UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIAS
CURSO DE CIÊNCIA DA COMPUTAÇÃO

WENNER BAIMA MUNIZ

Uma arquitetura para o desenvolvimento de jogos com múltiplos jogadores na plataforma Ginga

São Luís
2012

WENNER BAIMA MUNIZ

Uma arquitetura para o desenvolvimento de jogos com múltiplos jogadores na plataforma Ginga

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal do Maranhão como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Mário Antônio Meireles Teixeira

Doutor em Ciência da Computação – USP

São Luís

2012

Muniz, Wenner Baima.

Uma arquitetura para o desenvolvimento de jogos com múltiplos jogadores na plataforma Ginga / Wenner Baima Muniz – 2012.

Orientador: Mário Antônio Meireles Teixeira.

Monografia (Graduação) – Universidade Federal do Maranhão, Curso de Ciência da Computação, 2012.

1.Ciência da Computação 2.TV Digital 3.NCLua 4.Arquitetura orientada a eventos 5.Xadrez. I.Título.

WENNER BAIMA MUNIZ

Uma arquitetura para o desenvolvimento de jogos com múltiplos jogadores na plataforma Ginga

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal do Maranhão como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 12/07/2012

BANCA EXAMINADORA

Prof. Dr. Mário Antônio Meireles Teixeira (Orientador)

Prof. Msc. Geraldo Braz Júnior

Prof. Dr. Alexandre César Muniz de Oliveira

Ao meu pai, Wllisses Furtado Muniz, e à minha falecida avó, Maria José Furtado Muniz.

AGRADECIMENTOS

Agradeço primeiramente à minha família por todo o apoio e incentivo. Em especial, ao meu pai Wllisses Furtado Muniz por todo o apoio e à minha falecida avó Maria José Furtado Muniz por toda a educação que me foi depositada. A todos os professores e funcionários do Departamento de Informática e, em particular, ao meu orientador Prof. Mário Antônio Meireles Teixeira por todos os ensinamentos e confiança passados durante a elaboração desse trabalho. A todos os meus amigos que de alguma forma contribuíram para a conclusão dessa monografia.

“Temos o destino que merecemos. Nosso destino está de acordo com os nossos méritos.”
(Albert Einstein)

RESUMO

A TV Digital Brasileira ganhou projeção nacional no final de 2007 quando o sinal digital começou a ser distribuído na região metropolitana de São Paulo. Desde então, estima-se que até o ano de 2016 todo o território nacional esteja recebendo sinais digitais de TV. O atual cenário do Sistema Brasileiro de TV Digital representa uma oportunidade para estimular o desenvolvimento de novos serviços e aplicações e, dentro desse contexto, esta monografia apresenta uma solução para prover meios de interatividade e inclusão social a partir de uma plataforma compatível com o SBTDV. Esta solução fornece um suporte ao desenvolvimento de jogos para múltiplos jogadores no ambiente Ginga, utilizando-se de um protocolo aberto para comunicação em tempo real e baseada em uma arquitetura orientada a eventos do tipo cliente-servidor.

Palavras-chave: TV Digital, inclusão social, Ginga, Arquitetura orientada a eventos.

ABSTRACT

The Brazilian Digital TV got national prominence in late 2007 when the digital signal began to be distributed in the metropolitan area of Sao Paulo. Since then, it is estimated that by the year 2016 all the country will be receiving digital TV signals. The current scenario of Brazilian System of Digital TV represents an opportunity to stimulate the development of new services and applications and, within this context, this thesis presents a solution to provides a means of interactivity and social inclusion as a supported platform for SBTVD. This solution provides a support for developing multiplayer games in the Ginga environment, using an open protocol for real-time communication and based on an event-driven architeture client-server.

Keywords: Digital TV, social inclusion, Ginga, event-driven architetury.

SUMÁRIO

LISTA DE FIGURAS

1. INTRODUÇÃO

Com o avanço da tecnologia, a televisão teve de seguir a tendência mundial de convergência digital. Nesse cenário, novos recursos são disponibilizados como, por exemplo, a possibilidade de interação entre o usuário e a informação que está sendo mostrada, a partir de aplicações desenvolvidas para esse fim.

Na última década, a economia brasileira presenciou uma evolução significativa no mercado de jogos eletrônicos sendo este responsável pelo movimento de aproximadamente 87 milhões de reais somando-se hardware e software, de acordo com pesquisa realizada pela Associação Brasileira de Desenvolvedores de Jogos Eletrônicos (Abragames). Ainda, a pesquisa mostrou que a produção de software de jogos no Brasil vem crescendo, em média, 30% ao ano e, mundialmente falando, o segmento de jogos eletrônicos já ultrapassou o faturamento da indústria do cinema (Pesquisa Abragames, 2008).

O cenário atual de implantação do Sistema Brasileiro de Televisão Digital (SBTVD) representa uma grande oportunidade para estimular o desenvolvimento científico e tecnológico brasileiro, além de prover um aproveitamento industrial e comercial. Dentre as estratégias e diretrizes do decreto que institui o SBTVD, podemos destacar o incentivo à indústria local no que diz respeito ao desenvolvimento de aplicações digitais. Nesse aspecto, o que aparenta ser apenas uma adaptação do desenvolvimento de aplicações de um ambiente para outro, ou seja, do computador para a TV Digital, mostra-se bem diferente em se tratando da estrutura de funcionamento, recursos utilizados e metodologia de implementação, em especial a interação do usuário.

A inserção do SBTVD no Brasil pretende ainda introduzir alguns conceitos-chaves: a conectividade, a portabilidade e, principalmente, a interatividade, sendo este último o conceito onde o desenvolvimento de jogos mais tem destaque dentre todos os outros tipos de aplicações. Todavia, a construção de aplicações mais complexas e dinâmicas para o SBTVD ainda é considerada um desafio visto que essa é uma tecnologia relativamente nova.

A televisão é um meio de comunicação que possui um grande poder de penetração em massa. Com a adoção da TV Digital, os telespectadores podem usar a televisão como plataforma para jogos interativos. Nesse contexto, o desenvolvimento de jogos se torna um tópico de grande interesse à medida que eles podem auxiliar no processo de aceleração da inclusão digital e social, proporcionando entretenimento e informação a

uma parcela dos brasileiros que ainda não têm acesso à internet através dos computadores tradicionais.

O objetivo do presente trabalho é fazer uso da TV Digital para potencializar e acelerar a produção de jogos interativos para o SBTVD. A utilização da televisão como um meio de difusão e penetração dá a garantia de que uma enorme parcela da população tenha acesso a essa forma de interatividade, sendo que a ampliação do acesso à internet nas residências assim como a ampliação do sinal digital de TV são os requisitos necessários para possibilitar aos telespectadores interagirem com a proposta do trabalho.

Para que tal objetivo fosse atingido, foi necessário efetuar um estudo do protocolo XMPP e da linguagem NCL, assim como de sua linguagem de *script* Lua, analisando as características e estrutura de cada um. A partir daí, foi desenvolvida uma arquitetura baseada no modelo cliente-servidor para dar suporte ao desenvolvimento de jogos de tabuleiro para o SBTVD. Também, foi desenvolvido um protótipo de um jogo de xadrez para validar a arquitetura.

Esta monografia está dividida em 5 capítulos. O Capítulo 2 descreve alguns trabalhos relacionados. No Capítulo 3 será apresentada uma breve descrição acerca do Sistema Brasileiro de Televisão Digital bem como os conceitos básicos e características do middleware Ginga e das linguagens NCL e Lua. O Capítulo 4 aborda uma visão geral sobre os Serviços Web detalhando as características da sua linguagem base, o XML, e também um descrição do protocolo XMPP. Ainda nesse capítulo, a arquitetura proposta é definida. O Capítulo 5 apresenta o protótipo de uma aplicação *multiplayer* que simula um jogo de xadrez para o SBTVD. Finalmente, o capítulo 6 descreve a conclusão do trabalho além de sugestões para futuros trabalhos.

2. SBTVD E *MIDDLEWARE* GINGA

Há mais de meio século, os sinais da TV aberta são transmitidos de forma analógica. Desde a primeira emissora de TV, a BBC (*British Broadcasting Corporation*) de Londres, fundada em 1936, a televisão já passou por várias mudanças, que vai desde a cor até o aumento da quantidade de canais. Este último acarretou o surgimento do controle remoto, aumentando o conforto do telespectador a partir do momento que dispensou a locomoção do mesmo; nascia o primeiro componente digital integrado aos aparelhos receptores de sinal de TV.

O Brasil atualmente passa por uma fase de transição, onde seu sistema de televisão analógica vem sendo substituído por um sistema de televisão digital. Em diversas cidades a transmissão de sinal digital já está ativa e modelos de decodificadores cada vez mais sofisticados podem ser encontrados no mercado. Estima-se que até 2016 todos os telespectadores já estejam recebendo sinais digitais em seus decodificadores, extinguindo, dessa forma, o sinal analógico.

Instituído pelo Decreto 4.901, de 26/11/2003, o Sistema Brasileiro de Televisão Digital tem a finalidade de proporcionar uma melhora significativa na qualidade de áudio e vídeo, além de por um fim nos chamados "chuviscos" e "chiados". Estes efeitos que afetam a imagem são causados por conta de interferências e ruídos que agem sobre o sinal original. Em sinais digitais, é possível aplicar-se técnicas capazes de corrigir e restaurar esse sinal tal como ele foi transmitido a partir da fonte. Mas tal aplicação só é possível caso a taxa de erros provenientes das interferências não ultrapasse um certo limite (limiar), caso contrário o uso dessas técnicas poderia, além de ser completamente ineficaz, acarretar mais erros no sinal. Por isso é comum se dizer que o sinal digital é sempre perfeito.

A TV Digital é uma evolução tecnológica da TV Analógica. Além de trazer imagens em alta definição com som de alta qualidade, abre portas para o desenvolvimento de novos serviços e aplicações. É nesse cenário que entra uma das grandes inovações da Televisão Digital: a interatividade. Interatividade é a relação que se estabelece entre os campos de produção e da recepção, onde as audiências passam a interagir, em diferentes níveis, com os produtores de televisão, podendo participar, interferir ou comentar os programas (Barbosa e Castro, 2008). A interatividade permitirá ao telespectador experimentar um novo modo de assistir televisão podendo agora participar diretamente da programação que está sendo exibida através de aplicações

disponibilizadas paralelamente à transmissão de sinal.

Com o surgimento das novas oportunidades frente à televisão, ocorre em paralelo uma imensa revolução entre os tipos de telespectadores, pois surgem os telespectadores ativos, ou seja, o modelo de transmissão digital cria um novo modelo de relacionamento com seu público (Flores et al , 2008).

Neste capítulo, a Seção 2.1 aborda as características do *middleware* Ginga que torna possível a interatividade no SBTVD; a Seção 2.2 descreve a linguagem NCL; na Seção 2.3 discute-se a linguagem imperativa lua, adotada como a linguagem de *script* do NCL no Ginga; e finalmente na Seção 2.4 é mostrado como é feita a integração de NCL e Lua.

2.1 O *middleware* Ginga

Middleware é o nome dado à camada de *software* localizada entre o código das aplicações e a infraestrutura de execução. Sua função é oferecer o suporte necessário para o rápido e fácil desenvolvimento das aplicações interativas fornecendo, assim, uma abstração dos diferentes tipos de sistemas operacionais e arquiteturas de hardware, além de esconder os detalhes das camadas inferiores. Um middleware é uma classe de *software* desenvolvida para gerenciar a complexidade e heterogeneidade inerentes ao sistema (Bakken, 2001).

Ginga[1] é o nome do *middleware* aberto do Sistema Brasileiro de TV Digital. Sendo compatível com as recomendações ITU-T[2], o Ginga foi desenvolvido levando em consideração as últimas inovações tecnológicas e as necessidades de inclusão digital no país, objetivos estes que não seriam alcançados caso fosse adotado qualquer um dos *middlewares* já existentes. As aplicações executadas sobre ele são classificadas em duas categorias: declarativas e procedurais. Aplicações desenvolvidas em linguagem declarativa utilizam o subsistema Ginga-NCL e as aplicações desenvolvidas em linguagem procedural utilizam o subsistema Ginga-J. Além disso, existe um terceiro subsistema, o Ginga-CC (*Common Core*), que fornece todas as funcionalidades comuns ao suporte dos ambientes declarativo e procedural. A Figura 2.1 mostra a arquitetura do *middleware* Ginga.

1 O termo Ginga está associado ao movimento e a atitude que os brasileiros possuem para tudo que fazem. Por isso foi adotado o nome Ginga para o middleware brasileiro.

2 Conjunto de recomendações publicadas pela ITU (International Telecommunication Union) com o propósito de promover a interatividade.

