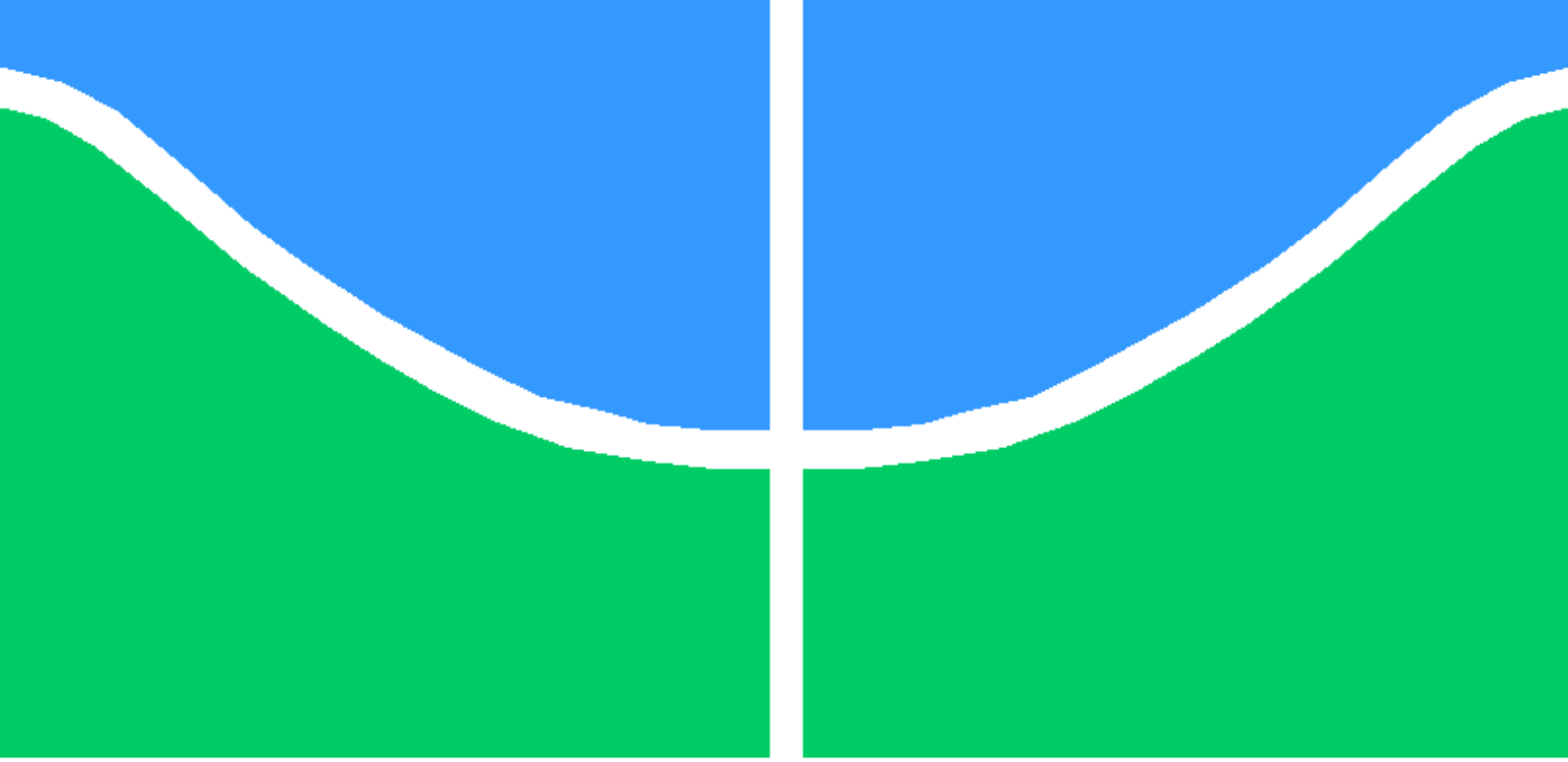


ARQUITETURA ORIENTADA A SERVIÇOS PARA COMÉRCIO ELETRÔNICO NO SISTEMA BRASILEIRO DE TV DIGITAL

MANOEL CAMPOS DA SILVA FILHO

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA ELÉTRICA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

ARQUITETURA ORIENTADA A SERVIÇOS PARA COMÉRCIO ELETRÔNICO NO SISTEMA BRASILEIRO DE TV DIGITAL

MANOEL CAMPOS DA SILVA FILHO

Orientador: PROF. DR. PAULO ROBERTO DE LIRA GONDIM, ENE/UNB

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA ELÉTRICA

PUBLICAÇÃO PPGENE.DM - 439/2011
BRASÍLIA-DF, 16 DE JUNHO DE 2011.

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

ARQUITETURA ORIENTADA A SERVIÇOS PARA COMÉRCIO ELETRÔNICO NO SISTEMA BRASILEIRO DE TV DIGITAL

MANOEL CAMPOS DA SILVA FILHO

DISSERTAÇÃO DE MESTRADO ACADÊMICO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA ELÉTRICA.

APROVADA POR:

Prof. Dr. Paulo Roberto de Lira Gondim, ENE/UnB
Orientador

Prof. Dr. Divanilson Rodrigo Campelo, ENE/UnB
Examinador interno

Prof. Dr. Díbio Leandro Borges, CIC/UnB
Examinador externo

BRASÍLIA, 16 DE JUNHO DE 2011.

FICHA CATALOGRÁFICA
MANOEL CAMPOS DA SILVA FILHO
Arquitetura orientada a serviços para comércio eletrônico no Sistema Brasileiro de TV Digital
2011xv, 107p., 201x297 mm
(ENE/FT/UnB, Mestre, Engenharia Elétrica, 2011)
Dissertação de Mestrado - Universidade de Brasília
Faculdade de Tecnologia - Departamento de Engenharia Elétrica

REFERÊNCIA BIBLIOGRÁFICA

MANOEL CAMPOS DA SILVA FILHO (2011) Arquitetura orientada a serviços para comércio eletrônico no Sistema Brasileiro de TV Digital. Dissertação de Mestrado em Engenharia Elétrica, Publicação 439/2011, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 107p.

CESSÃO DE DIREITOS

AUTOR: Manoel Campos da Silva Filho
TÍTULO: Arquitetura orientada a serviços para comércio eletrônico no Sistema Brasileiro de TV Digital.
GRAU: Mestre ANO: 2011

Manoel Campos da Silva Filho
Universidade de Brasília, Faculdade de Tecnologia, Departamento de Engenharia Elétrica, 70910-900, Brasília-DF, Brasil.

