Universidade Federal do Maranhão
Centro de Ciências Exatas e Tecnologia
Curso de Graduação em Ciência da Computação

Ricardo Lucio Braga Reis

Convergência entre Aplicativos de TV Digital Interativa e a Web: uma arquitetura integradora

São Luís
2013

Ricardo Lucio Braga Reis

Convergência entre Aplicativos de TV Digital Interativa e a Web: uma arquitetura integradora

Monografia apresentada ao Curso de Ciência da Computação da UFMA, como requisito parcial para a obtenção do grau de BACHAREL em Ciência da Computação.

Orientador: Mário Antonio Meireles Teixeira

Prof. Dr. em Ciência da Computação - USP

São Luís

2013

Reis, Ricardo Lucio Braga.

Convergência entre aplicativos de TV digital interativa e a WEB: uma arquitetura integradora/ Ricardo Lucio Braga Reis. – São Luís, 2013.

51 f.

Impresso por computador (fotocópia).
Orientador: Mário Antonio Meireles Teixeira.

Monografia (Graduação) – Universidade Federal do Maranhão, Curso de Ciência da Computação, 2013.

1. TV digital interativa - WEB. 2. Aplicações. 3. Ginga. I. Título.

CDU 621.398: 654.93: 004.272.45

Ricardo Lucio Braga Reis

Convergência entre Aplicativos de TV Digital Interativa e a Web: uma arquitetura integradora

Monografia apresentada ao Curso de Ciência da Computação da UFMA, como requisito parcial para a obtenção do grau de BACHAREL em Ciência da Computação.

Aprovado em 01 / 08 / 2013

BANCA EXAMINADORA

Mário Antonio Meireles Teixeira

Prof. Dr. em Ciência da Computação - USP

Geraldo Braz Junior

Mestre em Engenharia de Eletricidade - UFMA

Maria Auxiliadora Freire

Mestre em Ciência de Engenharia - PUC-Rio

Dedico este trabalho de conclusão de curso aos meus pais, José Lucio Campos Reis e Elisabete Carvalho Braga Reis.

Agradecimentos

A Deus, pela oportunidade de ter chegado aqui, sempre intercedendo por mim em toda a caminhada. Aos meus pais, José Lucio Campos Reis e Elisabete Carvalho Braga Reis, por todo o amor, ensinamentos, apoio e por terem me proporcionado a melhor educação possível. A minha irmã Luciana Braga Reis, por todo incentivo e torcida durante essa trajetória. A minha namorada Ana Carolina Oliveira de Castro Moreira, pela compreensão, incentivo e auxílio constantes. Ao meu orientador, Professor Dr. Mário Antonio Meireles Teixeira, por todos os ensinamentos e apoio. Aos meus amigos, que direta ou indiretamente me ajudaram na realização do trabalho.

Resumo

O surgimento da TV Digital Interativa trouxe novas possibilidades de interação entre os telespectadores e a programação de TV. Há uma clara tendência de convergência entre TV e web. Um dos desafios neste cenário é fornecer soluções para auxiliar o desenvolvimento de aplicações nesta nova mídia. Este trabalho define e implementa uma arquitetura *open source* baseado no protocolo XMPP que simplifica o acesso de *widgets* NCL a conteúdo da web, através do modelo *publish/subscribe* e também aplicações interativas multi-usuário. Essa arquitetura minimiza os tempos de desenvolvimento e implantação de aplicações NCLua que usam o canal de interatividade da plataforma Ginga-NCL servindo como base para um espectro completo de *widgets* Ginga com acesso a conteúdo web.

Palavras-chave: TV Digital Interativa, Aplicações, Ginga, NCL, Web, XMPP.

Abstract

The emergence of Interactive Digital TV brought in novel possibilities of interaction between TV viewers and TV programming. There is a clear tendency of convergence between TV and the web. One of the challenges in this scenario is to provide solutions to assist application development in this new media. This work defines and implements an open-source architecture based in the XMPP protocol that simplifies the access of NCL widgets to web content by supporting the publish/subscribe model and also interactive multi-user applications. This architecture minimizes the development and deployment time of NCLua applications that use Ginga-NCL's platform interactivity channel thus serving as the basis to a thorough spectrum of web-enabled Ginga widgets.

Keywords: Interactive Digital TV, Applications, Ginga, NCL, Web, XMPP.

"A Simplicidade é a sofisticação máxima."

(Leonardo da Vinci)

Sumário

Lista de Figuras

1 INTRODUÇÃO

Desde o início da TV, as emissoras tentam proporcionar interação com o telespectador através das mais diversas mídias. No início da TV analógica, essa interação acontecia através de cartas, forma mais tradicional de comunicação. A interatividade acompanhou o avanço tecnológico e logo passou para o telefone, seguida das mensagens SMS, e-mails e chats. No cenário atual, com o advento da TV Digital e a convergência entre tecnologias, tornou-se inevitável o surgimento de um elo entre web e TV.

Com a expansão da internet, a web passou a ser entendida como uma plataforma (Web 2.0), onde ao invés de software, se desenvolvem serviços. Mais dinâmica e colaborativa nesse novo conceito, a Web 2.0 tornou a publicação de conteúdo flexível. Através de serviços como wikis, blogs ou redes sociais, qualquer pessoa gera conteúdo sem necessidade de requisitos técnicos específicos.

No Brasil, a TV está presente em mais de 95% dos lares [12], enquanto a internet e a web ainda não atingem, em algumas regiões, nem 20% das residências. Por outro lado, observa-se que o Brasil é o quinto país do mundo onde mais se utiliza redes sociais [13]. Ademais, serviços de governo eletrônico como previdência social, imposto de renda e serviços de saúde, além de bancos e lojas virtuais são acessados por um número cada vez mais significativo de brasileiros, mesmo entre aqueles que não possuem internet em suas casas. Espera-se que, nos próximos anos, com a disseminação da plataforma Ginga de TV digital interativa, a TV passe a se constituir numa opção popular para acesso a internet, substituindo em alguns casos os computadores. Nesse cenário, torna-se fundamental a existência de ferramentas e arquiteturas que permitam o rápido e eficiente desenvolvimento de aplicativos de TVDI com acesso a serviços disponibilizados através da web.

Este trabalho tem como objetivo propor uma arquitetura genérica e escalável para acesso a conteúdo web a partir de aplicativos de TVDI, particularmente para o caso de widgets NCLua. Genérica a fim de permitir o acesso a qualquer conteúdo web disponível, de maneira transparente, a partir da plataforma Ginga-NCL [14], isto é, sem que o desenvolvedor da aplicação NCLua tenha que produzir código específico para cuidar

de detalhes de comunicação, dentre outros. A proposta também deve ser escalável, a fim de suportar múltiplos acessos e clientes concorrentes, sem sobrecarregar o ambiente Ginga-NCL do decodificador.

Deseja-se, inicialmente, permitir acessos do tipo Publish/Subscribe, em que um cliente NCLua declara seu interesse em consumir informações fornecidas periodicamente por um dado site. Além disso, pretende-se também suportar aplicações de interação entre usuários, sejam baseadas em mensagens instantâneas ou as novas redes sociais. Assim, será possível contemplar uma ampla gama das aplicações comuns da web a partir de widgets de TVDI.

A arquitetura proposta baseia-se em padrões abertos, como o protocolo XMPP, amplamente utilizado em serviços atuais da web, o que garante sua interoperabilidade com diversas soluções já existentes. Além disso, a opção pelo XMPP torna a arquitetura escalável, capaz de suportar diversos clientes simultâneos. No seu desenvolvimento, foram observados e utilizados design patterns catalogados e produzido também um framework de comunicação em Lua. Com isso, conseguiu-se uma implementação mais organizada e oportunizou-se o reuso de código. Por fim, foram desenvolvidas algumas aplicações a fim de validar a arquitetura proposta.

Este trabalho está organizado como segue. O capítulo 2 traz uma visão geral da plataforma Ginga de TVDI e o funcionamento do portocolo XMPP. O capítulo 3 especifica a arquitetura proposta neste trabalho, bem como o framework de comunicação desenvolvido. O capítulo 4 apresenta estudos de casos que validam a proposta e, finalmente, o capítulo 5 discute as conclusões e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 A Plataforma Ginga de TVDI

2.1.1 Ginga

Com o advento da TV Digital interativa no Brasil, ganha-se não somente em qualidade de vídeo e áudio, mas também, como o próprio nome já diz, em interatividade. Os telespectadores poderão interagir com a programação de TV recebida em quatro níveis: interatividade local, unidirecional, bidirecional assimétrico e interatividade plena.

A *interatividade local* refere-se à interação apenas com os dados transmitidos por difusão, ou seja, essa categoria opera sem o canal de retorno. Um exemplo de aplicação desse nível é o EPG (*Eletronic Programming Guide*). A interação pode também ser *unidirecional*, onde o telespectador apenas envia dados à emissora, como numa votação, por exemplo. Já no nível *bidirecional assimétrico* pode-se obter dados não provenientes da difusão, como a navegação web. Por fim, na *interatividade plena* o telespectador já é capaz de fazer upload de dados, tornando-se assim uma pequena emissora. Esse nível permite o desenvolvimento de aplicações sociais que tornam possível a interação e compartilhamento de conteúdo entre telespectadores de uma determinada programação. Essa categoria de aplicações vem sendo chamada de *Social TV* [11].

Para que essa interatividade ocorra no Sistema Brasileiro de TV Digital (SBTVD) é necessário uma camada de *software* que torne as aplicações interativas independentes da plataforma de *hardware* e *software* de um fabricante de receptor específico, essa camada é chamada de *middleware* [10]. O *middleware* do padrão brasileiro de tv digital é o Ginga.

Arquitetura

O *middleware* Ginga é dividido em 3 subsistemas como mostra a sua arquitetura de referência na Figura 2.1: Ginga-NCL, Ginga-J e Ginga-CC. Ao Ginga-NCL é atribuída a tarefa de processar documetos escritos com a linguagem declarativa NCL, e

para tarefas que requerem algoritmos, a linguagem de *script* Lua. Já o Ginga-J processa aplicações imperativas escritas com a linguagem Java.

O Ginga-CC reune todas as funcionalidades comuns de suporte aos ambientes declarativos (Ginga-NCL) e imperativo (Ginga-J). Entre as principais funções do Ginga-CC estão: tratar da exibição dos vários objetos de mídia que fazem parte de uma aplicação, como JPEG, MPEG-4, MP3 entre outros, controle gráfico, controle para obtenção dos dados transmitidos por difusão (broadcast) e pelo Canal de Interatividade (ou canal de retorno).

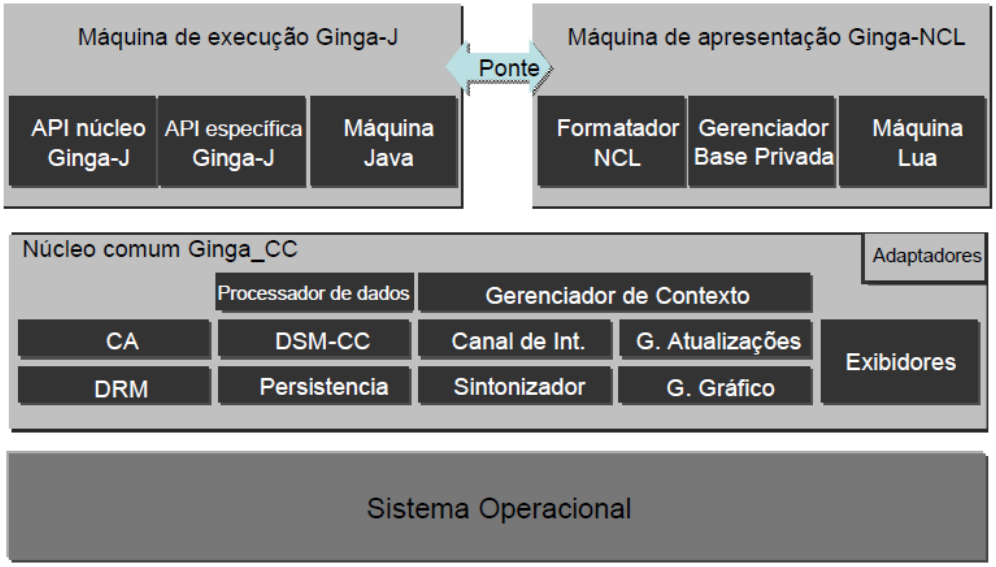

Figura 2.1: Arquitetura do *middleware* Ginga.