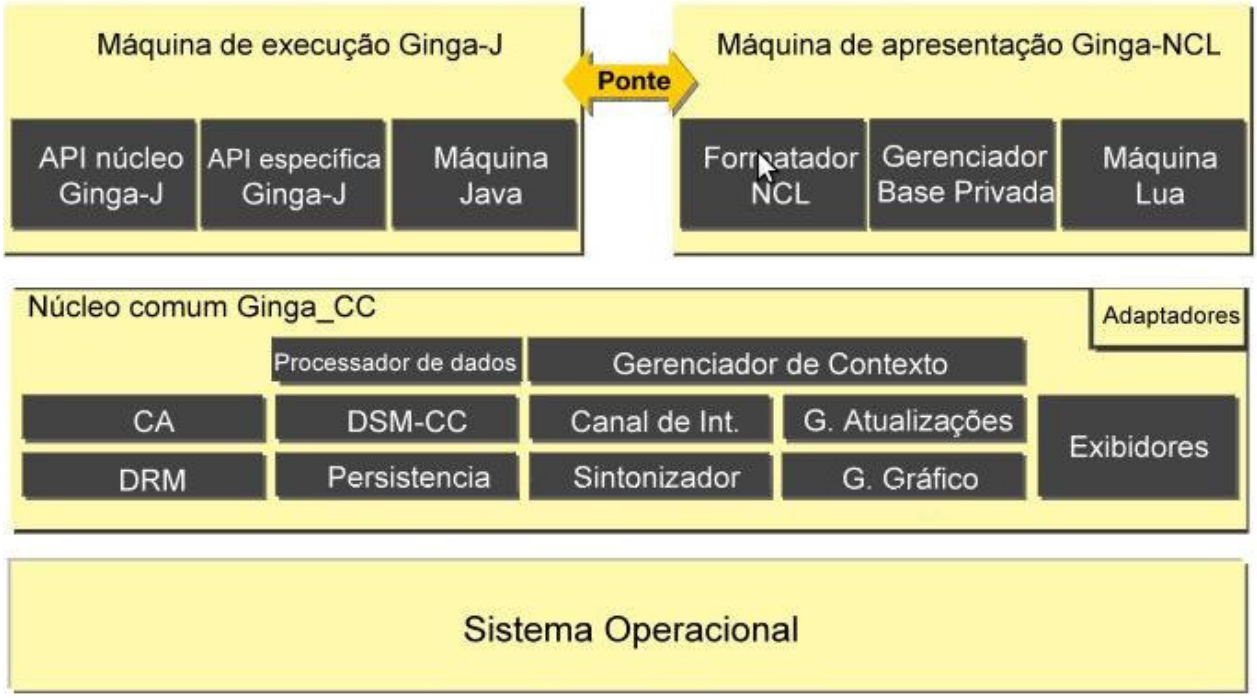

Figura 2.1: Arquitetura do *middleware* Ginga (Middleware Ginga, 2010).

2.1.1 Ginga-CC

O Ginga-CC (Ginga-Common Core ou Ginga-Núcleo Comum) é uma camada que oferece suporte aos dois ambientes existentes no Ginga. Se encontra entre os ambientes e o sistema operacional, fazendo com este uma interface direta resultando em uma ponte estreita com o hardware. Tem a responsabilidade de tratar a exibição dos vários objetos de mídia, tais como JPEG, MPEG-4, MP3, GIF, entre outros formatos.

Ainda na Figura 2.1, são mostrados seus módulos. O Gerenciador Gráfico fica encarregado de definir o modelo pelo qual as mídias serão apresentadas. O módulo Gerenciador de Atualizações é responsável por manter o funcionamento do *middleware* impedindo que ocorra interrupções quando houver possíveis atualizações. O Gerenciador de Contexto fornece informações referentes ao perfil do usuário, como sua localização, por exemplo, às aplicações para que haja adaptação de conteúdo nas mesmas. O canal de interatividade, também conhecido como canal de retorno, é o módulo responsável por controlar o acesso à camada de rede. O Sintonizador, como sugere o nome, encarrega-se da sintonização dos canais de radiofrequência. Os módulos DRM (Digital Right Management) e CA (Conditional Access) especificam o acesso às mídias da aplicação de TV Digital. Processador de Dados e DSM-CC (Digital storage media command and control) capturam o fluxo MPEG-2 e disponibilizam ao sistema. O módulo Persistência fica com o cargo de gerenciamento dos dados armazenados necessários à aplicação.

Finalmente aos Exibidores cabe a interpretação de cada um dos vários tipos de mídia já citados.

2.1.2 Ginga-J

O ambiente procedural Ginga-J foi inicialmente desenvolvido pela UFPB e utiliza a linguagem Java como ferramenta de autoria para aplicações nessa abordagem. Um diferencial que o Ginga-J possui é a utilização da API[3] Java DTV, que foi inserida recentemente ao conjunto de API's do ambiente substituindo o padrão GEM (Globally Executable MHP). O objetivo da API é prover uma solução livre de royalties e permitir a produção de aparelhos de TV e conversores por um custo mais acessível.

O Ginga-J é subdividido em três módulos como mostrado na Figura 2.1. O módulo API núcleo é composto pelas API's verdes que são responsáveis por manter a compatibilidade com outros padrões de TV Digital. A API específica por sua vez é dividida em amarelas e vermelhas. As API's específicas amarelas são compatíveis com outros sistemas através de um adaptador e fornecem algumas funcionalidades como múltiplas redes, múltiplos usuários e múltiplos dispositivos, dentre outras facilidades. As API's específicas vermelhas fornecem facilidades para integração de um conteúdo declarativo e outro procedural numa mesma aplicação; possuem caráter inovador e são exclusivos do SBTVD.

2.1.3 Ginga-NCL

O Ginga-NCL foi criado pela PUC-Rio para prover uma infraestrutura de apresentação para aplicações hipermídia/multimídia de acordo com o modelo declarativo, e utiliza a linguagem NCL (*Nested Context Language*) como ferramenta de autoria das mesmas. É dividido em três módulos: O Formatador NCL, o Gerenciador de Base Privadas e a Máquina Lua.

O Formatador NCL é o principal componente e seu papel consiste em direcionar todas as ações a serem executadas. É o responsável por receber o documento NCL e controlar sua apresentação, fazendo com que as relações de sincronismo entre os objetos de mídia existentes sejam respeitadas. Ele recebe um documento, converte-o em uma estrutura chamada "base privada", armazena-o e inicia sua execução.

O módulo gerenciador de Bases Privadas tem como principal função armazenar a estrutura processada pelo Formatador NCL. Também é de sua responsabilidade receber

3 Application Programming Interface.

os comandos para ativação e manipulação das aplicações. Usando um conjunto de comandos de edição, uma aplicação pode ser gerada ou modificada em tempo real (ao vivo). Esse conjunto é dividido em três grupos: o primeiro grupo de comandos tem a responsabilidade de ativar ou desativar uma base privada, ou seja, comanda a habilitação de aplicações de um determinado canal de TV. O segundo grupo tem a função de iniciar, desativar, pausar e retomar uma aplicação NCL. O
terceiro fica com os comandos de edição do conteúdo em tempo de execução, ou seja, modificações de uma aplicação ao vivo.

A máquina Lua é o último módulo do Ginga-NCL e é responsável pela execução dos *scripts* escritos na linguagem imperativa Lua. Por ser rápida, leve, eficiente e exatamente projetada para estender aplicações, além de ser sete vezes mais rápida que a ECMAScript (linguagem de *script* adotada pela maioria dos outros padrões) e fazer um uso quarenta vezes menor de memória[4], Lua foi escolhida a linguagem de *script* ideal para essa abordagem declarativa.

Assim como o Ginga-J, o Ginga-NCL dá suporte a exibição de conteúdo em múltiplos dispositivos de entrada e saída. O uso da linguagem NCL e sua linguagem de *script* lua é uma das inovações totalmente brasileiras do SBTVD, assim como os comandos de edição do conteúdo ao vivo. A facilidade da abordagem declarativa oferece suporte para o grande domínio de aplicações interativas para TV Digital como, por exemplo, as aplicações para as chamadas TV em comunidade (Social TV ou Community) onde torna-se possível a criação de conteúdo por parte do usuário, ou seja, uma comunidade compartilha conteúdos, e cria novos sobre as aplicações recebidas, em tempo real ou sob demanda (Santos, 2010).

2.2 A linguagem NCL

NCL é uma linguagem declarativa destinada à autoria de documentos hipermídia. Baseada no modelo conceitual NCM (Nested Context Model) ela trás uma separação clara entre os conteúdos de mídia e a estrutura da aplicação, ou seja, um documento NCL apenas define como os objetos de mídia são estruturados e relacionados no espaço e no tempo. Por ser uma linguagem de colagem, tem a função de reunir conteúdos de mídia em uma apresentação multimídia, onde estes podem ser de qualquer tipo: áudio, vídeo, imagem, um documento HTML ou até mesmo um objeto procedural.

4 Pode-se fazer um teste comparativo no portal http://shootout.alioth.debian.org/.

Genericamente, um documento hipermídia é composto por nós e elos. As mídias utilizadas no documento são representadas pelos nós, sendo estes responsáveis por trazerem informações sobre sua apresentação. Os elos por sua vez fazem a sincronização espacial e temporal dos nós que formam o documento.

O NCM (Soares e Rodrigues, 2005), também conhecido como Modelo de Contextos Aninhados, representa os componentes acima citados de uma forma mais ampla. O modelo estende o conceito de nó em dois tipos: nó de conteúdo e nó de composição. O primeiro traz informações referentes a uma mídia que compõe o documento NCL. O segundo é formado por um conjunto de nós de conteúdo e/ou outros nós de composição e é responsável pela estrutura e organização do documento. Um documento hipermídia é construído levando-se em consideração algumas informações básicas. É preciso indicar o que deve ser apresentado, como, quando e onde.

Em um documento hipermídia, deve-se definir uma área onde um nó será exibido. Os componentes denominados regiões são encarregados de desempenhar tal papel indicando o tamanho e a posição da área onde o nó será apresentado, independente do seu tipo, seja de conteúdo ou de composição. Desse modo, o documento NCL define a posição inicial do elemento (vídeo, texto, imagem, etc). A Figura 2.2 ilustra uma região.

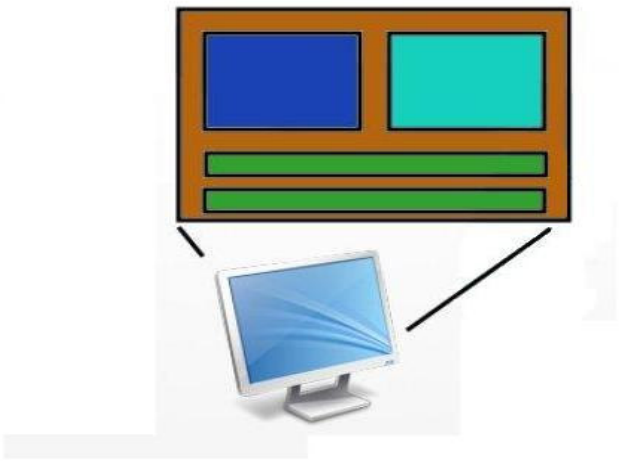

Figura 2.2: Ilustração de uma região (Carvalho et al, 2009).

Para que uma região seja definida, é necessário que ela seja associada a outras informações que indicam como o nó será exibido. Os descritores são responsáveis por definir como os conteúdos de mídia são apresentados nas áreas da tela indicando a região a qual ele está associado. Com isso pode-se descrever especificidades sobre a apresentação das mídias, assim como a região da tela onde serão mostradas,

transparência, tempo de duração de sua exibição, etc. A Figura 2.3 ilustra um descritor.

Figura 2.3: Ilustração de um descritor (Carvalho et al, 2009).

Cada nó de um documento hipermídia é representado por elementos que guardam seu conteúdo denominados mídias. Assim como um descritor está associado a uma região, uma mídia por sua vez está associada a um descritor. De acordo com o modelo NCM, uma mídia deve necessariamente estar dentro de um nó do tipo composição, denominado contexto. Um contexto é usado para representar um documento completo ou somente uma parte. A Figura 2.4 ilustra mídias e contextos.

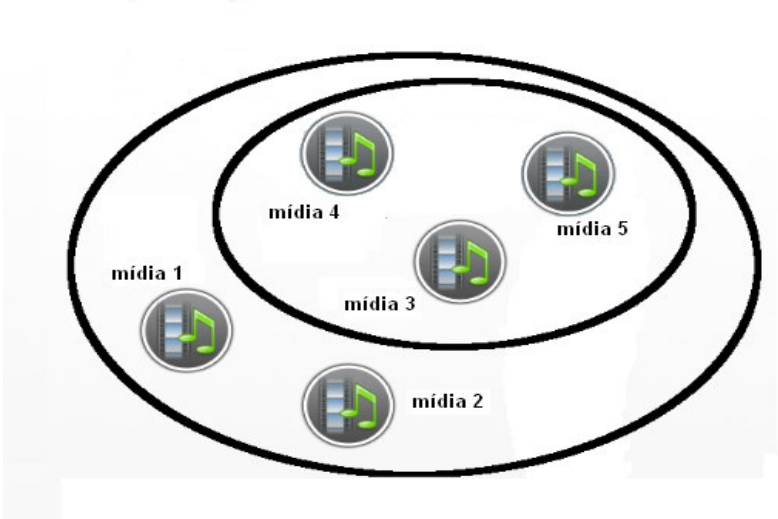

Figura 2.4: Ilustração de mídias e contextos (Carvalho et al, 2009).

Após serem definidos os nós que darão forma ao documento hipermídia, se faz necessário definir a ordem de apresentação dos mesmos, ou seja, precisa-se dizer qual será o primeiro nó a ser exibido, o segundo, e assim por diante. Para isso são usados os elementos denominados portas e links. Quando iniciado um nó de contexto, as portas ficam com a responsabilidade de definir quais nós serão mostrados e os links ditam os

relacionamentos de sincronização entre os nós definindo, desse modo, a interatividade da apresentação. A Figura 2.5 ilustra portas e links.

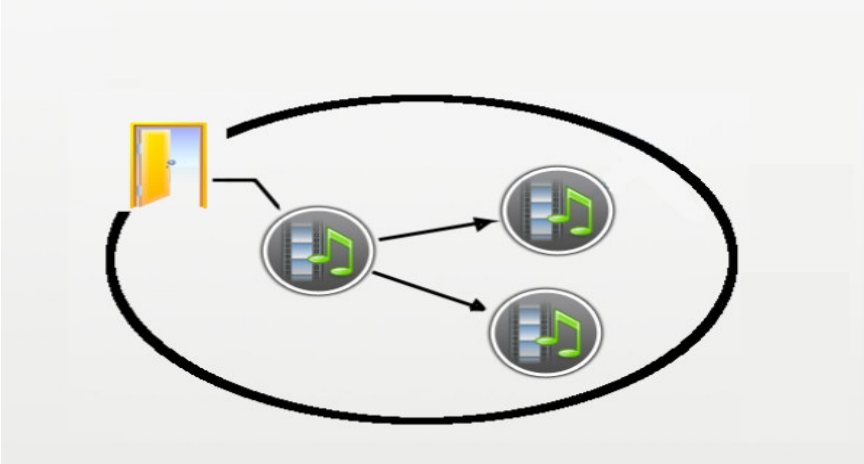

Figura 2.5: Ilustração de portas e links (Carvalho et al, 2009).

