Ginga-NCL: Implementação de Referência para Dispositivos Portáteis

Vítor Medina Cruz
Marcio Ferreira Moreno
Luiz Fernando Gomes Soares
Departamento de Informática – PUC-Rio
Rua Marquês de São Vicente, 225
Rio de Janeiro/RJ – 22453-900 - Brasil
{vcruz,mfmoreno,lfgs}@inf.puc-rio.br

ABSTRACT
This paper aims to present the first reference implementation of Ginga-NCL for portable receivers. Although based on a particular platform, the implementation not only works as a concept proof, but also raised several issues that should be taken into account when embedding this system. Ginga is the standard middleware of the Brazilian Digital TV System and Ginga-NCL is the unique required environment for portable devices.

RESUMO
Este artigo tem como objetivo apresentar a primeira implementação de referência do Ginga-NCL para receptores portáteis. Embora desenvolvida para uma plataforma particular, a implementação serviu não apenas como prova de conceito, mas também para levantar diversas questões e principais dificuldades sobre o porte de tal ambiente. Ginga é o middleware padrão do Sistema Brasileiro de TV Digital e Ginga-NCL é o único ambiente necessário em dispositivos portáteis.

Categories and Subject Descriptors
I.7.2 [Document Preparation]: Languages and systems, Hypertext/hypermedia, Markup languages, Standards.

General Terms
Design, Standardization, Languages.

Keywords
Declarative Middleware, Ginga-NCL, Interactive DTV, NCL, Multimedia Systems, Hypermedia Systems, Mobile iTV.

1. Introdução
O uso de dispositivos portáteis no contexto da TV Digital já é uma realidade em diversas partes do mundo [1]. Permitir acesso ao conteúdo televisivo a qualquer instante, em qualquer lugar, e até mesmo em movimento, torna a TV Digital (TVD) para dispositivos portáteis um modelo de negócios, no mínimo, interessante.

WebMedia'08, October 26-29, 2008, Vila Velha, ES, Brazil.

Nesse modelo, é possível oferecer, além dos serviços de TV tradicionais, serviços com interatividade, no qual o usuário telespectador não reage apenas aos estímulos audiovisuais, mas também às aplicações interativas que despertam seu interesse. Para oferecer uma interface padronizada e um suporte de programação para a autoria dessas aplicações, os dispositivos receptores devem possuir uma camada de software, denominada middleware, usualmente padronizada por cada sistema de TV Digital particular [2].

Os principais middlewares de TV Digital [2] oferecem como suporte à autoria de aplicações interativas tanto linguagens declarativas quanto imperativas. De forma análoga, o Sistema Brasileiro de TV Digital Terrestre (SBTVD-T) especifica seu middleware, denominado Ginga, subdividido em dois ambientes: um para suporte à autoria baseada na linguagem declarativa NCL (*Nested Context Language*) e em sua linguagem imperativa de script Lua (Ginga-NCL [3] [4]) e outro para suporte à autoria de aplicações na linguagem Java (Ginga-J [5]).

O objetivo deste artigo é apresentar a primeira implementação de referência do Ginga-NCL para receptores portáteis, salientando suas singularidades quando comparada à implementação para receptores fixos. Em [6] são discutidas as particularidades da implementação de referência do Ginga-NCL para terminais fixos e móveis. É importante salientar que para receptores portáteis apenas o ambiente Ginga-NCL é obrigatório.

A implementação de referência para receptores fixos serviu como base para a realização do trabalho proposto. Entretanto, ela teve que ser adaptada às características e limitações diferenciais encontradas nos dispositivos portáteis. Normalmente, os recursos disponíveis nesses aparelhos, como, por exemplo, resolução gráfica e memória primária/secundária, assim como a definição de prioridade de serviço (o serviço principal em um celular, por exemplo, deve ser o de atender às chamadas telefônicas) são distintos dos encontrados nos receptores fixos.

Além de expor as adaptações necessárias para tratar essas diferenças, a implementação de referência para receptores portáteis é discutida neste artigo com o intuito de servir como um modelo a ser seguido por implementações futuras do Ginga-NCL para esses dispositivos. Ou seja, a implementação realizada é apresentada juntamente com os problemas decorrentes da mudança de plataforma.

Outro ponto discutido neste artigo é a validação da arquitetura da implementação de referência do Ginga-NCL para dispositivos fixos, que foi desenvolvida com o objetivo de tornar

procedimentos de porte mais simplificados, mesmo para outros perfis. O trabalho apresentado neste artigo constitui em mais um passo para essa validação, demonstrando que a arquitetura realmente facilita os procedimentos de porte.

O artigo está organizado da seguinte forma. A Seção 2 discute os principais middlewares de TVD existentes para receptores portáteis, realizando uma breve comparação entre esses middlewares e o Ginga-NCL. A Seção 3 discute os principais sistemas operacionais e plataformas de desenvolvimento para dispositivos portáteis. A Seção 4 discorre sobre a arquitetura do Ginga-NCL proposta. A Seção 5 descreve a implementação realizada. Finalmente, a Seção 6 é reservada para as considerações finais.

2. Trabalhos Relacionados

O escopo de uma linguagem declarativa deve atender ao máximo possível os principais requisitos para o desenvolvimento de aplicações interativas de TVD, tais como: sincronismo das mídias, adaptabilidade, suporte a múltiplos dispositivos, suporte a edição em tempo de exibição e, finalmente, suporte ao reuso. Quanto maior o suporte oferecido, mais fácil é o desenvolvimento de aplicações, deixando para a parte imperativa do middleware apenas tarefas complementares.

