PROPOSTA DE ARQUITETURA PARA DISTRIBUIÇÃO DE INFORMAÇÕES RDF NO AMBIENTE DE TV DIGITAL

Lucas Carnicelli

JUIZ DE FORA
DEZEMBRO, 2011

Proposta de Arquitetura para Distribuição de Informações RDF no Ambiente de TV Digital

LUCAS CARNICELLI

Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharel em Ciência da Computação

Orientador: Eduardo Barrére

JUIZ DE FORA
DEZEMBRO, 2011

PROPOSTA DE ARQUITETURA PARA DISTRIBUIÇÃO DE INFORMAÇÕES RDF NO AMBIENTE DE TV DIGITAL

Lucas Carnicelli

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

_____

Eduardo Barrére, orientador.
DSc em Engenharia de Sistemas e Computação COPPE/UFRJ

_____

Michel Heluey Fortuna
DSc em Engenharia de Sistemas e Computação COPPE/UFRJ

_____

José Maria Nazar David
DSc em Engenharia de Sistemas e Computação COPPE/UFRJ

JUIZ DE FORA, MG - BRASIL
DEZEMBRO, 2011

Agradecimentos

Agradeço
em primeiro e mais importante lugar, à minha família, pelo amor e apoio,
ao meu orientador Barrére, pela paciência e dedicação
e aos amigos, que estiveram ao meu lado durante toda a jornada.

Resumo

A interatividade proporcionada pela TV Digital possibilita a exploração de uma vasta área na distribuição de informação através da televisão. Esse trabalho propõe uma arquitetura para a disponibilização de conteúdo retirado de arquivos RDF através de um método flexível de obtenção da informação. Para provar a viabilidade da arquitetura, foi criada uma aplicação exemplo, desenvolvida na linguagem Java. Com isso, o trabalho fornece um panorama do ambiente atual de desenvolvimento Java para TV Digital.

Palavras-chave: TV Digital. Ginga-J. RDF.

Abstract

The interactivity of the Digital TV allows the exploration of a vast area in the distribution of information through television. This work proposes an architecture to provide content extracted from the RDF files through a flexible method of obtaining information. To prove the viability of the architecture, was created a sample application developed in Java. With this, the paper provides an overview of the current environment of Java development for Digital TV.

Keywords: *Digital TV. Ginga-J. RDF.*

Sumário

Lista de Siglas e Abreviações

API	*Application Programming Interface*
CSV	*Comma Separated-Values*
DOM	*Document Object Model*
HTTP	*Hipertext Transfer Protocol*
IDE	*Integrated Development Environment*
JSON	*JavaScript Object Notation*
NCL	*Nested Context Language*
PSI	Provedores de Serviços Interativos
RDF	*Resource Description Framework*
SBTVD	Sistema Brasileiro de Televisão Digital
SQL	*Structured Query Language*
SPARQL	*SPARQL Protocol and RDF Query Language*
STB	*Set-Top Box*
TVDI	Televisão Digital Interativa
W3C	*World Wide Web Consortium*
URI	*Uniform Resource Identifier*
URL	*Uniform Resource Locator*
XML	*Extensible Markup Language*

Lista de Figuras

1. Introdução

O tema saúde é de vital importância para o ser humano e é o principal requisito para uma vida com qualidade. Isso faz com que cada indivíduo busque informações sobre saúde, com o objetivo de melhorar sua qualidade de vida. Uma pessoa com conhecimento em uma determinada doença está mais bem preparada para lidar com a mesma, caso venha a sofrê-la, do que outra sem o mesmo conhecimento. Portanto, para que essa informação possa chegar à população é preciso utilizar meios de comunicação em massa e que sejam facilmente acessíveis, pois o público que merece mais atenção nessa questão é justamente as classes inferiores da sociedade. Reside nessas informações a motivação para utilizar dados da área de saúde como fontes de informação para a aplicação exemplo criada.

Este trabalho propõe a utilização da transmissão do sinal digital na televisão para a disseminação de informações no formato RDF (*Resource Description Framework*) [1]. Tendo em vista o fato de que a televisão é o meio de comunicação mais popular na nossa sociedade, aliar a distribuição de informação, no nosso caso da área de saúde, com os programas televisivos, torna o objetivo de aumentar o conhecimento da população sobre saúde muito viável. Para atrair o público será explorada a interatividade possibilitada pela TV Digital, na qual o usuário pode se relacionar através do controle remoto com a programação assistida e programas interativos disponíveis.

De acordo com o Decreto 4.901, de 16 de Novembro de 2003, o governo brasileiro objetiva com o Sistema Brasileiro de Televisão Digital (SBTVD) "promover a inclusão social, a diversidade cultural do País e a língua pátria por meio do acesso à tecnologia digital, visando à democratização da informação" [2]. Reside neste fato a principal motivação do trabalho, atingindo a proposta de inclusão social através da distribuição de informações utilizando as tecnologias da TV Digital.

Como exposto em "*Inclusão digital através de serviços de saúde na TV digital interativa"* [3], percebe-se que o tema da saúde é explorado por vários programas e que a interatividade proposta pela TV Digital pode aprofundar ainda mais o tema. Porém, deve-se levar em conta o fato de que o público alvo do Serviço da Saúde são, em sua maioria, pessoas de baixa ou nenhuma escolaridade [3]. Com isso, torna-se clara a necessidade de passar essas informações utilizando uma didática que seja compreensível para este público.

O trabalho também foca o desenvolvimento de uma aplicação utilizando a linguagem Java, o lado imperativo da maquina de execução do *middleware* Ginga [4], a qual possibilita a criação de aplicações mais ricas para a TV Digital.

1.1 Objetivos

A meta principal deste trabalho é propor uma arquitetura de acesso à dados no formato RDF através de um método flexível para obtenção da informação. Para viabilizar essa arquitetura, foi criada uma aplicação exemplo que permite explorar o ambiente atual de desenvolvimento em Java para TV Digital, fazendo uso de interatividade.

Em relação ao conteúdo distribuído pela aplicação, o trabalho visa à utilização de metadados ligados à saúde, os quais serão retirados de uma base de dados governamental.

1.2 Organização

O trabalho está organizado da seguinte forma: a Seção 2 apresenta o sistema de TV Digital no Brasil, assim como fundamentos da sua arquitetura e o panorama atual do desenvolvimento Java para o mesmo. Na Seção 3, é explicado o acesso aos metadados e especificado qual padrão utilizado e de qual local foram tirados os arquivos com o conteúdo. Além disso, define as estruturas das aplicações exemplo criadas e discursa sobre os aspectos gráficos de cada uma, fornecendo um panorama do cenário atual de construção de interfaces para aplicações Java na TV Digital. A Seção 4 detalha os processos de implementação da aplicação exemplo que viabiliza a arquitetura proposta. Finalmente, a Seção 5 apresenta as considerações finais relevantes e possibilidades de trabalhos futuros.

2. TV Digital

Um sistema de TV Digital é um sistema típico cliente e servidor, no qual o cliente é o ambiente do usuário telespectador e o servidor consiste em um ambiente de uma radiodifusora e/ou de um servidor de conteúdo [5]. A figura 1 ilustra esse sistema.