Agradecimentos

Gostaria de agradecer primeiramente ao Instituto Federal de Educação, Ciência e Tecnologia do Tocantins (IFTO), instituição onde leciono, por ter me liberado por dois anos para cursar o programa de mestrado em Engenharia Elétrica da Universidade de Brasília; à CAPES pela ajuda financeira durante este período; e um agradecimento especial a meus grandes amigos e companheiros (em ordem alfabética) Cláudio Monteiro, Helder Cleber, Leandro Vaguetti, Lilissanne Marcelly, Maurício Júnior, Vanice Cunha e Vinícius Rios, que tornaram esta jornada menos cansativa e mais divertida, que foram amigos de todas as horas, tanto nas de alegria quanto nas de tristeza. Por fim, e não menos importante, gostaria de agradecer a meu orientador, o professor Paulo Gondim, por todos os ensinamentos e apoio durante esta jornada.

Resumo

Arquitetura orientada a serviços para comércio eletrônico no Sistema Brasileiro de TV Digital

Esta dissertação descreve uma arquitetura orientada a serviços para provimento de comércio eletrônico pela TV Digital, por meio do Sistema Brasileiro de TV Digital (SBTVD), desenvolvida para o sub-sistema Ginga-NCL do *middleware* Ginga. A arquitetura proposta utiliza serviços de diferentes provedores (nas áreas de telecomunicações, logística e outros) para compor uma estrutura de *T-Commerce*. Tais serviços são desenvolvidos considerando aspectos de interoperabilidade, utilizando o protocolo SOAP, para o qual é apresentada uma implementação, juntamente com o HTTP, como base para o desenvolvimento de toda a arquitetura e um dos objetivos principais do projeto.

Com a arquitetura elaborada, uma aplicação cliente, desenvolvida em NCL e Lua, é apresentada como prova de conceito do uso das implementações dos protocolos e da arquitetura proposta. Tal aplicação utiliza o *framework* LuaOnTV para a construção da interface gráfica de usuário para a TV Digital, o qual foi estendido neste trabalho, com as melhorias sendo apresentadas ao longo do mesmo.

O trabalho ainda apresenta um conjunto de aplicações desenvolvidas a partir dos *frameworks* construídos, que complementam as funcionalidades da aplicação de T-Commerce, como leitor de RSS e rastreamento de encomendas.

A partir do ambiente de desenvolvimento montado para a construção das aplicações, contendo a implementação de referência do sub-sistema Ginga-NCL do *middleware* Ginga, nativamente instalada, foi gerada uma distribuição Linux que permite que tal ambiente seja instalado em qualquer computador ou máquina virtual, para permitir o desenvolvimento de arquitetura semelhante ou extensão da arquitetura proposta.

Palavras-chave: SBTVD, Ginga, Ginga-NCL, *Web Services*, HTTP, SOAP, SOA, Lua, NCL, NCLua, NCLua HTTP, NCLua SOAP, LuaOnTV 2.0, *E-Commerce*, *T-Commerce*

Abstract

Service-oriented architecture for electronic commerce in the Brazilian Digital Television System

This dissertation describes a service-oriented architecture for providing of digital TV electronic commerce, through the Brazilian Digital Television System, developed to the Ginga-NCL sub-system of the Brazilian Ginga middleware. The proposed architecture uses services from distinct providers (at telecommunication, logistics and other areas) to compose a T-Commerce structure. Such services are developed considering interoperability aspects, using the SOAP protocol, for wich is presented an implementation, together with the HTTP protocol, as a basis for the development of the entire architecture and one of the project main goals.

With the architecture designed, a client application, developed in NCL and Lua languages, is presented as a proof of concept of the protocols implementations and proposed architecture use. Such application uses the LuaOnTV framework to build a Digital TV graphical user interface, wich was extended in this dissertation, with the improvements being presented along it.

The work also presents a set of applications developed from the constructed frameworks that complement the T-Commerce application functionalities, such as RSS reader and orders tracking.

From the mounted development environment for applications building, containing the reference implementation of the Ginga-NCL sub-system of the Ginga middleware, natively installed, a Linux distribution was generated that enables such environment to be installed on any computer or virtual machine, to allow the development of similar architecture or extension of the proposed one.

Keywords: ISDB-TB, Ginga, Ginga-NCL, Web Services, HTTP, SOAP, SOA, Lua, NCL, NCLua, NCLua HTTP, NCLua SOAP, LuaOnTV 2.0, E-Commerce, T-Commerce

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS FONTE

Capítulo 1

Introdução

Nas últimas décadas, observou-se uma evolução gradual das tecnologias da informação, e como desdobramento desse evento, a organização e o comportamento humano sofreram modificações. Prova disso são os sistemas de compras pelas redes de computadores, que nos últimos tempos tiveram um crescimento acelerado. O Brasil, assim como outros países pelo mundo, possui vários casos de sucesso de lojas de comércio eletrônico (*E-Commerce*)[1].

Esse fato se deve à comodidade de, a partir de um computador ligado à *Internet*, poder-se realizar compras de bens e serviços. Nesse modelo, o usuário tem a facilidade e a liberdade de se usar o tempo que julgar necessário para avaliar o produto que deseja adquirir, além de efetuar pesquisas em várias lojas ao mesmo tempo.

Outro fator importante, está na qualidade do atendimento, devido aos atendentes possuírem em mãos um sistema de informação para prover respostas ágeis e credenciadas. Assim, o usuário economiza tempo e se exime de problemas naturais da organização contemporânea como: as filas, gastos de deslocamento, trânsito e estacionamento.

Além disso, as lojas virtuais possuem sistemas de recomendação de produtos, que com base no perfil e nas aquisições habituais do usuário, oferecem produtos incentivando o usuário a utilizar o *E-Commerce*. Todos esses benefícios são alguns dos fatores de sucesso das maiores lojas de *E-Commerce* no Brasil e no mundo.

A Pesquisa sobre o Uso das Tecnologias de Informação e Comunicação no Brasil/TIC Domicílios em 2009 (última divulgada até a data de finalização desta dissertação)[CETIC.br 2009], realizada pelo Centro de Estudos sobre as Tecnologias da Informação e da Comunicação (CETIC.br) [2] e ao Comitê Gestor da *Internet* no Brasil (CGI.br) mostrou que, de 2008 para 2009, houve um aumento de 8% na consulta a preços de produtos ou serviços na *Internet*, passando de 44% para 52% em todo o Brasil, e que houve um crescimento nas compras *on line* de 3%, passando de 16% para 19% nacionalmente. A pesquisa ainda revela que, do total de domicílios pesquisados em 2009, 32% tinham computador

[1] `www.americanas.com.br`, `www.submarino.com.br`, `www.pontofrio.com.br` e outros

[2] Vinculado ao Núcleo de Informação e Coordenação do Ponto BR (NIC.br)

(contra 25% em 2008) e 25% tinham acesso à *Internet* (contra 18% em 2008).