Definir o sincronismo como um requisito dessas linguagens é importante, pois, comumente, as aplicações para TVD devem lidar com a sincronização, espacial e temporal, de objetos de diferentes tipos de mídia, além dos objetos de vídeo e áudio que compõem o fluxo principal. A adaptabilidade, por sua vez, permite aos autores especificarem aplicações com comportamentos diferentes, de acordo com o contexto de apresentação, como, por exemplo, localidade e perfil do telespectador, tipo de dispositivo receptor etc. Já o suporte a múltiplos dispositivos é interessante por permitir às aplicações fazerem uso, simultâneo ou não, de outros dispositivos que não o de exibição principal [7], evitando muitas vezes que uma interação de um usuário perturbe o resto da audiência, pois seu resultado poderia ser exibido apenas para o usuário em questão. O suporte à edição em tempo de apresentação oferece dinamismo na medida em que possibilita a geração e modificação de aplicações ao vivo. Por fim, o reuso é responsável por permitir que estruturas reutilizem outras, reduzindo o volume de código necessário para especificar uma aplicação. Além disso, o reuso pode reduzir o esforço na criação de novas aplicações, uma vez que elementos da nova aplicação podem referenciar outros elementos de aplicações já especificadas.

Entre os principais middlewares de TVD existentes no contexto dos dispositivos portáteis [1], pode-se citar o LASeR (*Lightweight Application Scene Representation*) [8] e o BML (*Broadcast Markup Language*) para dispositivos *one-seg* [9].

O middleware LASeR especifica, como interface para a autoria de aplicações, uma linguagem baseada em SVG (*Scalable Vector Graphics - Tiny Profile*) [10]. O perfil *Tiny* SVG é uma linguagem declarativa padronizada pelo W3C (*World Wide Web Consortium*) e foi desenvolvida especificamente para dispositivos com as limitações de recursos dos dispositivos portáteis.

Além de *Tiny* SVG, Extensões LASeR são incorporadas à arquitetura do middleware, como ilustrado na Figura 1. Essas extensões foram especificadas para permitir o desenvolvimento de alguns serviços específicos [8]. Ainda na arquitetura do LASeR, foi definido o componente denominado SAF (*Simple Aggregation Format*), responsável por oferecer suporte a um formato específico para o transporte de dados. O SAF foi criado com o objetivo de oferecer uma solução mais leve e simples do que outras opções existentes para o transporte de dados, como é fundamentado em [8]. É interessante mencionar que o LASeR é totalmente independente do protocolo de transporte, mas o padrão recomenda o uso do SAF.

A Figura 1 ilustra outros dois componentes na arquitetura LASeR: Comandos LASeR e Codificação Binária. O componente Comandos LASeR foi definido para permitir atualizações dinâmicas das aplicações. Com o objetivo de reduzir o *footprint* das aplicações, uma codificação binária é realizada sobre os documentos XML *Tiny* SVG.

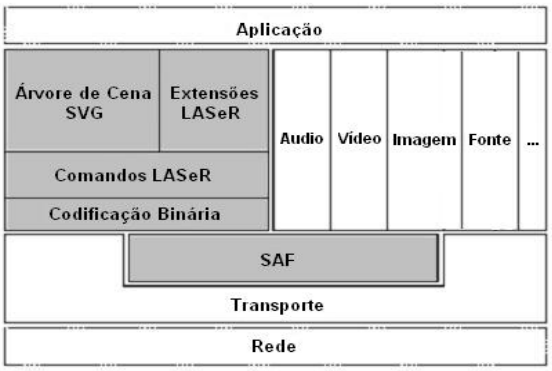

Figura 1. Arquitetura do Middleware LASeR. Retirado e adaptado de [8].

O foco da linguagem definida pelo LASeR é no sincronismo das mídias, que é alcançado com o uso e adaptação do modelo temporal definido pelo SMIL (*Synchronized Multimedia Integration Language*) [11], também uma linguagem declarativa padronizada pelo W3C.

No LASeR, o suporte à adaptabilidade é realizado através da seleção de elementos que referenciam conteúdo (objetos de mídia), por meio de comparações entre os valores de seus atributos e as características do receptor e do sistema. Somente um objeto de mídia que tem satisfeitos todos os valores definidos nos atributos é apresentado. Como exemplo, um objeto que possuir o atributo *systemLanguage*=”en”, só será apresentado caso idioma do sistema seja o inglês.

Para prover suporte simples ao reuso, o perfil *Tiny* SVG permite a existência de referências entre os objetos de mídia. Quando um objeto referencia outro, novas coordenadas (x, y, largura e altura) de apresentação devem ser especificadas [10]. Por fim, o suporte à edição em tempo de apresentação é obtido através de comandos LASeR, que são encapsulados em metadados MPEG-4 [8]. Durante a apresentação de uma aplicação, os comandos podem alterar o leiaute atual ou inserir, remover e alterar os objetos de mídia. Até onde as pesquisas dos autores deste artigo alcançaram, LASeR não oferece suporte a múltiplos dispositivos. Apesar de ser apontado [12] como um dos principais middlewares especificados para os receptores portáteis, nenhuma implementação comercial do LASeR foi encontrada, apenas a implementação de referência, detalhada em [13].

O middleware do padrão de TV Digital Japonês especifica em sua arquitetura, ilustrada na Figura 2, um *BML User Agent*, que é responsável por receber e apresentar aplicações BML. A

linguagem BML é baseada em XHTML [14], CSS [15], DOM [16] e ECMAScript [17], tecnologias padronizadas pelo W3C. O foco da linguagem BML é na interação com o usuário, e não no sincronismo entre as mídias em sua forma mais geral. Qualquer outra forma de sincronismo que não seja a interatividade precisa ser implementada com o uso de scripts.

Figura 2. Arquitetura do Middleware BML.