2.2.1 Estrutura de um documento NCL

Um documento hipermídia NCL é uma aplicação XML (Extensible Markup Language) que segue a abordagem de modularização. Um módulo é uma coleção de elementos, atributos e valores de atributos XML relacionados semanticamente que representam uma unidade de funcionalidade (Soares et al, 2006).

Todo o conteúdo de um documento NCL é definido dentro do elemento <ncl>. Sua estrutura é dividida em duas partes: cabeçalho e corpo do documento. No primeiro, são definidas as características da apresentação do documento; no segundo são definidos os elementos de mídia e a sincronização entre eles. Tomemos como base a Figura 2.6 para exemplificação e descrição da estrutura básica da linguagem NCL. A estrutura descreve a apresentação de um vídeo que é mostrado na tela durante um curto intervalo de tempo sobreposto a um outro vídeo.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="nclClicks" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">

<head>

<regionBase>
    <region width="800"  height="800"  right="560" top="0" id="rg_video_1"/>
    <region width="20%"  height="20%"  left="10%"  top="40%" id="rg_video_2"/>
</regionBase>

<descriptorBase>
    <descriptor id="ds_video_1" region="rg_video_1"/>
    <descriptor id="ds_video_2" region="rg_video_2"/>
</descriptorBase>

<connectorBase>
    <causalConnector id="onBeginStart">
        <simpleCondition role="onBegin"/>
        <simpleAction role="start"/>
    </causalConnector>
    <causalConnector id="onEndStop">
        <simpleCondition role="onEnd"/>
        <simpleAction role="stop"/>
    </causalConnector>
</connectorBase>

</head>

<body>
    <port id="pr_video_1" component="video_1"/>

    <media id="video_2" src="video_2.avi" descriptor="ds_video_2">
    <media id="video_1" src="video_1.avi" descriptor="ds_video_1">
        <area id="ar_video_1" begin="5s" end="15s"/>
    </media>

    <link id="link_ini" xconnector="onBeginStart">
        <bind role="onBegin" component="video_1" interface="ar_video_1"/>
        <bind role="start" component="video_2"/>
    </link>
    <link id="link_fim" xconnector="onEndStop">
        <bind role="onEnd" component="video_1" interface="ar_video_1"/>
        <bind role="stop" component="video_2"/>
    </link>
</body>
</ncl>
```

Figura 2.6: Estrutura básica de um documento NCL.

O cabeçalho é representado pelo elemento <head>. Nele são definidas as regiões (<region>), os descritores (descriptor) e os conectores. As regiões ditam posicionamente e dimensão através de seus atributos *width, height, left e top,* além de outros como o *z-Index* usado na ordenação do empilhamento das regiões. Um elemento <region> sempre deve estar dentro de sua base <regionBase>. O elemento <descriptor> é responsável por determinar de que forma a mídia será exibida na tela usando os atributos *explicitDur,*

focusBorderColor, focusBorderWidth , moveUp, moveDown, etc. Além disso, utiliza também o atributo *region* para especificar a que região o descritor está associado. Sua base é definida pelo elemento <descriptorBase>. Por fim, os conectores são definidos dentro de sua base <connectorBase> e servem para estabelecer os relacionamentos entre as mídias.

No corpo do documento são encontrados os nós de conteúdo, os nós de composição e os elos. Como já mencionado, as mídias são descritas pelos nós de conteúdo, e são representadas pelo elemento <media>. Fazem uso de alguns atributos como *src* que faz uma referência explícita à mídia, *descriptor* para referenciar um descritor previamente declarado e *type* para definir o tipo da mídia (*audio, video, text, etc*). As mídias possuem um elemento chamado âncora (<area>) que é utilizado para permitir a sincronização com outros objetos de mídia. Uma mídia define o elemento <area> e seus atributos *begin* e *end* servem para dizer o intervalo em que a mídia, respectivamente, inicia e finaliza sua apresentação na tela. Também é definida a mídia principal da composição, papel exercido pelo elemento porta (<port>). Este usa o atributo *component* para fazer referência ao objeto de mídia definindo-o como a mídia principal, ou seja, aquela que será chamada primeiro no começo da apresentação.

Por fim, ainda no corpo do documento, os relacionamentos são especificados através dos elos (<link>) que usam o atributo *xconnector* para definir a ação a ser tomada a partir de uma referência a um conector. Um elo possui um elemento filho identificado por <bind> que define os objetos de mídia participantes da relação usando o atributo *component* e o papel exercido pelos mesmos com o uso do atributo *role.*

Assim, baseado no exemplo da Figura 2.6, o elemento <port> chama a mídia principal, "video_1", que possui uma âncora. O elo "link_ini" determina que no intervalo 5s da mídia principal a mídia secundária, "video_2", deve ser exibida na tela. Igualmente, o outro elo, "link_fim", determina que no intervalo 25s a mídia secundária deve ter sua exibição encerrada em decorrência do fim da âncora.

2.3 A linguagem Lua

Lua é uma linguagem de *script* criada em 1993 na PUC-Rio pelo Grupo de Tecnologia em Computação Gráfica (Tecgraf), sendo desenvolvida por brasileiros. É uma linguagem de programação leve, porém poderosa, projetada para estender aplicações onde combina sintaxe simples para programação procedural com poderosas construções para descrição de dados baseadas em tabelas associativas e semânticas extensíveis

(Ierusalimschy et al, 2007). Como mencionado anteriormente, Lua é utilizada como a linguagem de *script* de NCL.

Atualmente Lua vem se destacando como a linguagem mais usada para *scripting* em jogos e é parte do ambiente Ginga-NCL do Sistema Brasileiro de TV Digital. Lua apresenta as seguintes características:

- **Interpretação dinâmica:** significa dizer que é possível executar trechos de códigos que são criados dinamicamente, ou seja, durante a execução do programa.
- **Tipagem dinâmica forte:** os tipos das variáveis são verificados em tempo de execução do programa, ou seja, não é necessária ser feita a declaração de tipos no código do programa.
- **Gerência automática de memória dinâmica:** também chamada de "coleta de lixo", significa que não há necessidade de ser feito o gerenciamento de memória de forma explícita no código.

Por ser muito simples, Lua reduz muito o trabalho dos programadores, podendo ser utilizada em testes de algoritmos antes de suas implementações definitivas (Celles et al, 2003). A partir da Figura 2.7 podemos fazer algumas análises para entender a estrutura básica de um *script* em Lua.

```
local tabela_valores = {1, 2, 3, 4, 5}

function verifica(valor)
        if numero % 2 == 0 then
                print("Este número é par!")
        end
end

duplic = funtion(valor)
        valor = valor * 2
        return valor
end

for i, v ipairs(tabela_numeros) do
        verifica(numero)
end
```

Figura 2.7: Parte de um código em Lua.

Quando uma variável qualquer é declarada, esta sempre terá escopo global a não ser que o autor do código utilize a palavra-chave *local* antes da mesma. Esta decisão pode parecer um pouco relevante mas na realidade é de suma importância para o desempenho do programa; em lua, variáveis locais são acessadas com muito mais rapidez que variáveis globais. Vale ressaltar que uma variável pode receber *nil* (nulo) indicando a ausência de valor.

Uma tabela é a única estrutura de dados existente em Lua. No nosso exemplo ela foi inicializado com valores numéricos de 1 a 5. Ainda, foi declarada como uma variável local.

Dando sequência nas linhas do código, encontramos duas função, que são declaradas usando a palavra-chave *function* sendo a segunda atribuída a uma variável. Em Lua as funções não são tratadas de modo especial como em outras linguagens pois da mesma maneira que atribuímos a uma variável uma *string,* podemos fazer o mesmo com uma função. Dentro da primeira função se encontra uma estrutura de controle *if.* Esta testa se o valor recebido pela função é um número par.

Mais adiante se encontra outra estrutura de controle, um *for*. O laço repete o bloco de código, no caso a chamada à função "verifica", enquanto a variável de controle "i" varia de acordo com uma progressão aritmética e a variável "v" guarda o valor de índice "i" da tabela para posteriormente ser testado. Todos os blocos e funções devem estar delimitados pela palavra-chave *end* indicando o fim dos mesmos.

2.3.1 Funções básicas em Lua

Existe um conjunto de funções básicas em Lua que são essenciais para o desenvolvimento de uma aplicação. Elas oferecem recursos onde torna-se possível gerar código lua para a persistência de um objeto em diferentes momentos da execução do programa. Abaixo são listadas algumas dessas funções:

- **assert (v [, msg])**: retorna um erro caso o parâmetro "v" seja nulo (*nil* ou *false*). "msg" é a mensagem de erro mostrada e no caso da ausência desta, a mensagem mostrada é "assertion failed!".
- **tostring (v):** recebe um parâmetro que pode ser de qualquer tipo e o converte para uma cadeia de caracteres.
- **tonumber (v):** recebe um parâmetro e o converte em número, caso ele já seja um número ou uma cadeia de caracteres que pode ser convertida em número.

Caso contrário, retorna *nil*.

- **print (...):** recebe um número qualquer de parâmetros. Primeiro converte-os em cadeias de caracteres usando a função já citada *tostring* e imprime o resultado. Normalmente é usada para depuração por ser uma maneira rápida de mostrar um valor.
- **type (v):** recebe um parâmetro qualquer e retorna seu tipo. O retorno é uma cadeia de caracteres que pode ser "string", "table", "boolean", dentre outros.
- **dofile (a):** recebe como parâmetro um arquivo e executa seu conteúdo como um código Lua. Se executado com sucesso, retorna todos os valores retornados pelo trecho de código; senão, retorna *nil*.
- **next (t [, ind]):** recebe uma tabela "t" e retorna o índice seguinte ao parâmetro "ind" juntamente com seu valor associado. Quando o segundo parâmetro for *nil* ou estiver ausente, retorna o primeiro índice e seu respectivo valor.
- **unpack (t [, i [, j]]):** recebe uma tabela e retorna seus elementos em ordem crescente. O parâmetro "i" indica o índice pelo qual se quer começar a listar e "j" é a última posição. Em caso de omissão dos índices, por padrão "i" é 1 e "j" é o comprimento da tabela.
- **rawequal (v1, v2):** recebe dois parâmetros de qualquer tipo e verifica se são iguais, retornando um booleano.
- **error (msg):** tem o propósito de gerar uma codificação de erro de execução. Recebe como parâmetro uma cadeira de caracteres (texto) que será a mensagem de erro.

Além das funções citadas, Lua possui outras funções a serem consideradas, além de módulos. Um módulo nada mais é que a combinação de tabelas com funções de primeira classe. Lua não fornece nenhuma maneira específica para se construir um módulo, já que este pode ser implementado diretamente como uma tabela. A função básica *dofile* pode ser usada para carregar um módulo já que este pode ser definido como um trecho de código em Lua armazenado em um arquivo.

A linguagem Lua possui muitos outros aspectos a serem considerados como o uso de co-rotinas, também conhecidas como fluxos de execução independentes (*threads*), a API C que é o conjunto de funções disponíveis em C para o programa hospedeiro se comunicar com Lua, dentre outros.

2.4 A integração de NCL e Lua: NCLua

No Sistema Brasileiro de TV Digital, a linguagem Lua pode ser integrada à linguagem NCL, como já mencionado, aumentando o potencial das programações no que se refere à interatividade. Dessa união surgem os chamados scripts NCLua (Cerqueira et al, 2088). Eles usam a mesma abstração para objetos de mídia utilizada para vídeos, imagens e outros, ou seja, Lua pode ser embutida e funcionar normalmente em um documento NCL já que este apenas tem o papel de fazer o relacionamento entre objetos de mídia, não importando os tipos de conteúdo.

Um *script* NCLua deve ser escrito em um arquivo com sintaxe idêntica a Lua (de extensão .lua) e referenciado pelo documento NCL como um nó de mídia qualquer. Em particular, seu ciclo de vida é controlado pelo documento sendo este responsável por determinar em que momento o NCLua inicia.

NCLua possui um conjunto de bibliotecas divididas em 4 módulos, essenciais para o desenvolvimento de aplicações para TV Digital. Seguem abaixo:

- **módulo event:** define uma ponte entre os *scripts* NCLua e o Ginga-NCL, permitindo a comunicação entre os *scripts* e o Formatador NCL através de eventos.
- **módulo canvas:** disponibiliza uma API para desenhar imagens e primitivas gráficas.
- **módulo settings:** exporta uma tabela contendo variáveis definidas pelo autor do documento NCL e variáveis de ambiente reservadas.
- **módulo persistent:** exporta uma tabela contento variáveis persistentes definidas em uma área reservada do Ginga.

O módulo *event* é o módulo mais importante pois é através dele que ocorre toda a dinâmica de um *script* NCLua. Eventos são transmitidos na forma de uma tabela Lua e de maneira bidirecional, ou seja, tanto os *scripts* quanto o Formatador enviam e recebem eventos. A Figura 2.8 ilustra a tabela gerada quando a apresentação de um nó NCLua é iniciada.

