimentações no *middleware*, de forma que fosse possível avaliar o desempenho de aplicações similares àquelas que são construídas por provedores de serviços para televisão digital. As aplicações foram desenhadas buscando a utilização da maior parte dos recursos providos pela linguagem NCL, de forma que se criassem cenários que representassem situações mais próximas do cotidiano.

Apesar de as avaliações não colocarem o dispositivo em seu estado limite de carga, ainda assim dentro de escopo de avaliações realizadas, que evidentemente não cobrem todas as situações possíveis, através delas é possível ter uma perspectiva inicial de como o *middleware* se comporta em situações reais de apresentações de mídias em ambiente de televisão digital, no padrão brasileiro.

No próximo capítulo serão apresentadas as conclusões do trabalho e as perspectivas de trabalhos futuros serão discutidas.

6 Conclusões e trabalhos futuros

"É preciso provocar sistematicamente confusão. Isso promove a criatividade. Tudo aquilo que é contraditório gera vida."

Salvador Dali

Com foco nos terminais de recepção portáteis, o presente trabalho descreve as principais etapas de projeto de uma implementação do *middleware* Ginga-NCL para a plataforma Android™, bem como apresenta detalhes da implementação proposta. Essa é, no nosso conhecimento, a primeira implementação do Ginga-NCL para esse novo e promissor sistema operacional para dispositivos embarcados, bem como a primeira implementação pública de código aberto para dispositivos portáteis e que pode ser obtida em http://www.lprm.inf.ufes.br.

A implementação resultante do trabalho descrito nesta dissertação, se deu de modo correto dado que procurou-se seguir as normas, especificações e recomendações referentes ao padrão nacional Ginga-NCL. Ressalta-se, contudo, com base no protótipo desenvolvido e na experimentação não exaustiva efetuada para validar a implementação, que ainda há um esforço de desenvolvimento para tornar a implementação mais robusta e leve para permitir a execução de uma ampla gama de aplicações NCL em dispositivos portáteis com limitações em seus recursos de processamento e memória.

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.

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. Deve ser implementado também o módulo *Context Manager*, previsto na arquitetura conceitual do Ginga. A sua integração no *middleware* permitirá a construção de aplicações NCL mais elaboradas, que associam mobilidade, interatividade e sensibilidade ao contexto.

Quanto ao processamento de documentos NCL, deve ser recodificado o módulo Conversor utilizando as bibliotecas SAX e deve ser realizado um comparativo de desempenho com a versão aqui proposta.

Deve ser estudado e implementado um mecanismo mais ágil para manipulação das mídias no Gerenciador Gráfico e posteriormente, também comparado com a versão aqui proposta. Atualmente o *middleware* ainda não suporta o redimensionamento de uma região de vídeo durante a exibição de uma imagem, devido a uma limitação da versão 1.5 do SDK utilizado. É necessário também, portar o *middleware* para a versão mais atual (2.2) do SDK e realizar testes para avaliar se tal limitação já foi contemplada nesta versão.

No que diz respeito ao acesso de mídias remotas via canal de retorno pelo *middleware*, deve ser estudado um procedimento para realização de pré-busca em arquivos NCL e talvez manter um cache local de mídias já utilizadas. Para programadores de aplicações NCL, caso seja necessário esperar o carregamento via canal de retorno, existem técnicas para amortizar o tempo de espera percebido pelo usuário, como manter um indicador visual de carregamento ou exibir frases temporizadas. Esse procedimento também pode ser implementado.

Em vista da futura disponibilidade de dispositivos baseados em Android^{TM}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. Embora ainda não existam dispositivos portáteis executando Android^{TM}que contenha o receptor de televisão digital no padrão Brasileiro, é possivel utilizar esta implementação para estudar novas aplicações devido ao suporte a múltiplos dispositivos provido pela linguagem NCL [63] através de seu modelo hierárquico suportado no ambiente de apresentação Ginga-NCL.