Linguagens de scripts são imperativas e possuem um poder de expressão maior que aquele oferecido pelas linguagens declarativas. Apesar disso, o uso desse tipo de linguagem é muito mais difícil e suscetível a erros. O *footprint* das aplicações que possuem componentes imperativos também aumenta. Isso porque, ao se fazer uso de uma linguagem imperativa, é preciso descrever algoritmicamente o que é necessário fazer. No contexto da TV Digital, ao se confrontar com a necessidade do uso de uma linguagem imperativa, o autor da aplicação precisa atuar como um desenvolvedor de software. Adaptabilidade e edição em tempo de exibição são os outros requisitos suportados em BML apenas através de scripts [9].

Outra limitação de BML é quanto ao reuso. Na BML, ele se restringe ao reuso que os elementos XHTML podem fazer sobre elementos CSS especificados. Ou seja, com exceção do reuso de elementos CSS, não existe a possibilidade de reutilizar elementos XHTML sem replicar o código. Finalmente, BML não oferece suporte a múltiplos dispositivos e à edição em tempo de apresentação (mais uma vez, ela deve ser feita via scripts e eventos gerados externamente [9]).

O Ginga-NCL oferece como suporte à autoria de aplicações a linguagem declarativa NCL (*Nested Context Language*) [3]. NCL atende a todos os requisitos mencionados no início desta seção. O foco dessa linguagem é no sincronismo das mídias, no qual a interatividade do usuário é um caso particular. Em NCL, o sincronismo é definido por especificações dos relacionamentos de causalidade (representados por elos) entre estruturas que referenciam conteúdo (chamadas de objetos de mídia).

NCL oferece suporte à adaptabilidade de conteúdo e da forma como um conteúdo particular deve ser apresentado. Uma adaptação pode ser realizada com base em regras definidas pela aplicação, por características do dispositivo receptor ou pelo contexto de apresentação (perfil do usuário, localização etc.). O suporte a múltiplos dispositivos pode ser especificado de forma transparente na definição do leiaute da aplicação, onde o autor pode definir o dispositivo de exibição desejado.

NCL também provê um grande suporte ao reuso, tanto de objetos de mídia quanto dos elementos que especificam o leiaute da aplicação. Como em NCL o conteúdo é separado dos relacionamentos de sincronismo, o reuso é provido de maneira trivial [6]. O suporte à edição em tempo de exibição é oferecido através da especificação de uma API de edição. Essa API define comandos que podem ser transportados em metadados de sincronização ou acionados por código imperativo. Comandos podem ser enviados pelo provedor de conteúdo, obtidos através de um canal de retorno, ou virem de aplicações residentes. Os comandos definidos permitem a adição, remoção ou alteração das estruturas definidas pela linguagem [4].

Com base nos requisitos discutidos no início desta seção, pode-se entender o porquê de Ginga-NCL ter sido padronizado como o middleware do SBTVD-T. Uma implementação do Ginga-NCL embarcada em aparelhos portáteis era, no entanto, importante, para servir como uma prova de conceito capaz de fundamentar a especificação brasileira. Antes de apresentar a arquitetura e a primeira implementação de referência do Ginga-NCL para os dispositivos portáteis, contudo, é importante mostrar as opções de plataforma de desenvolvimento existentes e expor a escolha realizada para a implementação.

3. Infra-estrutura de Desenvolvimento

Para permitir o desenvolvimento da implementação realizada, um sistema operacional adequado às características e limitações diferenciais dos dispositivos portáteis teve que ser escolhido, bem com uma plataforma de desenvolvimento para esse sistema. Windows Mobile, Linux e Symbian foram os três sistemas operacionais que se destacaram entre os pesquisados [1].

O Windows Mobile, desenvolvido pela Microsoft, possui como vantagem permitir o desenvolvimento das suas aplicações através das mesmas ferramentas usadas na versão do Windows para computadores pessoais. Por outro lado, esse sistema e suas ferramentas possuem custos atrelados [1].

O crescimento do mercado de dispositivos portáteis tem despertado o interesse de desenvolvedores das distribuições do sistema operacional Linux. Um ponto interessante do Linux é o suporte a sistemas heterogêneos [18]. Além disso, Linux possui uma plataforma aberta e a maioria das suas ferramentas de desenvolvimento pode ser usada sem custo adicional. Entretanto, no instante em que eram iniciados os esforços para este trabalho, existia uma limitação da disponibilidade de um dispositivo portátil com Linux embarcado.

O sistema operacional Symbian foi desenvolvido especificamente para dispositivos portáteis. É um sistema robusto e se encontra no mercado desde 1998. Foi desenvolvido com o objetivo de ser econômico no consumo de recursos, em especial memória e bateria. Além disso, possui plataforma aberta e as aplicações podem ser desenvolvidas em uma série de tecnologias diferentes, como Symbian C++, Java e Flash. Embora tenha muitas vantagens, os custos atrelados à plataforma constituem um ponto negativo. O sistema em si não é gratuito para o fabricante do dispositivo e as ferramentas de desenvolvimento gratuitas possuem recursos limitados, apesar de serem estáveis.

Entre os três sistemas apresentados, Symbian foi o escolhido para a implementação realizada por possuir maior disponibilidade de dispositivos para desenvolvimento, além de oferecer suporte através de comunidades e fóruns sobre o sistema.

As principais plataformas de desenvolvimento do sistema operacional Symbian são a plataforma nativa e a plataforma JavaME. Na plataforma nativa, a linguagem Symbian C++, que é baseada na linguagem C++, é usada no desenvolvimento das

aplicações. Symbian C++ é uma linguagem complexa, mas permite ao desenvolvedor ter maior controle sobre o código da sua aplicação, facilitando, com isso, a definição de otimizações mais refinadas. O desenvolvimento na plataforma nativa do sistema Symbian também implica na possibilidade do uso direto de APIs aperfeiçoadas [19], e do acesso completo aos recursos de hardware.