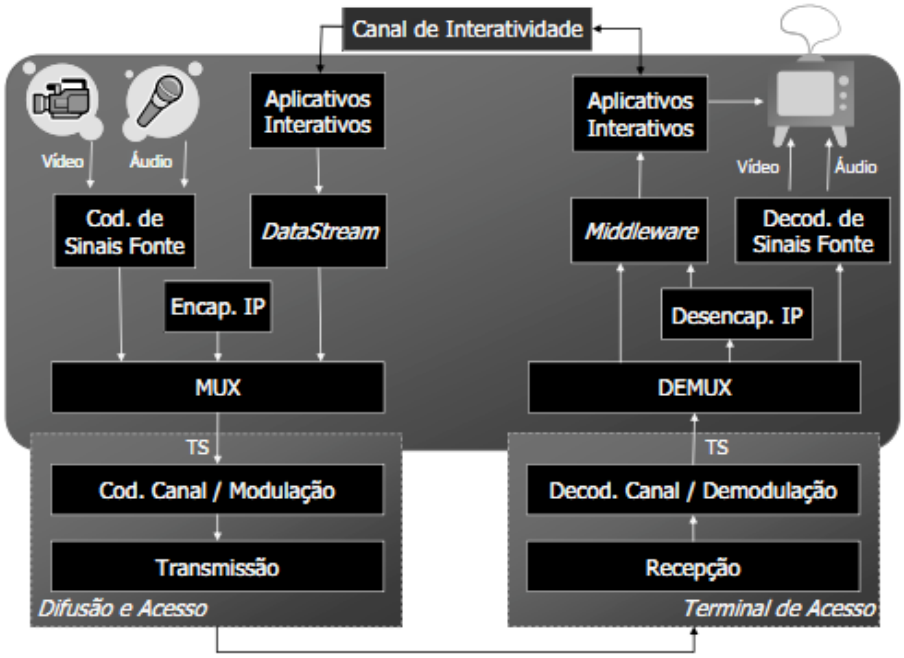

Figura 1. Sistema da TV Digital [5].

Um programa de televisão é composto de um áudio principal e de um vídeo principal, capturado ao vivo de uma câmera, como ilustrado no lado esquerdo da figura 1, ou proveniente de um servidor de vídeo e de dados adicionais, incluindo o aplicativo que define o relacionamento entre os vários objetos de mídia. Esses dados adicionais, nos quais se enquadra a aplicação exemplo desenvolvida nesse trabalho, na maioria das vezes são encapsulados no formato IP ou em um *stream* de dados para sua transmissão.

O vídeo e o áudio são entregues aos codificadores digitais no lado do servidor que são responsáveis pela geração dos respectivos fluxos comprimidos. A esses fluxos, são adicionados os fluxos de dados e então eles são multiplexados em um único sinal, denominado fluxo de transporte (TS). Esse sinal é modulado para um canal de freqüência e transmitido.

Do lado cliente, ilustrado na parte direita da figura 1, o sinal é recebido e retirado do canal de freqüência ao qual foi sintonizado (demodulado). Depois é entregue ao

demultiplexador que separa os fluxos de áudio e vídeo principais do fluxo de dados. O processamento dos dados recebidos pode demandar novos dados, que são buscados pelo canal de retorno (interatividade completa). Dados também podem ser enviados pelo aplicativo à emissora ou outro destino qualquer na rede do canal de retorno (interatividade parcial).

2.1 Fundamentos e arquitetura

Em relação ao modelo de distribuição da informação através da TV Digital, é possível observar que, de acordo com a arquitetura proposta utilizada por Barrére [6], um ambiente de Televisão Digital Interativa (TVDI) é composto basicamente por três grupos: as emissoras, os usuários e os Provedores de Serviços Interativos (PSI). A figura 2 ilustra esse ambiente.

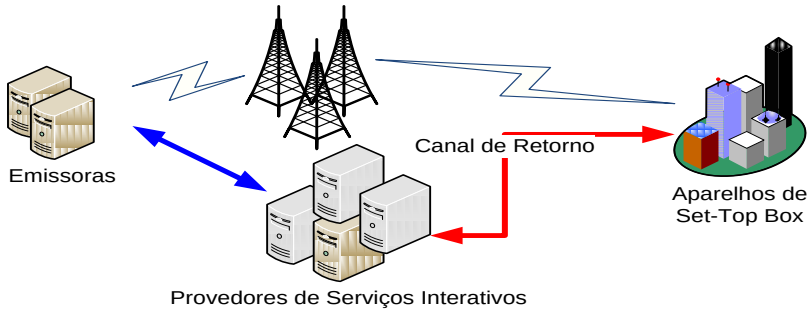

Figura 2. Arquitetura de um ambiente de TVDI [6].

Um PSI tem como função intermediar a interatividade entre os usuários finais e as emissoras de televisão. Para que esses usuários possam desfrutar do sinal digital e do serviço de interatividade, é necessário que possuam um aparelho chamado *set-top box* (STB), responsável por codificar o sinal digital e interpretar o conteúdo interativo que é transmitido junto à programação. Porém, nem todos os STB contam com a funcionalidade de interatividade completa. Para isso, o aparelho deve se comunicar com o canal de retorno [7] ilustrado na figura 2. É por este canal que o usuário solicita informações junto aos PSI, garantindo assim a interatividade na TV Digital e a inclusão digital mencionada.

A respeito dos fundamentos técnicos do trabalho, é importante salientar o papel do *middleware* envolvido no desenvolvimento de aplicações para a TV Digital.

Middleware é uma camada de software posicionada entre o código das aplicações e a infraestrutura de execução (plataforma de hardware e sistema operacional). No caso do ambiente de TVDI, essa camada de software atua no STB.

De acordo com Soares [8], um *middleware* para aplicações de TV Digital consiste de máquinas de execução das linguagens oferecidas e bibliotecas de funções que permitem o desenvolvimento rápido e fácil de aplicações. As linguagens oferecidas se dividem em dois tipos: as declarativas, que descrevem claramente as tarefas que devem ser executadas, e as imperativas, que decompõem passo a passo suas descrições em uma definição algorítmica do fluxo de execução das suas tarefas.

O Sistema Brasileiro de Televisão Digital (SBTVD) utiliza como *middleware* o Ginga. O seu ambiente declarativo provê o desenvolvimento de aplicações utilizando-se a linguagem NCL (*Nested Context Language*) [9], que podem conter entidades imperativas especificadas na linguagem Lua. Em seu ambiente imperativo, o Ginga suporta aplicações feitas com a linguagem Java. Existe também a possibilidade do desenvolvimento de aplicações híbridas, que utilizem NCL, Lua e Java.

Com base em Soares [8], o sistema brasileiro é, atualmente, o mais avançado sistema de TV Digital terrestre, não apenas por usar as tecnologias mais avançadas, mas, principalmente, por dispor de tecnologias inovadoras, como é o caso de seu *middleware* Ginga.

2.2 Desenvolvimento em Java para TV Digital

O Fórum do Sistema Brasileiro de TV Digital Terrestre (SBTVD) [10] está em atividade desde 2006 e tem como objetivo padronizar o sistema de transmissão e recepção de áudio e imagens digitais no Brasil.

A especificação JavaDTV [11] foi definida pelo Fórum SBTVD em parceria com a Sun Microsystems Inc. com o intuito de propor uma adoção livre de royalties para o desenvolvimento imperativo para a TV Digital brasileira. O JavaDTV é o núcleo da máquina de execução Java do sistema de interatividade brasileiro, o *middleware* Ginga.

A figura 3 representa a especificação do Ginga destacando a estrutura da máquina de execução Ginga-J.

Figura 3. Especificação do *middleware* Ginga [14].

A Universidade Federal da Paraíba (UFPB) é responsável pelo projeto atual de desenvolvimento e integração das API´s necessárias, incluindo o JavaDTV, para o uso da linguagem Java na criação de aplicações para TV Digital. O projeto se chama GingaCDN [12] e consiste em uma rede de desenvolvedores de componentes e ferramentas para o *middleware* brasileiro Ginga. Esta iniciativa faz parte do programa Centro de Pesquisa e Desenvolvimento em Tecnologias Digitais para Informação e Comunicação (CTIC), atualmente incubado pela Rede Nacional de Ensino e Pesquisa (RNP), criado pelo governo federal com o objetivo de desenvolver a competência nacional para inovação em comunicações digitais.