Assim os dados da pesquisa supracitada mostram que, no Brasil, as buscas por produtos e serviços na *Internet* é crescente e que gradualmente mais pessoas têm acesso ao comércio eletrônico.

O comércio eletrônico possui outros suportes além do computador, tais como o celular e, mais recentemente, a TV digital. A compra de produtos pela TV não é algo novo. Atualmente existem até canais específicos para tal atividade, no entanto, o usuário precisa utilizar um outro canal para finalizar o processo de compra. A TV Digital traz a facilidade de permitir que todo este processo seja iniciado e finalizado diretamente do controle remoto da TV, trazendo mais comodidade para os usuários, em uma modalidade denominada *T-Commerce*.

As possibilidades dos recursos de interatividade da TV Digital (TVD) são inúmeras, dependendo da criatividade das produtoras de conteúdo e desenvolvedores de *software*. Um dos sonhos atualmente possíveis com as tecnologias de TVD é a compra de produtos que estejam sendo exibidos em um programa de TV convencional, como um tênis, uma bolsa, um quadro, ou qualquer outro.

Tendo em vista esta tendência crescente de provimento de serviços nas mais diversas plataformas e o sucesso de vários serviços de comércio eletrônico no Brasil e no mundo, a presente dissertação descreve uma arquitetura para provimento de comércio eletrônico pela TV Digital (*T-Commerce*). A arquitetura proposta é composta por diversos serviços *Web* que, juntos, agregam todos os serviços que são disponibilizados aos usuários dos sistemas de *T-Commerce* a serem desenvolvidos a partir de tal arquitetura. Esta arquitetura é então enquadrada como uma Arquitetura Orientada a Serviços (*Service Oriented Architecture* - SOA), voltada para o provimento de serviços de *T-Commerce*.

Tal arquitetura foi elaborada devido não terem sido encontrados outros trabalhos que tratem de uma proposta de comércio eletrônico para a TV Digital. Assim, a proposta aqui apresentada serve como base para o desenvolvimento de aplicações de *T-Commerce*, visando a popularização de serviços de comércio eletrônico pelo Sistema Brasileiro de TV Digital. Considerando que o sinal analógico de TV Digital está previsto para ser desligado em 2016, segundo previsão do Ministério das Comunicações[3], todos os brasileiros que desejarem receber sinal de TV, precisarão de um televisor com conversor integrado ou um conversor para conectar à um televisor convencional. Desta forma, sistemas de *T-Commerce* podem ter um grande alcance.

A arquitetura proposta, elaborada a partir de requisitos funcionais e não funcionais, serve de base para a implementação de aplicativos para comércio eletrônico, para o qual são também elicitados requisitos funcionais e não funcionais. Esses aplicativos são construídos com base em um modelo de *templates* para definir a identidade visual da mesma, permitindo que, ao ser alterado o *template*, toda a identidade visual da aplicação seja alterada.

[3] `http://goo.gl/nLVnW`

A arquitetura e a aplicação de *T-Commerce* propostas utilizam *Web Services* para disponibilizar funcionalidades aos usuários. Desta forma, tal proposta vai ao encontro da também crescente tendência de integração entre *Web* e TV. Tal integração é possível por meio de protocolos de comunicação padronizados, como o caso dos protocolos HTTP e SOAP, que neste trabalho são objeto de implementações específicas.

Para esta integração entre *Web* e TV, apresenta-se uma proposta de *framework* de comunicação de dados que possibilita a comunicação entre aplicações de TV Digital e serviços disponíveis na *Web*, o qual tem seu emprego demonstrado não somente por meio de aplicações de *T-Commerce*, mas também de rastreamento de encomendas, leitor de RSS e outras.

Um *framework* é um conjunto de classes que incorporam um projeto abstrato de soluções para uma família de problemas relacionados[Johnson e Foote 1988]. O mesmo pode ser também denominado como arcabouço, estrutura, esqueleto, suporte e outros termos.

Alguns dos requisitos do aplicativo de *T-Commerce* supracitado são contemplados com a extensão do *framework* LuaOnTV para permitir o uso de temas e a adaptação automática da interface de usuário da aplicação para diferentes tamanhos de TV ou até mesmo em dispositivos móveis como telefones celulares.

1.1 Objetivos

1.1.1 Geral

Propor e desenvolver uma arquitetura orientada a serviços, por meio de *Web Services* SOAP, para provimento de comércio eletrônico pela TV Digital, favorecendo a convergência *Web*-TV.

1.1.2 Específicos

Como objetivos específicos, propõe-se:

A) elicitar requisitos funcionais e não funcionais a serem atendidos por uma arquitetura baseada em padrões de serviços *Web*, a ser utilizada para comércio eletrônico via TV Digital;

B) propor uma arquitetura de serviços de *T-Commerce*, que atenda aos requisitos citados;

C) a partir da arquitetura , implementar um *framework* para comunicação de dados, baseado nos protocolos HTTP e SOAP, para permitir a comunicação de aplicações de TV Digital, desenvolvidas nas linguagens NCL e Lua, que permita a interoperação com serviços *Web*;

D) caracterizar o emprego do *framework* de comunicação de dados para desenvolvimento de aplicações tais como Leitor de RSS, Rastreador de Encomendas, Cliente de Twitter, Enquete e *Quiz*;

E) estender o *framework* LuaOnTV[Júnior e Gondim 2009] incluindo recursos de temas para permitir a definição centralizada das características visuais das aplicações desenvolvidas, além de incluir suporte a múltiplos dispositivos com diferentes resoluções de tela, como TV's e aparelhos celulares;

F) um modelo de desenvolvimento de aplicações para TV Digital (TVD), de forma que todos os formulários da aplicação (páginas/telas) tenham um mesmo conjunto de componentes básicos, permitindo a criação de *templates* para definir a identidade visual da mesma. Assim, ao ser alterado o *template*, toda a identidade visual da aplicação deverá ser alterada;

1.2 Justificativa

Como o início das operações do Sistema Brasileiro de TV Digital (SBTVD) é bastante recente, onde a primeira transmissão de sinal digital foi realizada em 2007 na cidade de São Paulo, até a data de elaboração da presente dissertação, só conhecia-se um caso de aplicação de *T-Commerce* no Brasil, como citado em [Síntese 2010], no qual apenas exibem-se os produtos e o processo de compra deve ser realizado pelo usuário utilizando outro canal como a loja virtual (*Internet*) ou o *call center* da empresa.

Além disso, tal aplicação de comércio eletrônico foi desenvolvida para o *middleware ByYou* (implementação do Ginga desenvolvida pela empresa TOTVS)[TeleTime 2010], e seu uso fica restrito aos conversores digitais e televisores que possuam tal *middleware*. Por usar API's específicas do *ByYou*, a aplicação só executa em tal implementação do *middleware* Ginga.