A plataforma JavaME (*Micro ou Mobile Edition*) [20] é um subconjunto da J2SE [21] e foi desenvolvida para atender aos requisitos dos dispositivos portáteis. Na prática, a portabilidade, comum em plataformas Java, não é trivial na JavaME. Além disso, essa plataforma não oferece suporte a otimizações mais refinadas. Um ponto negativo a ser salientado é a ausência da definição de uma API de acesso aos recursos de hardware.

As características da plataforma JavaME apresentadas dificultam um possível processo de implementação do Ginga-NCL. Por outro lado, a plataforma nativa do Symbian OS reúne algumas características interessantes para esse projeto.

A plataforma Symbian C++ oferece alguns perfis de desenvolvimento para interface com o usuário, denominadas UIs (*User Interface*). As UIs determinam as características específicas do sistema Symbian, e são associadas às famílias de dispositivos. Na implementação realizada, o uso de APIs Symbian genéricas foi priorizado, entretanto, parte da implementação necessariamente [22] teve de ficar atrelada a uma UI específica.

A UI escolhida foi a S60, com o SDK que oferece suporte a versão 9.2 do sistema operacional Symbian. Ela é a mais recente UI disponibilizada para desenvolvimento [23]. Os critérios de escolha da UI foram: a disponibilidade de documentação, o suporte ao desenvolvimento e a quantidade de dispositivos S60 disponíveis no mercado, o que possibilitaria testes em dispositivos com maior flexibilidade e rapidez.

4. Arquitetura

Como já mencionado, o trabalho proposto neste artigo é baseado na implementação de referência do Ginga-NCL para terminais fixos [6]. Nessa implementação, foram definidos níveis de abstração para centralizar as características específicas da plataforma e do sistema operacional em uma única camada, apresentada na Figura 3. Essa "camada de abstração" foi desenvolvida com o objetivo de facilitar procedimentos de porte e facilitou o desenvolvimento do trabalho proposto.

A Figura 3 ilustra a arquitetura modular do Ginga-NCL para dispositivos portáteis, que é dividida em dois subsistemas lógicos: a Máquina de Apresentação Ginga-NCL e o Núcleo Ginga.

No Núcleo Ginga, o módulo Sintonizador é responsável por receber conteúdo de TV Digital para portáteis transmitido por provedores de conteúdo. As aplicações interativas podem chegar ao dispositivo portátil multiplexadas nesse conteúdo, ou por outra interface de rede (por exemplo, canal de dados oferecido pela operadora de celular, conexão Bluetooth ou IEEE 802.11 etc.). No primeiro caso, a aplicação é obtida através de um processamento, realizado pelo módulo Processador de Dados, sobre o conteúdo recebido. No segundo caso, um componente de Transporte foi definido para controlar protocolos e interfaces de rede e atender a demanda da Máquina de Apresentação por conteúdo e aplicações. Para gerenciar o armazenamento das aplicações de TV e o conteúdo que essas aplicações referenciam, o módulo de Persistência foi definido.

Figura 3. Arquitetura Ginga-NCL para Dispositivos Móveis.

Ainda no Núcleo Ginga, o módulo Exibidores é responsável por prover, à Máquina de Apresentação, o decodificador adequado para a apresentação de um conteúdo específico. O módulo Gerenciador Gráfico foi definido para realizar o controle espacial da renderização de objetos de acordo com o especificado pelas aplicações de TV. Para atender às especificações da norma brasileira [3] [4], que define Lua [24] como a linguagem script para NCL, uma máquina Lua é agregada ao Núcleo. Os módulos definidos na arquitetura podem ser atualizados de forma independente. As atualizações podem ser enviadas pelo provedor de conteúdo ou obtidas através do módulo de Transporte. Cabe ao módulo Gerenciador de Atualizações realizar o procedimento de atualização, sem interromper o funcionamento do middleware. Finalmente, um Gerenciador de Contexto foi definido para gerenciar as informações sobre o sistema embarcado no dispositivo e sobre o perfil do usuário telespectador.

Na arquitetura apresentada na Figura 3, o Núcleo Ginga é responsável por oferecer os serviços anteriormente mencionados à Máquina de Apresentação Ginga-NCL. A Máquina de Apresentação Ginga-NCL é um subsistema lógico capaz de iniciar e controlar aplicações NCL. O núcleo da Máquina de Apresentação NCL é o Formatador. Esse módulo é responsável por receber e controlar as aplicações NCL entregues pelo Núcleo Ginga. Ao receber uma aplicação NCL, o Formatador solicita uma tradução das especificações NCL em estruturas de dados adequadas à apresentação das aplicações. Essa tradução é realizada pelo módulo Conversor. O resultado da tradução é então agrupado em uma estrutura de dados denominada Base Privada.

Em seguida, o módulo Escalonador é solicitado, pelo Formatador, para orquestrar a apresentação da aplicação NCL. Para permitir que cada conteúdo seja exibido de forma adequada, o Escalonador solicita que o Gerenciador de Exibidores instancie cada Exibidor de mídia apropriado, de acordo com o tipo de conteúdo a ser exibido em um dado instante. Com o intuito de padronizar a comunicação entre a Máquina de Apresentação e a API de decodificação de conteúdo dos exibidores, um componente, denominado Adaptadores, foi definido na arquitetura.

A exibição de um conteúdo deve ser realizada em regiões do dispositivo, conforme especificado pelo autor da aplicação NCL. O módulo Gerenciador de Leiaute foi definido para fazer a associação do conteúdo, tratado por um exibidor específico, a essas regiões.

O módulo Gerenciador de Bases Privadas é responsável por gerenciar todas as Bases Privadas criadas pelo Formatador, assim como processar os comandos de edição ao vivo mencionados na Seção 2. Ao processar um comando de edição, mudanças sobre as estruturas presentes nas Bases Privadas podem ser realizadas pelo Gerenciador de Bases Privadas [3]. Finalmente, um Gerenciador de Contexto NCL foi definido para realizar adaptações na apresentação das aplicações NCL, de acordo com as informações providas pelo Núcleo Ginga e/ou pela própria aplicação NCL.