Referências Bibliográficas

[1] ANDROID. User interface. Último acesso em fevereiro de 2010, disponível em http://developer.android.com/guide/topics/ui/index.html. , 36

[2] SOARES, L. F. G.; RODRIGUES, R. F.; MORENO, M. F. Ginga-ncl: the declarative environment of the brazilian digital tv system. In: *Journal of the Brazilian Computer Society*. Porto Alegre, RS, Brasil: [s.n.], 2007. p. 37–46. 13, 19

[3] FILHO, G. L. de; LEITE, L. E. C.; BATISTA, C. E. C. F. *Ginga-J: The Procedural Middleware for the Brazilian DigitalTV System*. Porto Alegre, RS, Brasil: [s.n.], Março 2007. 47-56 p. 13, 19

[4] CRUZ, V. M.; MORENO, M. F.; SOARES, L. F. G. Ginga-ncl: Implementação de referência para dispositivos portáteis. p. 67–78, 2008. 13, 19, 20, 43, 66

[5] FERREIRA, G. D. et al. Ginga-ncl em dispositivos portáteis: Uma implementação para a plataforma android. In: *WebMedia 2010 - Artigos completos e Resumos*. Belo Horizonte, Brasil: [s.n.], 2010. 13, 42

[6] SOARES, L. F. G.; BARBOSA., S. D. J. *Programando em NCL 3.0*. São Paulo, SP, Brasil: Campus, 2009. 13

[7] GIANSANTE, M. et al. Projeto de sistema brasileiro de televisão digital. *Relatório Técnico*, Outubro 2004. 14

[8] SOARES, L. F. G. et al. Ginga-ncl: Declarative middleware for multimedia iptv services. In: *IEEE Communications Magazine*. [S.l.: s.n.], 2010. v. 8, p. 74–81. 17

[9] WEBER, J.; NEWBERRY, T. IPTV Crash Course. 2006. 17

[10] CRUZ, V. M.; MORENO, M. F.; SOARES, L. F. G. Tv digital para dispositivos portáteis - middlewares. Rio de Janeiro, RJ, Brasil, 2008. 18, 20

[11] SET. 2010. Disponível em: <http://www.set.com.br>. 18

[12] ABERT. 2010. Disponível em: <http://www.abert.org.br>. 18

[13] SET/ABERT. 2010. Disponível em: <http://www.mc.gov.br/tv-digital/apresentacao-do-grupo-tecnico-abert/>. 18

[14] SOARES, L. F. G.; CASANOVA, M. A.; COLCHER, S. An architecture for hypermedia systems using mheg standard objects interchange. information services and use. In: *IOS Press*. Amsterdam, The Netherlands: [s.n.], 1993. p. 131–139. 18

[15] SOARES, L. F. G.; CASANOVA, M. A.; RODRIGUEZ, N. de L. R. Modelo de contextos aninhados - um modelo conceitual hipermídia. In: *Revista Brasileira de Computação*. Rio de Janeiro, Brasil: [s.n.], 1994. p. 35–48. 18

[16] SOARES, L. F. G.; RODRIGUEZ, N. de L. R.; CASANOVA, M. A. Nested composite nodes and version control in an open hypermedia systems. In: *Information Systems Journal*. Oxford, UK: [s.n.], 1995. p. 501 – 519. 18

[17] COLCHER, S. et al. Modelo de objetos baseado no corba para sistemas hipermídia abertos com garantias de sincronização. In: *XIV Simpósio Brasileiro de Redes de Computadores*. Fortaleza, Brasil: [s.n.], 1996. p. 18–37. 18

[18] ABNT. 2010. Disponível em: <http://www.abnt.org.br>. 18

[19] MELO, J. C. P. de et al. Os módulos ncl e nclua do middleware ginga para aplicações em tv digital interativa. Natal, Rio Grande do Norte, Brasil, Junho 2008. 18