As aplicações exemplos desenvolvidas neste trabalho foram criadas a partir de exemplos fornecidos pelo site do projeto GingaCDN e compatível com o Emulador Ginga-J [13] também disponibilizado pelo mesmo projeto. Atualmente esse é o melhor e mais completo caminho para o desenvolvimento Java para TV Digital, visto que algumas API´s não estão finalizadas. Como exemplo, no momento do desenvolvimento das aplicações exemplos, apenas os componentes *Form* e *Label* estavam implementados no JavaDTV do emulador. Com isso, o quesito interface das aplicações não pôde ser muito explorado, sendo objeto de estudos futuros. Porém, toda a estrutura das API´s de interfaces gráficas já estão definidas e bem encaminhadas pelo projeto GingaCDN [14].

3. Arquitetura proposta

Foram criadas duas arquiteturas com o objetivo de comparação entre o custo de acesso ao conteúdo e a interface gráfica. A primeira arquitetura é denominada A1 e a segunda A2 e as mesmas são ilustradas nas figuras 4 e 5 respectivamente.

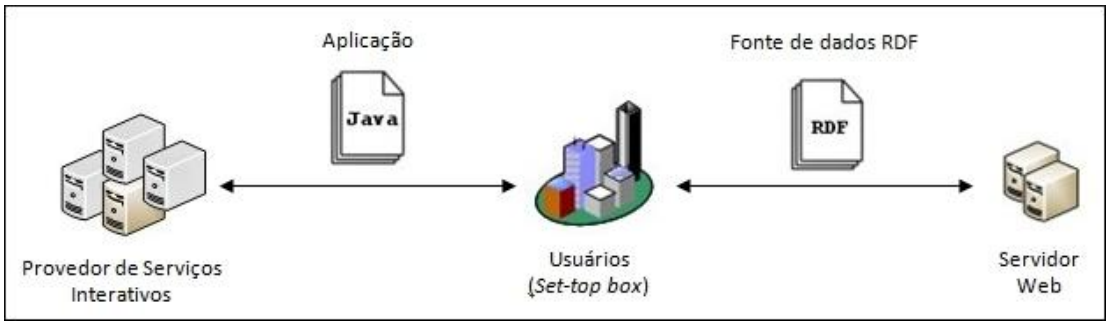

Figura 4. Arquitetura A1.

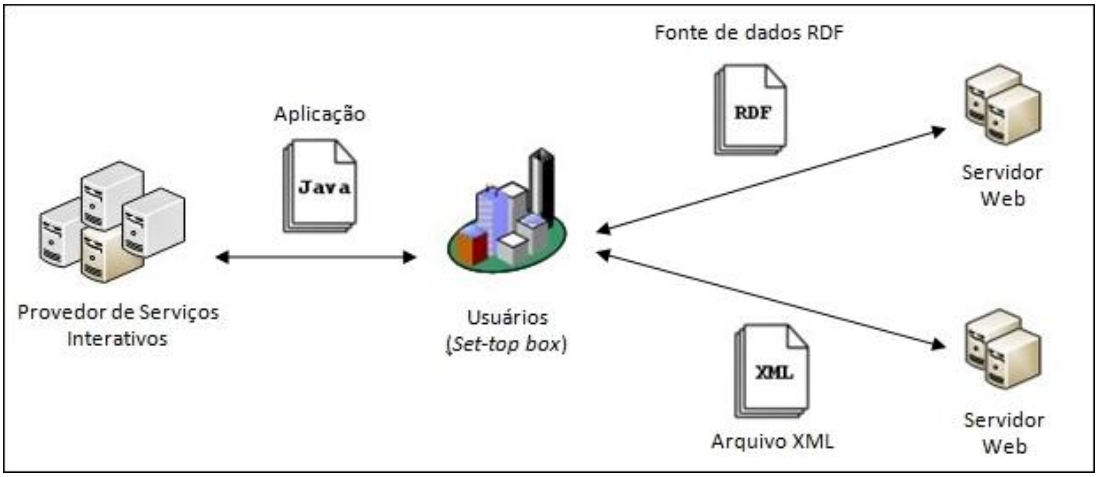

Figura 5. Arquitetura A2.

Em ambas as arquiteturas, a aplicação Java é passada para o usuário a partir dos Provedores de Serviços Interativos. Porém, em A1 apenas uma fonte de dados RDF é acessada tornando o custo de acesso à outra fonte de dados qualquer muito alto, pois o código da aplicação utilizada deve ser alterado. Mas ao mesmo tempo sua interface gráfica pode ser mais bem adaptada ao conteúdo já previamente conhecido. Já em A2, foi adicionado um item a mais, um arquivo XML (*Extensible Markup Language*) localizado em um servidor web que contém as informações necessárias para localizar a fonte RDF, que pode estar em outro servidor web qualquer. Esse item torna o método de acesso aos dados mais flexível e torna a arquitetura A2 a proposta deste trabalho. Em relação à

interface gráfica, A2 perde qualidade já que a aplicação utilizada não tem conhecimento da quantidade e tamanho das informações vindas da fonte RDF.

3.1 Acesso a metadados - RDF

O conceito de metadados consiste em um modelo de dados construído sobre outro modelo já existente, a fim de que esse conteúdo se torne inteligível por sistemas computacionais. Além disso, os metadados padronizam o formato de informações distribuídas em diversas estruturas diferentes. O principal objetivo na sua utilização é a facilidade de entendimento da informação disponibilizada.

A respeito dos metadados, o trabalho propõe a utilização de dados no formato RDF. O RDF é um padrão do W3C (*The World Wide Web Consortium*) para codificação do conhecimento [1]. Como exposto por Santos [15], estabelece um modelo, codifica e transmite metadados com o objetivo de maximizar a interoperabilidade de dados entre fontes heterogêneas. Ele possui uma semântica formal baseada em URI (*Uniform Resource Indentifier*) e uma sintaxe baseada em XML. O meio de extração das informações contidas em um arquivo RDF é feito através de uma linguagem de consultas (*query*) chamada SPARQL [16], que é a padrão do W3C, e possui uma sintaxe muito semelhante a do SQL.

Para a fonte de dados RDF a ser utilizada nas aplicações, a ideia inicial foi à utilização de arquivos contendo informações de saúde referentes ao governo federal do Brasil. Porém não foram encontrados, no momento da pesquisa do trabalho, arquivos RDF disponibilizados ao público. Existem informações referentes a dados abertos [17] [18], porém os arquivos disponibilizados são em sua maioria nos formatos CSV, XML e JSON, não atendendo ao objetivo deste trabalho.

Com isso, a alternativa para a fonte de dados foi buscar arquivos com informação de saúde fornecida pelo governo federal dos EUA. O site utilizado para extração dos arquivos foi o DataGov [19], do governo dos EUA.

É importante ressaltar que a escolha de uma fonte de dados provenientes do governo foi uma escolha para tornar a informação distribuída mais atraente para a população. Porém, desde que a fonte de dados seja um arquivo RDF, qualquer arquivo de fonte distinta funcionará na aplicação desenvolvida.

3.2 Definição das aplicações exemplo

A fim de exemplificar as arquiteturas A1 e A2 foram desenvolvidas duas aplicações. A primeira aplicação é denominada Ap1 e a segunda Ap2, seguindo, respectivamente, A1 e A2. Ambas as aplicações possuem uma estrutura básica que consiste em ler uma fonte de dados RDF em um servidor web, processar as informações desse arquivo, exibir os dados para o usuário e permitir que o mesmo possa navegar entre algumas opções. No caso de Ap2, que segue a arquitetura A2, sua estrutura possui um passo a mais que é a leitura do arquivo XML de configuração.

As aplicações Ap1 e Ap2 foram desenvolvidas com base no emulador Ginga-J criado pelo projeto GingaCDN da Universidade Federal da Paraíba (UFPB) [13]. Esse emulador tem o objetivo de simular a máquina de execução Java que rodará nos aparelhos set-top box presentes nas residências dos usuários ou nas televisões que possuírem o conversor digital integrado com suporte ao Ginga. De acordo com o portal do Fórum SBTVD, o governo vai exigir que até 2015 todos os aparelhos televisores fabricados no Brasil deverão implantar o Ginga [20].