O Ginga-NCL por sua vez também é subdividido em três componentes: o formatador NCL, o gerenciado de base privada e a máquina Lua. O formatador NCL tem a tarefa de controlar toda a apresentação de um documento NCL, assegurando assim que as relações especificadas entre os objetos de mídia sejam cumpridas. Cada documento NCL é chamado de base privada e o gerenciador é responsável por receber e executar comandos de edição de documentos NCL, incluindo edições ao vivo [20]. Já a máquina Lua tem como objetivo executar os *scripts* escritos na linguagem Lua.

O canal de interatividade do Ginga é habilitado para o protocolo TCP. Através desse canal pode-se implementar aplicações distribuídas, como as de passagem de mensagem por exemplo, que atualmente são um meio de comunicação bastante utilizado e difundido. O Ginga oferece, então, mais uma forma de pessoas interagirem, pois, além

do computador, elas poderão se relacionar também através da TV, e não apenas com aplicativos de rede consagrados como GTalk e Skype, mas também por meio de mídias sociais, como o popular Facebook ou o Twitter.

2.1.2 NCL

A NCL é a linguagem declarativa do padrão brasileiro de TV Digital para o desenvolvimento de aplicações multimídia com sincronismo espaço-temporal entre objetos de mídia como imagens, vídeos, áudio. Ou seja, ela define como as mídias são estruturadas e relacionadas no tempo e espaço [10].

Um documento Hipermídia é composto basicamente por nós e elos. Os nós são as abstrações de mídias utilizadas no documento, podendo trazer informações adicionais sobre como a mídia será apresentada. Os elos são responsáveis pela sincronização dos nós no espaço e no tempo.

A linguagem NCL é baseada no modelo conceitual NCM (*Nested Context Model*), ou Modelo de Contextos Aninhados, que estende os conceitos acima aumentando a flexibilidade dos documentos hipermídia. Basicamente ele perimite que os nós sejam ou de conteúdo ou de composição. Os nós de conteúdo (nós de mídias) representam os objetos de mídia (vídeo, áudio, imagem, texto e etc). Os nós de composição são utilizados para dar estrutura a um documento hipermídia, podendo conter um conjunto de nós de conteúdo, outros nós de composição e um conjunto de elos.

Estrutura de um Documento NCL

Como pode ser visto na Figura 2.2, um documento NCL utiliza de marcações para declarar quais mídias serão exibidas, onde se posicionarão, como serão exibidas e quando se relacionarão com outras mídias.

As *tags ncl*, *head* e *body* definem a estrutura de um documento e são de uso obrigatório. Dentro do elemento *head* define-se onde as mídias serão posicionadas, como serão exibidades e os seus relacionamentos. Essas tarefas são atribuições do *regionBase*, *descriptorBase* e *connectorBase* respectivamente. É importante resaltar que essas bases não precisam obrigatóriamente estar dentro do documento principal. Existem ainda outras bases não mostradas na Figura 2.2, a *ruleBase* (Base de Regras) e *(transitionBase*

(Base de transições).

No elemento *body* declaram-se os contextos da aplicação, os *links*, que informam quando as mídias se relacionarão e quais mídias serão usadas, juntamente com suas portas. Os contextos englobam *links*, mídias e portas. São usados para organizar uma aplicação e são chamados quando necessários através de uma porta. As portas são o acesso da aplicação às mídias e contextos. Os *links* utilizam regras como *onSelection*, *onPause*... para informar a aplicação quando as mídias se relacionarão.


```
<ncl id="main" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
    <head>

        <regionBase>
            <region id="rgVideo" width="100%" height="100%" >
                <region id="regBotao" width="10" height="10"/>
            </region>
        </regionBase>

        <descriptorBase>
            <descriptor id="deVideo" region="rgVideo" focusIndex="1" />
            <descriptor id="deBotao" region="regBotao" focusIndex="2"/>
        </descriptorBase>

        <connectorBase>
            <causalConnector id="onSelectionPause">
                <simpleCondition role="onSelection" key="RED"/>
                <simpleAction role="pause"/>
            </causalConnector>
        </connectorBase>

    </head>

    <body>

        <port id="ptIntro" component="ctxIntro"/>

        <context id="ctxIntro">

            <port id="ptVideo" component="video"/>
            <media id="video" src="video.avi" descriptor="deVideo" />

            <port id="ptBotao" component="botao"/>
            <media id="botao" src="pause.png" descriptor="deBotao" />

            <link xconnector="onSelectionPause">
                <bind role="onSelection" component="botao"/>
                <bind role="pause" component="video" />
            </link>

        </context>

    </body>
</ncl>
```

Figura 2.2: Exemplo de um documento NCL. [16]

2.1.3 Objetos Imperativos Lua em NCL (NCLua)

A linguagem Lua pode ser integrada à linguagem NCL através de *scripts* NCLua. Os *scripts* possuem a extensão .lua e são adicionados à aplicação NCL como

uma mídia, ou seja, os *scripts* NCLua são subordinados ao NCL. Eles são chamados pelo NCL para fazer alguma computação, depois param e retornam um resultado.

Os *scripts* NCLua possuem a mesma sintaxe do Lua e bibliotecas similares, algumas foram desenvolvidas especialmente para o contexto da TV Digital. Esses módulos especias estão na API Lua descritos na Norma ABNT NBR 15606-2:2007 (ABNT, 2007).

Os 4 módulos especiais são descritos abaixo:

- **Canvas**: Oferece uma API para desenhar objetos gráficos e imagens na região do NCLua.
- **Event**: Permite que objetos NCLua se comunique com o documento NCL.
- **Settings**: Fornece acesso às variáveis definidas no objeto *settings* do documento NCL.
- **Persistent**: Exporta uma tabela com variáveis persistentes disponíveis aos *scripts* NCLua.

Comunicação Entre NCL e Lua

No *middleware* Ginga a comunicação entre o documento NCL e *scripts* NCLua é feita através do paradigma de programação orientada a eventos, ou seja, toda a comunicação é feita através da difusão e recepção de eventos. Para isso é necessário o correto entendimento do módulo *event* já que ele é essencial no envio e recebimento de eventos.

A Figura 2.3 mostra o *script* NCLua ao centro e as várias entidades como as quais ele pode se comunicar. Para se comunicar uma entidade deve enviar um evento para a fila do *script* onde aguardará até sua vez, pois ele trata apenas 1 por vez. Quando chegar a sua vez o evento será redirecionado ao seu respectivo tratador, definido pelo programador do *script*. O *script* também pode se comunicar com as entidades postando eventos para elas, como podemos conferir na imagem.

É função do programador implementar os tratadores de eventos, porém para ser informado quando eventos são recebido o programador não deve se esquecer de registrar os tratadores através de uma chamada à função *event.register*(tratador).

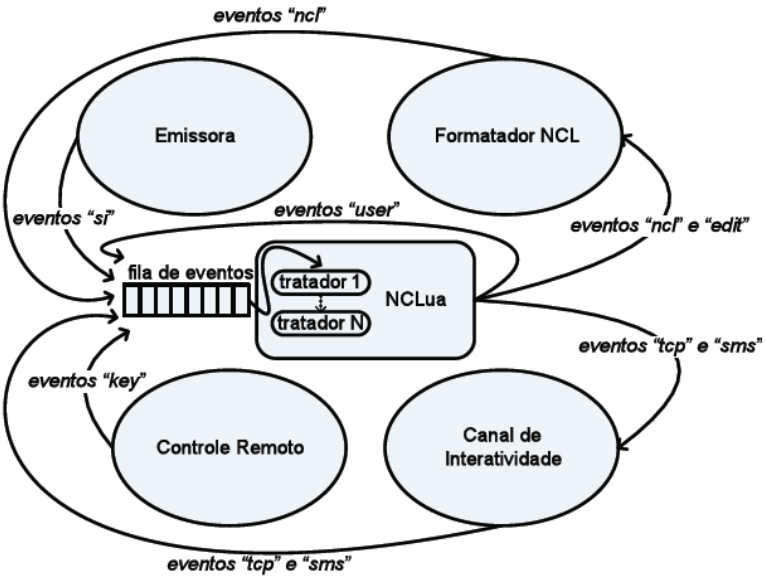

Figura 2.3: Paradigma de programação orientada a eventos. [17]

Eventos são descritos por tabelas Lua com valores descrevendo atributos. Na Figura 2.4 pode-se ver um exemplo onde o evento indica que o botão verde do controle remoto foi pressionado pelo telespectador.

```
evt = {
        class = 'key',
        type  = 'press',
        key   = 'green'
}
```

Figura 2.4: Exemplo de um evento.

No âmbito do *framework* desenvolvido, utiliza-se o módulo *event* pois como dito anteriormente o Lua é subordinado ao NCL, e o processamento é todo feito no Lua, já que o NCL não foi feito para esse fim. Necessita-se que o formatador NCL gere um evento ao se acionar o botão para chamar o aplicativo, a partir daí o Lua faz o resto, desde conectar-se ao servidor XMPP até a exibição do *widget* na tela do telespectador.

O módulo *canvas* oferece uma API para desenhar objetos gráficos na região do NCLua e o *framework* utiliza esse artifício para exibir o *widget* na tela. O *widget* em si nada mais é do que uma imagem, e através de uma posição (x, y) passada pelo desenvolvedor desenha-se ele e os títulos de notícias recebidos.

A viabilidade deste trabalho deu-se pelo fato do Ginga oferecer o canal de interatividade (ou canal de retorno) que permite o desenvolvimento de aplicações distribuídas. Utilizando-se uma biblioteca TCP para o Lua desenvolveu-se toda a parte de comunicação permitindo assim ao *framework* a conexão com um servidor. Pode-se observar na Figura 2.5 que basicamente utilizam-se as primitivas *tcp.connect()* para conectar-se ao servidor, *tcp.send()* para enviar uma mensagem e *tcp.receive()* para receber uma resposta.

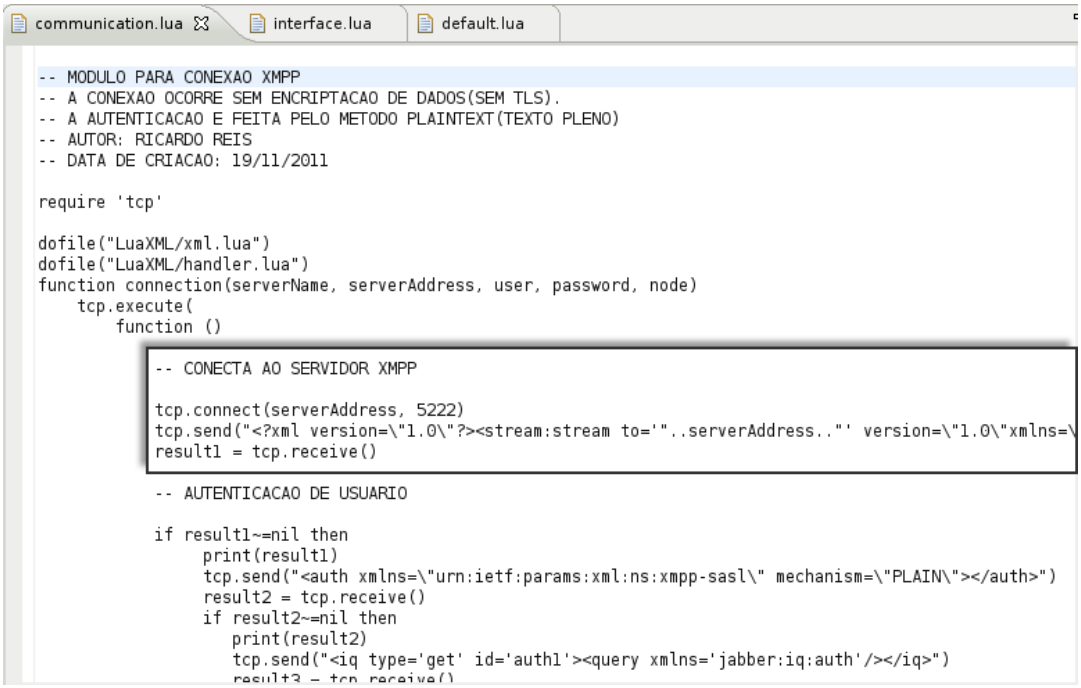

Figura 2.5: Exemplo de um evento.

2.2 Protocolo XMPP

Como alternativa para implementação da arquitetura, foi escolhido o protocolo XMPP [1] [2], acrônimo de eXtensible Messaging and Presence Protocol, um padrão da IETF (*Internet Engineering Task Force*) para a comunicação em tempo real através de passagem de mensagens XML. O protocolo oferece serviços de criptografia de canal, autenticação, presença, descoberta de serviços, entre outros.