[20] ABNT/NBR15606-2. *ABNT NBR 15606-2. 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.* Rio de Janeiro, RJ, Brasil, Abril 2009. 19, 22, 43, 48, 52

[21] ABNT/NBR15606-5. *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.* Rio de Janeiro, RJ, Brasil, Abril 2009. 19, 22, 43

[22] ETSI. ETSI EN 302 304. Digital Video Broadcasting (DVB); Transmission System for Handheld Terminals (DVB-H). Novembro 2004. 20

[23] TELISONERA. Mobile Broadcast Multicast Service (MBMS) . 2004. 21, 23

[24] ARIB. *ARIB STD-B24. Data Coding and Transmission Specification for Digital Broadcasting.* [S.l.], Junho 2008. 21

[25] ATSC. *ATSC Standard: ACAP Service Signaling and Announcement.* Washington, D.C., EUA, Setembro 2006. 21

[26] FRANCA, B. et al. *DTV Report on COFDM and 8-VSD Performance.* Office of Engineering and Technology, Setembro 1999. 21

[27] ISO/IEC. *ISO/IEC 14496-14:2003. Information technology – Coding of audio-visual objects – Part 14: MP4 file format.* [S.l.], 2003. 22

[28] ISO/IEC. *ISO/IEC 13818-3:1998. Generic coding of moving pictures and associated audio information, Part 3: Audio.* 1998. 22

[29] ABNT/NBR15606-4. *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.* Rio de Janeiro, RJ, Brasil, 2007. 22, 43

[30] ISO/IEC. *ISO/IEC 14496-11:2005. Coding of audio-visual objects, Part 11: Scene description and Application engine (BIFS, XMT, MPEG-J)*. [S.l.], Outubro 2005. 23

[31] HOFFMAN, D.; FERNANDO, G.; GOYAL, V. *RTP Payload Format for MPEG1/MPEG2 Video, RFC 2038*. [S.l.], Outubro 1996. 23

[32] SMITH, B. *A Quick Guide to GPLv3*. [S.l.], 2007. 24

[33] ISO/IEC. *ISO/IEC 9899:1999. Programming languages - C*. [S.l.], 1999. 24

[34] ADMOB. Admob mobile metrics report. Último acesso em fevereiro de 2010, disponível em http://metrics.admob.com/wp-content/uploads/2010/03/AdMob-Mobile-Metrics-Feb-10.pdf. 24, 25, 26, 27, 28

[35] ADMOB. Admob mobile metrics report. Último acesso em fevereiro de 2010, disponível em http://metrics.admob.com/wp-content/uploads/2010/01/AdMob-Mobile-Metrics-Dec-09.pdf. 24, 25, 26, 28, 29

[36] EPL. Eclipse public license - v 1.0. Último acesso em fevereiro de 2010, disponível em http://www.eclipse.org/org/documents/epl-v10.php. 24

[37] ADMOB. Admob mobile metrics report. Último acesso em fevereiro de 2010, disponível em http://metrics.admob.com/wp-content/uploads/2009/09/AdMob-Mobile-Metrics-Aug-092.pdf. 25, 26, 27

[38] ADMOB. Admob mobile metrics report. Último acesso em fevereiro de 2010, disponível em http://metrics.admob.com/wp-content/uploads/2010/02/AdMob-Mobile-Metrics-SEAsia-Q409.pdf. 25

[39] ALLIANCE, O. H. *http://www.openhandsetalliance.com/press_110507.html*. 31

[40] MARCONDES, C.; MARTINELLO, M.; GUARDIA TITLE = Redes Sociais Móveis no Sensoriamento Participativo do Meio Urbano, B. . A. y. . . A. . V. p. . . m. . O. H. C. In: . [S.l.: s.n.]. 31