5. Implementação

Como discutido na Seção 3, a implementação realizada foi desenvolvida utilizando a plataforma nativa do sistema operacional Symbian, utilizando Symbian C++. Apesar de ser uma linguagem similar a C++, algumas diferenças devem, no entanto, ser ressaltadas.

Symbian C++ oferece suporte a todos os tipos básicos da linguagem C++, mas todos esses tipos são redefinidos ou possuem classes que devem substituí-los. Por exemplo, o `TInt` no Symbian é, na verdade, uma definição (`typedef`) para o tipo básico `int`. Um outro exemplo é o `TChar` que é uma classe usada na substituição do tipo básico `char`. A recomendação de uso desses tipos específicos do Symbian C++ serve mais para forçar os programadores a manterem uma padronização no código do que para aumentar a sua eficiência. Ou seja, desconsiderar essa recomendação não significa comprometer a eficiência e o desempenho da implementação.

A implementação de referência Ginga-NCL para terminais fixos faz uso de estruturas definidas nas bibliotecas padrões C [25], C++ [26] e STL (*Static Template Library*) [26]. As bibliotecas encontradas no Symbian C++, porém, diferem dessas bibliotecas, o que poderia afetar a arquitetura da implementação realizada como um todo. Em um esforço para promover uma padronização entre as linguagens e facilitar procedimentos de porte, as bibliotecas PIPS (*P.I.P.S. Is Posix on Symbian*) [27] e *Open C* [28] foram desenvolvidas pelo consórcio Symbian e pela Nokia, respectivamente. Juntas, essas bibliotecas cobrem boa parte das bibliotecas padrões C e C++, inclusive as funcionalidades utilizadas na implementação de referência do Ginga-NCL para terminais fixos.

Por outro lado, na tentativa de manter a consistência entre as duas implementações, seria necessário ainda concentrar esforços em portar a biblioteca STL. Para isso, duas soluções se apresentaram: mapear todas as APIs STL C++ em APIs Symbian C++, ou portar uma implementação STL disponível em C++ para Symbian C++. A solução do mapeamento foi parcialmente implementada e consumiu um tempo excessivo de desenvolvimento. Acredita-se que a segunda solução demandaria maior tempo de desenvolvimento por envolver todas as funcionalidades internas da STL.

Concomitantemente ao processo de mapeamento das APIs, uma implementação da STL foi portada para Symbian C++, por um membro da comunidade Symbian, surgindo a STLPort [29]. Como os esforços sobre o mapeamento de APIs ainda não estavam em fase de conclusão, a biblioteca STLPort foi considerada. Uma vez verificado o seu desempenho, através da implementação do Ginga-NCL para dispositivos portáteis, os esforços sobre o mapeamento de interfaces foram interrompidos.

Uma funcionalidade importante da arquitetura do Ginga-NCL para terminais fixos é a de escalonamento de tarefas. Através do escalonamento de tarefas, é possível executar algoritmos em linhas independentes, simulando ou criando (de acordo com o tipo de processador) ações que executam em paralelo, um dos requisitos mais importantes em aplicações que exijam sincronismo como as de TVD. Na implementação de referência para receptores fixos, o escalonamento de tarefas é realizado através da manipulação de processos leves, ou *threads*, da biblioteca *Posix* [31]. Essa funcionalidade também é suportada pelas bibliotecas PIPS e Open C. Existe, entretanto, na plataforma Symbian, uma limitação que faz com que a instanciação e manipulação de objetos específicos de exibição, como vídeo e áudio, só possam ser realizadas no processo principal da aplicação.

Para tentar resolver esse problema, foi desenvolvido um esquema de requisições entre as *threads* e o processo principal, onde as *threads* requisitam instanciações ou ações pré-definidas de objetos, sendo o processo principal o responsável por atender a essas requisições. A idéia consiste em deixar o processo principal em espera até que as *threads* em execução realizem essas requisições durante o seu processamento. Essa solução tornou possível a realização de testes sobre a implementação realizada. Entretanto, esses testes, bem como relatos de desenvolvedores da comunidade Symbian, deixaram claro que o uso de *threads* na arquitetura Symbian é proibitivo, devido à grande perda de desempenho. A alternativa foi o uso de *Active Objects* [32], uma estrutura específica da plataforma Symbian.

Os *Active Objects* são utilizados, normalmente, com o objetivo de controlar o funcionamento de funções assíncronas [32], mas também podem ser usados para simular o comportamento de uma *thread*. A diferença básica é que os diferentes *Actives Objects* de uma mesma aplicação executam todos na mesma linha de execução do processo principal.

A substituição das *threads* pelos *Actives Objects* resolveu o problema da perda de desempenho, mas introduziu um erro no tratamento do sincronismo temporal entre trechos de conteúdos, especificado nas aplicações NCL. Esse tipo de sincronismo, em NCL, consiste em um intervalo de tempo dentro da duração da apresentação de um conteúdo qualquer.

Na implementação de referência do Ginga-NCL para terminais fixos, uma *thread* é criada para tratar o sincronismo temporal. Essa *thread* é responsável por contabilizar os intervalos de tempo, especificados na aplicação NCL, aguardando o início ou o fim de um intervalo. Nesses instantes, a *thread* deve notificar, ao módulo Escalonador, a ocorrência desses eventos.