Desenvolvido inicialmente por Jeremie Miller em 1998, surgiu com o intuito de disponibilizar aos desenvolvedores, um protocolo de passagem de mensagem aberto, pois na época cada aplicativo de mensagem instantânea tinha seu protocolo proprietário, isso

significa que, um usuário de um aplicativo não podia se comunicar com um usuário de outro.

Antes de ser padronizado, o protocolo era conhecido como Jabber e em 1999, por meio de uma comunidade adepta do código-fonte aberto, o primeiro servidor Jabber foi desenvolvido, o Jabberd. Em agosto de 1999 Jeremie pediu o apoio da comunidade no processo de padronização do IETF. O objetivo dele segundo a missão do projeto Jabber era fomentar a liberdade de conversação e apoiar padrões abertos e interoperabilidade nas comunicações em tempo real [18].

Com o crescente número de projetos *open-source* e entidades comerciais utilizando o Jabber formou-se o *Jabber Software Foundation* (JSF) para coordená-los. Em 2002 o XMPP foi aprovado pelo IETF, definindo dois *Request For Comments* (RFC) básicos (3920 e 3921) e as extensões. O RFC (3920) trata das especificações básicas do protocolo, tais como: arquitetura, métodos de conexão, segurança e transporte de dados e semântica das mensagens XML. Já o RFC (3921) define como ele faz o gerenciamento de contatos, funcionamento da *presença* e etc [18].

2.2.1 Arquitetura

O protocolo XMPP utiliza uma arquitetura cliente-servidor descentralizada como mostrado na Figura 2.6. O sistema consiste em uma rede de servidores que se comunicam entre si, permitindo assim a comunicação entre clientes conectados a servidores diferentes. Os servidores têm basicamente o trabalho de gerenciar as conexões dos clientes e fazer o roteamento das stanzas. Por questões históricas, Stanza é o nome dado às mensagens XMPP. Os clientes se conectam aos servidores através de conexões TCP, podendo estar conectados ao mesmo servidor a partir de lugares diferentes.

O esquema de endereçamento é feito através do Jabber ID, ou apenas JID. O JID é um identificador único que cada cliente adquire ao se cadastrar em um servidor. Ele é composto por 3 partes: *usuário@domíno/recurso*. O *domínio* é o identificador primário, ou seja, representa o servidor onde o cliente está conectado. O *usuário* é utilizado para identificar um usuário particular dentro de um domínio, por isso são válidos apenas em determinado domínio. Os *recursos* servem para identificar uma conexão particular do cliente, permitindo assim a um usuário se conectar várias vezes a um mesmo servidor.

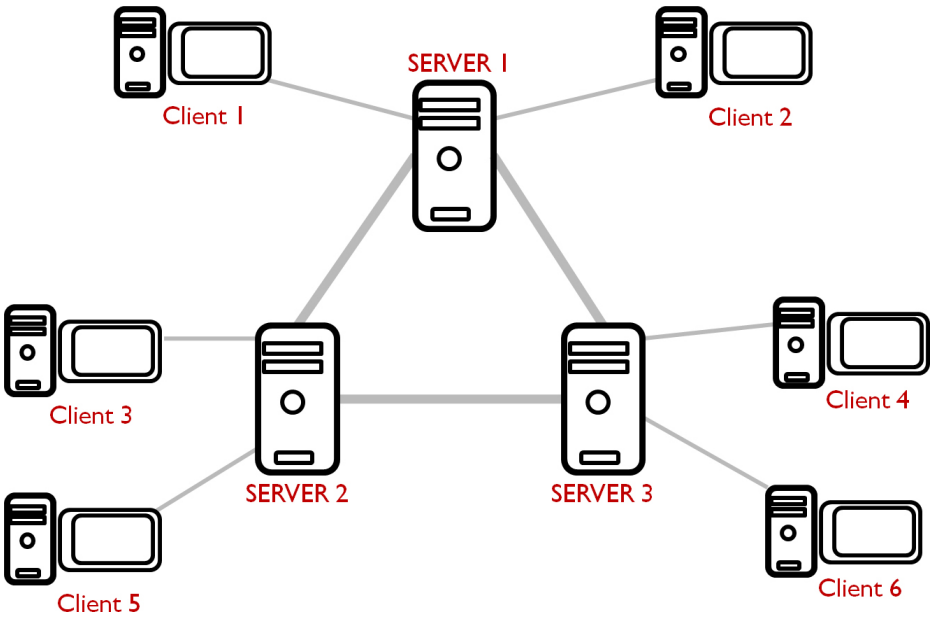

Figura 2.6: Arquitetura XMPP.

Logo, quando um usuário manda uma mensagem para outro, ele passa primeiramente pelo servidor do remetente, o servidor analisa o JID do destinatário e extrai o endereço do domínio destinatário para enviar a mensagem. Quando o servidor do destinatário recebe a mensagem, ele analisa a parte *usuário* do JID do destinatário para conseguir identificar entre todos os usuários cadastrados nele, para qual ele deve enviar. Como podemos observar, a comunicação não é feita diretamente entre clientes, as mensagens passam por no mínimo um servidor, no caso de os dois clientes estarem cadastrados no mesmo servidor, e no máximo dois, caso estejam em servidores diferentes.

Como podemos observar a arquitetura é muito parecida à dos servidores de correio eletrônico (e-mail). Porém existe uma diferença fundamental, o serviço de correio eletrônico utiliza o princípio de *store-and-forward* ([19]) enquanto que o XMPP entrega a mensagem quase em tempo real, pois a mensagem de *presença* enviada pelo cliente quando se conecta permite aos servidores saberem quem está esta conectado.

2.2.2 Comunicação

A comunicação entre todas as entidades da rede é feita através de mensagens XML (*Stanzas*). O cliente XMPP se comunica com o servidor através da porta 5222 e os servidores se comunicam entre si através da porta 5269.

Existem três tipos básicos de stanzas fazem parte do core do XMPP:

- **presence**
- **message**
- **iq**

As *stanzas* de *presença* controlam e reportam a disponibilidade da entidade, ou seja, logo que um cliente conecta-se a um servidor ele precisa enviar uma mensagem de presença ao servidor para informá-lo que ele está *on-line*. Fica a cargo do servidor repassar essa mensagem a todos os contatos do usuário que acabou de se conectar. As *stanzas* de presença podem ter os formatos mostrados na Figura 2.7.

```
<presence/>

<presence type='unavailable'/>

<presence>
        <show>away</show>
        <status>Estudando...</status>
</presence>
```

Figura 2.7: Formatos da mensagem de Presença.

As *stanzas* de *mensagem* permitem aos usuários da rede trocarem informações no momento de uma conversa. Elas tem o formato mostrado na Figura 2.8.

```
<message from='usuario1@chat.meryton.lit' to='usuario2@longbourn.lit' type='groupchat'>
        <body> XMPP é legal ☺</body>
</message>
```

Figura 2.8: Formatos da mensagem de Mensagem.

Com as *stanzas* de *iq* (*Info/Query*) os clientes podem fazer requisições ao servidor, como por exemplo a sua lista de contatos (*Roster*) mostrado na Figura 2.9.

A opção pelo XMPP, neste trabalho, deu-se pelo fato desse protocolo atuar com padrões abertos, o que garante a interoperabilidade da arquitetura proposta com os mais variados tipos de serviços e ambientes operacionais.

```
<iq from='usuario1@longbourn.lit/garden'
  type='get'  id='roster1'>

    <query xmlns='jabber:iq:roster'/>

</iq>
```

Figura 2.9: Formato da mensagem de *iq* (*Info/Query*).

2.3 Convergência entre TV digital e Web

Atualmente existe uma infinidade de conteúdos disponíveis na web, universalmente acessíveis a partir de browsers. Entretanto, considerando-se o cenário de aplicativos de TVDI, vê-se que este acesso é ainda restrito e geralmente tratado caso a caso, de maneira específica para cada tipo de aplicação.

No cenário atual, a integração da TV Digital com serviços da internet é feita através de uma abordagem simples, que consiste em utilizar um gateway que faça a integração entre o aplicativo de TV Digital e o serviço desejado [5]. O gateway faz pleno uso da API (*Application Programming Interface*) fornecida pelo serviço da internet, a fim de gerenciar a troca de mensagens HTTP entre os ambientes de TV digital interativa e da web, funcionando como uma camada de tradução entre esses dois mundos. A Figura 2.10 mostra o caso de um aplicativo Ginga, transmitido por uma emissora de TV, conectando-se através do gateway integrador a uma rede social disponível na internet.

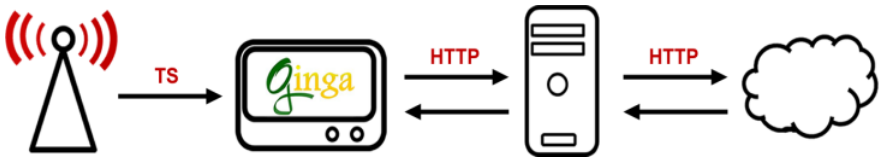

Figura 2.10: Arquitetura para integração entre TVDI e serviços da internet.

A emissora transmite a aplicação NCLua, juntamente com a programação, para os receptores Ginga; quando solicitadas, as aplicações abrem uma conexão com o gateway integrador para fazerem a requisição de informações de redes sociais. O gateway atua como um intermediador recebendo a requisição do aplicativo e repassando ao serviço na web, no caso a uma determinada rede social, por meio da API disponibilizada por ela. O serviço envia a resposta com as informações requisitadas ao gateway, que por sua vez repassa ao aplicativo que a solicitou.

Tal solução, além de estar fortemente atrelada à API fornecida pelo serviço, o que compromete sua aplicabilidade de forma mais genérica, padece, ainda, de um grave defeito no que tange à sua escalabilidade. Note-se que ela traz consigo o problema do gargalo no gateway [5], dado que muitas solicitações simultâneas irão certamente sobrecarregá-lo, consequentemente aumentando o tempo de resposta percebido pelos usuários e podendo até mesmo inviabilizar o uso da aplicação em si.

No ambiente de televisão, não se fala numa escala de dezenas de milhares de usuários, mas sim de milhões ou dezenas de milhões, daí a preocupação que se deve ter desde o início com o desempenho da arquitetura concebida como um todo.

3 ARQUITETURA DE INTEGRAÇÃO

Este trabalho propõe uma arquitetura genérica e escalável para permitir que usuários de TV Digital recebam e enviem conteúdo para a Web, utilizando-se de seus serviços mais comuns, como *sites*, *blogs* e redes sociais. Como já foi comentado, o tradicional modelo requisição/resposta adotado pelo HTTP mostra-se ineficaz neste cenário, pois não é capaz de sustentar a alta audiência esperada dos programas televisivos.

No cenário aqui considerado (Figura 3.1), interpõe-se um servidor, executando o protocolo XMPP, entre o *widget* NCL e o serviço da web, servidor este que irá desempenhar com vantagens as funções do *gateway* integrador mencionado na Seção 1.1. A aplicação (*widget*) NCL executando no decodificador (*set-top box*) da residência do telespectador registra inicialmente seu interesse por uma informação específica, disponível no servidor XMPP, através do cliente XMPP Ginga. O cliente estabelece uma conexão com o servidor XMPP a fim de receber as mensagens (*stanzas*) com a informação solicitada. Do outro lado, o servidor XMPP é periodicamente atualizado com dados provenientes do cliente XMPP baseado na web, que os obtém de *sites* previamente registrados e os repassa ao servidor XMPP.

Note que o servidor XMPP representa um ponto de concentração de informações na arquitetura proposta, de um lado sendo alimentado por informações obtidas pelos clientes XMPP Web e, de outro, fornecendo-as aos consumidores (clientes XMPP Ginga) interessados. Os clientes Ginga devem se registrar no servidor XMPP em modo *push*, sinalizando seu interesse em receber informações periodicamente. O servidor XMPP alivia, assim, a carga nos servidores web originais, compilando e disponibilizando a informação em um formato mais conciso, estruturado (um documento XML) e de forma mais imediata do que se fosse obtido diretamente do servidor web original. A interação com os servidores Web fica, portanto, a cargo dos clientes XMPP Web, que periodicamente consultam os servidores originais. Os *widgets* NCL, por sua vez, comunicam-se com o servidor XMPP por meio do cliente XMPP Ginga, liberando-se da tarefa (e da sobrecarga) de consultar diferentes *sites* da Web.