Como o Brasil possui diversos casos de sucesso no mercado de comércio eletrônico em larga escala, a disponibilização de aplicações de comércio para TV Digital é uma tendência natural, podendo aumentar consideravelmente as vendas das empresas do ramo, devido à grande penetração do aparelho de TV nos lares brasileiros (cerca de 96%)[IBGE 2009], além de ser uma nova plataforma de *E-Commerce* para os usuários.

Da falta de uma arquitetura para provimento de comércio eletrônico para o SBTVD, surgiu este trabalho de dissertação, que apresenta uma proposta de arquitetura orientada a serviços (*Service Oriented Architecture* - SOA) para a estruturação de um ambiente de *T-Commerce*. Considerando que tal arquitetura é bastante utilizada para a integração de sistemas heterogêneos, ele vai ao encontro de um dos objetivos do projeto: prover uma arquitetura de *T-Commerce* que utilize serviços *Web* que, juntos, atuem de forma interoperável com o SBTVD e os padrões W3C.

Desta forma, como a arquitetura SOA pode, comumente, ser baseada em *Web Services* SOAP, faz-se necessária a implementação de protocolos como *Hypertext Transfer Protocol* (HTTP) e *Simple Object Access Protocol* (SOAP) em que se baseiam as tecnologias relacionadas a SOA, sendo tais implementações objetivos principais deste trabalho.

Com a implementação dos protocolos HTTP e SOAP (onde não se tem conhecimento, até a data de elaboração desta dissertação, de nenhuma implementação de código aberto dos mesmos para o ambiente de TVD) criam-se enormes possibilidades de construção de aplicações para integração entre *Web* e TV, um dos objetivos deste projeto com sua arquitetura de *T-Commerce*. Como durante as pesquisas observou-se que a falta de implementação de tais protocolos era uma queixa recorrente nos fóruns da Comunidade Ginga no Portal do Software Público[4] e em outros fóruns de discussão, a implementação de tais protocolos representa uma grande contribuição para o desenvolvimento do SBTVD. Assim, após a publicação inicial das implementações dos protocolos HTTP e SOAP, alguns trabalhos

[4] `http://www.softwarepublico.gov.br/dotlrn/clubs/ginga/`

importantes foram desenvolvidos, como será apresentado no Capítulo 5.

1.3 Metodologia

Para o desenvolvimento da presente dissertação foi feito um levantamento bibliográfico sobre as tecnologias, linguagens, ferramentas e trabalhos relacionados para nortear a implementação da arquitetura e solução proposta.

O projeto, análise e desenvolvimento da solução seguiu o processo de desenvolvimento em cascata juntamente com o processo de componentização[Sommerville 2011]. O processo em cascata foi adotado pois os requisitos estavam bem definidos, havendo pequena probabilidade de mudanças radicais. Já o processo de componentização foi também adotado devido ao uso de alguns componentes reutilizáveis (como *Web Services* próprios e de terceiros) realizando a integração destes diversos componentes.

No processo de desenvolvimento em cascata, foram seguidas as seguintes etapas:

- especificação de requisitos - foram definidos os requisitos funcionais e não funcionais, que são apresentados ao longo do trabalho;
- projeto - foi adotado o paradigma de Orientação a Objetos (OO), utilizando-se a *Unified Modeling Language* (UML) para modelagem, com o auxílio de ferramenta CASE (*Computer-Aided Software Engineering*);
- implementação - a implementação seguiu o paradigma OO, conforme definido na fase de projeto;
- testes - foram feitos testes de interoperabilidade/integração para verificar se os componentes da arquitetura estavam se comunicando conforme o esperado.

Optou-se, para o desenvolvimento do projeto, pelo paradigma de orientação a objetos pois o mesmo permite um alto grau de reutilização de código, tornando o mesmo mais organizado e fácil de manter.

Para a comunicação entre os componentes da arquitetura da solução, foi escolhido o protocolo de comunicação SOAP, uma vez que a aplicação de TV Digital desenvolvida faz parte de uma arquitetura distribuída e precisa realizar comunicação com servidores *Web* por meio da *Internet*. Assim, foi preciso usar um protocolo que não tivesse problemas de bloqueio em *firewalls*. Desta forma, o protocolo SOAP foi escolhido também por ser padrão do *World Wide Web Consortium* (W3C).

Para as aplicações de TV Digital, foi escolhido o sub-sistema Ginga-NCL do *middleware* Ginga do Sistema Brasileiro de TV Digital (SBTVD), devido à grande quantidade de documentos, fóruns de discussão, ferramentas e exemplos disponíveis em relação à outra vertente de desenvolvimento utilizando a linguagem Java no sub-sistema Ginga-J. Considerando-se ainda que o Ginga-NCL é o único sub-sistema obrigatório para dispositivos móveis, isto foi decisivo para a escolha, pois assim, a arquitetura e as aplicações desenvolvidas podem, em

tese, ser executadas em receptores (conversores/*Set-bop Boxes*) de TV Digital fixos, móveis ou portáteis.

Com a escolha do Ginga-NCL, foi necessária a implementação do protocolo SOAP para este sub-sistema, utilizando-se a linguagem Lua, uma vez que o primeiro só disponibiliza protocolos até a camada de transporte do modelo OSI/ISO, como o TCP. Com isto, foi necessário realizar testes de interoperabilidade entre aplicações de TV Digital desenvolvidas com as linguagens NCL e Lua e *Web Services* desenvolvidos em diferentes plataformas e linguagens, pois um dos grandes benefícios do protocolo SOAP é a integração de sistemas heterogêneos. Assim, tais testes foram necessários para garantir esta interoperabilidade.

Os requisitos da aplicação foram levantados tomando por base as funcionalidades existentes na grande maioria dos *Web sites* de comércio eletrônico disponíveis na *Internet* e bastante difundidos no Brasil.

Para a construção da interface de usuário das aplicações de TV Digital, optou-se pela utilização do *framework* LuaOnTV para abstrair as primitivas gráficas para renderização da interface. A mesma foi projetada baseada em recursos de *templates*, o que permite a personalização e adaptação automática para diferentes resoluções de tela, tendo sido estendido o LuaOnTV para incluir tais recursos.

1.4 Organização do Trabalho

- O Capítulo 2 apresenta uma revisão bibliográfica das tecnologias empregadas e relacionadas ao desenvolvimento do trabalho.
- O Capítulo 3 apresenta a arquitetura de *T-Commerce* proposta.
- O Capítulo 4 apresenta o *Framework* LuaOnTV, utilizado na construção das interfaces gráficas das aplicações desenvolvidas.
- O Capítulo 5 apresenta um *framework* de comunicação de dados, utilizado para a integração das aplicações desenvolvidas com serviços *Web*.
- Por fim, o Capítulo 6 apresenta as conclusões e trabalhos futuros.