```
{ class='ncl', type='presentation', action='start' }
```

Figura 2.8: Tabela gerada para representar um evento.

Para que o NCLua receba eventos do Formatador, o script deve registrar pelo menos um tratador de eventos. Um tratador de eventos nada mais é que uma função dentro do *script* NCLua que será chamada toda vez que ocorrer a transmissão de um evento. Para tanto, usa-se o método *event.register* passando como parâmetro o nome da função tratadora. Para um NCLua enviar eventos utiliza-se o método *event.post* passando como parâmetro a tabela que representa o evento.

Ainda na Figura 2.8, a tabela possui um campo *class* responsável por identificar a classe a qual o evento pertence. São definidos 4 tipos de classes: *ncl, key, user* e *tcp.*

A classe *ncl* tem a responsabilidade de fazer a comunicação entre o NCLua e o documento ao qual está referenciado. Para essa classe, há dois tipos de eventos definidos através do campo *type*: *presentation* e *attribution*. O primeiro está relacionado a eventos de apresentação e controla a exibição do nó NCLua. Podem estar associados a áreas específicas (âncoras) ou ao nó por completo. O último está relacionado a eventos de atribuição e controla as propriedades do nó NCLua. Os dois tipo de eventos são bastante similares já que são guiados pelo mesmo modelo de máquina de estados. Logo, o campo *action* pode assumir os valores *start*, *stop*, *abort*, etc em ambos.

A classe *key* está associada ao uso do controle remoto. Como este é um dispositivo de entrada somente, não existem nesta classe eventos gerados, ou seja, o *script* apenas recebe eventos. O campo *type* pode assumir os valores *press* ou *release* para indicar que a tecla, definida pelo campo *key*, foi pressionada ou liberada, respectivamente.

A classe user é usada para a uma aplicação estender sua funcionalidade criando eventos próprios. Nenhum campo é definido além, obviamente, do campo *class*.

O uso do canal de interatividade é representado pela classe *tcp*. Por meio dessa classe, é possível enviar e receber dados através de uma conexão previamente estabelecida. Para abrir uma conexão, o campo *type* deve receber o valor "connect" e ao campo *host* deve-se atribuir o endereço do *host.* Para fechar uma conexão, o campo *type* deve receber o valor "disconnect". O envio de dados é feito definindo *type* com o valor "data" e ao campo *value* é passado cadeia de caracteres como sendo o dado.

Até aqui descreveu-se algumas características do módulo *event* por ser o mais importante, como mencionado. Uma descrição completa desse e também dos outros módulos pode ser encontrada em (de Melo e Araújo, 2008).

3. ACESSO A SERVIÇOS DA INTERNET

A evolução das Tecnologias da Informação e Comunicação (TIC) pode ser considerada um dos maiores marcos na história moderna no que diz respeito aos segmentos de ofertas de serviços. É possível perceber uma tendência cada vez mais acentuada de adoção dessas tecnologias por escolas, empresas, especialmente com a disseminação dos aparelhos digitais no cotidiano atual.

Pode-se entender um serviço web, como sendo uma aplicação ou um componente, identificado por uma URI (Uniform Recurses Identificator), onde suas interfaces e ligações podem ser descritas, definidas e acessadas por meio de outras aplicações. Essa tecnologia permite interações diretas entre aplicações de *software* diferentes através do envio e recebimento de mensagens em formato XML. Os serviços web apresentam algumas características tais como:

- Conectam computadores e dispositivos através da internet, tornando possível novas maneiras de troca e manipulação de dados.
- Os dados são transmitidos no formato XML, podendo ser tratados e lidos em qualquer plataforma.
- Possuem uma interface abstrata para acesso aos métodos disponíveis tornando ocultos ao usuário do serviço os detalhes da implementação.
- Permite o desenvolvimento de *softwares* baseados no modelo *cloud- computing* (computação nas nuvens).
- Permite que um *software* deixe de ser comercializado da forma tradicional e passe a atuar como um serviço (SaaS – *Software as a Service*).

Algumas tecnologias como CORBA e RMI são soluções tradicionais para a integração de sistemas, porém possuem desenvolvimento muito complexo e caro e são soluções tecnológicas proprietárias, ou seja, cada fabricante tem seu próprio modo de construção e o suporte na maioria das vezes não é suficiente a ponto de garantir segurança e estabilidade aos sistemas baseados nessas soluções de *middleware*.

3.1 XML

XML (Extensible Markup Language) (W3C, 2008) é uma linguagem de marcação de dados que possibilita descrever informações em um formato estruturado. É uma linguagem inspirada no SGML (Standard Generalized Markup Language) e tem o

propósito de facilitar o compartilhamento de informações através da internet. A linguagem XML possui algumas características tais como:

- Tem seu foco na estrutura da informação, e não na sua forma de apresentação.
- Possibilita a troca de informações entre sistemas diferentes.
- Permite a criação de um número indefinido de etiquetas (*tags).*
- É simples e legível, tanto para pessoas quanto para computadores.
- É extensível, suporta praticamente todos os tipos de aplicações.

A linguagem vem se tornando um formato padrão para a troca de informações entre entidades distintas. Usando XML é possível criar documentos com dados organizados hierarquicamente como nota-se frequentemente em documentos formatados, banco de dados, imagens vetoriais, etc. Por conta de sua portabilidade, já que não depende do tipo de plataforma, é possível a um banco de dados, por exemplo, ler um arquivo XML enviado por outro banco de dados distinto, escrito por este por meio de uma aplicação.

Um documento XML é escrito de forma muito similar a de um documento HTML. A diferença está no fato de que este último se encarrega da formatação dos dados que se deseja mostrar, ou seja, preocupa-se também com a formatação, enquanto o primeiro se preocupa com a estruturação da informação que se pretende armazenar. Sua estrutura consiste em um conjunto de marcações chamadas de elementos que delimitam e identificam as partes do documento. Tais marcações são representadas pelos caracteres "<" e ">" que, por sua vez, delimitam o nome do elemento. Cada elemento é definido como um nó, sendo chamado de elemento raiz o nó mais externo e de elementos filhos os nós internos definindo uma estrutura na forma de árvore (hierárquica). A figura a seguir demonstra a sintaxe flexível do XML descrevendo informações pessoais em um currículo.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<curriculo>
    <dadosPessoais>
        <nomecompleto>Hórus de Nazaré</Nomecompleto>
        <dataNascimento>05-10-1986</dataNascimento>
        <contatos>
            <endereco>
                  <rua>Rua 10</rua>
                  <numero>50</numero>
                  <Cidade>São Luís</Cidade>
                  <Pais>Brasil</Pais>
            </endereco>
            <telefone>3333-3333</telefone>
            <email>email@email.com</email>
        </contatos>
        <nacionalidade>Brasileira</nacionalidade>
        <sexo>Masculino</sexo>
    </dadosPessoais>
    <objetivo>Atuar na área de desenvolvimento para web</objetivo>
    <experiencia>
        <cargo>Programador</cargo>
        <empresa>Empresa X</empresa>
    </experiencia>
    <formacao>Superior Completo</formacao>
</curriculo>
```

Figura 3.1: Documento XML.

Na Figura 3.1, pode-se notar que o elemento <curriculo> é o elemento mais externo, portanto a raiz do documento. Ainda, cada elemento deve possuir seu respectivo par que fecha o escopo da marcação. Por exemplo, o elemento <endereco> possui seu delimitador </endereco>, demarcando o espaço para as informações referente ao endereço do candidato.

Com o XML é possível processar a informação com muita facilidade e simplicidade. Tudo está organizado de uma maneira lógica podendo ser compreendido facilmente por uma pessoa comum. Por ser extensível, como já citado, a linguagem XML permite-se servir de suporte para uma enorme variedade de aplicações.

3.2 XMPP

XMPP (*Extensible Messaging and Presence Protocol*) é um protocolo aberto e extensível, baseado no formato XML para comunicação em tempo real. Originalmente foi desenvolvido para aplicações de mensagens instantâneas, porém hoje pode ser

empregado no desenvolvimento de uma infinidade de outras aplicações tais como jogos, localização geográfica, VoIP[5], distribuição de conteúdo, dentre outros, graças a suas inúmeras extensões (XMPP Standards Foundation, 2012).

A princípio chamado de Jabber, o protocolo XMPP foi criado em 1998 por Jeremie Miller e surgiu como uma alternativa aos protocolos de comunicação fechados. A ideia era prover a interoperabilidade entre os protocolos existentes já que, até então, somente usuários de um mesmo aplicativo podiam se comunicar. Em meados de 2004, foi aprovado e padronizado pelo IETF (*Internet Engineering Task Force*) sendo definidas duas RFC-s (*Request for Comments*) básicas. A primeira RFC (3920) trata das especificações base do XMPP, ou seja, arquitetura, métodos de conexão, segurança, transporte de dados e semântica das mensagens XML. A segunda (3921) define o modo como as mensagens são transmitidas, solicitação e definição de presença, gerenciamento de contatos, etc. No ano seguinte, 2005, agora padronizado junto a um órgão reconhecido, o protocolo XMPP foi base para a criação do serviço Google Talk[6], o que acabou por popularizar o protocolo por conta do grande número de usuários do serviço. A partir daí, inúmeras empresas como Nokia, Apple e IBM passaram a desenvolver seus produtos utilizando também o XMPP. Pelo fato de as mensagens terem o formato XML, este sendo uma recomendação do W3C (*World Wide Web Consortium*), o XMPP tem um grande suporte por parte dos fornecedores de software.

3.2.1 Conceitos Básicos

Para entender melhor a tecnologia XMPP, faz-se necessário descrever seus componentes básicos e suas respectivas funções durante a comunicação. Os componentes básicos são: JID*, roster e stanza.*

O esquema de endereçamento é feito através de um identificador único. Cada usuário XMPP deve possuir um identificador e a este foi dado o nome de Jabber ID (ou simplesmente JID). Tem um formato muito semelhante ao do endereço de e-mail mas possui um diferencial de ser totalmente internacionalizado, ou seja, pode-se criá-lo usando qualquer caractere Unicode UTF-8. Além disso, um JID diz, de forma explícita, em que servidor o usuário se encontra, facilitando assim o repasse das mensagens no caso de dois usuários se comunicando por servidores diferentes. Um JID é composto por três partes: domínio, usuário e recurso, onde:

5 Voice over Internet Protocol, tecnologia para transmissão de conversação humana através da internet.
6 Mensageiro instantâneo desenvolvido pela empresa Google.

- Domínio é o identificador primário do JID, responsável por identificar a qual servidor o usuário pertence. Todo JID válido deve conter ao menos um domínio válido na sua estrutura.
- Usuário é o identificador secundário, onde este será válido apenas no domínio em que está registrado. É um identificador opcional já que existem alguns domínios específicos que são JID's válidos.
- Recurso é o identificador terciário e é usado para identificar dispositivos ou locais. Também é um identificador opcional.

Um *roster* pode ser entendido como a lista de contatos que o usuário possui e com os quais pode interagir. Essa lista fica armazenada no servidor onde podem ser organizados em grupos seus itens.

Stanza é o nome dado à unidade básica de comunicação no XMPP. Pode-se imaginar como sendo um pacote ou uma mensagem em outros protocolos. Uma *stanza* pode ser classificada em três tipos: presença (<presence>), mensagem (<message>) e info/query (<iq>). Cada tipo é enviado ao servidor de uma forma específica e tratado pelo destinatário também de forma específica quando recebido.

A *stanza* <presence> anuncia e controla a disponibilidade de uma entidade. A presença permite a uma entidade saber quais outras estão disponíveis para comunicação. Para anunciar sua presença, um usuário envia uma stanza ao servidor que fica com a responsabilidade de espalhar para o seu *roster*. É ilustrado na Figura 3.2 um exemplo de *stanza* <presence>.

```
<presence from="usuario1@servidor1.com/casa">
    <show>away</show>
    <status>Almoçando!</status>
</presence>
```

Figura 3.2: Estrutura de uma stanza <presence>.

A *stanza* <message>, como sugere a nome, é usada para enviar mensagens de uma entidade para outra. Estas podem ser simples mensagens de conversação ou até mesmo mensagens carregando algum tipo de informação estruturada. Quando um cliente envia uma mensagem, o servidor não lhe retorna nenhuma confirmação, apenas transmite a mensagem ao destinatário se o usuário estiver online, senão armazena. A Figura 3.3 mostra um exemplo de uma *stanza* <message>.

```
<message from="usuario1@servidor1.com/casa"
         to="usuario2@servidor2.com"
         type="chat">
    <body>Estou em casa!</body>
</message>
```

Figura 3.3: Estrutura de uma stanza <message>.

A *stanza* <iq> se refere a informações ou consultas e provê um mecanismo de resposta durante a comunicação em uma rede XMPP promovendo uma interação estruturada entre as entidades. Seu funcionamento é muito semelhante ao de outros protocolos pois permite definir e obter consultas, semelhantes às ações GET e POST do protocolo HTTP. Uma entidade que envia uma *stanza* <iq> sempre deve receber uma resposta, normalmente gerada pelo servidor do destinatário. A Figura 3.4 exemplifica uma interação promovida pela *stanza* <iq>. A primeira parte indica uma *stanza* enviada ao servidor solicitando a lista de contatos disponíveis e a segunda parte indica a resposta do servidor contendo essa lista.

```
<iq from="usuario1@servidor1.com/casa"
        type="get">
    <query xmls="jaber:iq:roster"/>
</iq>

<iq to="usuario1@servidor1.com/casa"
        type="result">
    <query xmls="jaber:iq:roster">
        <item jid="usuario2@servidor1.com" name="Usuário2"/>
        <item jid="usuario3@servidor2.com" name="Usuário3"/>
    </query>
</iq>
```

Figura 3.4: Interação promovida pela *stanza* <iq>.

3.2.2 Arquitetura XMPP

A arquitetura de uma rede XMPP é bem parecida com a de uma rede de e-mail. Podemos destacar algumas características inerentes a ambas:

- As mensagens não são entregues diretamente de um usuário para outro. Antes de chegar ao destinatário, uma mensagem deve passar por, pelo menos, um servidor. É a chamada comunicação cliente-servidor.

- Usuários são identificados usando basicamente o mesmo esquema de endereçamento: usuário@domínio.
- Quando o domínio do destinatário não é conhecido pelo remetente, os servidores se comunicam entre si a fim de entregar as mensagens.

O principal fator que diferencia as duas redes está no fato de que uma rede de e-mail é baseada no princípio de *store-and-forward* onde as mensagens ficam armazenadas no servidor aguardando o usuário "perguntar" se existem novas em sua caixa de entrada. Já em uma rede XMPP, as mensagens são entregues em tempo real pois o servidor sabe quando o usuário está conectado. A Figura 3.5 mostra a estrutura de uma rede XMPP.