A característica genérica dessa arquitetura advém do fato que qualquer *site*

provedor de informações, que as disponibilize como *feeds*, pode utilizar a arquitetura aqui proposta para manter atualizado o *widget* NCLua, no ambiente de TV Digital. E sua escalablidade é garantida pela presença do servidor XMPP intermediário, que pode evoluir para um *cluster*, se assim for necessário.

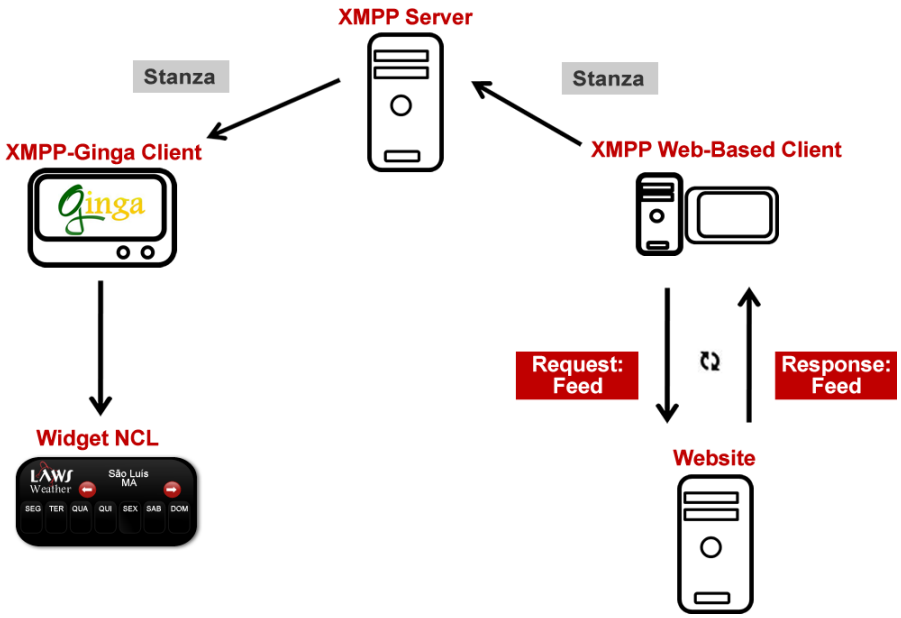

Figura 3.1: Arquitetura para integração entre widgets NCLua eweb.

A arquitetura proposta neste artigo vai além do domínio de aplicativos *publish/subscribe*. Ela visa também dar suporte a uma ampla gama de domínios, como o tradicional *request/response*, *chats* de comunicação e jogos *multiplayer* (com cada jogador no seu próprio ambiente de TV). Neste trabalho aplicamos a arquitetura nos domínios de jogos *multiplayer* e *publish/subscribe*, que serão abordados na seção de estudo de caso.

3.1 Cliente XMPP Web

Nessa arquitetura, o cliente XMPP Web utiliza técnicas de AJAX (*Asynchronous Javascript and XML*) [15] para fazer requisições automáticas, periodicamente, visando obter o *feed* com dados atualizados do *site* provedor de informações.

O cliente XMPP Web implementado utiliza um método de conexão que não requer a manutenção da conexão TCP aberta por muito tempo, embora o padrão para uma rede de passagem de mensagens seja manter a conexão TCP aberta enquanto o

usuário estiver *on-line*. Essa otimização no tempo de uso das conexões é possível graças à opção feita pelo BOSH (*Bidirectional-streams Over Synchronous HTTP*) [6] [4] [3], um protocolo de extensão do XMPP.

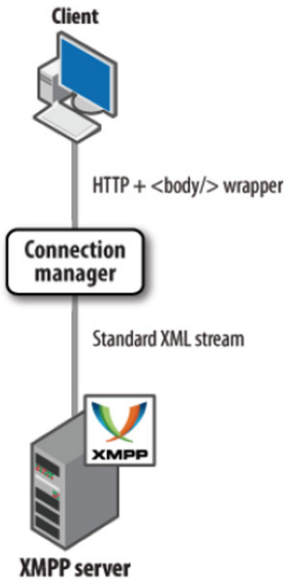

Figura 3.2: No BOSH, o cliente *fala* HTTP com o gerenciador de conexão, que *fala* XMPP com o servidor.

Sistemas baseados em BOSH utilizam um gerenciador de conexões que atua como um tipo de *proxy* entre o cliente e o servidor XMPP, como mostrado na Figura 3.2 [4]. Assim, a cada resposta obtida do *site*, o cliente XMPP Web encapsula o *feed* em uma *stanza* XMPP e a envia, dentro de um pacote HTTP, para o gerenciador de conexões, que extrai essa *stanza* e a repassa ao servidor. O servidor se encarrega de enviar essa *stanza* a todos os usuários interessados conectados a ele.

3.2 Cliente XMPP Ginga

No contexto da TV Digital, ao se ligar a televisão, o aplicativo abre um *socket* de conexão com o servidor XMPP e todo um processo de negociação e autenticação é executado para o cliente adquirir o direito de receber as *stanzas* de mensagens lá armazenadas.

Após a negociação e autenticação, o aplicativo fica à espera das mensagens enviadas pelo cliente XMPP Web, que alimenta o servidor XMPP com informações da

web. Ao receber as *stanzas* de mensagem, o cliente Ginga extrai os dados da mensagem, trata-os e por fim os exibe, quando solicitado, através do *widget* NCLua no aparelho de TV.

3.3 Utilização de Padrões de Projeto

Ao se projetar a arquitetura em questão, decidiu-se fazê-lo segundo alguns padrões de projeto (*design patterns*) comuns, que posteriormente foram utilizados na implementação do *framework* descrito na seção seguinte. Observando a Figura 3.3 nota-se uma dependência um-para-muitos, do servidor XMPP em direção aos clientes XMPP Ginga, de maneira que quando um objeto muda de estado todos os seus dependentes são notificados e atualizados automaticamente [8].

Esse comportamento do tipo *publish/subscribe* (Padrão *Observer*) [8] consiste em um objeto que atua como um canal de notificação em tempo real, ou seja, um nó que publica informações, sendo que qualquer outro nó pode assinar esse canal para receber as informações transmitidas pelo publicador. Essa dependência um-para-muitos se traduz em comunicação *broadcast*, onde o publicador não se preocupa com quantos observadores (Assinantes) existem, ele apenas transmite os dados para todos da sua lista.

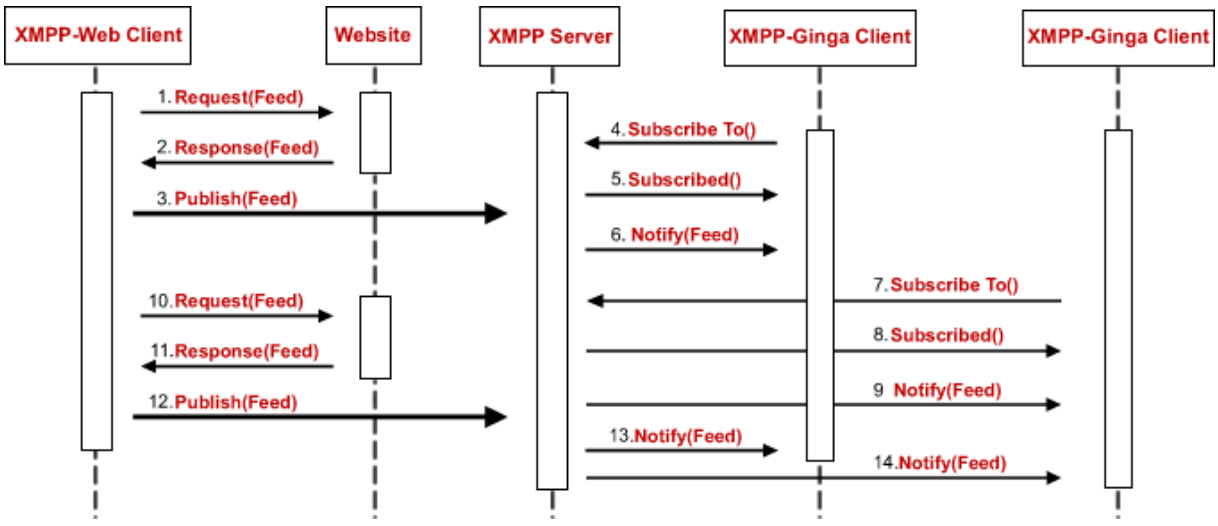

Figura 3.3: Arquitetura para integração entre widgets NCLua e web.

O padrão Observer possui dois modos de implementação: *pull* e *push*. No modelo *pull*, o publicador emite uma notificação apenas avisando que novos dados estão

disponíveis, ficando a cargo dos assinantes a requisição dos dados junto ao publicador. Note que esse modelo gera o mesmo problema do *gateway* integrador citado na Seção 1.1, onde muitas requisições seriam feitas ao publicador causando assim uma sobrecarga. A arquitetura proposta neste trabalho se encaixa no modelo *push* do padrão, onde o publicador envia os novos dados junto com a notificação, quer os assinantes os solicitem ou não, poupando o publicador de uma possível sobrecarga.

O nó publicador não mantém uma lista dos assinantes, ele apenas envia as informações ao servidor, ficando a cargo deste repassá-las a todos os assinantes do canal de publicação. Ele atua neste caso como um mediador (*Mediator*) [8], mantendo o canal e fazendo a interface entre publicador e assinantes, ou seja, os objetos conhecem apenas o mediador, diminuindo assim o número de interconexões e promovendo o fraco acoplamento das partes. Pode-se observar na Figura 3.3 como o servidor XMPP faz essa mediação, tornando-se o elo de comunicação entre as duas partes, ou seja, entre os aplicativos Ginga e a Web.

Para a implementação dos padrões *Observer* e *Mediator*, fez-se o uso do XEP (*XMPP Extension Protocol*) *Publish/Subscribe* [7], disponibilizado pelo *XMPP Standards Foundation*.

3.4 O Framework de Comunicação

Como forma de facilitar o desenvolvimento de aplicações do tipo *publish/subscribe*, implementou-se a arquitetura na forma de um *framework*. Este oferece uma API eficiente a fim de permitir o rápido desenvolvimento desse tipo de aplicação, de forma que o desenvolvedor não precise se envolver com detalhes de comunicação, ou seja, com o protocolo propriamente dito.

O *framework* é baseado em uma arquitetura com três camadas:

- **Camada de dados**: Composta por um *website* que fornece os *feeds*.
- **Camada de aplicação**: Composta por um cliente XMPP Web que obtém o *feed* periodicamente do site, e por um servidor XMPP responsável por enviar o *feed* a todos os assinantes do canal.
- **Camada de Apresentação**: Composta por um cliente XMPP Ginga que assina o

canal, recebe os dados e exibe ao telespectador os *feeds* através de um *widget* NCL.

3.4.1 Implementação da Arquitetura

A camada de dados faz o papel de provedor dos dados. Ela consiste em um *site* que fornece *feeds* nos padrões de publicação de conteúdo (*content syndication*): RSS 1.0 (*RDF Site Summary*); RSS 2.0 (*Really Simple Syndication*); e Atom. O *framework* foi projetado para ser genérico no sentido de aceitar qualquer um desses formatos sem que o desenvolvedor tenha que se preocupar com o *parsing* de cada um deles.

Na camada de lógica da aplicação, o cliente XMPP Web é o nó publicador, que utiliza técnicas de AJAX (*Asynchronous Javascript and XML*) para fazer requisições automáticas, de tempos em tempos, visando obter o *feed* com dados atualizados do *site* provedor de informações. Após a obtenção do *feed*, o cliente faz o *parsing* do documento e monta uma mensagem XMPP com as informações obtidas do *feed*. Em seguida essa mensagem é enviada ao servidor XMPP, que se encarrega de repassá-la a todos os telespectadores que assinaram o canal de publicação.

Podemos observar na Figura 3.4 como o *Google Feed API* é utilizado para fazer a requisição do feed. Ao se criar o objeto *feed*, passa-se a URL (*Uniform Resource Locator*) do *feed* desejado como parâmetro para que ele possa ser requisitado. Pode-se determinar também o número de nós desejados do *feed* em *feed.setNumEntries(framework.entries)* e o formato que o *Google Feed API* retornará para a aplicação em *feed.setResultFormat(google.feeds.Feed.JSON_FORMAT)*. No caso desta aplicação o formato escolhido foi o JSON (*JavaScript Object Notation*).

```
/**
 * Google Feed API sendo utilizado para fazer a requisição e  parsing do feed.
 */

      var feed = new google.feeds.Feed(framework.url);
      var time = new Date();
      feed.setNumEntries(framework.entries);
      feed.setResultFormat(google.feeds.Feed.JSON_FORMAT);
```

Figura 3.4: Google Feed API sendo utilizado para fazer a requisição do feed.