Capítulo 2

Revisão bibliográfica

2.1 Comércio Eletrônico: *E-Commerce*, *M-Commerce* e *T-Commerce*

No início da *World Wide Web* (comumente denominada apenas *Web*) as primeiras páginas de *Internet* possuíam apenas conteúdo estático permitindo pouca ou nenhuma interação do usuário. A *Web* era apenas um repositório de informações. Com a crescente demanda de troca de informações entre empresas, o advento de padrões abertos de comunicação, e a necessidade das mesmas de alcançarem novos clientes e mercados, durante a década de 90 começou a surgir um novo gênero de *Web sites*: os *sites* de comércio eletrônico[Chu et al. 2007].

Tais *Web sites* possibilitaram a realização de negócios entre compradores e vendedores e entre vendedores e seus parceiros. Desta forma surgiu o *E-Commerce*. Este é baseado na possibilidade de realização de transações de forma eletrônica. Segundo [Veijalainen, Terziyan e Tirri 2006]:

> "uma transação eletrônica é uma venda ou compra de produtos ou serviços, entre empresas, familiares, indivíduos, governos e outras organizações públicas ou privadas, conduzidas por redes mediadas por computador."

A troca de informações entre essas empresas era feita por meio de *Eletronic Data Interchange* (EDI) (troca eletrônica de informações), no entanto, isto requeria realização de acordos entre as organizações[Chu et al. 2007], o que poderia dificultar tais parcerias. O advento de padrões abertos como o XML permitiu a evolução de tais processos de intercâmbio de dados e integração entre empresas.

Atualmente existem diversas empresas consolidadas na área de comércio eletrônico. Algumas nasceram na era digital e vendem exclusivamente pela *Internet*, alcançando novos clientes e mercados nacionais e internacionais.

Com a recente popularização de dispositivos móveis e da *Internet* sem fio como rede de grande abrangência (por exemplo, utilizando as tecnologias de comunicação móvel de 3ª geração como o *Universal Mobile Telecommunications System* - UMTS), surgem novas oportunidades para as lojas de comércio eletrônico. Estas entram em uma nova era, com novas perspectivas de captação de clientes e mercados. Com isto surgem novas tendências como o denominado Comércio Móvel (*M-Commerce*), onde as transações são feitas utilizando-se dispositivos e redes de acesso móveis, tais como *Wireless Local Area Networks* (WLAN's), redes de telecomunicações 2G ou 3G, conexões *Bluetooth* ou infravermelho[Veijalainen, Terziyan e Tirri 2006].

Tais dispositivos móveis permitem que os usuários possam realizar compras em qualquer lugar que eles estejam, até mesmo em suas horas livres, durante o trajeto para o trabalho, ou no horário de almoço. Desta forma, as lojas virtuais têm um mercado em potencial para aumentar suas vendas.

Com o advento dos sistemas de televisão digital interativa e a possibilidade de se ter aplicativos executando juntamente com a programação áudio-visual, abre-se um novo mercado para as lojas de comércio eletrônico: a venda de produtos pela TV. Tais serviços de vendas pela TV não são novos, mas antes da TV Digital Interativa (TVDi), os canais de venda pela TV apenas anunciavam produtos e os telespectadores que desejavam comprar um produto precisavam utilizar outro canal de comunicação, como o telefone ou recorrer ao *Web site* da loja na *Internet*. Utilizando-se os recursos da TV Digital Interativa, todo o processo de compra pode ser realizado diretamente pelo controle remoto da TV, desde que a mesma esteja conectada à *Internet*.

As tecnologias da TV Digital trazem um novo benefício aos usuários: a possibilidade de comprar produtos a partir de um receptor de TV Digital, caracterizando uma nova modalidade de comércio eletrônico, denominada *T-Commerce*.

Em países norte-americanos e europeus, que têm sistemas de TV Digital Interativa há mais tempo que o Brasil, existem algumas empresas disponibilizando soluções para a área de *T-Commerce* [1] [2] [3]. No Brasil, apesar de as transmissões de TV Digital terem iniciado somente em 2007 na cidade de São Paulo[G1 2007], em 2010 já há soluções iniciais para *T-Commerce*[Síntese 2010], apesar de a finalização da compra ainda precisar ser feita utilizando-se outros canais como o telefone.

Devido ao Brasil ser um país de características bem diferentes dos outros países, espera-se que a interatividade seja um grande diferencial para o país. Como o Governo Federal pretende realizar inclusão social/digital por meio da TV, de acordo com o Decreto número 4.901, de 26 de novembro de 2003, espera-se que a disponibilização de serviços pela TV seja crescente. Um fator importante para a difusão da interatividade no país é a existência de aparelhos de TV em 96% das residências[IBGE 2009], tendo grandes possibilidades de

[1] `http://www.ensequence.com/t-commerce`
[2] `http://www.digisoft.tv/applications.html`
[3] `http://www.icuetv.com/ets_platform/applications/t_commerce`

as aplicações interativas terem um enorme público, considerando ainda as dimensões continentais do Brasil e sua enorme população de 185.712.713 habitantes, segundo o Censo 2010[IBGE 2010].

Outra medida tomada pelo governo que beneficiará a interatividade pela TV Digital é a criação do Plano Nacional de Banda Larga (PNBL), por meio do Decreto número 7.175, de 12 de maio de 2010, visando prover *Internet* banda larga, a um custo reduzido, em municípios onde não exista tal serviço.

2.2 Sistema Brasileiro de TV Digital (SBTVD)

O Sistema Brasileiro de TV Digital (SBTVD) atualmente é o mais avançado do mundo. A necessidade de o Brasil implantar um sistema de TV Digital levou o Governo Federal a emitir o Decreto 4.901, de 26 de novembro de 2003, instituindo o referido sistema.

A partir daí, o governo passou a investir diretamente na pesquisa de tecnologias para TV Digital. Considerando que já havia outros padrões de TV Digital no mundo, como o americano ATSC (*Advanced Television Systems Committee*), o europeu DVB (*Digital Video Broadcast*) e o japonês ISDB-T (*Integrated Services Digital Broadcasting Terrestrial*), tais estudos concluíram que o melhor para o país seria a adoção do sistema japonês ISDB-T, que seria estendido para atender às características e necessidades do Brasil.

Desta forma, o Decreto 5.820, de 29 de junho de 2006 estabelece o uso do ISDB-T como base para o Sistema Brasileiro de TV Digital Terrestre (SBTVD-T). O decreto ainda define a criação do Fórum SBTVD (Fórum do Sistema Brasileiro de TV Digital). Tal fórum, como menciona o decreto supracitado, tem como atribuições:

> "a assessoria acerca de políticas e assuntos técnicos referentes à aprovação de inovações tecnológicas, especificações, desenvolvimento e implantação do SBTVD-T".