Além da estrutura básica seguida, foram adicionados às aplicações componentes específicos para a criação da interface, para leitura da fonte de dados RDF e, apenas em Ap2, para leitura do arquivo XML de configuração.

Em relação à leitura do arquivo RDF foi criado um método específico que lê, executa a *query* e processa os dados recebidos pela fonte de dados relacionada. Para a manipulação dos arquivos RDF foi escolhida o framework Jena [21] que consiste em um projeto *open source* que fornece um ambiente de programação completo para trabalhar com RDF e SPARQL. O principal fator que motivou na escolha do Jena foi a simplicidade para a codificação da leitura dos arquivos e execução das *queries*.

Quanto à leitura do arquivo XML na Ap2, foi criado um método de leitura utilizando a biblioteca DOM (*Document Object Model*) [22], que se encontra no pacote "org.w3c.dom" da API do Java para processamento de arquivos XML. Por seguir os padrões W3C e ser muito completo no manuseio de arquivos XML, o DOM foi escolhido para esse projeto.

3.3 Interface para TV Digital

O Portal do Software Público Brasileiro [23] é um site que compartilha softwares de interesse público e trata os mesmos como um bem da população. Ele é composto

principalmente por espaços específicos (comunidades) para cada software cadastrado onde é possível a utilização de fóruns, notícias, chats, armazenamento de arquivos, wiki entre muitas outras opções.

O projeto de desenvolvimento do Ginga possui uma comunidade no Portal de Software Público Brasileiro e a partir de setembro de 2010, a equipe de desenvolvedores do GingaCDN, projeto responsável pela criação do Ginga-J, passou a dar suporte através do fórum dessa comunidade.

Como colocado pela própria equipe de desenvolvimento do Ginga-J no fórum da comunidade Ginga, muitos componentes gráficos ainda não foram implementados na máquina de execução e até o momento do desenvolvimento das aplicações exemplo desse trabalho, apenas os componentes *Form* e *Label* estavam completamente implementados. Com isso, o quesito interface gráfica atualmente não pode ser muito explorada para a criação de aplicações Java para TV Digital. Porém, com apenas esses dois componentes é possível realizar aplicações totalmente capazes de fornecer informações relevantes para a população, no caso desse trabalho, informações na área da saúde, tão importantes para nossa sociedade.

O importante a ressaltar em relação às aplicações Ap1 e Ap2 quanto à interface gráfica é o fator que diferencia a estrutura das mesmas. Como Ap1 é restrita a uma única fonte de dados, sua interface pode ser desenvolvida especificamente para ela, devido ao prévio conhecimento dos dados que serão recebidos. Já em Ap2, qualquer fonte RDF pode ser acessada, utilizando-se uma *query* diferente de acesso aos dados para cada uma. Com isso, não se tem o conhecimento prévio das informações recebidas pela aplicação nem de sua quantidade, dificultando a criação de uma interface rica que se adapte aos dados.

As figuras 6, 7a e 7b mostram as telas das aplicações Ap1 e Ap2. Vale observar que na figura 6 a interface foi totalmente desenhada para apresentar as informações contidas no arquivo RDF previamente conhecido e com recursos de navegação também adaptados à navegação dessas informações. Já nas figuras 7a e 7b, a interface é genérica, sem adaptações, *layout* e navegação, à informação oriunda do arquivo RDF.

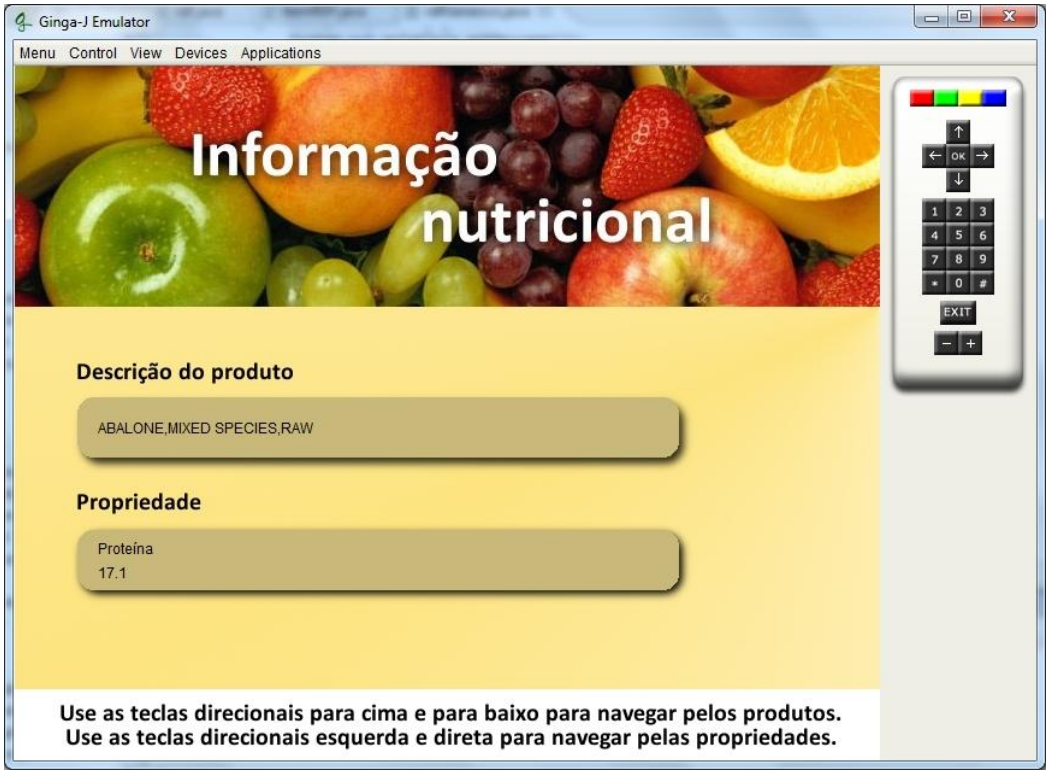

Figura 6. Interface da aplicação Ap1.

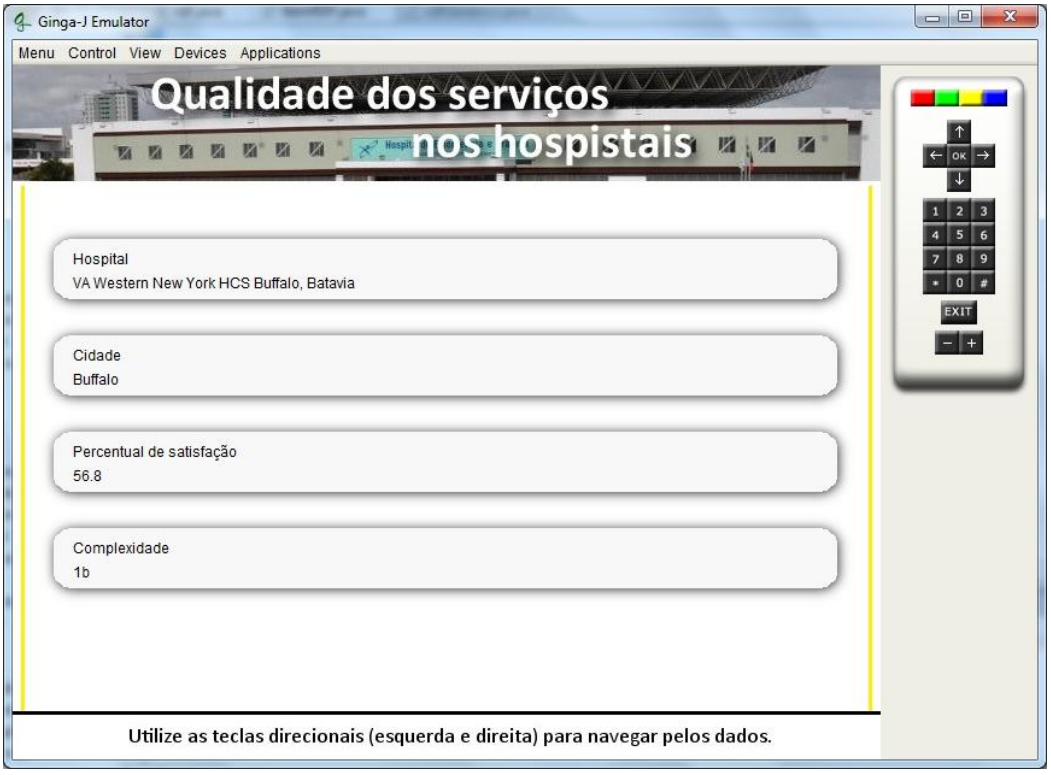

Figura 7a. Interface da aplicação Ap2 utilizando uma fonte RDF com dados relacionados à qualidade dos hospitais nos EUA.