Em seguida, o *feed* no formato JSON retornado pelo *Google Feed API* passa por um *parsing* onde o título é retirado de cada nó do *feed* para formar a mensagem

XMPP. A hora em que a requisição foi feita também é anexada para informar ao usuário o quão recente é a notícia. Como é mostrado na Figura 3.5, ao filtrar todos os títulos dos nós, a aplicação chama a função *XMPP.publish_action(feeds);* que montará a mensagem XMPP e enviará ao servidor.

```
feed.load(function(result) {
        if (!result.error) {
           var container = document.getElementById("feed");
           $('.entries').remove();
            var div = document.createElement("div");
            div.className = "entries";
            container.appendChild(div);
          var feeds = [];
            for (var i = 0; i < result.feed.entries.length; i++) {
               var entry = result.feed.entries[i];
            var p = document.createElement("p");
               p.appendChild(document.createTextNode(entry.title));
               div.appendChild(p);

            feeds[i] = entry.title;
            }
          var divTime = document.createElement("div");
           divTime.className = "time";

           var hour = time.getHours();
           var min = time.getMinutes();
           var sec = time.getSeconds();

           divTime.appendChild(document.createTextNode("Requested at: " + hour + ":" + min + ":" + sec));
           div.appendChild(divTime);

          XMPP.publish_action(feeds);

          }else{
            alert(result.error.code);
          }
       });
```

Figura 3.5: Função onde o *feed* é filtrado.

Na Figura 3.6 pode-se observar a biblioteca XMPP *Strophe* baseada em *javascript* sendo utilizada para montar a mensagem XML e enviar a *stanza*. Observa-se que a mensagem esta sendo montada no parâmetro da função *XMPP.connection.sendIQ()*, que por sua vez está encarregada de enviá-la ao servidor.

O cliente XMPP Web foi desenvolvido em *javascript* com o auxílio da biblioteca *Strophe.js* para o tratamento da comunicação XMPP, e do XEP (*XMPP Extension Protocol*) BOSH (*Bidirectional-streams Over Synchronous HTTP*) [6], que permite o desenvolvimento de clientes XMPP baseados na web. Para o *parsing* dos *feeds* utilizou-se

o *Google feed API*, uma interface de programação para solicitar e tratar *feeds* [9]. O servidor XMPP utilizado foi o *Openfire*.

O cliente XMPP Web foi dividido em dois componentes: de comunicação e de interface. O componente de comunicação é o de mais baixo nível, trata da comunicação e troca de mensagens com o servidor XMPP, da requisição e *parsing* do *feed*. O componente de interface é o de mais alto nível, responsável por fazer a interação entre o desenvolvedor e a camada de comunicação. Essa camada é a API do *framework*, que esconde do desenvolvedor todos os detalhes de comunicação, requisição e tratamento de *feeds*, dessa forma poupando tempo de codificação e permitindo o foco em outras partes da aplicação a ser desenvolvida.

```
//Publica informações
   publish_action: function (action) {

      XMPP.connection.sendIQ(
         $iq({to: XMPP.service, type: "set"})
            .c('pubsub', {xmlns: XMPP.NS_PUBSUB})
            .c('publish', {node: XMPP.node})
            .c('item')
            .c('x', {xmlns: XMPP.NS_DATA_FORMS,
                  type: "result"})
            .c('field', {"var": "title1"})
            .c('value').t(action[0])
            .up().up()
            .c('field', {"var": "title2"})
            .c('value').t(action[1])
            .up().up()
            .c('field', {"var": "title3"})
            .c('value').t(action[2])
            .up().up()
            .c('field', {"var": "title4"})
            .c('value').t(action[3])
            .up().up()
            .c('field', {"var": "title5"})
            .c('value').t(action[4]));

      var timePublication = new Date();
      var hourPublication = timePublication.getHours();
      var minPublication = timePublication.getMinutes();
      var secPublication = timePublication.getSeconds();
      XMPP.log("Published at: <b>" + hourPublication + ":" + minPublication + ":" + secPublication + "</b>");
   },
```

Figura 3.6: Função encarregada de montar a mensagem XML e enviá-la.

A camada de apresentação é composta por um cliente XMPP Ginga desenvolvido em NCLua, que tem como funcionalidades: assinar um canal; fazer o *parsing* das mensagens recebidas; e exibir as informações através de um *widget* NCL. Ao assinar

um canal, o cliente mantém a conexão aberta com o servidor XMPP e fica aguardando a atualização das informações vindas dele. Quando a mensagem chega do servidor, o cliente faz o *parsing* dela e as exibe através de um *widget*.

Como o cliente XMPP Web, o cliente XMPP Ginga está dividido em dois componentes: de comunicação e de interface. O componente de comunicação trata da troca de mensagens com o servidor XMPP e do *parsing* delas. Já o componente de interface, do mesmo modo que no cliente XMPP Web, esconde os detalhes de comunicação do desenvolvedor, oferecendo-lhe uma API para este fim. É através dessa API que o desenvolvedor poderá carregar e posicionar seu *widget* NCL, assinar um canal de publicação e conectar um cliente ao servidor XMPP, por exemplo.

Cliente Web

Método	Descrição	Parâmetro
set_serverNameAddress()	Utiliza o nome, o endereço e a porta do servidor XMPP para permitir aos clientes conectarem-se a ele.	(serverName, serverAddress)
set_time()	Determina a periodicidade das requisições ao *feed*.	(time)
set_url()	Determina o endereço do *feed*.	(url)
connect()	Utiliza o *usuário* e a *senha* do cliente para se conectar ao servidor.	(jid, password)

Figura 3.7: API do cliente web.

Podem-se observar na Figura 3.7 os métodos da API do cliente web. A API é composta por quatro métodos: no método *connect()* são passadas duas variáveis como parâmetro, o usuário e a senha do cliente, a fim de se conectar com o servidor XMPP. Os métodos *set_url()* e *set_time()* recebem, respectivamente, a URL onde o cliente requisita o *feed* e o intervalo de tempo entre duas requisições sucessivas. O método *set_serverNameAddress()* utiliza o nome, o endereço e a porta do servidor XMPP como parâmetro para permitir aos clientes conectarem-se a ele.

Na API do cliente Ginga (Figura 3.8), o método *connect()* funciona de modo similar ao do cliente Web, podendo disconectar também através do *disconnect()*. Após a conexão como o servidor XMPP, pode-se assinar um canal de publicação através do método *set_Subscribe()* indicando o nome do nó publicador, e pode-se cancelar a assinatura através do método *set_Unsubscribe()*. O método *set_messagePull()* funciona de modo similar ao *set_serverNameAddress()* do cliente Web, onde é necessário informar o nome e endereço do servidor XMPP.

Cliente Ginga

Método	Descrição	Parâmetro
connect()	Utiliza o *usuário* e a *senha* do cliente para se conectar ao servidor.	(user, password)
disconnect()	Disconecta do servidor XMPP.	-
set_Subscribe()	Assina um nó publicador utilizando o seu nome.	(node)
set_Unsubscribe()	Cancela a assinatura de um nó publicador.	(unsubscribe)
set_messagePull()	Utiliza o nome e o IP do servidor XMPP para permitir aos clientes conectarem-se a ele.	(serverName, serverAddress)
get_messages()	Captura as mensagens depois de tratadas e faz uma chamada a drawWidget().	(messages)
set_entriesStyles()	Determina fonte, tamanho e cor das notícias que será exibidas no *widget*.	(font, size, color)
set_entriesPositions()	Determina a posição (x, y) de cada título de notícia no *canvas*.	(x1, y1, x2, y2, x3, y3, x4, y4, x5, y5)
set_widget()	Determina o caminho da imagem do *widget*.	(path)
set_widgetPosition()	Determina a posição (x, y) do *widget* no canvas.	(x, y)
draw_widget()	Desenha o *widget* e os títulos das notícias no canvas, de acordo com as respectivas posições (x, y) passadas.	-

Figura 3.8: API do cliente Ginga.

Já o método *get_messages()*, captura as mensagens recebidas, depois de tratadas, e informa ao método *draw_widget()* que já pode desenhá-las junto ao seu *widget* de acordo com as suas respectivas posições (x,y) passadas pelos métodos *set_entriesPositions()* e *set_widgetPosition()*. O método *set_entriesStyles()* determina a formatação aplicada aos títulos das notícias, tais como fonte, tamanho e cor. Por fim através do método *set_widget()* indica-se o caminho (*path*) da imagem do *widget*.

4 ESTUDO DE CASO

Foram desenvolvidas aplicações classificadas em duas categorias, *Publish/Subscribe* e Interativas, para demonstrar o uso da arquitetura de integração proposta em cenários diferentes.

As aplicações foram desenvolvidas no ambiente de uma máquina virtual Ginga executando sobre Debian 5.0.6 e com o Eclipse IDE 3.5 utilizando o *plugin* NCL-Eclipse 1.5.1 para o desenvolvimento de aplicações voltadas ao Ginga NCL.

4.1 Aplicações *Publish/Subscribe*

Uma aplicação que acessa o *site* Clima Tempo foi desenvolvida utilizando a arquitetura proposta, como forma de validá-la com um exemplo realista (Figura 4.1). O *site* do Clima Tempo disponibiliza um *feed* em formato XML que foi utilizado para alimentar o *widget* com informações sobre a previsão do tempo em todas as capitais do país.

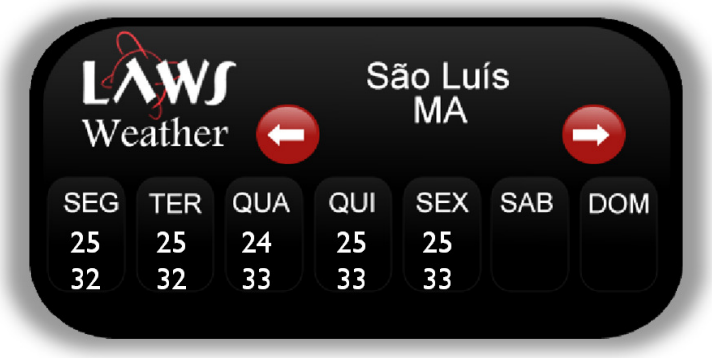

Figura 4.1: *Widget* da aplicação do clima tempo.

A aplicação do Clima Tempo se encaixa perfeitamente na arquitetura, pois o *feed* do site é atualizado diariamente, com isso foi possível desenvolver o cliente XMPP Web para enviar os dados ao *widget* na mesma frequência do *site*. Entretanto, esta abordagem obriga que o cliente XMPP Web tome a iniciativa de consultar o site e, caso

isso tenha que ser feito várias vezes, pode acabar gerando uma sobrecarga desnecessária pelas muitas mensagens de *polling* e resposta. Um modo de diminuir essa sobrecarga por parte do cliente XMPP Web é utilizar um protocolo de extensão do XMPP do tipo *Publish/Subscribe* [7] [4] [3]. Podemos observar uma comparação entre XMPP e HTTP na Figura 4.2.

No caso do cliente web convencional, a frequência de requisições ao servidor pode se tornar alta, gerando uma diferença significativa de desempenho se muitos clientes estiverem ativos simultaneamente [4]. Já com o uso do modelo *Publish/Subscribe*, os clientes recebem as informações (por meio de uma notificação) quando o site publicador as disponibiliza, economizando assim processamento e largura de banda.