O mesmo foi composto, como cita o decreto:

> "entre outros, por representantes do setor de radiodifusão, do setor industrial e da comunidade científica e tecnológica"

Dentre as contribuições do Brasil para a área de TVD, destaca-se a construção do Ginga[4] como padrão de *middleware* para aplicações interativas e não interativas. O Ginga é uma inovação nacional, atuando como responsável pela execução e controle do ciclo de vida das

[3] `http://www.forumsbtvd.org.br`
[4] `http://www.ginga.org.br`

aplicações interativas, abstraindo o sistema operacional e o hardware para elas, além de realizar tarefas especializadas como a disponibilização de *Application Programming Interfaces* (API's) para facilitar o desenvolvimento de aplicações interativas[Soares e Barbosa 2009].

O Ginga é composto por dois sub-sistemas: um para a execução de aplicações declarativas, denominado Ginga-NCL[5], e outro para execução de aplicações procedurais, denominado Ginga-J[6]. Aplicações declarativas são aquelas implementadas utilizando-se uma linguagem declarativa, como XML, de forma não algorítmica, apenas declarando elementos que comporão a mesma. Tais linguagens abstraem muitos detalhes do desenvolvedor. Aplicações procedurais são aquelas implementadas utilizando uma linguagem procedural, como Java, onde o desenvolvedor precisa escrever um algoritmo e especificar cada operação a ser realizada, necessitando de conhecimento em linguagens de programação.

O sub-sistema Ginga-NCL permite a construção de aplicações declarativas por meio da linguagem NCL, a *Nested Context Language*[7]. Esta é conhecida como uma linguagem de cola, que permite criar aplicações declarativas juntando-se diferentes tipos de mídias como imagens, texto, hipertexto (HTML), vídeos e outros. Um de seus principais recursos é a sincronização de mídias, muito importante no contexto de aplicações interativas. Tal recurso garante que, por exemplo, uma legenda seja sincronizada com um vídeo em exibição, ou que uma imagem apareça depois de determinado tempo que o vídeo iniciou. As aplicações NCL podem ter seu poder estendido com a inclusão de mídias especiais: os *scripts* Lua[8], conhecidos em aplicações NCL como NCLua. Tais *scripts* adicionam características procedurais às aplicações NCL.

O outro sub-sistema que compõe o Ginga é o chamado Ginga-J, que permite a construção de aplicações interativas utilizando a linguagem Java. Os outros padrões de *middleware* de TV Digital pelo mundo, na parte procedural, normalmente utilizam a linguagem Java e a API JavaTV. No entanto, os fabricantes, para embarcarem a máquina virtual Java e tal API nos conversores de TV Digital precisam pagar *royalities* à Oracle, empresa que atualmente detém os direitos sobre a marca Java. Com isto, o preço dos conversores é encarecido. Além disto, tal API é baseada no *toolkit* gráfico AWT, que é bastante antigo e não possui componentes com visual bonito e elegante. Devido às questões de pagamento de *royalities*, o Fórum SBTVD decidiu fazer um acordo entre a então Sun, hoje Oracle, para a definição de uma nova API para ser utilizada pelo SBTVD, a qual foi batizada de JavaDTV[9]. Tal API é livre de *royalities* e é baseada no *toolkit* gráfico LWUIT[10], o *Light Weight UI Toolkit*. O LWUIT é um *toolkit* que possui componentes bonitos e elegantes e é o mesmo utilizado em aplicações para celulares.

Com as contribuições brasileiras, o ISDB-T e o SBTVD-T passaram a compartilhar uma

[5] `http://www.gingancl.org.br`
[6] `http://dev.openginga.org`
[7] `http://www.ncl.org.br`
[8] `http://www.lua.org`
[9] `http://www.forumsbtvd.org.br/materias.asp?id=75`
[10] `http://lwuit.java.net`

denominação internacionalmente conhecida como ISDB-TB, onde "B"representa as contribuições do Brasil para o ISDB-T[SBTVD 2010].

As melhorias resultantes da parceria realizada levaram o ISDB-TB a se tornar norma internacional de TV Digital no ITU-T (*International Telecommunications Union - Telecommunication Standardization Sector*)[InfoExame 2009][IDGNow 2010]. O Ginga se tornou padrão de *middleware* para TVD, relativo à recomendação ITU-T J.200[ITU-T 2010]. Seu sub-sistema declarativo Ginga-NCL se tornou recomendação ITU-T J.201[ITU-T 2009] e seu sub-sistema imperativo Ginga-J se tornou recomendação ITU-T J.202[ITU-T 2010].

Ressalta-se, também, a recente normatização, no âmbito do ITU-T, referente a IPTV, conhecida como H.761, baseada fortemente no *middleware* Ginga e seu sub-sistema Ginga-NCL. Desta forma, o Ginga pode chegar a novas plataformas que não apenas a TV aberta, e as aplicações interativas e não interativas desenvolvidas em NCL/Lua para o SBTVD podem também ser executados em sistemas de IPTV que adotem o Ginga-NCL.

2.3 A Tecnologia de *Web Services*

Há alguns anos as aplicações corporativas eram desenvolvidas principalmente em um modelo cliente/servidor de duas camadas, existindo uma aplicação *desktop* que fazia acesso direto a um banco de dados. Pela natureza destas aplicações, que precisam ser instaladas em cada computador onde serão executadas, existe um grande esforço para atualizar todos os computadores com novas versões do sistema. Mesmo que haja um processo automatizado para esta atualização, isto demanda recursos como tempo e largura de banda. Com o avanço das tecnologias de comunicação de dados, o avanço da *Web* e suas linguagens de programação e, principalmente, com o advento da tecnologia AJAX, atualmente é possível desenvolver aplicações de *Internet* ricas (*Rich Internet Applications*, RIA) com componentes visuais que vão além dos componentes básicos oferecidos pela linguagem HTML. Isto permitiu que muitas empresas migrassem seus sistemas *desktop* para a plataforma *Web*, sem perder funcionalidades existentes naquele tipo de interface.

Essa mudança de paradigma *desktop* para *Web* vem seguida ainda de outra tendência: a dos sistemas distribuídos. Estes são sistemas que comumente utilizam a arquitetura cliente/-servidor, no entanto, são compostos de três ou mais camadas. Segundo [Sommerville 2011]:

> "um sistema distribuído é uma coleção de computadores independentes que aparece para o usuário como um único sistema coerente."

A Figura 2.1 apresenta uma arquitetura de um sistema distribuído em quatro camadas. As mesmas são descritas a seguir.

- Camada de Apresentação: pode contar tanto com aplicações *desktop* leves, conhecidas como *thin client* (cliente magro/leve), possuindo apenas uma interface gráfica que faz

acesso a um servidor de aplicação, por meio de chamadas de procedimento remoto; quanto com um *browser*, que faz acesso a um servidor *Web*, responsável por gerar as interfaces HTML.