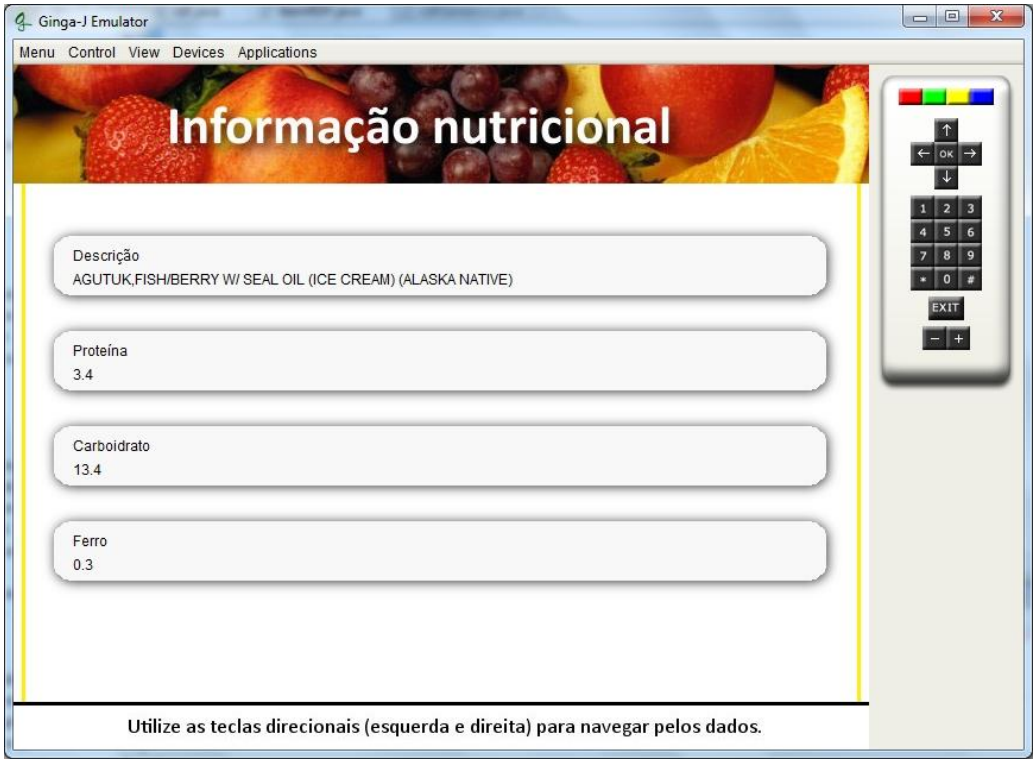

Figura 7b. Interface da aplicação Ap2 utilizando uma fonte RDF com dados relacionados às informações nutricionais dos alimentos.

4. Implementação

As explicações referentes à implementação a partir de agora expostas, serão baseadas na aplicação Ap2, uma vez que ela realiza as mesmas funções de Ap1, porém possui outras funcionalidades e segue a arquitetura proposta neste trabalho. Foi utilizada a IDE Eclipse para todos os códigos implementados.

O primeiro passo necessário para o desenvolvimento da aplicação exemplo Ap2 é baixar o emulador do Ginga-J no site do projeto GingaCDN. O emulador consiste em um projeto Java para o Eclipse com todos os arquivos fontes e bibliotecas necessárias para simular o ambiente de execução imperativo do Ginga.

Após baixar o emulador é necessário compilar o mesmo adicionando as configurações necessárias. Para tudo funcionar corretamente na criação da aplicação foi necessário seguir os passos explicados no tutorial [24] localizado no site do projeto GingaCDN.

Com o emulador compilado foi possível iniciar o desenvolvimento da aplicação exemplo. Para isso, foi criado um projeto no Eclipse seguindo as configurações padrões. O passo importante é adicionar às bibliotecas do projeto, a pasta "bin" situada no projeto do emulador compilado. Feito isso, todas as bibliotecas do emulador ficam disponíveis para utilização na aplicação.

A base da aplicação criada é a classe que implementa a interface *Xlet* presente na biblioteca "xjavax.tv.xlet". Nessa classe está presente todo o controle do ciclo de vida da aplicação no ambiente de execução do Ginga. Ela obrigatoriamente deve implementar os métodos *initXlet*, *startXlet*, *pauseXlet* e *destroyXlet*, que realizam as operações descritas em seus nomes e detalhadas mais a frente. Para completar os elementos básicos é necessário um objeto contexto do tipo *XletContext*, que é obrigatório em uma aplicação *Xlet* e que serve para isolar a mesma do resto da máquina de execução, além de um objeto do tipo *DTVContainer*, que serve como um container que armazena todos os componentes gráficos adicionados à aplicação. No caso da nossa aplicação exemplo, ao *DTVContainer* foi adicionado apenas um *Form* no qual foram adicionados todos os *Labels* utilizados. O esqueleto do código encontra-se a seguir e o código completo pode ser visto no Apêndice I.

```
public class aplicacao implements Xlet {

   private XletContext context;
   private DTVContainer dtvcontainer;


   public void initXlet(XletContext xletContext) throws
XletStateChangeException {
      context = xletContext;
   }
   public void startXlet() throws XletStateChangeException {
      //Captura a tela atual
      //Inicializa DTVContainer
      //Inicializa componentes gráficos
      //Le XML
      //Le RDF

      form.repaint();
   }

   public void pauseXlet() {
   }

   public void destroyXlet(boolean flag) throws
XletStateChangeException {
      context.notifyDestroyed();
   }
}
```

Conforme o código listado acima, o método *initXlet* recebe como parâmetro um objeto *XletContext* da máquina de execução e o mesmo é atribuído ao objeto *XletContext* da aplicação a ser utilizado. Além disso, outras operações de inicialização podem ser executadas nesse método.

O método *startXlet* é o responsável pela criação dos componentes gráficos e ligação dos mesmos com o contexto. A primeira ação tomada em seu escopo é a captura da tela atual, fornecida pela máquina de execução, que tem como objetivo fazer a ligação dos elementos da TV com os da aplicação. Após isso é inicializado o objeto *DTVContainer* assim como todos os outros componentes gráficos. No fim, os arquivos XML e RDF são lidos, o que será detalhado mais adiante.

Na aplicação exemplo não foi utilizado o método *pauseXlet*, porém sua utilização é muito útil quando é necessário realizar alguma ação na qual a aplicação deva ficar em segundo plano por algum momento. Por último, o método *destroyXlet* é responsável por notificar ao contexto que o objeto *Xlet* já pode ser descartado.