[41] RUBINI, A.; COBERT, J. *Linux Device Drivers*. [S.l.]: O REILLY, 2001. 32

[42] MURPHY, M. L. *The Busy Coder's Guide To Android Development*. [S.l.]: CommonsWare, LCC, 2009. 33

[43] ANDROID. Application fundamentals. Último acesso em fevereiro de 2010, disponível em http://developer.android.com/guide/topics/fundamentals.html. 33

[44] LECHETA, R. R. *Google Android: aprenda a criar aplicações para dispositivos móveis com Android SDK*. São Paulo, SP, Brasil: Novatec, 2009. 34

[45] ANDROID. Graphics. Último acesso em fevereiro de 2010, disponível em http://developer.android.com/guide/topics/graphics/. 37

[46] PEREIRA, L. C. O.; SILVA, M. L. da. *Android para desenvolvedores*. Rio de Janeiro, RJ, Brasil: BrasPort, 2009. 38

[47] ANDROID. Android developers. Último acesso em junho de 2010, disponível em http://developer.android.com. 38

[48] SJOBERG, J. et al. *RTP Payload Format and File Storage Format for the Adaptive Multi-Rate (AMR) and Adaptive Multi-Rate Wideband (AMR-WB) Audio Codecs, RFC 4867*. [S.l.], Abril 2007. 39

[49] KOSONEN, T.; WHITE, T. *Registration of Media Type audio/mobile-xmf, RFC 4723*. [S.l.], Dezembro 2006. 39

[50] GONCALVES, I.; PFEIFFER, S. *Ogg Media Types, RFC 5334*. [S.l.], Setembro 2008. 39

[51] FLEISCHMAN, E. *WAVE and AVI Codec Registries, RFC 2361*. [S.l.], Junho 1998. 39

[52] OTT, J. et al. *RTP Payload Format for ITU-T Rec. H.263 Video, RFC 4629*. [S.l.], Janeiro 2007. 39

[53] WENGER, S. et al. *RTP Payload Format for H.264 Video, RFC 3984*. [S.l.], Fevereiro 2005. 39

[54] BERC, L. et al. *RTP Payload Format for JPEG-compressed Video, RFC 2435*. [S.l.], Outubro 1998. 39

[55] ALVESTRAND, H. *X.400 Image Body Parts, RFC 2158*. [S.l.], Janeiro 1998. 39

[56] BOUTELL, T. *PNG (Portable Network Graphics) Specification Version, RFC 2083*. [S.l.], Março 1997. 39

[57] KATZ, A. *Format for Bitmap files, RFC 797*. [S.l.], Setembro 1981. 39

[58] ABNT/NBR15606-1. *ABNT NBR 15606-1. Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiodifusão digital Parte 1: Codificação de dados*. Rio de Janeiro, RJ, Brasil, Maio 2010. 43

[59] ABNT/NBR15606-3. *ABNT NBR 15606-3. Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiodifusão digital Parte 3: Especificação de transmissão de dados*. Rio de Janeiro, RJ, Brasil, Agosto 2008. 43

[60] OREN, Y. Sax parser benchmarks. Último acesso em fevereiro de 2010, disponível em http://piccolo.sourceforge.net/bench.html. 47

[61] DEVSPHERE. Sax versus dom. benchmark performed with xerces and crimson. Último acesso em fevereiro de 2010, disponível em http://www.devsphere.com/xml/benchmark/method.html. 47

[62] NETO, C. de S. S. et al. *Construindo Programas Audiovisuais Interativos Utilizando a NCL 3.0 e a Ferramenta Composer*. [S.l.]: PUC Rio, 2007. 56, 59

[63] SOARES, R. M. de Resende Costa e Marcio Ferreira Moreno e L. F. G. Ginga-ncl: Suporte a múltiplos dispositivos. In: *XV Simpósio Brasileiro de Sistemas Multimídia e Web - WebMedia2009*. Fortaleza, CE, Brasil: [s.n.], 2009. 73