A substituição de *threads* que tratam o sincronismo temporal por *Active Objects* apresenta uma idiossincrasia relacionada à espera dos instantes especificados. As esperas, realizadas após a notificação do inicio e antes da notificação do fim de um intervalo, bloqueiam o processo que está relacionado a todos os *Active Objects* existentes na implementação. Conforme já discutido, os *Actives Object* são executados na mesma linha de execução do processo principal. Assim, as chamadas de espera bloqueiam o processamento da aplicação como um todo. No caso das *threads*, chamadas de espera bloqueiam apenas o processo leve em execução, enquanto que o processo principal não é afetado.

Para resolver esse problema, foi preciso utilizar um *Active Object* específico, denominado `CTimer` [22]. O `CTimer` possui um

método, denominado `After`, capaz de realizar espera em um domínio especifico. Ou seja, o `After` cria um processo leve com prioridade de kernel [22] para aguardar um intervalo de tempo especificado, sem consumo de processamento. Após esse intervalo de tempo, o *Active Object* que fez a chamada ao método `After` é notificado, podendo prosseguir sua execução. Assim, o funcionamento do *Active Object* pode ser interrompido enquanto o processo principal permanece em funcionamento. Além disso, nesse caso, é permitido que outros *Actives Objects* possam ser executados.

Outra adaptação contida na implementação realizada, envolve o módulo Conversor, apresentado na Seção 4. Para realizar o *parser* das aplicações NCL, o módulo Conversor, na implementação do Ginga-NCL para terminais fixos, faz uso de um *parser* DOM [33]. Nesse tipo de *parser,* é criada, e alocada na memória, uma árvore de objetos (árvore DOM), que representa a estrutura do documento XML. A partir da árvore DOM, é possível acessar todos os elementos, e seus atributos, do documento XML, em qualquer instante, até que a árvore DOM seja explicitamente removida da memória.

Normalmente, o modelo DOM é utilizado quando é necessário acessar informações sobre os elementos do documento XML em instantes não determinísticos, e por diversas vezes. Existe um ganho no desempenho do processamento, por não ter que realizar uma nova interpretação do documento a cada consulta. Por outro lado, manter uma árvore DOM alocada na memória pode ser uma tarefa crítica para os dispositivos portáteis. Uma alternativa para interpretação de documentos XML, denominada SAX [33], consiste em processar documentos através do paradigma orientado a eventos. Nessa alternativa, o desenvolvedor deve criar um controlador que será notificado sobre a existência de elementos e atributos, quando o *parser* SAX encontrá-los no documento XML.

Conforme discorrido na Seção 4, o módulo Conversor traduz as especificações NCL em estruturas de dados adequadas à apresentação das aplicações. A partir disso, o módulo Formatador utiliza essas estruturas para apresentar a aplicação NCL. Através de testes sobre a interpretação de documentos NCL e a apresentação desses documentos, foi verificado que não existia a necessidade de fazer acessos repetidos à estrutura do documento NCL, descartando assim o modelo DOM da implementação realizada. O módulo Conversor foi, portanto, re-projetado utilizando a API SAX na implementação de referência do Ginga-NCL para dispositivos portáteis. Apesar de essa verificação ser uma contribuição interessante, existe um ponto aberto neste trabalho devido à necessidade de se realizar testes de desempenho com os dois tipos de parser mencionados, em função de diferentes aplicações NCL.

Um último ponto sobre a implementação realizada que merece atenção especial, refere-se ao acesso sobre o modelo de apresentação do dispositivo. Para realizar a exibição de conteúdo, é preciso fazer acesso à tela do dispositivo, procedimento que, no Symbian, é realizado através de uma estrutura denominada `Window`. A partir da `Window`, são definidas as regiões manipuladas pelo módulo Gerenciador de Leiaute, apresentado na Seção 4. Para representar cada região especificada na aplicação NCL, foram utilizadas estruturas denominadas `Controls`. As `Controls` são utilizadas para implementar as diferentes interfaces gráficas de uma aplicação e, normalmente, compartilham uma única `Window`. Cada `Control` é associada a um conteúdo, de acordo com a especificação da aplicação NCL. Ou seja, o conteúdo decodificado por um exibidor é renderizado em uma `Control`.

O módulo Exibidores da implementação de referência para dispositivos portáteis oferece suporte a decodificação de vídeo, áudio, imagem e documentos HTML. Os exibidores de mídia foram implementados através de APIs específicas da plataforma Symbian C++ [22] [34]. Em especial, para os exibidores de áudio e vídeo, foram utilizadas duas estruturas do arcabouço MMF (*Multi Media Framework*) [22] [34] denominadas `CMdaAudioPlayerUtility` e `CVideoPlayerUtility`, respectivamente. Para o exibidor de imagem, a estrutura `CFbsBitmap` foi utilizada. Uma limitação do exibidor de imagem, imposta pela estrutura `CFbsBitmap`, é a capacidade de exibir apenas imagens `mbm`, um formato específico do Symbian [22] [34]. Finalmente, para exibir documentos HTML, a estrutura `CBrCtlInterface` foi utilizada.

O módulo Sintonizador 1-seg, introduzido na Seção 3, não foi implementado devido à falta de disponibilidade de um dispositivo com esse tipo de interface durante a realização deste trabalho. Os módulos da arquitetura apresentada na Seção 4, que não foram discutidos nesta Seção, não apresentam diferenças entre a implementação realizada e a implementação de referência para terminais fixos. Exceto a máquina e o exibidor Lua que são citados entre os trabalhos futuros.

A implementação resultante de todo o trabalho descrito foi testada em um emulador de dispositivos Symbian e embarcada em dois dispositivos portáteis diferentes: *Nokia 5700 Express Music*[1] e *Nokia N81 1G*[2]. O *Nokia 5700 Express Music* possui um processador de 369 MHz e 64 MB de memória RAM, sendo aproximadamente 18 MB disponíveis para as aplicações. Já o

Figura 4. Aplicação de Compra Sendo Apresentada pelo Ginga-NCL Embarcado.

[1] http://www.forum.nokia.com/devices/5700_XpressMusic. 2008.