Para possibilitar a interação da aplicação com o controle remoto é necessário implementar a interface *UserInputEventListener*, presente na biblioteca "com.sun.dtv.ui.event". Para isso, foi adicionado o método *userInputEventReceived* que é

o responsável por captar a tecla que foi pressionada pelo usuário em seu controle remoto. Na aplicação exemplo A2, foram mapeadas ações para as teclas direcionais para cima e para baixo a fim de realizar a navegação pelos itens lidos da fonte de dados RDF, conforme o código listado abaixo.

```
public class aplicacao implements Xlet, UserInputEventListener
{

   //..

   public void userInputEventReceived(UserInputEvent
inputEvent) {
      com.sun.dtv.ui.event.KeyEvent event =
         (com.sun.dtv.ui.event.KeyEvent) inputEvent;

      if(event.getID() == KeyEvent.KEY_RELEASED) {
         return;
      }

      switch (event.getKeyCode()) {
         case KeyEvent.VK_LEFT: {
            //Executa naveção entre os itens para cima
            form.repaint();
            break;
         }
         case KeyEvent.VK_RIGHT: {
            //Executa naveção entre os itens para baixo

            form.repaint();
            break;
         }
      }
   }
}
```

4.1 Arquivo de configuração

Para a aplicação exemplo atingir o objetivo de ser genérica, funcionando para qualquer fonte de dados RDF passada, foi criado um arquivo XML responsável por definir alguns parâmetros. De acordo com a estrutura de A2 já mencionada, esse arquivo fica localizado em um servidor web e é acessado pela aplicação, via protocolo HTTP, no método *startXlet*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!--
   Arquivo de configuração para leitura de RDF
-->

<conteudoRDF>

   <caminho web="0">d:\\dataset-1458.rdf</caminho>

   <imagem
web="0">d:\\UFJF\\Monografia\\aplicacao\\img\\cabec1.jpg</image
m>

   <query><![CDATA[SELECT ?descricao ?proteina ?carboidrato
?ferro WHERE {?x
<http://www.data.gov/semantic/data/alpha/1458/dataset-
1458.rdf#shrt_desc>  ?descricao . ?x
<http://www.data.gov/semantic/data/alpha/1458/dataset-
1458.rdf#protein>  ?proteina . ?x
<http://www.data.gov/semantic/data/alpha/1458/dataset-
1458.rdf#carbohydrt> ?carboidrato . ?x
<http://www.data.gov/semantic/data/alpha/1458/dataset-
1458.rdf#iron> ?ferro } order by ?descricao]]></query>

   <campo label="Descrição">descricao</campo>
   <campo label="Proteína">proteina</campo>
   <campo label="Carboidrato">carboidrato</campo>
   <campo label="Ferro">ferro</campo>

</conteudoRDF>
```

De acordo com o código listado acima, o arquivo de configuração possui quatro parâmetros principais que são lidos pela aplicação. O rótulo *caminho* contém o parâmetro referente ao diretório onde o arquivo fonte RDF está localizado, caso o atributo *web* seja igual a zero, ou a URL de acesso ao arquivo, caso o atributo *web* seja diferente de zero.

O rótulo *imagem* é semelhante ao rótulo *caminho*, porém armazena o caminho da imagem utilizada como cabeçalho da aplicação e responsável por identificar o tema da fonte de dados. A imagem deve ter 720 pixels de largura e 100 pixels de altura, sendo redimensionada caso seja maior ou menor. Essa limitação se deve ao fato da aplicação ter o objetivo de generalizar a fonte de dados, com isso limitando o aspecto da interface da mesma. A adaptação da interface gráfica com as informações lidas pode ser mais elaborada, porém foge do escopo desse trabalho e pode motivar trabalhos futuros.

O rótulo *query* é responsável por passar o texto contendo a consulta SPARQL que será executada no arquivo RDF. Isso dá liberdade para a aplicação ler diferentes dados de um mesmo arquivo fonte, sendo necessário apenas alterar esse parâmetro em cada instância do programa.

Por último, o rótulo *campo* é o único que pode aparecer mais de uma vez no arquivo. Ele é responsável por passar para aplicação quais campos do arquivo RDF serão exibidos para o usuário de acordo com a *query* especificada. Então, para cada campo retornado na *query*, deve existir um rótulo *campo* no arquivo XML para a exibição da informação referente. Esse rótulo possui um atributo *label* que é utilizado para criação de um componente *Label* na aplicação com o texto passado como parâmetro. Esse componente tem como função identificar o campo que será exibido na tela. Por exemplo, se na *query* do arquivo XML estiver especificado o retorno de apenas um campo, na aplicação serão criados dois *Labels*, um contendo o texto do atributo *label* e outro logo abaixo que exibirá o dado vindo do arquivo RDF.

5.2 Acesso ao conteúdo

A ideia principal do trabalho quanto ao conteúdo, consiste em buscar informações governamentais e disponibilizá-las de forma adaptativa ao formato RDF que as contém. Uma aplicação importante é a apresentação de informações na área de saúde, permitindo assim que documentários e programas jornalísticos possam, através dessas arquiteturas, apresentar informações que complementem o conteúdo apresentado e ajudem a enriquecer o conhecimento do telespectador sobre o assunto, atingindo assim um grande conjunto de pessoas.

Como, no momento do desenvolvimento desse trabalho, não foram encontrados informações do governo brasileiro disponíveis em arquivos RDF, a solução foi utilizar a base de dados do site do governo dos EUA, o DataGov. Porém, no portal não são disponibilizados os links diretos para os arquivos, com isso foi necessário baixar os arquivos e armazená-los em um servidor web. O trecho de código listado abaixo exemplifica um dos arquivos RDF utilizados nos testes.

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    xmlns:dcterms="http://purl.org/dc/terms/"
    xmlns:dgtwc="http://data-gov.tw.rpi.edu/2009/data-gov-
twc.rdf#"

xmlns="http://www.data.gov/semantic/data/alpha/1290/dataset-
1290.rdf#"
```

```
xml:base="http://www.data.gov/semantic/data/alpha/1290/dataset-
1290.rdf" >

    <rdf:Description rdf:about="#entry1">
       <complexity>2</complexity>
       <st>ME</st>
       <station>402</station>
       <city>Togus</city>
       <facility_name>VAMC/RO</facility_name>
       <visn_name>VA New England Health Care System - VISN
1</visn_name>
       <visn>1</visn>
       <rdf:type rdf:resource="http://data-
gov.tw.rpi.edu/2009/data-gov-twc.rdf#DataEntry"/>
    </rdf:Description>

    ...

</rdf:RDF>
```

O método de leitura do arquivo RDF é muito simples, devido à utilização do framework Jena [21]. Conforme o código listado abaixo, um objeto do tipo *Model* lê o arquivo fonte e cria uma estrutura interna de representação do RDF, a qual fica disponível para a realização de consultas SPARQL. Um objeto do tipo *QueryExecution* recebe o texto da *query*, vindo do arquivo XML, e o objeto *Model* como parâmetros e fornece uma máquina de execução que retorna os resultados para posterior tratamento pela aplicação.

```
private void leRDF() {
   try {

      //Le o arquivo RDF
      InputStream in = fonteArquivo(rdfCaminho, rdfCaminhoWeb);
      Model model = ModelFactory.createDefaultModel();
      model.read(in,null);
      in.close();


      //Executa a query
      Query query = QueryFactory.create(rdfQuery);
      QueryExecution qe = QueryExecutionFactory.create(query,
model);
      ResultSet results = qe.execSelect();



      //Pega os resultados
      lstItem = new ArrayList<String>();
      String item = "";

      while(results.hasNext()) {
         //Processa os dados retornados
      }
```

```
      qe.close();
   } catch (FileNotFoundException e) {
      e.printStackTrace();
   } catch (IOException e) {
      e.printStackTrace();
   }
}
```

5. Considerações finais e trabalhos futuros

O sistema brasileiro é, atualmente, um dos mais avançados sistemas de TV Digital terrestre do mundo [8], não apenas por usar as tecnologias mais avançadas, mas, principalmente, por dispor de tecnologias inovadoras, como é o caso do *middleware* Ginga. Com base nisso, pode-se concluir que a área de desenvolvimento de aplicações para TV Digital é muito promissora, porém, quanto à máquina de execução Java, o Ginga-J, o ambiente atual de desenvolvimento ainda se encontra prematuro.