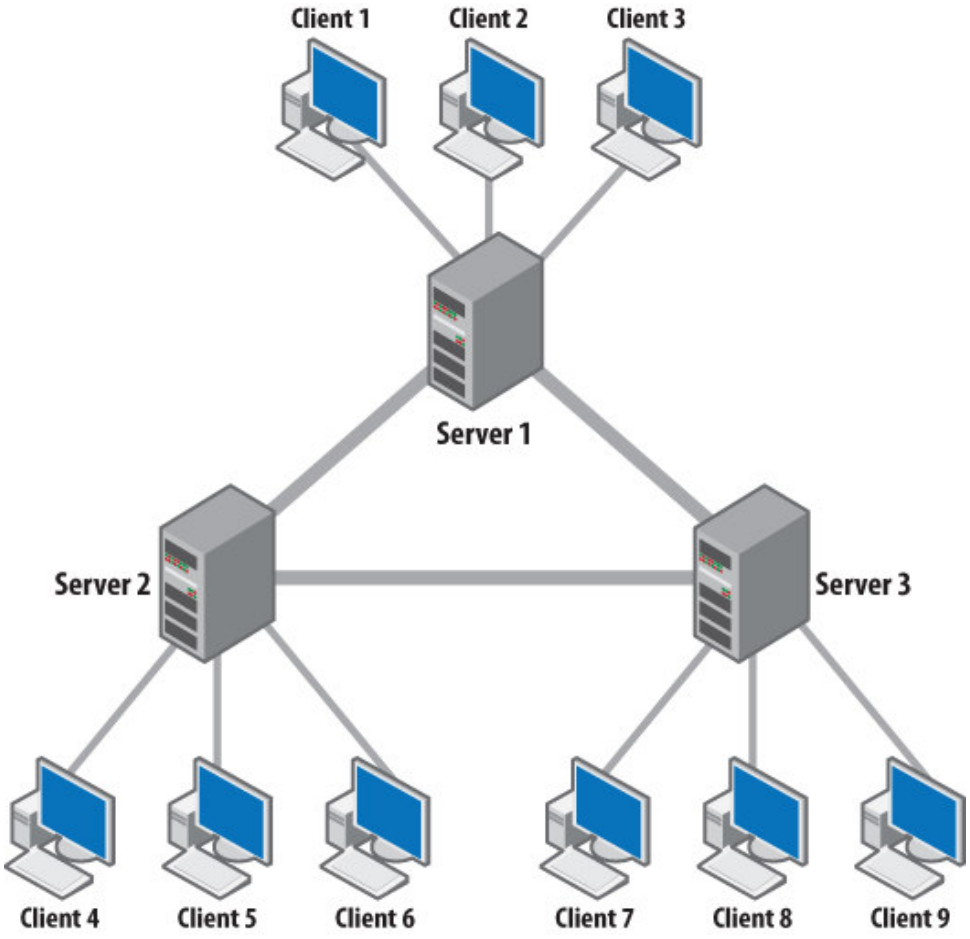

Figura 3.5: Rede XMPP (Saint-Andre et al, 2009).

A partir da Figura 3.5, podemos notar a presença de duas entidades que são essenciais para propiciar a comunicação dentro da arquitetura XMPP, assim como da arquitetura de e-mail: os servidores e os clientes.

Um servidor pode ser definido como um computador ou uma aplicação responsável por realizar tarefas e/ou fornecer serviços dentro de uma rede de computadores. Os

servidores XMPP são encarregados de fazer o gerenciamento das conexões e processar as mensagens XML antes de enviá-las ao destinatário, além de armazenar os dados do usuário como, por exemplo, sua lista de contatos. Existem inúmeros servidores XMPP disponíveis, entre eles se destacam o Openfire[7] e o Ejabberd[8].

Clientes são as entidades que usufruem dos serviços oferecidos por um servidor. Um cliente XMPP se conecta a um servidor XMPP através de uma conexão TCP (*Transmission Control Protol*) o que garante o envio dos dados de forma íntegra e na ordem especificada. Com o uso do identificador terciário, o recurso, o cliente XMPP pode se conectar ao mesmo servidor a partir de lugares ou dispositivos diferentes. Alguns dos clientes XMPP mais populares são o Spark[9] e o Adium[10].

Existe ainda um tipo especial de servidor XMPP que é chamado de *gateway* e serve para traduzir as mensagens oriundas de sistemas de protocolos estrangeiros para o protocolo de origem e vice-versa. Desse modo um cliente XMPP pode aceder a qualquer rede em que exista um *gateway*, sem qualquer código extra. Atualmente existem *gateways* para vários serviços como, por exemplo, o MSN Messenger[11] e o Google Talk.

A estrutura de uma rede XMPP é muito semelhante à da internet. Cada servidor é identificado por um endereço de rede estabelecendo uma rede de servidores que se intercomunicam. São necessários inúmeros servidores como o Openfire executando clientes como o Spark para que aconteça a transmissão de mensagens em uma rede XMPP. Caso se queira enviar uma mensagem para um destinatário de um domínio diferente, o cliente se conecta ao seu servidor de origem, este transmite a mensagem diretamente para o servidor do destinatário que finalmente faz a entrega.

3.2.3 Exemplo de comunicação cliente-servidor XMPP

As conexões em XMPP são persistentes, ou seja, enquanto o cliente possuir uma sessão ativa no servidor a conexão permanece aberta. Para se conectar a um servidor, o cliente deve lhe enviar a mensagem do tipo **<stream to="127.0.0.1: 5222" xmlns="jabber:client"/>** , onde o valor do atributo "to" é o endereço ip seguido da porta do servidor XMPP. O servidor então responde ao cliente com uma mensagem do tipo **<stream id="x" from:"127.0.0.1: 5222" xmlns="jabber:client"/>** onde o atributo "id" guarda um valor usado para identificar as respostas. Em seguida o cliente deve enviar

7 Servidor XMPP criado pela empresa Jive Software.
8 Servidor XMPP escrito na linguagem Erlang e criado por uma comunidade de desenvolvedores.
9 Cliente XMPP multiplataforma, aberto e que possui integração com o servidor Openfire.
10 Mais popular cliente XMPP para Mac OS, aberto e com licença GPL.
11 Mensageiro instantâneo criado pela empresa Microsoft.

uma mensagem com os parâmetros necessários para se conectar ao servidor. Ele então envia uma mensagem como essa: **<iq type="set" id="y"><query xmlns="jabber:iq:auth"><username>usuario1</username><password>123</password><resource>casa</resource></query></iq>**. Caso a autenticação seja efetuada com sucesso, o servidor lhe retorna a mensagem **<iq id="y" type="result"/>** ; caso contrário retorna uma mensagem de erro como essa: **<iq id="y" type="error"/>**. Após conectado, é necessário que o cliente envie ao servidor uma *stanza* de presença para indicar sua disponibilidade. O cliente faz o envio da *stanza* e, como já mencionado, fica a cargo do servidor repassar a mensagem anunciando sua presença.

O XMPP vem se mostrando ser bastante robusto apesar de ser um protocolo novo. É uma alternativa de fácil compreensão pois todo e qualquer tipo de comunicação acontece fazendo uso de mensagens em XML. Todo o processo de desenvolvimento de novas extensões é acompanhado pela XMPP Standards Foundation, o que garante um nível de qualidade e seriedade padrão para todos.

3.3 Arquitetura Proposta

O ambiente de um jogo *multiplayer* estabelece meios de interação entre os seus usuários onde estes podem conversar entre si, iniciar uma partida contra outro usuário, acompanhar o andamento de uma determinada partida, dentre outras funcionalidades. No cenário da TV Digital, já podemos encontrar algumas aplicações que simulam jogos de tabuleiro, por exemplo, porém cada tipo de aplicação é tratada de uma forma específica, no que se refere ao acesso, passagem de mensagens, etc. Para tanto, um sistema robusto para a troca de mensagens entre os jogadores e se faz necessário.

Foi proposta uma arquitetura genérica para um ambiente *multiplayer* de jogos de tabuleiro, para permitir a usuários jogarem com seus amigos utilizando o controle remoto, cada um no seu ambiente de TV Digital. A arquitetura é baseada em (Reis e Teixeira), do tipo cliente-servidor e possui 3 módulos: o módulo de comunicação, o módulo de regras e o módulo interface. O módulo de comunicação é responsável pelas trocas de mensagens dentro do sistema. Ele realiza o encaminhamento das mensagens de uma entidade para outra, seja usuário ou servidor. O módulo de regras fica encarregado de tratar todas as requisições relacionadas ao jogo como, por exemplo, mensagens relacionadas a desafios, abandono de partida, informações de jogadas, dentro outros. O módulo interface é a própria interface do usuário. Ele provê os meios necessários para o usuário enviar e receber mensagens. A Figura 3.6 ilustra a arquitetura em questão.

Figura 3.6: Arquitetura proposta.

A arquitetura utiliza-se constantemente da troca de mensagens envolvendo as entidades usuário e servidor. Dessa forma, é necessário utilizar um protocolo de comunicação robusto que suporte mensagens de requisição, confirmação e erros. Como alternativa para a implementação da arquitetura, optou-se pela utilização do protocolo XMPP. A seguir, são descritos cada um dos módulos da arquitetura.

3.3.1 Módulo de Comunicação

Este módulo tem como principal funcionalidade gerenciar o tráfego de mensagens de uma entidade para outra dentro da aplicação, como já mencionado. Para isso, o ambiente de comunicação deve oferecer mecanismos de autenticação e controle de acesso.

O componente de comunicação escolhido foi o servidor Openfire, pois este implementa o protocolo XMPP, além de possuir código aberto e distribuído sob a licença GPL (*General Public Licence*). A escolha dessa ferramenta se deu pelo fato de possuir uma boa implementação do XMPP e por apresentar facilidade na configuração e administração, além de um bom desempenho agregado à sua sólida segurança.

3.3.2 Módulo de Regras

Este módulo é responsável por tratar as mensagens que são trocadas entre as entidades. Ele funciona como um intermediador, cada mensagem trafegada passa por este módulo onde será avaliada e posteriormente convertida de acordo com a sua

natureza.

Como componente desse módulo, é sugerido o desenvolvimento de um cliente XMPP em Lua. A princípio, tem a função de realizar a conexão entre o usuário e o servidor XMPP. Todo um processo de negociação e autenticação é realizado para o usuário obter acesso ao jogo e poder interagir dentro dele.

Ainda nesse componente, devem ser implementados também os mecanismos de tratamento das *stanzas* de mensagens que irão coordenar as ações do jogo. O componente recebe, por exemplo, uma *stanza* contendo as informações da jogada, avalia se é um movimento válido e, em caso afirmativo, repassa a informação ao usuário oponente; caso contrário, não repassa.

3.3.3 Módulo Interface

Esse módulo representa a interface do cliente. É através dele que o usuário poderá visualizar a interface do jogo e tomar as decisões desejadas como fazer uma jogada, convidar um usuário para uma partida, etc.

Para tanto, um *widget*[12] NCLua deve ser implementado para prover uma forma transparente e amigável de interação entre os jogadores. Na prática, ao se ligar o aparelho de TV o miniaplicativo é carregado na tela e se conecta ao servidor através do cliente XMPP citado no módulo anterior. A partir daí, o usuário pode interagir dentro do jogo através de suas ações, senda estas realizadas com o uso do controle remoto.

A partir da arquitetura proposta, pode-se criar um cenário que suporte um jogo de tabuleiro *multiplayer*. Interpõe-se um servidor executando o protocolo XMPP entre o *widget* NCLua e a internet. O miniaplicativo, executado no *set-top-box* (decodificador) da residência do telespectador, inicialmente faz a autenticação do usuário por meio do cliente XMPP. Este, por sua vez, estabelece uma conexão com o servidor a fim de possibilitar ao usuário entrar no jogo.

Após a autenticação, o *widget* fica à espera das ações do jogador ou de eventuais mensagens vindas do servidor. Quando o servidor envia uma *stanza* de mensagem, esta passa antes pelo cliente XMPP que extrai a informação, faz o devido tratamento e exibe através do *widget* NCLua no aparelho de TV.

Pode-se notar que o servidor XMPP representa o ponto central e de concentração das informações na arquitetura dada. Ao mesmo tempo que faz o gerenciamento das

12 Miniaplicativo

conexões, ele é alimentado pelas mensagens que trafegam continuamente, oriundas de ambas as partes, fazendo o fornecimento das mesmas aos devidos jogadores. O servidor compila e disponibiliza a informação em um formato conciso e estruturado, um documento XML, já que se encontra sob a execução do protocolo XMPP. Dessa forma, a entrega da informação se torna mais imediata do que se fosse a partir de outro protocolo.

4. JOGO DE XADREZ COM MÚLTIPLOS JOGADORES

O xadrez é um dos jogos de tabuleiro mais populares do mundo. Existe desde a antiguidade e possui uma natureza recreativa e competitiva, podendo servir como entretenimento ou até mesmo no desenvolvimento do raciocínio lógico, aprendizado e tomada de decisões. Com a popularização da internet, surgiram servidores de xadrez que permitem a pessoas geograficamente distantes disputarem partidas via *web* através de um tabuleiro virtual.

Uma partida de xadrez é disputada entre dois jogadores, obrigatoriamente, em um tabuleiro contendo casas claras e escuras. Inicialmente, cada jogador, também conhecido como enxadrista, controla 16 peças da mesma cor com diferentes características e formato, resultando num total de 32 peças classificadas em 6 tipos. Cada tipo possui uma forma distinta de movimentação. O objetivo do jogo é dar um xeque-mate, ou seja, ameaçar o rei do adversário com uma captura inevitável. Uma partida não se encerra necessariamente com o xeque-mate; há casos em que um jogador opta pela desistência quando acredita que não conseguirá vencer ou ainda uma partida pode terminar empatada.

O xadrez é um jogo de estratégia e tática e, por conta disso, não envolve o fator sorte. Nesse caso, a única exceção seria no momento do sorteio das cores no início do jogo, já que as peças brancas , pela regra, sempre fazem o primeiro movimento e, em tese, daria uma pequena vantagem ao enxadrista (Stevens, 1969).