[2] http://www.forum.nokia.com/devices/N81. 2008.

Nokia N81 1G possui um processador de 369 MHz e 96 MB de memória RAM, sendo aproximadamente 42 MB disponíveis para as aplicações.

Foi observado o funcionamento correto e com desempenho satisfatório da implementação realizada, tanto no emulador quanto nos dispositivos embarcados. Os testes realizados foram do tipo sistêmico, onde várias funcionalidades são testadas em conjunto. A melhor forma de se fazer esse tipo de teste é desenvolver uma aplicação NCL propriamente dita, onde vários recursos são exercitados ao mesmo tempo. Um conjunto considerável de aplicações foi desenvolvido e testado na implementação realizada. Como exemplo, em uma aplicação mais abrangente, foi especificado o sincronismo da aplicação com um trecho do vídeo. Nesse teste, passados alguns segundos de exibição do vídeo principal, a aplicação oferece ao usuário a opção de realizar a compra de um CD com trilhas sonoras relacionadas ao vídeo. A Figura 4 apresenta a realização desse teste em um dos aparelhos citados.

Ao navegar pelas opções de sim e não, o usuário pode optar por fazer ou não a compra do CD, conforme ilustrado na Figura 4. Caso o não seja escolhido, o vídeo principal continua a tocar normalmente e a opção desaparece. Caso a opção o sim for a escolhida, o vídeo é pausado e uma outra tela com novas opções é apresentada, assim como ilustrado na Figura 5. Ainda nesse caso, são oferecidas ao usuário telespectador três trilhas sonoras como amostra gratuita. O usuário pode, então, selecionar uma das três opções de amostra para ouvir ou acessar uma página HTML, através da Web, para realizar a compra. Uma opção para voltar à exibição normal do vídeo principal é também oferecida.

Através do teste descrito, é possível observar todos os exibidores de mídia implementados. O mais interessante dessa aplicação, pode-se dizer, é o uso do exibidor HTML, por utilizar o canal de retorno (nessa demonstração, o acesso é realizado através de uma interface de rede IEEE 802.11 do dispositivo).

Figura 5. Ginga-NCL - Formulário de compra de CD.

Apesar dos testes terem demonstrado a eficiência do Ginga-NCL nos dispositivos portáteis, algumas limitações da implementação realizada, em relação à implementação do Ginga-NCL para dispositivos fixos, foram observadas nos dispositivos usados. A primeira limitação observada é que não é possível apresentar uma imagem sobre um vídeo devido à limitação da arquitetura dos dispositivos testados, que possuem uma única camada gráfica de apresentação. A segunda limitação é a impossibilidade de se definir níveis de transparência para o conteúdo exibido. Finalmente, como nos dispositivos onde o Ginga-NCL foi embarcado a decodificação do áudio e vídeo é realizada por software, a soma total do tamanho das mídias apresentadas ao mesmo tempo em uma aplicação NCL não pôde ser muito alta. Nos testes realizados, somas maiores que 2 MB causaram o término da aplicação por falta de recursos. Esse problema seria inexistente em dispositivos apropriados para TV digital, com decodificação de vídeo por hardware, bem como os outros problemas mencionados, em dispositivos seguindo a Norma SBTVD-T.

6. Considerações Finais

Em geral, um middleware de TV Digital [2] oferece suporte à autoria tanto em linguagens declarativas quanto imperativas. No escopo das linguagens declarativas, com o intuito de facilitar o processo de autoria e, ao mesmo tempo, oferecer a expressividade computacional adequada, é interessante que essas linguagens possuam características que atendam aos principais requisitos para o desenvolvimento de aplicações interativas de TV Digital. Como discutido na Seção 2, NCL é uma linguagem que atende a esses requisitos. Conforme apresentado ao longo deste trabalho, a especificação Ginga-NCL é suficientemente leve para ser embarcada mesmo em dispositivos portáteis que não foram direcionados ao uso como receptor de TV. Isso demonstra que seu uso em receptores seguindo a Norma SBTVD-T terá um excelente desempenho.

A vantagem da linguagem declarativa NCL frente a outras linguagens declarativas, aliada ao poder de expressão de Lua e, agora, a comprovada leveza da implementação fazem antever o acerto da escolha do Ginga-NCL como o padrão brasileiro para dispositivos portáteis.

Muitos dos problemas observados e apontados neste trabalho, embora decorrentes de uma plataforma particular, possivelmente ocorrerão no porte ou no desenvolvimento do Ginga-NCL para outras plataformas portáteis. Espera-se, portanto, que os resultados aqui obtidos possam ser usados como referência em implementações futuras do Ginga-NCL. Pelo menos três dos problemas discutidos na Seção 5 merecem destaque por seu caráter genérico:

- O escalonamento de tarefas realizado através da manipulação de processos leves, ou *threads*, pode ter seu desempenho comprometido e devem ser cuidadosamente considerados. Outros mecanismos substitutos podem ser utilizados com o mesmo efeito e melhor desempenho, como foi o caso dos *Active Objects* da plataforma Symbian.
- A manutenção de uma árvore DOM em memória pode ser uma tarefa crítica para os dispositivos portáteis, e pode mesmo ser desnecessária, mesmo até em receptores fixos. A alternativa para interpretação de documentos NCL usando SAX deve ser cuidadosamente considerada. Deve ser salientado, no entanto, que

existe um ponto em aberto nessa questão, exigindo a realização de testes de desempenho em função de diferentes aplicações NCL.

• A limitação no modelo conceitual gráfico do dispositivo receptor portátil muitas vezes inviabiliza a realização de aplicações mais elaboradas. Muitos dispositivos, mesmo aqueles construídos com o foco na recepção de TV, possuem uma única camada gráfica de apresentação, inviabilizando a apresentação de um outro objeto de mídia sobre um vídeo, em particular o vídeo principal de um programa.