A implementação da aplicação Ap2 viabiliza, mesmo com o precário cenário atual de desenvolvimento Java para o *middleware* Ginga-J, a arquitetura proposta com um método flexível de acesso ao conteúdo, que pode ser qualquer arquivo RDF localizado em um servidor web. Além disso, permite a exibição de informações para o telespectador, possibilitando o mesmo navegar pelos dados visualizados.

Como trabalhos futuros, existem as possibilidades de uma maior exploração dos aspectos gráficos de aplicações Java para TV Digital e também uma melhor integração da interface com os tipos de dados recebidos. Além disso, pode-se estudar a possibilidade de exploração de outros tipos de fontes de dados para enriquecer a gama de informações disponibilizadas assim como propor novas arquiteturas para que uma aplicação Java permita a interatividade completa do usuário com os programas televisivos.

Referências bibliográficas

[1] Especificação RDF, W3C. Disponível na Internet em: http://www.w3.org/RDF/, 03 de Julho, 2011.

[2] DOU (2003). *Decreto 4.901, de 16 de novembro de 2003, que institui o Sistema Brasileiro de TV Digital e dá outras providências*. Brasília, 28 de novembro de 2003.

[3] Becker, V., Filho, G. H. H., and Piccioni, C. A. (2006). *Inclusão digital através de serviços de saúde na TV digital interativa*. II Workshop de TV Digital, 2006, Curitiba. Anais do WTVD 2006 - II Workshop de TV Digital.

[4] Ginga. Disponível na Internet em: http://www.ginga.org.br/, 27 de Junho, 2011.

[5] Barbosa, S. D. J., Soares, L. F. G. (2008). *TV Digital Interativa no Brasil se faz com Ginga: Fundamentos, Padrões, Autoria Declarativa e Usabilidade*. Em T. Kowaltowski & K. Breitman (orgs.) *Atualizações em Informática 2008*. Rio de Janeiro, RJ: Editora PUC-Rio, 2008. pp. 105-174.

[6] Barrere, E. and Silva, C. K. P. E. (2010). *Acesso ao Moodle via Aplicação para TV Digital*. Moodle Moot Brasil, 2010, São Paulo. Moodle Moot Brasil 2010.

[7] Margalho, M., Francês, R., and Costa, J. C. W. A. (2007). *Canal de Retorno para TV Digital com Interatividade Condicionada por Mecanismo de Sinalização Contínua e Provisionamento de Banda Orientado a QoS.* IEEE LATIN AMERICA TRANSACTIONS, volume 5, number 5, september.

[8] Soares, L. F. G. (2009). *TV Interativa se Faz com Ginga*. Revista da SET-Sociedade Brasileira de Engenharia de Televisão, 2009. p. 30 - 35.

[9] Portal da linguagem NCL. Disponível na Internet em: http://www.ncl.org.br/, 20 de Outubro, 2011.

[10] Fórum do Sistema Brasileiro de TV Digital Terrestre. Disponível na Internet em: http://www.forumsbtvd.org.br, 12 de setembro, 2011.

[11] Fórum do Sistema Brasileiro de TV Digital Terrestre, *Módulo Técnico - JavaDTV*. Disponível na Internet em: http://www.forumsbtvd.org.br/materias.asp?id=200, 13 de setembro, 2011.

[12] Projeto GingaCDN. Disponível na Internet em: http://gingacdn.lavid.ufpb.br/, 05 de Agosto, 2011.

[13] Projeto GingaCDN, *Emulador Ginga-J*. Disponível na Internet em: http://gingacdn.lavid.ufpb.br/projects/gingaj-emulator, 05 de Agosto, 2011.

[14] Saraiva Júnior, E. G. *Arquitetura de APIs Gráficas do JavaDTV*. Disponível na Internet em: http://gingacdn.lavid.ufpb.br/projects/gingaj-emulator/wiki/Arquitetura_APIs_Graficas, 02 de Outubro, 2011.

[15] Santos, D. and Schiel, U. (2002). *RDF na Interoperabilidade entre Sistemas na Web. Simpósio Brasileiro de Engenharia de Software*, 16., 2002, Gramado, RS. Anais. Porto Alegre, RS.

[16] W3C, *SPARQL Query Language for RDF*. Disponível na Internet em: http://www.w3.org/TR/rdf-sparql-query/, 15 de Setembro, 2011.

[17] DataPrev, *Dados Abertos*. Disponível na Internet em: http://api.dataprev.gov.br/doc/dadosDisp.htm, 23 de Novembro, 2011.

[18] Portal Governo Aberto do Estado de São Paulo. Disponível na Internet em: http://www.governoaberto.sp.gov.br/view/, 23 de Novembro, 2011.

[19] DataGov, Governo dos EUA. Disponível na Internet em: http://www.data.gov/semantic/ data/alpha/page/1/count/100, 10 de Agosto, 2011.

[20] Fórum do Sistema Brasileiro de TV Digital Terrestre. Disponível na Internet em: http://www.forumsbtvd.org.br/materias.asp?id=737, 18 de Novembro, 2011.

[21] Framework Jena. Disponível na Internet em: http://jena.sourceforge.net/, 06 de Agosto, 2011.

[22] Documentação DOM. Disponível na Internet em: http://download.oracle.com/javase/1.4.2/docs/api/org/w3c/dom/package-summary.html, 26 de Novembro, 2011.

[23] Portal do Software Público Brasileiro. Disponível na Internet em: http://www.softwarepublico.gov.br/, 14 de Setembro, 2011.

[24] GingaCDN, Compilação no Emulador no Eclipse. Disponível na Internet em: http://gingacdn.lavid.ufpb.br/projects/gingaj-emulator/wiki/Compila%C3%A7%C3%A3o_do_Emulador_no_Eclipse, 12 de Julho, 2011.