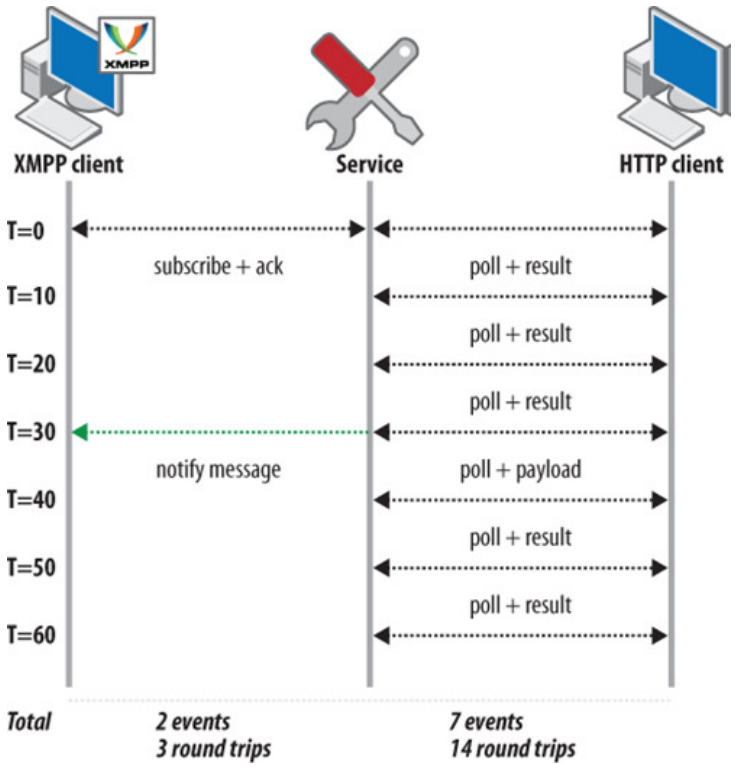

Figura 4.2: Comparação entre as abordagens de XMPP e HTTP.

A aplicação descrita acima foi pensada e concebida apenas para o Clima Tempo, o qual faz parte de um domínio de aplicações alimentadas por *feeds*, portanto o próximo passo foi tornar a arquitetura genérica para esse domínio. O *framework* já descrito foi projetado para dar suporte ao desenvolvimento de *widgets* alimentados por qualquer *feed* da web sem que o desenvolvedor se envolva com o protocolo de comunicação.

O site da SBC (Sociedade Brasileira de Computação) publica informações

constantemente na forma de *feeds* RSS 2.0. A aplicação consiste em um *widget* NCL para TVDI que é alimentado com informações desse *site*. Pode-se observar o cliente XMPP Web da aplicação em ação, na Figura 4.3 e o *widget*, na Figura 4.4. Observou-se durante o desenvolvimento que habilitar o *widget* para receber mensagens do cliente XMPP Web é facilitado pela API oferecida pelo *framework*, já que basta a configuração de alguns parâmetros para fazê-lo operar corretamente, sendo essa a etapa mais rápida do desenvolvimento. O maior trabalho, portanto, deve-se à concepção do leiaute do cliente XMPP Web e ao projeto do *widget*. Foram feitos testes com outros protocolos de publicação de conteúdo como o Atom e o RSS 1.0 e não foi preciso mudar uma linha de código sequer, pois essa parte genérica ficou a cargo do *Google Feed API*, que dá suporte aos padrões usados atualmente.

Figura 4.3: Cliente XMPP Web do framework.

Durante a fase de testes, notou-se que se a periodicidade de requisições do *feed* for menor que 2 segundos, o cliente XMPP Web encerra a conexão com o servidor, consequentemente as atualizações do *widget* terminam. Esse problema ocorre pois menos de 2 segundos não é tempo suficiente para se fazer uma requisição pelo *feed*, receber a

resposta, fazer o *parsing* e mandar ao servidor a tempo de repetir todo o ciclo novamente. Observa-se que a maior parte do tempo é gasta pela requisição/resposta do *feed* já que isso depende do tráfego na rede.

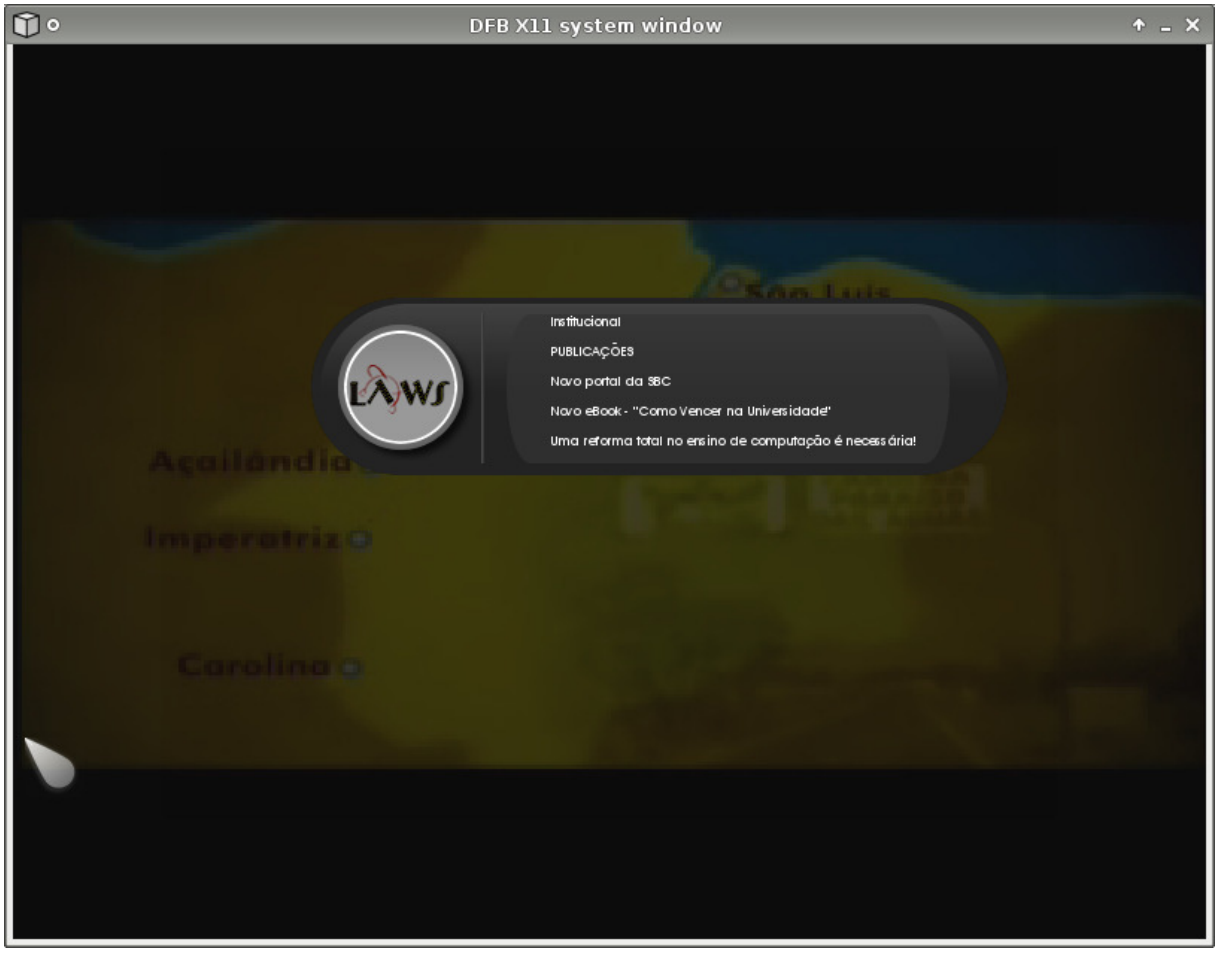

Figura 4.4: *Widget* genérico com informações trazidas de *feeds* do *site* da Sociedade Brasileira de Computação.

Na Figura 4.5 pode-se observar como o *widget* é carregado pelo Ginga. O método *drawWidget* utiliza o *canvas* para desenhar o *widget*, que nada mais é que uma imagem. O desenvolvedor utiliza o *framework* para carregar a imagem e depois determina uma posição (x, y) no *canvas* onde essa imagem será desenhada. Em seguida utiliza-se a fonte, tamanho e cor também passadas pelo desenvolvedor através da API, para serem aplicadas às notícias que serão exibidas no *widget*. Através de um *while* cada título de notícia será desenhada na posição (x,y) passada pelo desenvolvedor, que será por cima do widget.

```
drawWidget = function()
                canvas:attrColor(0,0,0,0)
                canvas:clear()

                img = canvas:new(framework.widget)
                canvas:compose(framework.widgetX, framework.widgetY, img)

                local i = 1
                local j = 1
                canvas:attrColor(framework.color)
                canvas:attrFont(framework.font, framework.size)

                 while(i<=5) do
                        canvas:drawText(framework.positions[j],framework.positions[j+1], framework.data[i])
                        i= i+1
                        j = j+2
                end

                canvas:flush()

            end
```

Figura 4.5: Método utilizado para desenhar o *widget* no *canvas*.

4.2 Aplicações Interativas

Assim como aplicações *publish/subscribe*, a arquitetura proposta também suporta aplicações interativas. Nessa categoria, desenvolveu-se um jogo de xadrez *multiplayer*, onde cada jogador se encontra no seu próprio ambiente de TV. Seu funcionamento se assemelha ao de uma aplicação de mensagens instantâneas, onde dois usuários estão conectados a um servidor XMPP e trocam mensagens de uma conversa entre si.

No jogo (Figura 4.6), cada mensagem enviada ou recebida, em vez de conter o texto de uma conversa, poderá conter informações de jogadas, estado da partida, disponibilidade dos jogadores, etc. Por exemplo, ao efetuar uma jogada, o *jogador A* envia uma mensagem contendo a peça movida e nova localização dela, ou seja, sua linha e coluna, ao *jogador B*. Os clientes de cada um irão fazer o *parsing* das mensagens e atualizar o tabuleiro de jogo, já que essa tarefa não cabe ao servidor. O papel do servidor é simplesmente encaminhar as mensagens.

Como dito anteriormente, o XMPP trabalha com padrões abertos, portanto isso garante a interoperabilidade da arquitetura. O cliente XMPP Web do *framework* pode se tornar um cliente do jogo de xadrez multiplayer apenas fazendo as alterações necessárias, uma vez que ele já é cliente de um servidor XMPP. Do mesmo modo, pode-se implementar um cliente *Android* ou um cliente *desktop* para o jogo, em vez do cliente XMPP Ginga aqui apresentado (Figura 4.6), sem perda de generalidade.

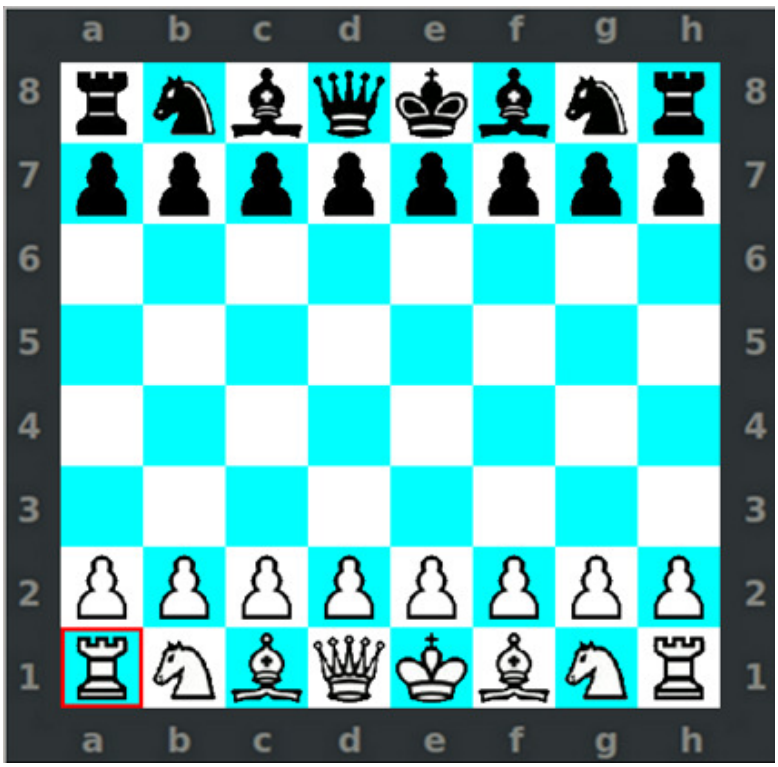

Figura 4.6: Jogo de xadrez multiplayer em NCLua.

Desse modo, utilizando a arquitetura de integração aqui detalhada, um jogador poderá se conectar ao servidor do jogo e jogar contra um usuário em qualquer plataforma, desde que tenha o cliente do jogo implementado na sua plataforma. Para o servidor XMPP, não faz diferença se as informações trocadas são mensagens instantâneas ou provenientes de um *site* como o Clima Tempo, ou do jogo de xadrez, o que confirma o caráter generalista da arquitetura proposta.

5 Conclusão

Este trabalho discutiu e implementou uma arquitetura genérica para convergência entre aplicativos de TV digital interativa, executando na plataforma Ginga NCL e serviços comuns da web, visando assim suprir uma demanda geral por acesso a conteúdo e aplicações da web a partir da plataforma Ginga NCL de TVDI.