- Camada *Web*: composta por um ou mais servidores *Web*, responsáveis por gerar a interface HTML para os clientes *Web*. As aplicações *desktop* não acessam esta camada, tendo ligação direta com a Camada de Aplicação.

- Camada de Aplicação: composta por um ou mais servidores de aplicação onde as regras de negócios da aplicação estão implementadas. Desta forma, alterações na lógica de negócios não requer a atualização dos clientes. Os servidores de aplicação adicionam tolerância a falhas e balanceamento de cargo ao sistema.

- Camada de Dados: composta por um ou mais servidores de banco de dados, acessíveis apenas pelos servidores de aplicação, o que garante transparência e independência de banco de dados aos clientes.

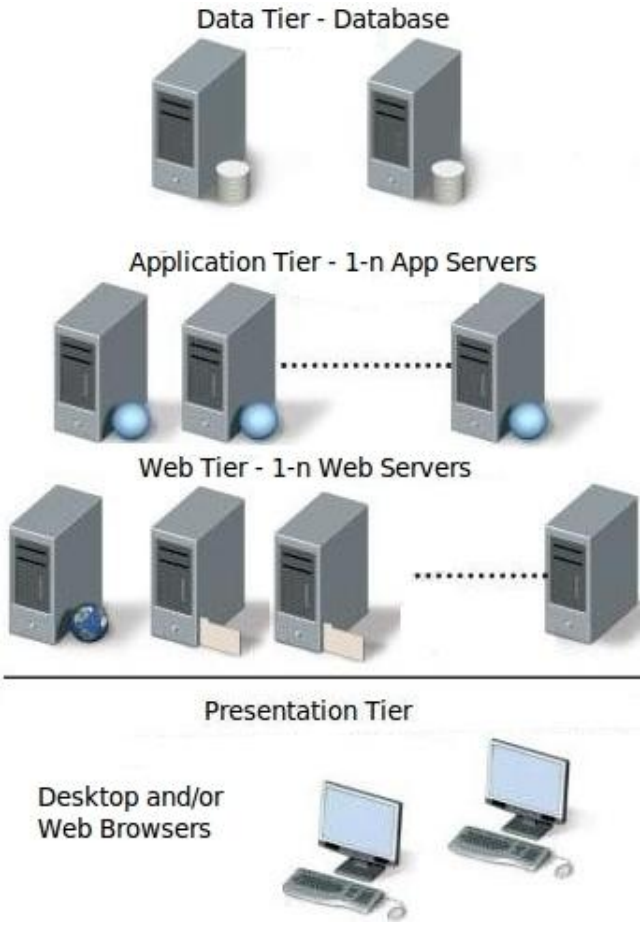


Figura 2.1: Arquitetura de Aplicação Distribuída (adaptada de [InterSystems 2010]).

Segundo [Coulouris, Dollimore e Kindberg 2007], um sistema distribuído traz benefícios como: permitir a heterogeneidade de componentes, possibilitar a escalabilidade, ser tolerante a falhas, dentre outros.

Neste contexto, *Web Services* (WSs) proveem um *framework* extensível para comunicação de aplicação para aplicação, que utiliza protocolos *Web* existentes e baseados em padrões XML abertos [Curbera et al. 2002]. Eles são a maior tecnologia para publicação de serviços na *Web* e têm significativa adoção em áreas como integração de aplicações, computação distribuída de larga escala e cooperação *business-to-business* (B2B) [KOPECKÝ e Simperl 2008]. Os mesmos são instalados na Camada de Aplicação, como mostrado na Figura 2.1, e proveem um conjunto de padrões para acesso a serviços pela *Internet*, sendo uma tecnologia estabelecida no mercado e cada vez mais adotada por empresas [Lausen e Haselwanter 2007].

Web Services permitem o desenvolvimento baseado em componentes, acessíveis por meio da *Internet*. Eles são componentes reutilizáveis, que as aplicações, independentemente da linguagem em que foram implementadas, podem utilizar sem se preocupar em como eles foram desenvolvidos [Vilas et al. 2007]. Desta forma, a construção de aplicações a partir de Web Service segue o processo de desenvolvimento de *software* conhecido como componentização, como apresentado em [Sommerville 2011].

Diferentemente de tecnologias como *Common Object Request Broker Architecture* (CORBA), *Distributed Component Object Model* (DCOM), *Component Object Model Plus* (COM+) e Java *Remote Method Invocation* (RMI), WSs usam protocolos *Web* e formatos de dados ubíquos, como HTTP e XML [Vilas et al. 2007].

O objetivo dos *Web Services* é alcançar a interoperabilidade entre aplicações, usando padrões *Web* (*Web standards*). Eles usam um modelo de integração fracamente acoplado para permitir a integração flexível de sistemas heterogêneos em uma variedade de domínios, incluindo *Business-to-Consumer* (*B2C*), *Business-to-Business* (*B2B*) e *Enterprise Application Integration* (*EAI*) [OASIS 2007].

Pela própria natureza distribuída da *Web*, o uso de WSs vem ao encontro da integração entre aplicações heterogêneas, executando em diferentes plataformas, desenvolvidas por diferentes empresas. Cada vez mais empresas disponibilizam serviços, públicos ou privados, para serem utilizados por terceiros, permitindo a interoperabilidade entre diferentes sistemas. Além disto, pelo fato de WSs serem transportados, principalmente, por protocolo HTTP, não sofrem com problemas de portas bloqueadas em *Firewalls* que outras tecnologias, como as mencionadas anteriormente, encontram. Isto facilita ainda mais a integração entre sistemas hospedados em diferentes redes.

Um *framework* de WSs pode ser dividido em três áreas: protocolos de comunicação, descrição de serviços e descoberta de serviços [Sommerville 2011], conforme apresenta a Figura 2.2 e as sub-seções seguintes.

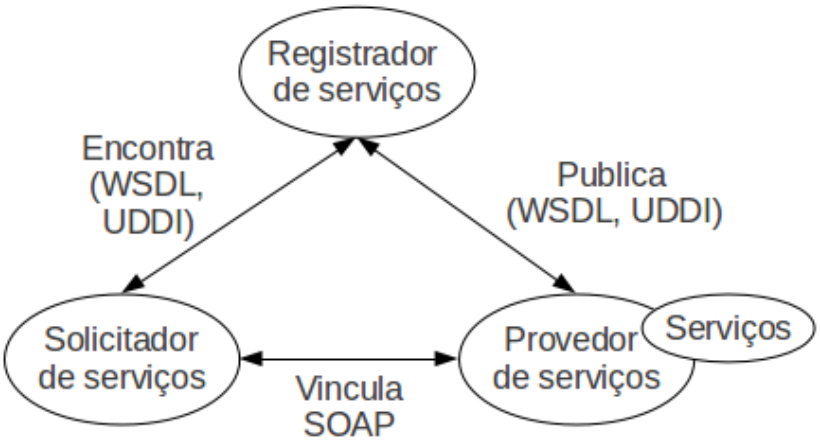


Figura 2.2: Arquitetura conceitual de sistemas orientados a serviços (adaptada de [Sommerville 2011]).