Partindo desse contexto, foi desenvolvido um protótipo de um jogo de xadrez com múltiplos jogadores para demonstrar o uso da arquitetura proposta no capítulo anterior e prover meios de entretenimento e interatividade no ambiente de TV Digital. A aplicação residirá nos conversores de TV Digital compatíveis com o Ginga e deve manter os mesmos níveis de desempenho, usabilidade e segurança das aplicações voltadas para computadores tradicionais, possibilitando, dessa forma, uma alternativa de diversão e inclusão social para o público televisivo.

Para ilustrar seu encaixe na arquitetura proposta no Capítulo 3, a Figura 4.1 reorganiza a arquitetura de uma forma mais adequada ao jogo de xadrez.

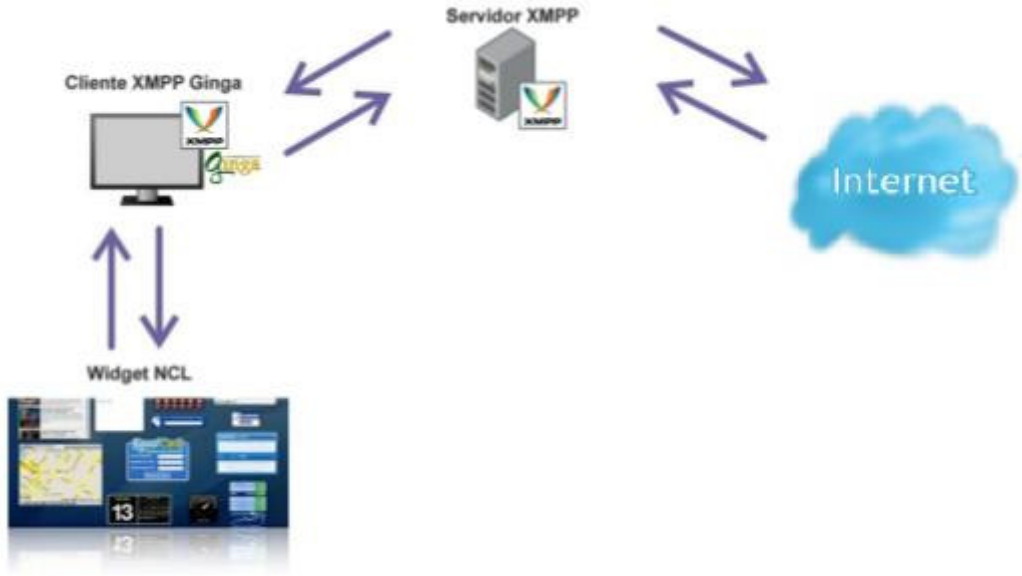

Figura 4.1: Arquitetura reorganizada.

4.1 Trabalhos Relacionados

Existem algumas aplicações que, assim como o trabalho apresentado, também buscam oferecer meios de interatividade para o público televisivo por meio do ambiente de TV Digital. Estas serão descritas aqui.

4.1.1 RummiTV

O projeto RummiTV, criado no laboratório MídiaCom da UFF (Universidade Federal Fluminense), está desenvolvendo o jogo eletrônico RummiTV, que é uma adaptação baseada no popular jogo de tabuleiro Rummikub para o ambiente de TV Digital Interativa. É um jogo de cunho educativo que tem como objetivo estimular a capacidade de raciocínio de jovens e crianças a partir de 8 anos de idade (Santos e Ratamero, 2007).

O jogo consiste de um tabuleiro contendo um conjunto de 106 peças, divididas em 4 cores (amarela, vermelha, azul e preta), sendo 26 peças de cada cor e mais 2 coringas. Cada grupo de peças de cada cor possui duas sequências numeradas de 1 a 13. O jogo se inicia com cada jogador recebendo aleatoriamente um conjunto de 14 peças, ficando as demais no "monte". Cada jogador, por sua vez, deve juntar suas peças entre si ou com as que foram descartadas sobre a mesa, formando combinações aceitáveis de acordo com a regra. O jogador que primeiro descartar todas as suas peças formando combinações é o vencedor (Santos e Ratamero, 2007).

A estrutura de dados orientada a objetos do RummiTV está toda especificada e implementada em java. O projeto encontra-se em fase de desenvolvimento, já consideradas as limitações de interface para a plataforma de TV Digital. Sua interface é mostrada abaixo.

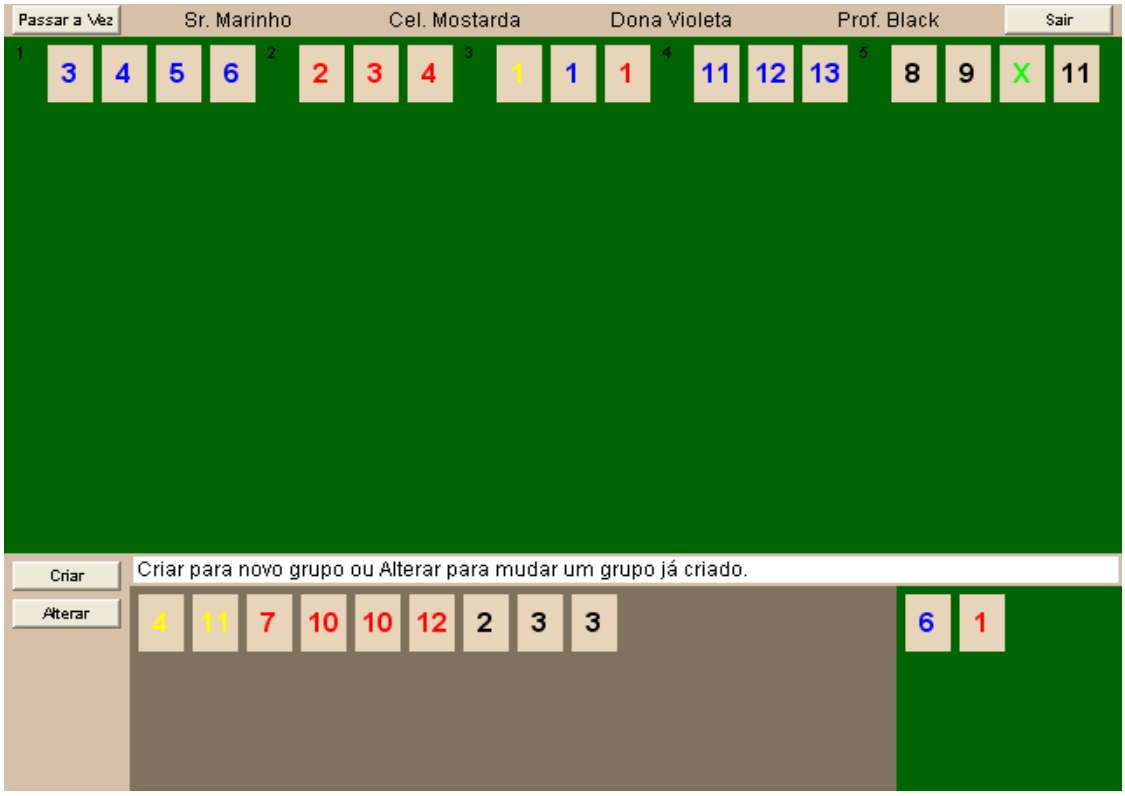

Figura 4.2: Interface do RummiTV (Manual RummiTV, 2008).

A natureza educativa do jogo foi determinante para a escolha do mesmo para o projeto. Ainda, além dos benefícios proporcionados pelo jogo em si, o projeto abre caminhos para o usuário entrar em contato com novos meios de interatividade proporcionado pela TV Digital. Para tanto, o jogo apresenta uma linguagem de simples entendimento com o intuito de facilitar a compreensão do mesmo por parte do usuário.

Mais informações referentes a esta aplicação podem ser encontradas em (Santos e Ratamero, 2007).

4.1.2 DamasTV

O DamasTV é um jogo de damas isométrico para TV Digital também desenvolvido pelo laboratório MídiaCom da UFF. O jogo oferece 2 modos de desafio, o *stand-alone*, onde o usuário joga sozinho ou contra alguém compartilhando do mesmo controle remoto, e o distribuído, onde dois usuários em ambientes distintos de TV podem jogar um contra o

outro.

O jogo foi feito em NCLua. Toda a lógica e a interface do jogo foram desenvolvidas usando a linguagem Lua enquanto a parte do código em NCL se resume à definição das mídias como, por exemplo, o vídeo introdutório e os efeitos sonoros. Além disso, o DamasTV possui um servidor criado em PHP que usa o MySql para guardar as informações dos usuários, dos jogos, dentre outras.

DamasTV utiliza-se do paradigma orientado a objetos para facilitar a organização do código e possibilitar seu reuso. Embora Lua não seja uma linguagem orientada a objetos, é possível implementar muitos conceitos a partir da estrutura em tabelas (Ellwanger e Balbinot, 2003).

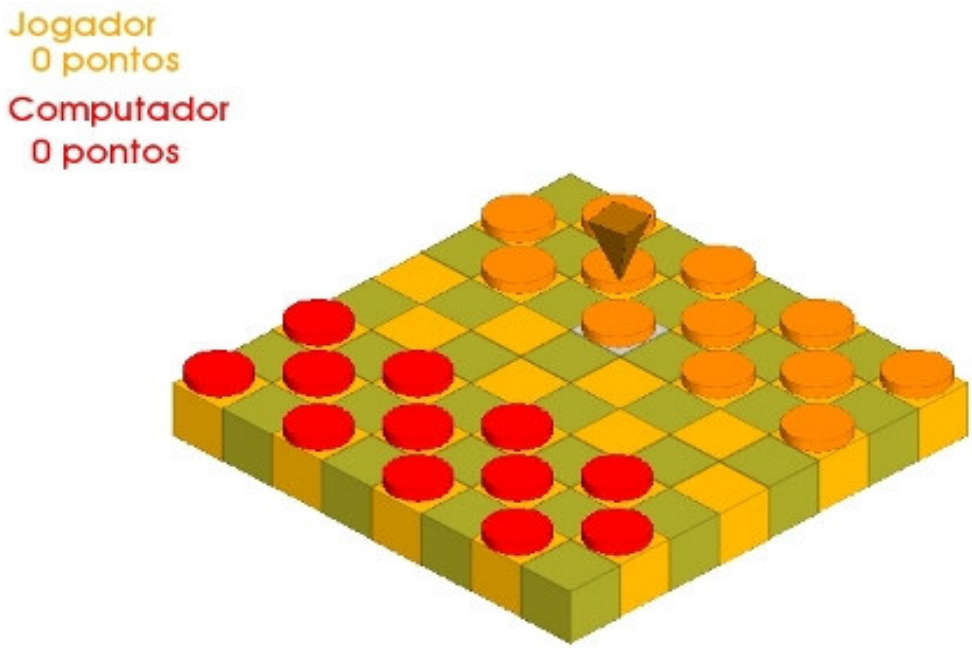

Figura 4.3: Interface do DamasTV (DamasTV, 2010).

Uma descrição completa dessa aplicação pode ser encontrada em (DamasTV, 2010).

4.2 Tipos de Aplicações

O protocolo XMPP pode ser empregado no desenvolvimento de uma grande variedade de aplicações, graças às suas inúmeras extensões, como mencionado anteriormente. Dentro desse contexto, duas categorias de aplicações podem ser

destacadas: aplicações *publish/subscribe* e aplicações interativas.

4.2.1 *Publish/Subscribe*

O paradigma *Publish/Subscribe*, ou simplesmente PubSub, implementado pelos chamados sistemas de notificações, tem um importante papel no contexto de desenvolvimento de sistemas distribuídos. É usado em sistemas ou serviços onde acontecem muitos eventos e há entidades interessadas em receber determinados tipos de eventos. Para tanto, entidades devem subscreverem-se a um mediador (handler) declarando que, sempre que ocorrer o evento desejado, ele deseja ser notificado. A subscrição só se efetiva mediante autorização prévia. Nesse paradigma, os componentes da aplicação são divididos entre funcionais, referentes ao domínio da aplicação, e os não-funcionais, referentes a suportes da aplicação como, por exemplo, as rotinas de comunicação.

O modelo PubSub é bastante comum no ambiente XMPP. A presença é uma forma especial de PubSub, uma vez que é necessário o envio de um pedido a ser aceito para que um usuário possa ser visto online numa rede.

O PubSub é estruturado em nodos onde cada um destes corresponde a uma folha, de maneira análoga à estrutura de uma árvore. Um nodo pode ser descrito como o elemento básico em uma rede PubSub/XMPP sendo que cada nodo tem a função de publicar ou subscrever um conteúdo na rede. Os nodos responsáveis por publicar um conteúdo ou informação são definidos como produtores e os nodos que recebem essa informação em forma de notificação de atualização são conhecidos como consumidores.