Concluindo, é importante ressaltar que a implementação corrente é uma prova de conceito. Como trabalho futuro, é necessário o embarque em dispositivos seguindo a especificação SBTVD e a realização da bateria de testes unitários e sistêmicos já desenvolvidos para o Ginga-NCL. Além disso, outros exibidores de mídia requeridos pelo padrão de middleware do SBTVD devem ser avaliados, em particular o porte da máquina de execução Lua, cuja eficiência em dispositivos portáteis já foi comprovada em diversos trabalhos [35].

7. Referências

[1] Cruz, V. M. Ginga-NCL para Dispositivos Portáteis. Dissertação de mestrado; PUC-Rio, DI, 2008.

[2] Morris, S., Smith-Chaigneau, A. Interactive TV Standards: A Guide to MHP, OCAP, and JavaTV. Focal Press, 2005.

[3] Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiofusão digital – Parte 2: Ginga-NCL para receptores fixos e móveis – Linguagem de aplicação XML para codificação de aplicações. Setembro 2007.

[4] 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. Março de 2008.

[5] Televisão digital terrestre - Codificação de dados e especificações de transmissão para transmissão digital – Parte 4: Ginga-J - Ambiente para a execução de aplicações procedurais. Setembro 2007.

[6] Soares, L.F.G., Rodrigues, R.F., Moreno, M.F. Ginga-NCL: the Declarative Environment of the Brazilian Digital TV System. 2006.

[7] Dick C.A. Bulterman, Pablo Cesar, A.J. Jansen. An Architecture for Viewer-Side Enrichment of TV Content. ACM, 2006.

[8] ISO 14496-20. Lightweight Application Scene Representation (LASeR) and Simple Aggregation Format (SAF). Jun. 2006.

[9] B24 Appendix 5 – Operational Guidelines for Implementing Extended Services for Mobile Receiving System. fev. 2004.

[10] Scalable Vector Graphics (SVG) 1.1 Specification. Disponível em: http://www.w3.org/TR/2003/REC-SVG11-20030114/. jun. 2003. Acesso em junho de 2008.

[11] Smil 2.1 – Timing and Syncronization. Disponível em: http://www.w3.org/TR/2005/REC-SMIL2-20051213/smil-timing.html. dec. 2005. Acesso em fevereiro de 2007.

[12] Jean-Claude Dufourd, Olivier Avaro, Cyril Concolato. LASeR: the MPEG Standard for Rich Media Services. 2006.

[13] LASeR. http://www.mpeg-laser.org/html/techSection_referenceSoftware.htm. Março de 2006. Acesso em julho de 2008.

[14] XHTML 1.0: The Extensible HyperText Markup. Disponível em: http://www.w3.org/TR/2000/REC-xhtml1-20000126/. Jan. 2000. Acesso em junho de 2008.

[15] Cascading Style Sheets, Level 2. http://www.w3.org/TR/REC-CSS2/. Mai. 1998. Acesso em junho de 2008.

[16] Document Object Model (DOM) Level 1 Specification. Disponível em: http://www.w3.org/TR/REC-DOM-Level-1/. Out. 1998. Acesso em junho de 2008.

[17] Ecma International. Standard ECMA-262 3rd Edition. December 1999.

[18] Yaghmour , Karim . Building Embedded Linux Systems. O'Reilly Media, Inc, 2003.

[19] Symbian Limited. Using Symbian OS GETTING STARTED, 2007 2nd edition. 2-6 Boundary Row, London SE1 8HP, UK, jan. de 2007.

[20] Johnson Thienne M. Java para Dispositivos Móveis - Desenvolvendo Aplicações com J2ME. Novatec, 2007.

[21] Friesen, Jeff. Beginning Java SE 6 Platform: From Novice to Professional. Apress, 2007.

[22] Babin, Steve, Harrison, Richard. Developing Software for Symbian OS An Introduction to Creating Smartphone Applications in C++. Jonh Wiley & Sons Ltd, 2006.

[23] Nokia. S60 Platform SDKs for Symbian OS, for C++. Disponível em: http://www.forum.nokia.com/info/sw.nokia.com/id/4a7149a5-95a5-4726-913a-3c6f21eb65a5/S60-SDK-0616-3.0-mr.html. 2008. Último acesso em Fevereiro de 2008.

[24] Ierusalimschy, Roberto. Programming in Lua. Lua.org, 2003.

[25] ISO/IEC 9899:1999. Programming languages – C. 1999.

[26] ISO/IEC 14882:2003. Programming languages -- C++. 2003.

[27] Symbian Ltd. Using Symbian OS P.I.P.S. fev de 2007.

[28] Forum Nokia. Open C for S60: Increasing Developer Productivity, Version 1.2. mar 2007.

[29] STLPort. Disponível em: http://www.stlport.org/. 2001. Último acesso em Fevereiro de 2008.

[30] Jez, Marco. Disponível em: http://marcoplusplus.blogspot.com/2007/05/stlport-for-symbian-os-released.html. maio 2007. Último acesso em Fevereiro de 2008.

[31] Nichols, Bradford, Buttlar, Dick, Farrell, Jacqueline. Pthreads programming: A Posix Standard for Better Multiprocessing. O'Reilly, 1996.

[32] Stichbury, J. Symbian OS Explained. Effective C++ Programming for Smartphones. J. Wiley & Sons Ltd, 2004.

[33] Arciniegas, Fabio. C++ XML: Exploring New Techniques and New Materials. Sams Publishing, 2001.

[34] Harrison, Richard et al. Symbian OS C++ for Mobile Phones. Volume2. John Wiley & Sons, 2004.

[35] Lua for Emvedded Devices and More. Disponível em http://elua.luaforge.net. Acesso em junho de 2008.