A solução arquitetural aqui apresentada utiliza-se de um servidor executando o protocolo XMPP, atendendo aos critérios de generalidade e escalabilidade colocados como premissa inicial deste trabalho. Um conjunto de aplicativos foi desenvolvido a fim de validar a funcionalidade da arquitetura, sendo aqui destacados dois deles: um *widget* NCLua alimentado por *feeds*, segundo o modelo *publish/subscribe* e outro, um jogo *multiplayer*, na categoria de *aplicações interativas*. A implementação de ambos foi bastante facilitada pelo uso da arquitetura aqui proposta.

Outra contribuição deste trabalho foi no sentido de identificar e aplicar padrões de projeto, como *observer* e *mediator*, no projeto de aplicações de TV digital interativa, culminando na implementação de um *framework* de comunicação para interação com servidores do tipo XMPP, o que tornou o desenvolvimento de novas aplicações bastante rápido e produtivo, a exemplo do *widget* implementado para consumir informações do *site* da SBC.

Ao se utilizar a presente arquitetura, é possível minimizar consideravelmente o tempo necessário para o desenvolvimento de aplicações para TV digital que façam uso do canal de interatividade da plataforma Ginga. O *framework* de comunicação proposto também simplifica o acesso a dados da web, oportuniza o reuso de código e permite uma integração com serviços mais direta e menos suscetível a problemas de escalabilidade e erros. O desenvolvedor, dessa forma, pode concentrar-se na lógica de aplicação do *widget* e no projeto de sua interface com o usuário, deixando os detalhes de comunicação por conta da arquitetura integradora Ginga/Web descrita neste trabalho.

Segundo Cesar et al. [11], Social TV constitui uma mudança fundamental na forma como as pessoas interagem e socializam em torno de conteúdos televisivos e pode ser dividido em um conjunto de categorias de aplicações, tais como: *content selection*

and sharing, *communication*, *community building* e *status update*. Acredita-se que a arquitetura proposta pode também ser a base para tal categoria de aplicações, já que o protocolo XMPP mostra-se robusto a ponto de suportar sistemas distribuídos complexos, porém flexível como mostrado neste artigo, podendo ser utilizado para uma ampla gama de domínios de aplicações.

Como trabalhos futuros, pretende-se investigar algumas variações na arquitetura, experimentando-se outros tipos de servidores além dos baseados em XMPP, a fim de melhorar ainda mais seu desempenho e escalablidade. E, ainda, deseja-se realizar sua integração com o Facebook e Twitter, duas das redes sociais mais populares do momento, ambas com APIs livremente disponibilizadas na web. A médio prazo, esta arquitetura pode tornar-se uma rede de serviços, onde cada nó oferecerá um tipo de serviço, podendo-se criar mais nós (serviços) apenas cadastrando-os no servidor. Seria interessante também disponibilizar o *framework* discutido na seção 3.4 para uma comunidade ampla de desenvolvedores, que poderão testá-lo em diversos cenários, contribuindo para o seu aprimoramento e da arquitetura em questão.

Referências Bibliográficas

[1] SAINT-ANDRE, Peter. *Extensible Messaging and Presence Protocol (XMPP): Core*, 10/2004. http://xmpp.org/rfcs/rfc3920.html.

[2] SAINT-ANDRE, Peter. *Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence*, 10/2004. http://xmpp.org/rfcs/rfc3921.html.

[3] MOFFITT, Jack. *Professional XMPP Programming with Javascript and JQuery*, Editora Wrox. 2010.

[4] SAINT-ANDRE, Peter; SMITH, Kevin; TRONÇON, Remko. *XMPP: The Definitive Guide*, Editora O'Reilly. 2009.

[5] C. GHISI, Bruno; F. LOPES, Guilherme; SIQUEIRA, Frank. *Integração de Aplicações para TV Digital Interativa com Redes Sociais*, WEBMEDIA 2010. Belo Horizonte – MG. 2010.

[6] PATERSON, Ian; SMITH, Dave; SAINT-ANDRE, Peter; MOFFITT, Jack. *[XEP-0124] Bidirectional-streams Over Synchronous HTTP (BOSH)* , http://xmpp.org/extensions/xep-0124.html.

[7] MILLARD, Peter; SAINT-ANDRE, Peter; MEIJER, Ralph. *[XEP-0060] Publish-Subscribe*, http://xmpp.org/extensions/xep-0060.html.

[8] GAMMA, E; HELM, R; JOHNSON, R; VLISSIDES, J. *Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos*, Porto Alegre: Bookman, 2000. 350 p.

[9] GOOGLE. *Google Feed API*, Disponível em: https://developers.google.com/feed/. Acessado em 09/2011.

[10] F. G. SOARES, Luiz; D. J. BARBOSA, Simone. *Programando em NCL 3.0: Desenvolvimento de Aplicações para o Middleware Ginga*, 2.ª ed., Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio).

[11] CESAR, Paulo; GEERTS, David. *Past, Present, and Future of Social TV: A Categorization*, Proceedings of the IEEE Consumer Communications and Networking Conference (IEEE CCNC 2011), Las Vegas (NV), USA, January 9-12, 2011, pp. 347-351.

[12] Portal R7. *TV e geladeira estão em quase todos os lares do Brasil*, http://noticias.r7.com/economia/noticias/tv-egeladeira-estao-em-quase-todos-os-lares-do-brasil-20100917.html 2010.

[13] Jornal Estadão. *Brasil é o quinto país que mais usa redes sociais*, http://www.estadao.com.br/noticias/impresso,brasile-o-quinto-pais-que-mais-usa-redes-sociais,600478,0.htm2010.

[14] Associação Brasileira de Normas Técnicas. *ABNT 15606-1. Televisão digital terrestre – Codificação de dados e especificações de transmissão para radiodifusão digital – Parte 1: Codificação de dados*, 2008.

[15] World Wide Web Consortium (W3C). *W3C Working Draft 17. XMLHttpRequest level 2*, 2012. http://www.w3.org/TR/XMLHttpRequest/.

[16] VIANA, Adriano. *Acesso a serviços de governo eletrônico a partir da TV Digital: o caso do Tribunal de Justiça do Estado do Maranhão*, São Luís, 2010.

[17] SANT'ANNA, Francisco; SOARES, Carlos S.; AZEVEDO, Roberto G. A.; BARBOSA, Simone D. J.. *Desenvolvimento de Aplicações Declarativas para TV Digital no Middleware Ginga com Objetos Imperativos NCLua*, minicurso, Webmedia, Fortaleza, 2009.

[18] XMPP Team. *History*. http://xmpp.org/about-xmpp/history/.

[19] KUROSE, James; F.ROSS, Keith. *Redes de Computadores e a Internet*, 3ª Edição, Editora Pearson. 2005.

[20] F. MORENO, Marcio; F. MORENO, Marcelo; F. G. SOARES, Luiz. *Transmissão de Aplicações e Comandos de Edição ao Vivo em IPTV e DTV Terrestre*, WEBMEDIA 2010. Belo Horizonte – MG. 2010.

A APÊNDICE

A.1 Publicações

- REIS, R. L. B. ; TEIXEIRA, M. M. . Uma Arquitetura para Interoperablidade entre Aplicativos NCLua e Serviços da Internet. In: XVII Simpósio Brasileiro de Sistemas Multimídia e Web, 2011, Florianópolis, SC. Workshop de Trabalhos de Iniciação Científica (WTIC), 2011.

- REIS, R. L. B. ; TEIXEIRA, M. M. . Uma Arquitetura para Interoperablidade entre Aplicativos NCLua e Serviços da Internet. In: 64º Reunião Anual da SBPC, 2012, São Luís, MA. Jornada Nacional de Iniciação Científica, 2012.

A.2 Prêmios

- Best Paper, Workshop de Trabalhos de Iniciação Científica (WTIC), WebMedia - 2011.

- 1º Lugar - Apresentação de Oral, XXIII Seminários de Iniciação Científica (SEMIC) - 2011

B APÊNDICE

Uma Arquitetura para Interoperablidade entre Aplicativos NCLua e Serviços da Internet

Ricardo Lucio Braga Reis
Laboratory of Advanced Web Systems – UFMA
Av. dos Portugueses, Campus do Bacanga
São Luís/MA – 65085-580 - Brasil
ricardo.l.b.reis@gmail.com

Mário Meireles Teixeira
Departamento de Informática – UFMA
Av. dos Portugueses, Campus do Bacanga
São Luís/MA – 65085-580 - Brasil
mario@deinf.ufma.br

ABSTRACT
The emergence of Digital TV brought in novel possibilities of interaction between TV viewers and TV programming. There is a clear tendency of convergence between TV and the web. One of the challenges in this scenario is to provide solutions to assist application development in this new media. This work defines and implements an open source architecture that not only streamlines the access of NCL applications to web content but also supports the so-called social networks, one of the strongest current trends.

Categories and Subject Descriptors
C.2.1 Network Architecture and Design – *Distributed Networks*; C.2.4 Distributed Systems – *Distributed Applications*.

General Terms
Design, Experimentation, Languages, Performance.

Keywords
Interactive Digital TV, Applications, NCL, Ginga, Web, XMPP.

1. INTRODUÇÃO
Desde o início da TV, as emissoras tentam proporcionar interação com o telespectador através das mais diversas mídias. No início da TV analógica, essa interação acontecia através de cartas, forma mais tradicional de comunicação. A interatividade acompanhou o avanço tecnológico e logo passou para o telefone, seguida das mensagens SMS, e-mails e *chats*. No cenário atual, com o advento da TV Digital e a convergência entre tecnologias, tornou-se inevitável o surgimento de um elo entre web e TV.

Este artigo tem como objetivo especificar e implementar uma arquitetura para acesso e interação com serviços da internet a partir de mini-aplicativos (*widgets*) NCLua. Ao final, são apresentadas algumas aplicações desenvolvidas a fim de validar a arquitetura proposta.

O artigo está organizado como segue: a Seção 2 apresenta o modelo comumente utilizado no cenário de TV digital atual para integração com a web, além de definir a arquitetura aqui proposta. A Seção 3 discute alguns exemplos de aplicações desenvolvidas sobre essa arquitetura e, por fim, a Seção 4 discute as conclusões e trabalhos futuros.

2. ARQUITETURA DE INTEGRAÇÃO ENTRE APLICATIVOS DE TVDI E A WEB
Atualmente existe uma infinidade de conteúdos disponíveis na web, universalmente acessíveis a partir de *browsers*. Entretanto, considerando-se o cenário de aplicativos de TVDI, vê-se que este acesso é ainda restrito e geralmente tratado caso a caso, de maneira específica para cada tipo de aplicação.

Este trabalho, desenvolvido em um contexto de iniciação científica, tem como objetivo propor uma arquitetura genérica e escalável para acesso a conteúdo web a partir de aplicativos de TVDI, particularmente para o caso de *widgets* NCLua. Genérica a fim de permitir o acesso a qualquer conteúdo web disponível, de maneira transparente a partir da plataforma Ginga-NCL, isto é, sem que o desenvolvedor da aplicação NCLua tenha que produzir código específico para cuidar de detalhes de comunicação, dentre outros. E escalável, a fim de suportar múltiplos acessos e clientes concorrentes, sem sobrecarregar o ambiente Ginga-NCL do decodificador.

Deseja-se, em particular, permitir acessos do tipo *Publish/Subscribe*, em que um cliente NCLua declara seu interesse em consumir informações fornecidas periodicamente por um dado site. Além disso, pretende-se também suportar aplicações de interação entre usuários, seja baseadas em mensagens instantâneas ou as novas redes sociais. Assim, será possível contemplar uma ampla gama das aplicações comuns da web a partir de *widgets* de TVDI.

2.1 Integração da TV Digital com Serviços da Internet
Atualmente a integração da TV Digital com serviços da internet é feita através de uma abordagem simples, que consiste em utilizar um *gateway* que faça a integração entre o aplicativo de TV Digital e o serviço desejado [5]. O *gateway* faz pleno uso da API (*Application Programming Interface*) fornecida pelo serviço da internet, a fim de gerenciar a troca de mensagens HTTP entre os ambientes de TV digital interativa e da web, funcionando como uma camada de tradução entre esses dois mundos. A Figura 1 mostra o caso de um aplicativo Ginga, transmitido por uma emissora de TV, conectando-se através do *gateway* integrador a uma rede social disponível na internet.