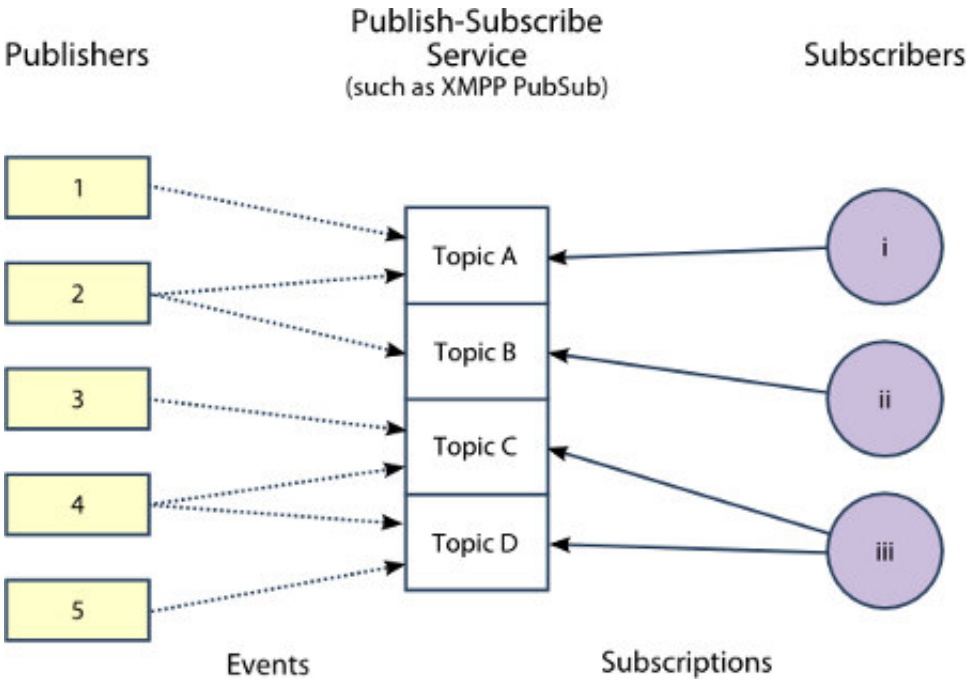

Figura 4.4: Arquitetura PubSub (XMPP Pubsub, 2011).

A extensão XEP-0060 (XEP-0060, 2010) do protocolo XMPP provê funcionalidades para o desenvolvimento de uma ampla variedade de aplicações, incluindo *feeds* de notícias, distribuição de conteúdo, a presença prolongada, geolocalização, *bookmarks* compartilhados, leilão e sistemas de negociação, sistemas de gerenciamento de rede, gerenciamento de perfil, e qualquer outra aplicação que exija notificações de eventos. A extensão permite a entidades XMPP criar nodos em um serviço PubSub e publicar informações; a informação é então transmitida em tempo real a todas as entidades que subscreveram o nodo. O envio das mensagens segue o padrão 1-N, isto é, um para muitos, onde apenas as entidades interessadas as recebem em forma de notificação de atualização.

Para subscrever um nodo, o usuário deve enviar uma requisição de subscrição ao provedor PubSub. Essa solicitação nada mais é que uma *stanza* <iq>. A *stanza* deve conter o elemento filho <pubsub/> que, por sua vez, deve conter o elemento <subscribe/>. Neste último, deve ser informado como atributos o nodo a ser subscrito como também o JID do usuário assinante.

```
<iq type="set"
        from="usuario1@servidor1.com/casa"
        to="pubsub.info.com"
        id="sub1">
    <pubsub xmlns="http://jabber.org/protocol/pubsub">
        <subscribe
            node="nodo1"
            jid="usuario1@servidor1.com"/>
    </pubsub>
</iq>
```

Figura 4.5: Pedido de assinatura de um nodo.

Após o envio do pedido de subscrição, o servidor retorna ao usuário uma *stanza* de resposta à solicitação informando que a assinatura do nodo foi realizada com sucesso ou que houve falha na subscrição.

```
<iq type="result"
        from="pubsub.info.com"
        to="usuario1@servidor1.com"
        id="sub1">
    <pubsub xmlns="http://jabber.org/protocol/pubsub">
        <subscrition
            node="nodo1"
            jid="usuario1@servidor1.com"
            subid="28307eu0928e75dfh"
            subscription="subscribed"/>
    </pubsub>
</iq>
```

Figura 4.6: Resposta de assinatura realizada com sucesso.

O modelo PubSub é atualmente utilizado em inúmeros sistemas como, por exemplo, o Twitter e o One Channel.

O Twitter é um *microblogging* e rede social baseado em PubSub e que se popularizou bastante nos últimos anos. Ele permite aos usuários enviar e receber atualizações através de textos de até 140 caracteres. Para que um usuário receba determinada atualização ele deve "seguir" (subscrever) um outro usuário de sua escolha. Na teoria, essas atualizações são os nodos subscritos no serviço PubSub, que são enviadas a todos os assinantes do nodo.

O One Channel é uma aplicação *desktop* para a visualização e leitura de mensagens distribuídas por meio do protocolo PubSub XMPP. É um sistema que permite a um usuário subscrever artigos e notícias para que receba atualizações dos mesmos quando ocorrer. O sistema disponibiliza uma variedade de canais para subscrição. A única desvantagem que a aplicação apresenta é que, por ainda estar em fase de testes, ela só pode se conectar ao servidor XMPP da Process One, não permitindo, assim, uma conexão com diferentes provedores PubSub.

4.2.2 Interativas

Aplicações Interativas não são muito comuns no ambiente XMPP. Além disso, o fato do XMPP ser um protocolo relativamente novo faz com que não seja muito comum encontrar algum aplicativo para TV Digital que faça uso do protocolo. O protótipo de jogo desenvolvido como escopo desse trabalho pode ser encaixado perfeitamente na categoria.

4.3 Descrição do Protótipo

O jogo pode funcionar em dois modos: *stand-alone* ou distribuído. No primeiro, dois usuários podem se enfrentar num mesmo ambiente de TV, isto é, compartilhando do mesmo controle remoto; no segundo modo, usuários podem se confrontar cada um em seu próprio ambiente televisivo, ou seja, dois jogadores distintos geograficamente podem interagir através do canal de retorno.

Numa visão geral, a estrutura do jogo se assemelha à de uma aplicação de mensagens instantâneas, já que se sustenta na troca de mensagens. Como ele é baseado no protocolo XMPP, cada *stanza* de mensagem enviada ou recebida pode conter vários tipos de informações como, por exemplo, o movimento de uma peça, um convite para uma partida, disponibilidade de jogadores, etc. Cada usuário, por sua vez, deve possuir um JID para poder autenticar-se e entrar no jogo.

O protótipo foi desenvolvido usando as linguagens NCL e Lua, seguindo as normas do *middleware* declarativo Ginga-NCL. A parte do código em Lua ficou responsável por toda a lógica e interface do jogo enquanto a parte em NCL se resumiu à definição das mídias. Somente dois módulos da biblioteca NCLua foram usados na implementação, o módulo *event* e o módulo *canvas.* O módulo *event* é responsável pelos eventos gerados durante a execução da aplicação. O módulo *canvas* se encarrega de desenhar os objetos do jogo na tela. O ambiente de desenvolvimento utilizado foi o Eclipse IDE[13] 3.5 juntamente com o *plugin* NCL-Eclipse[14]. Para simular um ambiente de TV interativa foi utilizado o VMware Player[15] bem como a máquina virtual que emula o Ginga.

13 Ambiente para desenvolvimento em java e outras plataformas.
14 Plugin para o desenvolvimento de aplicações voltadas ao Ginga-NCL.
15 Player desenvolvido pela Vmware para a emulação de máquinas virtuais.

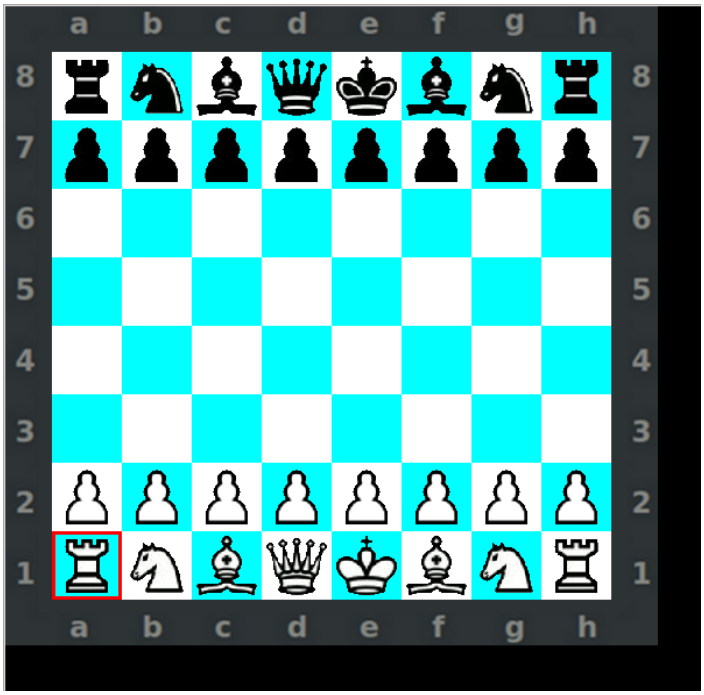

Figura 4.7: Interface do jogo.

4.3.1 Parte declarativa NCL

A parte da aplicação implementada utilizando a linguagem NCL refere-se à definição das mídias, como são apresentadas e de que modo estão estruturadas. Todo o código está definido no arquivo main.ncl.

A base de regiões é composta por três regiões, **rgTabuleiro**, **rgInfo** e **rgWin**, regiões responsáveis por representar a área do tabuleiro, a área destinada à notificações do jogo e uma pequena área no centro do tabuleiro, respectivamente. A área de notificações serve para informar ao jogador as jogadas quando ocorrem, quando um jogador abandona o jogo, o fim do jogo com seu respectivo vencedor, etc. A área central é destinada a mostrar uma mídia indicando o fim do jogo.

A base de descritores é formada por três descritores, **dsTabuleiro**, **dsInfo** e **dsWin**, cada um associado a uma das regiões. O primeiro está associado à região destinada ao tabuleiro (rgTabuleiro), o segundo à região onde serão mostradas as notificações (rgInfo) e o último referencia a região onde será mostrada a mensagem de fim do jogo.

A base de conectores contém apenas um conector, **onBeginStart**. Ele define que uma determinada âncora assumirá o papel *start* assim que uma outra âncora de um nó de mídia iniciar. Um exemplo de seu uso ocorre quando uma partida termina, onde é sinalizado que a âncora de "fim" está ocorrendo; é iniciada então a apresentação de outro

nó na tela com uma mensagem de vitória ou derrota.

Após definidas as bases de regiões, descritores e conectores, que se limitam ao cabeçalho do documento NCL, pode-se especificar os objetos de mídias e seus respectivos links (elos), que compreendem o corpo do documento.

Foram declarados dois nós de mídia: **tabuleiro** e **win**. O primeiro nó está associado ao descritor dsTabuleiro e corresponde ao *script* Lua, a parte imperativa da aplicação que será detalhada mais adiante. O último está associado ao descritor dsWin e define uma mídia que é mostrada no centro do tabuleiro assim que uma partida de xadrez termina. Não foi necessário declarar um terceiro nó de mídia para referenciar o descritor dsInfo, já que este está associado a uma região destinada apenas à notificações do jogo, não sendo alocada, portanto, para a apresentação de qualquer mídia. Ainda, o nó **tabuleiro** define uma âncora **fim** que sinaliza o final da partida.

A porta **iniciaTabuleiro** foi definida com a finalidade de mapear o nó **tabuleiro** para dar início à apresentação, isto é, carregar o tabuleiro na tela. Inicialmente é atribuído o foco da aplicação ao nó em questão, permanecendo neste enquanto perdurar uma partida. Um elo também foi definido, para identificar que os nós de mídia mantém uma relação. A relação descreve que, assim que a âncora **fim** é ativada, a apresentação do nó **win** é iniciada no centro do tabuleiro.

Como pode ser percebido, a parte declarativa do protótipo não é muito vasta, visto que se encarrega apenas da apresentação e estrutura dos nós de mídias.

4.3.2 Parte Imperativa Lua

A parte da aplicação desenvolvida utilizando a linguagem de *script* Lua compreende toda a lógica e interface do protótipo, isto é, basicamente todo o funcionamento do xadrez, e foi definida em três *scripts* NCLua. Abaixo são descritos os *scripts* envolvidos:

- **xml.lua**: módulo que fornece um conjunto mínimo de funções para o tratamento de dados XML em Lua. Foi utilizada a versão 1.7.4 que pode ser encontrada em *http://lua-users.org/wiki/LuaXml*.
- **tcp.lua**: módulo responsável pela realização de conexões TCP dentro do código Lua. Utiliza co-rotinas para simular fluxos de execução independentes (multi-threads). Está disponível em *http://www.lua.inf.puc-rio.br/~francisco/nclua/tutorial/exemplo_06.html.*
- **xadrez.lua**: representa o *script* NCLua principal da aplicação. Nele estão

contidas a inclusão dos dois módulos anteriores, além de toda a estrutura lógica do jogo.

O *script* xml.lua fica com o encargo de auxiliar no tratamento das *stanzas* trocadas entre os ambientes de TV, ou seja, entre os jogadores. Sempre que uma *stanza* é recebida por outra entidade, seu conteúdo é convertido em uma tabela Lua contendo um xml estruturado. Assim, todo o seu conteúdo fica acessível de maneira mais simplificada.

O *script* tcp.lua é fundamental para a comunicação. Ele realiza conexões através de canal de retorno a fim de possibilitar a comunicação entre os jogadores. Assim que o jogo é carregado, uma conexão TCP é aberta e assim permanece até que o usuário se desconecte.

O *script* principal, xadrez.lua, é responsável por todo o restante da estrutura do jogo, incluindo a comunicação com o documento NCL. Nele estão implementados a montagem do tabuleiro, as regras do jogo, a função tratadora de eventos, etc. A Figura 4.8 mostra o trecho do código responsável por receber os eventos oriundos do Formatador NCL.

```
1  require 'tcp'
2  dofile('includes/xml.lua')
3
4  -- Funcao de tratamento de eventos:
5  function handler (evt)
6
7          if evt.class ~= 'key' then
8                  return
9          end
10         if (evt.key == 'e') then
11                 encerra_jogo()
12         end
13
14         if evt.type == 'press' then
```

Figura 4.8: Trecho da função tratadora de eventos.

Na linha 1, o módulo *tcp* é carregado para posteriormente suas funções serem utilizadas na realização de uma conexão; na linha 2 é incluso o módulo *xml* para o tratamento das *stanzas*; na linha 5 se encontra a função tratadora de eventos do *script*, tendo como parâmetro a variável *evt*, uma tabela representando os eventos que vierem a ocorrer; a linha 7 verifica se o evento passado como argumento é diferente do tipo *key*, ou seja, apenas eventos oriundos do pressionamento de teclas do controle remoto

interessam aqui; na linha 10 é verificado se a tecla de encerramento foi pressionada e, em caso afirmativo, chama a função encerra_jogo() que desconecta o usuário do jogo; a linha 14 testa se uma tecla foi pressionada e, a partir daí, o evento começa a ser tratado.