Apêndice I – Código fonte completo da aplicação Ap2

```
import com.sun.dtv.lwuit.*;
import com.sun.dtv.lwuit.geom.Dimension;
import com.sun.dtv.resources.ScarceResource;
import com.sun.dtv.resources.ScarceResourceListener;
import com.sun.dtv.ui.Capabilities;
import com.sun.dtv.ui.DTVContainer;
import com.sun.dtv.ui.Device;
import com.sun.dtv.ui.Plane;
import com.sun.dtv.ui.PlaneSetup;
import com.sun.dtv.ui.Screen;
import com.sun.dtv.ui.event.KeyEvent;
import com.sun.dtv.ui.event.RemoteControlEvent;
import com.sun.dtv.ui.event.UserInputEvent;
import com.sun.dtv.ui.event.UserInputEventListener;
import com.sun.dtv.ui.event.UserInputEventManager;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import xjava.lang.System;
import xjavax.tv.xlet.Xlet;
import xjavax.tv.xlet.XletContext;
import xjavax.tv.xlet.XletStateChangeException;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.net.URLConnection;
import java.util.ArrayList;
import com.hp.hpl.jena.query.Query;
import com.hp.hpl.jena.query.QueryExecution;
import com.hp.hpl.jena.query.QueryExecutionFactory;
import com.hp.hpl.jena.query.QueryFactory;
import com.hp.hpl.jena.query.QuerySolution;
import com.hp.hpl.jena.query.ResultSet;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;

public class rdfGenerico implements Xlet, UserInputEventListener,
     ScarceResourceListener {

     private XletContext context;
     private DTVContainer dtvcontainer;
     private Plane plane;
     private PlaneSetup planeSetup;
     private Form form;
     private Label lblImgCabecalho;
     private Label lblImgRodape;
     private ArrayList<Label> lstLabel;

     private ArrayList<String> lstItem;
     private int itemAtual;

     //Propriedades do XML
     private String rdfCaminho;
     private boolean rdfCaminhoWeb;
     private String rdfImgCaminho;
     private boolean rdfImgWeb;
     private String rdfQuery;
```

```
private ArrayList<String> rdfLstCampo;
private ArrayList<String> rdfLstLabel;

public rdfGenerico() {
}

private InputStream fonteArquivo(String caminho, boolean web) throws
IOException {
 if (web) {
        URL arq = new URL(caminho);
        URLConnection uc = arq.openConnection();
        return uc.getInputStream();
 } else {
        return new FileInputStream(new File(caminho));
 }
}

private void leXML() {
 try {
        DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();
        DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
        Document doc =
dBuilder.parse(fonteArquivo("D:\\UFJF\\Monografia\\appConfig.xml", false));
        doc.getDocumentElement().normalize();

        //Nó caminho
        NodeList nList = doc.getElementsByTagName("caminho");
        Node n = nList.item(0);
        rdfCaminho = n.getTextContent();
        if (rdfCaminhoWeb =
n.getAttributes().getNamedItem("web").getTextContent().equals("1"))
rdfCaminhoWeb = true;

        //Nó imagem
        nList = doc.getElementsByTagName("imagem");
        n = nList.item(0);
        rdfImgCaminho = n.getTextContent();
        if
(n.getAttributes().getNamedItem("web").getTextContent().equals("1"))
rdfImgWeb = true;

        //Nó query
        nList = doc.getElementsByTagName("query");
        n = nList.item(0);
        rdfQuery = n.getTextContent();

        //Nós campos
        rdfLstCampo = new ArrayList<String>();
        rdfLstLabel = new ArrayList<String>();
        nList = doc.getElementsByTagName("campo");

        for (int i = 0; i < nList.getLength(); i++) {
               n = nList.item(i);
               rdfLstCampo.add(n.getTextContent());

 rdfLstLabel.add(n.getAttributes().getNamedItem("label").getTextContent());
        }

 } catch (Exception e) {
        System.out.println(e.getMessage());
 }
}

private void leRDF() {
 try {
        //Le o arquivo RDF
```

```
            InputStream in = fonteArquivo(rdfCaminho, rdfCaminhoWeb);

            Model model = ModelFactory.createDefaultModel();
            model.read(in,null);
            in.close();

            Query query = QueryFactory.create(rdfQuery);

            //Executa a query
            QueryExecution qe = QueryExecutionFactory.create(query, model);
            ResultSet results = qe.execSelect();

            //Pega os resultados
            lstItem = new ArrayList<String>();
            String item = "";

            while(results.hasNext()) {
                  QuerySolution qs = results.next();

                  item = "";

                  for (int i = 0; i < rdfLstCampo.size(); i++) {
                        item += qs.get(rdfLstCampo.get(i)).toString();

                        if (i < rdfLstCampo.size()-1)  item += "#&#";
                  }

                  lstItem.add(item);
            }

            qe.close();

     } catch (FileNotFoundException e) {
            e.printStackTrace();
     } catch (IOException e) {
            e.printStackTrace();
     }
    }

    private void addLabel(Label lbl, String img, boolean imgWeb, int x, int y,
    int width, int height){
     lbl.setX(x);
     lbl.setY(y);
     lbl.setSize(new Dimension(width,height));

     if (img != "") {
            try {
                  lbl.setIcon(Image.createImage(fonteArquivo(img, imgWeb)));
            } catch (IOException e) {
                  System.out.println("Erro ao carregar a imagem: " +  img);
            }
     }
    }

    public void initXlet(XletContext xletContext) throws
    XletStateChangeException {
       context = xletContext;
   }

    public void startXlet() throws XletStateChangeException {
     Device device = Device.getInstance();
     Screen currentScreen = device.getDefaultScreen();
     UserInputEventManager manager =
    UserInputEventManager.getUserInputEventManager(currentScreen);

     RemoteControlEvent anyColoredKeyTyped = new RemoteControlEvent(null,
    com.sun.dtv.ui.event.KeyEvent.KEY_TYPED, 0, 0,
```

```
RemoteControlEvent.VK_COLORED,
com.sun.dtv.ui.event.KeyEvent.CHAR_UNDEFINED);

 try {
       anyColoredKeyTyped.reserve(true, -1, null);
 } catch (Exception e) {
       e.printStackTrace();
 }

 manager.addUserInputEventListener(this, anyColoredKeyTyped);

 Plane[] planes = currentScreen.getAllPlanes();

 for(int i=0; i<planes.length; i++) {
       Capabilities cap = planes[i].getCapabilities();
       if (cap.isGraphicsRenderingSupported()) {
             plane = planes[i];
             planeSetup = plane.getCurrentSetup();
             break;
       }
 }

 try {
       plane.reserve(false,-1,this);
 } catch (Exception e) {
       e.printStackTrace();
 }

 //Inicializa componentes
 //Form
 this.form = new Form();
 form.setLayout(null);
 form.getContentPane().setLayout(null);
 form.setSize(planeSetup.getScreenResolution());

 //DtvContainer
 dtvcontainer = DTVContainer.getDTVContainer(plane);
 dtvcontainer.setLayout(null);
 dtvcontainer.setSize(planeSetup.getScreenResolution());
 dtvcontainer.addComponent(form);
 dtvcontainer.setVisible(true);

 leXML();

 //lblImgCabecalho
 lblImgCabecalho = new Label();
 addLabel(lblImgCabecalho, rdfImgCaminho, rdfImgWeb, 0, 0, 720, 100);
 form.addComponent(lblImgCabecalho);

 //lblImgRodape
 lblImgRodape = new Label();
 addLabel(lblImgRodape, "img/instrucao.jpg", false, 0, form.getHeight()-40,
720, 40);
 form.addComponent(lblImgRodape);

 leRDF();
 itemAtual = 0;

 //Cria os labels
 int posY = 150;
 lstLabel = new ArrayList<Label>();
 Label lblFundo, lbl, lblCampo;
 String[] arrValor = lstItem.get(itemAtual).split("#&#");

 for (int i = 0; i < rdfLstCampo.size(); i++) {
       lblFundo = new Label();
```

```
        addLabel(lblFundo, "img/fundoGenerico.png", false, 20, posY-20, 680,
 80);
        form.addComponent(lblFundo);

        lbl = new Label(rdfLstLabel.get(i));
        addLabel(lbl, "", false, 50, posY, 700, 20);
        form.addComponent(lbl);

        posY += 20;

        lblCampo = new Label(arrValor[i]);
        addLabel(lblCampo, "", false, 50, posY, 700, 20);
        form.addComponent(lblCampo);
        lstLabel.add(lblCampo);

        posY += 60;
  }

  form.repaint();
 }

 public void pauseXlet() {
 }

 public void destroyXlet(boolean flag) throws XletStateChangeException {
    context.notifyDestroyed();
}

 public void userInputEventReceived(UserInputEvent inputEvent) {

  com.sun.dtv.ui.event.KeyEvent event = (com.sun.dtv.ui.event.KeyEvent)
 inputEvent;

  if(event.getID() == KeyEvent.KEY_RELEASED) {
        return;
  }

  String[] arrValor;

  switch (event.getKeyCode()) {
        case KeyEvent.VK_LEFT: {
              if (itemAtual > 0) {
                    itemAtual--;
                    arrValor = lstItem.get(itemAtual).split("#&#");

                    for (int i = 0; i < rdfLstCampo.size(); i++) {

                          lstLabel.get(i).setText(arrValor[i]);

                    }

                    form.repaint();
              }
              break;
        }
        case KeyEvent.VK_RIGHT: {
              if (itemAtual < lstItem.size() - 1) {
                    itemAtual++;
                    arrValor = lstItem.get(itemAtual).split("#&#");
                    for (int i = 0; i < rdfLstCampo.size(); i++) {
                          lstLabel.get(i).setText(arrValor[i]);

                    }

                    form.repaint();
              }
              break;
```

```
            }
        }

    }

    public boolean releaseRequested(ScarceResource resource) {
     return false;
    }

    public void releaseForced(ScarceResource resource) {
    }

    public void released(ScarceResource resource) {
    }
}
```