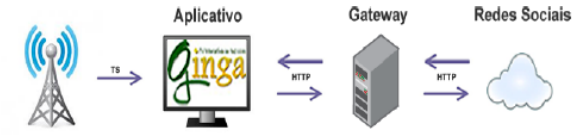

Figura 1: Arquitetura para integração entre TVDI e serviços da internet

Tal solução, além de estar fortemente atrelada à API fornecida pelo serviço, o que compromete sua aplicabilidade de forma mais genérica, padece, ainda, de um grave defeito no que tange à sua escalabilidade. Note-se que ela traz consigo o problema do gargalo no *gateway* [5], dado que muitas solicitações simultâneas irão certamente sobrecarregá-lo, consequentemente aumentando o tempo de resposta percebido pelos usuários e podendo até mesmo inviabilizar o uso da aplicação em si .

No ambiente de televisão, não se fala numa escala de dezenas de milhares de usuários, mas sim de milhões ou dezenas de milhões, daí a preocupação que se deve ter desde o início com o desempenho da arquitetura concebida como um todo.

2.2 Arquitetura Proposta

Este trabalho propõe uma arquitetura genérica e escalável para permitir que usuários de TV Digital obtenham dados da internet, utilizando-se de seus serviços mais comuns, como sites, *blogs* e redes sociais. Como já foi comentado, o tradicional modelo requisição/resposta adotado pelo HTTP mostra-se ineficaz neste cenário, pois não é capaz de sustentar a alta audiência esperada dos programas televisivos.

Como alternativa para implementação da arquitetura, foi escolhido o protocolo XMPP [1][2], acrônimo de *eXtensible Messaging and Presence Protocol*, um padrão da IETF (*Internet Engineering Task Force*) para a comunicação em tempo real através de passagem de mensagens XML. Inicialmente chamado de *Jabber*, nome da comunidade que o desenvolveu, o XMPP surgiu para suprir a falta de um protocolo aberto de passagem de mensagens instantâneas. O protocolo XMPP utiliza uma arquitetura cliente-servidor descentralizada. O sistema consiste em uma rede de servidores que se comunicam entre si, permitindo assim a comunicação entre clientes conectados a servidores diferentes. A opção pelo XMPP, neste trabalho, deu-se pelo fato desse protocolo trabalhar com padrões abertos, o que garante a interoperabilidade da arquitetura proposta com os mais variados tipos de serviços e ambientes operacionais.

No cenário aqui proposto (Figura 2), interpõe-se um servidor, executando o protocolo XMPP, entre o *widget* NCL e o serviço da internet, servidor este que irá desempenhar com vantagens as funções do gateway integrador mencionado na Seção 2.2. A aplicação (*widget*) NCL executando no decodificador (*set-top box*) da residência do telespectador registra inicialmente seu interesse por uma informação específica, disponível no servidor XMPP, através do cliente XMPP Ginga. Este estabelece uma conexão com o servidor XMPP a fim de receber as mensagens (*stanzas*) com a informação solicitada. Do outro lado, o servidor XMPP é periodicamente atualizado com dados provenientes do cliente XMPP baseado na web, que os obtém de sites previamente registrados e os repassa ao servidor XMPP.

Note-se que o servidor XMPP representa um ponto de concentração de informações na arquitetura proposta, de um lado sendo alimentado por informações obtidas pelos clientes XMPP Web e, de outro, fornecendo-as aos consumidores (clientes XMPP Ginga) interessados. Os clientes Ginga devem se registrar no servidor XMPP em modo *push*, sinalizando seu interesse em receber informações periodicamente. O servidor XMPP alivia, assim, a carga nos servidores web originais, compilando e disponibilizando a informação em um formato mais conciso, estruturado (um documento XML) e de forma mais imediata do que se fosse obtido diretamente do servidor web original. A interação com os servidores web fica, portanto, a cargo dos clientes XMPP Web, que periodicamente consultam os servidores originais. Os *widgets* NCL, por sua vez, comunicam-se com o servidor XMPP por meio do cliente XMPP Ginga, liberando-se da tarefa (e da sobrecarga) de consultar diferentes sites da web.

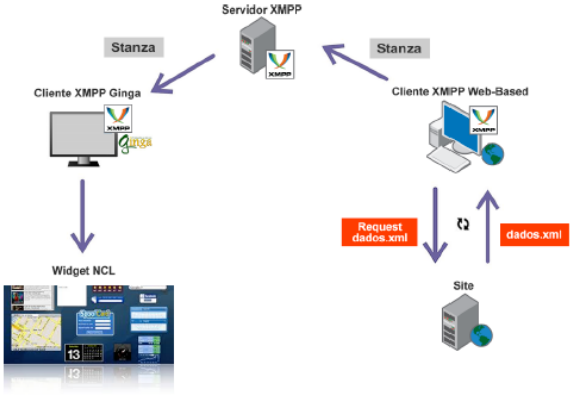

Figura 2: Arquitetura para integração entre *widgets* NCLua e redes sociais / web.

A característica genérica desta arquitetura advém do fato que qualquer site provedor de informações, que as disponibilize como *feeds*, pode utilizar a arquitetura aqui proposta para manter atualizado o seu *widget* NCL, no ambiente de TV Digital. E sua escalablidade é garantida pela presença do servidor XMPP intermediário, que pode evoluir para um cluster, se assim for necessário.

2.2.1 Cliente XMPP Web

Nessa arquitetura, o cliente XMPP Web utiliza técnicas de AJAX (*Asynchronous Javascript and XML*) para fazer requisições automáticas, de tempos em tempos, visando obter o *feed* com dados atualizados do site provedor de informações.

O cliente XMPP Web utiliza um método de conexão que não requer a manutenção da conexão TCP aberta por muito tempo, embora o padrão para uma rede de passagem de mensagens seja manter a conexão TCP aberta enquanto o usuário estiver *on-line*. Esta otimização no tempo de uso das conexões é possível graças à opção feita pelo BOSH (*Bidirectional-streams Over Synchronous HTTP*) [6][4][3], um protocolo de extensão do XMPP.

Sistemas baseados em BOSH utilizam um gerenciador de conexões que atua como um tipo de *proxy* entre o cliente e o servidor XMPP [4]. Assim, a cada resposta obtida do site, o cliente XMPP Web encapsula o *feed* em uma *stanza* XMPP e a envia, encapsulada em um pacote HTTP, para o gerenciador de conexões, que extrai essa *stanza* e a repassa ao servidor. O servidor se encarrega de enviar esta *stanza* a todos os usuários interessados conectados a ele.

2.2.2 *Cliente XMPP Ginga*

No contexto da TV Digital, ao se ligar a televisão, o aplicativo abre um *socket* de conexão com o servidor XMPP e todo um processo de negociação e autenticação é executado para o cliente adquirir o direito de receber as *stanzas* de mensagens lá armazenadas.

Após a negociação e autenticação, o aplicativo fica à espera das mensagens enviadas pelo cliente XMPP Web, que alimenta o servidor XMPP com informações da web. Ao receber as *stanzas* de mensagem, o cliente Ginga extrai os dados da mensagem, trata-os e por fim os exibe, quando solicitado, através do *widget* NCLua no aparelho de TV.

3. EXEMPLOS DE APLICAÇÕES

3.1 Clima Tempo

Uma aplicação que acessa o site Clima Tempo[1] foi desenvolvida utilizando a arquitetura proposta, como forma de validá-la com um exemplo realista. O site do Clima Tempo disponibiliza um *feed* em formato XML que foi utilizado para alimentar o *widget* com informações sobre a previsão da tempo em todas as capitais do país.

A aplicação do clima tempo se encaixa perfeitamente na arquitetura, pois o *feed* do site é atualizado uma vez por dia, com isso foi possível desenvolver o cliente XMPP Web para enviar os dados ao *widget* na mesma frequência do site. Entretanto, esta abordagem obriga que o cliente XMPP Web tome a iniciativa de consultar o site e, caso isso tenha que ser feito várias vezes, pode acabar gerando uma sobrecarga desnecessária pelas muitas mensagens de *polling* e resposta. Um modo de diminuir essa sobrecarga por parte do cliente XMPP Web é utilizar um protocolo de extensão do XMPP do tipo *Publish/Subscribe* [7][4][3]. Ele consiste em um cliente que atua como um canal de notificação em tempo real, ou seja, é um nó que publica informações, onde qualquer pessoa pode assinar esse canal para receber as informações desejadas.

No caso do cliente web convencional, a freqüência de requisições ao servidor pode se tornar alta, gerando uma diferença significativa de desempenho se muitos clientes estiverem ativos simultaneamente [4]. Já com o uso do modelo *Publish/Subscribe*, os clientes irão receber as informações (por meio de uma notificação) quando o site publicador as disponibilizar, economizando assim processamento e largura de banda.

3.2 Google Talk

A arquitetura descrita neste artigo não suporta somente aplicativos do tipo *Publish/Subscribe*, podendo ser utilizada também para aplicações interativas. Como estudo de caso, decidiu-se utilizar o Google Talk, serviço de mensagens instantâneas da Google, cujo código está disponibilizado para desenvolvedores[2].

Para tanto, algumas modificações se fazem necessárias. O servidor do Google Talk toma o lugar do site provedor de informações e o cliente XMPP Web continua sendo o mesmo, só que com uma funcionalidade diferente, pois passa a funcionar como um *gateway*, repassando mensagens entre o servidor XMPP e o serviço do Google Talk. O cliente XMPP Web, para atuar como *gateway*, conecta-se com dois servidores XMPP, o do GTalk e o nosso servidor XMPP intermediário, logo para uma mensagem chegar ao seu destino, esta precisa passar por pelo menos dois servidores.

Dessa forma, utilizando a arquitetura de integração, um usuário dos sistemas do Google poderá se conectar à rede Google Talk por meio de um *widget* da TV Digital interativa. Note que, para o *widget* NCL, não faz nenhuma diferença se as informações recebidas do servidor XMPP são mensagens instantâneas ou provenientes de um site como o Clima Tempo, o que demonstra o caráter genérico da arquitetura aqui detalhada.

4. CONCLUSÃO

Este trabalho discutiu e implementou uma arquitetura genérica para integração entre aplicativos de TV digital interativa e serviços comuns da internet, visando assim suprir uma demanda geral por acesso a conteúdo e aplicações da internet a partir da nova plataforma Ginga-NCL de TVDI.

A solução arquitetural apresentada utiliza-se de um servidor executando o protocolo XMPP, atendendo aos critérios de generalidade e escalabilidade colocados como premissa inicial deste trabalho. Um conjunto de aplicativos foi desenvolvido a fim de validar a funcionalidade da arquitetura, sendo aqui destacados dois deles, um segundo o modelo *publish/subscribe* e outro, de mensagens instantâneas. Ambos foram implementadas como *widgets* NCLua, com resultados bastante satisfatórios.

Como trabalhos futuros, pretende-se investigar algumas variações na arquitetura, com vistas a melhorar ainda mais seu desempenho e escalablidade. E ainda, deseja-se realizar sua integração com o Facebook e Twitter, duas das redes sociais mais populares no momento, ambas com APIs livremente disponibilizadas na web.

REFERÊNCIAS

[1] SAINT-ANDRE, Peter. “Extensible Messaging and Presence Protocol (XMPP): Core”. 10/2004. http://xmpp.org/rfcs/rfc3920.html.

[2] SAINT-ANDRE, Peter. “Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence”. 10/2004. http://xmpp.org/rfcs/rfc3921.html.

[3] MOFFITT, Jack. Professional XMPP Programming with Javascript and JQuery. Wrox. 2010.

[4] SAINT-ANDRE, Peter; SMITH, Kevin; TRONÇON, Remko. XMPP: The Definitive Guide. O’Reilly. 2009.

[5] C. GHISI, Bruno; F. LOPES, Guilherme; SIQUEIRA, Frank. Integração de Aplicações para TV Digital Interativa com Redes Sociais. WEBMEDIA 2010. Belo Horizonte – MG. 2010.

[6] PATERSON, Ian; SMITH, Dave; SAINT-ANDRE, Peter; MOFFITT, Jack. [XEP-0124] Bidirectional-streams Over Synchronous HTTP (BOSH). http://xmpp.org/extensions/xep-0124.html.

[7] MILLARD, Peter; SAINT-ANDRE, Peter; MEIJER, Ralph. [XEP-0060] Publish-Subscribe. http://xmpp.org/extensions/xep-0060.html.

[1] http://www.climatempo.com.br/

[2] http://code.google.com/intl/pt-BR/apis/talk