Quando o usuário pressiona a tecla de encerramento e a respectiva função é chamada, o *script* envia ao formatador NCL um evento sinalizando o fim da apresentação do nó do tabuleiro, indicando que o usuário está se desconectando.

```
event.post {
    class  = 'ncl',
    type   = 'presentation',
    action = 'stop'
}
```

Figura 4.9: Evento indicando o fim da apresentação do nó.

Quando uma partida chega ao seu final, o *script* envia ao formatador um evento semelhante ao anterior; a diferença está no fato de que o valor do seu atributo *action* agora é "start" e ele dispara uma âncora indicando o início da apresentação de um outro nó de mídia, que fica sobreposto ao nó do tabuleiro.

```
event.post {
    class  = 'ncl',
    type   = 'presentation',
    label  = 'fim',
    action = 'start'
}
```

Figura 4.10: Evento sinalizando que uma âncora está ocorrendo.

4.3.3 Comunicação entre as entidades

Para cada envio de uma *stanza*, a função *send* do módulo tcp é chamada tendo como parâmetro uma cadeia de caracteres representando a *stanza* que se deseja enviar. Cada chamada a essa função é precedida por uma chamada à função *receive* que retorna uma resposta, seja ela do servidor ou do jogador oponente, à mensagem enviada.

Quando o jogo é carregado, dá-se início a todo um processo de negociação entre cliente e servidor XMPP com o intuito de abrir uma conexão e fazer a autenticação do usuário. Primeiramente é feita uma chamada à função *connect* do módulo tcp tendo dois parâmetros passados: o endereço ip do servidor e a sua respectiva porta. Em seguida,

um conjunto de *stanzas* necessárias para realizar a autenticação do usuário no servidor são enviadas. Essas *stanzas* carregam consigo informações referentes a métodos de conexão, segurança, dentre outros, incluindo os dados de acesso do usuário. Vale ressaltar que o descrito se encontra dentro de uma função que é passada como parâmetro para o método *execute* do módulo tcp. Abaixo é mostrado o trecho do código que faz essa autenticação.

```
790 tcp.execute(
791   function ()
792     -- CONECTA AO SERVIDOR XMPP PASSANDO COMO PARAMETRO SEU IP E A PORTA, DEPOIS
793     -- MANDA UM FLUXO DE MENSAGENS XML PARA ABRIR UM CANAL DE COMUNICAÇÃO
794     tcp.connect('192.168.1.34', 5222)
795 
796     local xml = "<?xml version='1.0' ?>"
797       .."<stream:stream to='192.168.1.34' version='1.0' xmlns='jabber:client'"
798       .."xmlns:stream='http://etherx.jabber.org/streams'>"
799 
800     xml ..= "<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl' mechanism='PLAIN'></auth>"
801 
802     xml ..= "<iq type='set' id='auth2'><query xmlns='jabber:iq:auth'><username>usuario1</username>"
803       .."<password>senha</password><resource>casa</resource></query></iq>"
804 
805     tcp.send(xml)
806 
807     result = tcp.receive()
```

Figura 4.11: Trecho responsável pela autenticação do usuário.

Após autenticado, o usuário já estará apto a jogar partidas de xadrez online contra seus contatos.

Cada usuário deve possuir uma lista de contatos, assim como em uma aplicação de mensagens instantâneas, com os quais pode selecionar um deles e desafiá-lo para uma partida. Uma vez que o usuário entra no jogo, uma *stanza* de presença (<presence/>) é enviada ao servidor que fica encarregado de anunciar sua disponibilidade a todos da sua lista. O mesmo ocorre quando ele se desconecta do jogo, sua saída também é transmitida a todos através do envio de uma presença, porém, nesse caso, explicitando sua indisponibilidade (<presence type='unavailable'/>).

A fim de iniciar uma partida, um usuário deve enviar um convite a um de seus contatos, solicitando um desafio. Essa solicitação é feita por meio do envio de uma stanza <message>. Como as *stanzas* são representadas por fluxos em XML, foi possível introduzir *tags* e atributos de acordo com a necessidade em questão. A figura abaixo ilustra a estrutura da solicitação.

```
<message to='usuario2@pc-lord'
  from='usuario1@pc-lord/casa'>
    <invite id='id1'>
        <game id='game1' mycolor='white' />
    </invite>
</message>
```

Figura 4.12: Convite para uma partida.

O jogador "usuario1" envia um convite para o "usuario2" solicitando um desafio. Caso o destinatário aceite o desafio, o solicitante recebe uma resposta definida pelo fluxo abaixo.

```
<message to='usuario1@pc-lord'
  from='usuario2@pc-lord/trabalho'>
    <invite id='id2' invite='accepted'>
        <game id='game1' />
    </invite>
</message>
```

Figura 4.13: Resposta ao convite.

Os dois fluxos mostrados são suficientes para dar início a uma partida de xadrez. Nota-se que as mensagens apresentam algumas informações como, por exemplo, na primeira *stanza,* o atributo *mycolor* que define a cor das peças escolhida pelo jogador. Outra informação é o atributo *invite* que possui o valor "acepted" declarando a aceitação do desafio. Caso o jogador convidado já esteja jogando uma outra partida ou simplesmente tenha recusado o convite, o valor do atributo será "recused".

Durante o andamento de uma partida, os jogadores fazem jogadas cada um em seu respectivo turno, sendo estas transmitidas também por meio de *stanzas* <message> de acordo com a representada a seguir.

```
<message to='usuario2@pc-lord'
  from='usuario1@pc-lord/casa'>
    <move id='id4' turn='black'>
        <piece>cavalo</piece>
        <x>a</x>
        <y>6</y>
    </move>
</message>
```

Figura 4.14: Usuário enviando o movimento de uma peça.

Sempre que uma peça for movida, um fluxo do tipo acima será enviado ao oponente. O atributo *turn,* contido na tag <move>, dita a cor do turno, isto é, da peça que está sendo movida. As *tags* <piece>, <x> e <y> informam o tipo da peça e suas coordenadas finais horizontal e vertical, respectivamente.

Como já mencionado, todas as *stanzas* transmitidas durante a execução do jogo são tradadas por meio do módulo xml. Por exemplo, como uma mensagem de convite e uma de movimento de peça são determinadas pela mesma *stanza*, pode-se varrer o xml em busca de suas *tags* internas e detectar sua natureza, bem como obter as informações requeridas.

De acordo com a arquitetura em que o jogo se sustenta e o protocolo de comunicação descrito, percebe-se que não há a necessidade de um controle intermediário por parte de, por exemplo, um servidor de jogo que responda pelas solicitações de convites ou até mesmo aja como um árbitro. A manipulação de *tags* proporcionada pelo padrão adotado pelo XMPP, o XML, torna possível que ambas as partes (jogadores) sejam capazes de "conversar diretamente".

5. CONCLUSÃO

O trabalho apresentado teve como meta principal propor uma solução para a implementação de jogos de tabuleiro com múltiplos jogadores para a plataforma Ginga. Para tal feito foi necessário um estudo aprofundado das características e funcionalidades do Ginga-NCL, em especial, e da linguagem NCL juntamente com sua linguagem de *script* Lua. Também vale destacar o estudo do protocolo XMPP, que foi fundamental para prover um ambiente onde jogadores geograficamente distantes possam interagir.

O desenvolvimento de aplicações desse ramo voltadas para o ambiente de TV Digital representa um grande passo no que diz respeito à interatividade e à inclusão digital e social já que a TV Digital no país ainda se encontra em processo de implantação. Desse modo, este trabalho pode ser considerado inovador na categoria à medida que desenvolvedores podem contribuir com a implementação de novas aplicações tomando como base o seu contexto e podendo até estendê-lo.

É importante salientar que a solução proposta neste trabalho usou de um protótipo de jogo de xadrez desenvolvido como exemplo de validação, porém a arquitetura e o protocolo de comunicação definidos suportam uma variedade de jogos de tabuleiro, bastando, para tanto, alguns poucos ajustes nas mensagens de comunicação.

Como trabalhos futuros pode ser realizado um refinamento do protótipo de xadrez implementado adicionando-lhe novas funcionalidades. Um exemplo de refinamento seria a implementação de novos recursos como a criação de um mini *chat* embutido no jogo, isto é, a possibilidade de usuários conversarem entre si durante o andamento de uma partida assim como fora dela. Para cada nova funcionalidade, deve-se formatar uma mensagem XML nova ou realizar a adaptação de uma já existente.

6. REFERÊNCIAS BIBLIOGRÁFICAS

(Pesquisa Abragames, 2008) Pesquisa Abragames - A Indústria Brasileira de Jogos Eletrônicos. Disponível em: http://www.abragames.org/docs/Abragames-Pesquisa2008.pdf. Acessado em: abril de 2012.

(Barbosa e Castro, 2008) BARBOSA FILHO, A.; CASTRO, C.; Comunicação digital: educação, tecnologia e novos comportamentos. 1. ed. – SP: Paulinas, 2008.

(Flores et al , 2008) FLORES, L. V.; FAUST, R.; PIMENTA, M. S. Definindo uma Proposta para Avaliações de Usabilidade de Aplicações para o Sistema Brasileiro de TV Digital, Anais do IHC (Simpósio Brasileiro de Fatores Humanos em Sistemas Computacionais) 2008, pág. 88-97. PUCRS.

(Bakken, 2001) BAKKEN, D. E.; Middleware Encyclopedia of distributed computing, Kluwer Academic Press, 2001.

(Middleware Ginga, 2010) Middleware Ginga - TV Interativa se faz com Ginga!. Disponível em: http://www.ginga.org.br. Acessado em: maio de 2012.

(Santos, 2010) SANTOS, R. C. M.; Contribuições para ferramentas de autoria desenvolvidas para a comunidade Ginga. São Luís, Brasil. 2010.

(Soares e Rodrigues, 2005) SOARES, L. F. G.; RODRIGUES, R. F.; Nested Context Model 3.0: Part 1 – NCM Core. Technical Report, Departamento de Informática PUC-Rio. 2005.

(Carvalho et al, 2009) CARVALHO, R.; DOS SANTOS, J. A. F.; DAMASCENO, J. R.; DA SILVA, J. V.; SAADE, D. C. M. Introdução às Linguagens NCL e Lua: Desenvolvendo Aplicações Interativas para TV Digital, Semana de Engenharia, 2009 .

(Soares et al, 2006) SOARES, L. F. G.; RODRIGUES, R. F.; MORENO, M. F. Ginga-ncl: Environment for the execution of declarative applications. 2006.

(Ierusalimschy et al, 2007) IERUSALIMSCHY, R.; FIGUEIREDO, L. H.; CELES, W. The evolution of Lua. Proceedings of ACM HOPL III. 2007.

(Celles et al, 2003) CELES, W.; FIGUEIREDO, L.; H. IERUSALIMSCHY, R. A Linguagem Lua e suas Aplicações em Jogos. Disponível em: http://www.tvdi.inf.br/site/artigos/Lua/A_Linguagem_Lua_suas_Aplicacoes_em_Jogos_-_CELES_FIGUEIREDO_IERUSALIMSCHY.pdf. Acesso em: maio de 2012.

(Cerqueira et al, 2088) CERQUEIRA R.; SANT'ANNA F.; SOARES L. F. G. NCLua: objetos imperativos lua na linguagem declarativa NCL, XIV Simpósio Brasileiro de Sistemas Multimídia e Web - WebMedia, Outubro 26-29, 2008, Vila Velha, Brasil.

(de Melo e Araújo, 2008) DE MELO, J. C. P.; ARAÚJO, R. M. Os Módulos NCL e NCLua Do Middleware Ginga Para Aplicações Em TV Digital Interativa. Natal. Brasil. 2008.

(W3C, 2008) W3C. Xml 1.0 extensible markup language (xml) 1.0 (fourth edition), 2008. Disponível em: http://www.w3.org/TR/REC-xml. Acesso em: maio 2012.

(XMPP Standards Foundation, 2012) The XMPP Standards Foundation. Disponível em: http://xmpp.org/. Acesso em: maio de 2012.

(Saint-Andre et al, 2009) SAINT-ANDRE, P.; SMITH, K.; TRONÇON, R. XMPP: The Definitive Guide, Building Real-Time Applications with Jabber Technologies, O'Reilly Media. 2009.

(Reis e Teixeira) REIS, R. L. B.; TEIXEIRA, M. M. Uma Arquitetura para Interoperablidade entre Aplicativos NCLua e Serviços da Internet. Webmedia / WTIC 2011, Florianópolis. Brasil.

(Santos e Ratamero, 2007) SANTOS, J.; RATAMERO, E.; RummiTV - Jogo Eletrônico para TV Digital Interativa. Rio de Janeiro, 2007.

(Manual RummiTV, 2008) Manual do RummiTV. Disponível em: http://www.midiacom.uff.br/gtvd/rummitv/downloads/Manual.pdf. Acessado em: maio de

2012.

(Ellwanger e Balbinot, 2003) ELLWANGER, F.; BALBINOT, G.; A Linguagem de Programação Lua, III CONSIPA 2003, UNISINOS.

(DamasTV, 2010) DamasTV | Clube NCL. Disponível em: http://clube.ncl.org.br/node/62. Acessado em: maio de 2012.

(Stevens, 1969) STEVENS, A. M.; The Blue Book of Charts to Winning Chess. [S.l.]: A. S. Barnes, 1969.

(XMPP Pubsub, 2011) XMPP Pubsub. 2011. Disponível em: http://http://www.isode.com/whitepapers/xmpp-pubsub.html. Acessado em: junho de 2012.

(XEP-0060, 2010) XEP-0060: Publish-Subscribe. 2010. Disponível em: http://http://xmpp.org/extensions/xep-0060.html. Acessado em: junho de 2012.