2.3.1 Protocolos de Comunicação

Dada a natureza distribuída e heterogênea da *Web*, mecanismos de comunicação devem ser independentes de plataforma, seguros e leves quanto possível. A linguagem XML é um padrão altamente utilizado para codificação e intercâmbio de dados entre sistemas, sendo totalmente independente de plataforma. Por esse motivo, a mesma é utilizada no protocolo de comunicação usado por WSs.

O *Simple Object Access Protocol* (SOAP), é um protocolo padronizado pelo *World Wide Web Consortium* (W3C), como protocolo de comunicação para WSs. O SOAP é um protocolo, baseado em XML, para a troca de mensagens e chamadas de procedimentos remotos (*Remote Procedure Call*, RPC). No lugar de definir um novo protocolo para empacotar as mensagens, SOAP utiliza protocolos *Web* existentes como HTTP e SMTP.

Ele é um protocolo leve, adequado para comunicação em um ambiente distribuído e descentralizado [Vilas et al. 2007].

Uma mensagem SOAP, também denominada "envelope SOAP", é composta por um arquivo XML, contendo um elemento *header* e um *body*. A Listagem 2.1 mostra a estrutura de um envelope SOAP.

Listagem 2.1: Estrutura de um envelope SOAP [Curbera et al. 2002]

```
1  <SOAP:Envelope
2    xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
3    <SOAP:Header>
4        <!-- content of header goes here -->
5    </SOAP:Header>
6
7    <SOAP:Body>
8        <!-- content of body goes here -->
9    </SOAP:Body>
10 </SOAP:Envelope>
```

2.3.1.1 Troca de Mensagens SOAP

Como exemplo para esta seção, vamos utilizar um sistema *Web* de uma companhia aérea, como apresentado em [Curbera et al. 2002]. O sistema disponibiliza um *Web Service* (WS) para a compra de passagens. Uma empresa que vende pacotes turísticos pode utilizar este WS para integrar o seu sistema com o da companhia aérea, e assim, fazer todo o processo de registro da passagem dentro do seu próprio sistema. Desta forma, o sistema da empresa de turismo deve enviar um envelope SOAP para o WS disponibilizado pela companhia aérea, contendo os dados do cliente e dados da passagem a ser adquirida, como data/hora, número do voo e do assento escolhido pelo cliente. A Listagem 2.2 mostra um exemplo de tal envelope SOAP, empacotado numa mensagem HTTP.

Listagem 2.2: Envelope SOAP transportado via HTTP [Curbera et al. 2002]

```
1  POST /travelservice
2  SOAPAction: "http://www.acme-travel.com/checkin"
3  Content-Type: text/xml; charset="utf-8"
4  Content-Length: nnnn
5
6  <SOAP:Envelope
7      xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
8      <SOAP:Body>
9          <et:eTicket xmlns:et=
10             "http://www.acme-travel.com/eticket/schema">
11             <et:passengerName first="Joe" last="Smith"/>
12             <et:flightInfo airlineName="AA"
13                  flightNumber="1111"
14                  departureDate="2002-01-01"
15                  departureTime="1905"/>
16         </et:eTicket>
17      </SOAP:Body>
18 </SOAP:Envelope>
```

Observe que o envelope SOAP da Listagem 2.2 não possui um *header*. O mesmo é um elemento opcional e utilizado para prover informações específicas da aplicação, como credenciais para acesso ao serviço e informações de cobrança, dentre outras[Point 2010].

2.3.1.2 Chamadas de Procedimento Remoto Usando SOAP

Para que SOAP possa usar RPC para chamar procedimentos remotos, é necessário especificar alguns detalhes para o protocolo RPC, por exemplo:

- como valores de tipos são transportados entre a representação SOAP (XML) e a representação da aplicação, e vice-versa (para indicar, por exemplo, como deve ser feita a conversão de uma classe Java para XML e vice-versa), e

- onde as várias partes do protocolo RPC são transportadas (identificação de objeto, nome da operação e parâmetros de entrada e saída).

A especificação XML *schema* do W3C [11] provê uma linguagem padrão para definir a estrutura do documento e os tipos de dados da estrutura do XML. Isto é, dado um tipo básico como *integer* ou um tipo complexo como uma *struct*, XML *schema* oferece uma forma padrão de escrever dados destes tipos em um documento XML. Para habilitar o transporte de valores tipados, SOAP assume um sistema de tipos baseado em XML *schema* e define sua codificação em XML. Usando este estilo de codificação pode-se produzir uma codificação XML para qualquer tipo de dado estruturado. Parâmetros RPC e retornos são especificados usando esta codificação.

Usando o mesmo WS da companhia aérea, a empresa de turismo deseja obter informações a respeito de um determinado voo. Assim, seu sistema pode enviar um envelope SOAP ao WS da companhia aérea. O envelope da Listagem 2.3 mostra uma chamada RPC, encapsulada dentro de um envelope SOAP. O envelope permite a invocação do método remoto *GetFlightInfo*, que recebe dois parâmetros: uma *string* contendo o nome da companhia aérea, e um inteiro contendo o número do voo. Tal método retorna um valor estruturado (uma *struct*) com dois campos: o número do portão de embarque e a situação do voo.

No envelope SOAP, a chamada para *GetFlightInfo* é um elemento XML com atributos que incluem informações sobre a codificação (note a referência para o XML *schema*). Os elementos filhos são os argumentos do método: *airlineName* e *flightNumber*. Os tipos são definidos no atributo *type*, onde *xsd* refere-se as definições de um XML *schema*. Quando o servidor SOAP recebe o envelope, ele converte os valores *string*, dos atributos *airlineName* e *flightNumber*, para seus respectivos tipos *string* e inteiro, chamando o método *GetFlightInfo* passando estes valores.

Listagem 2.3: Chamada SOAP RPC empacotada em HTTP [Curbera et al. 2002]

```
1  POST /travelservice
2  SOAPAction: "http://www.acme-travel.com/flightinfo"
3  Content-Type: text/xml; charset="utf-8"
4  Content-Length: nnnn
5
6  <SOAP:Envelope
7    xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
8    <SOAP:Body>
9      <m:GetFlightInfo
10       xmlns:m="http://www.acme-travel.com/flightinfo"
11       SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
12       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
13       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
14          <airlineName xsi:type="xsd:string">UL</airlineName>
15          <flightNumber xsi:type="xsd:int">506</flightNumber>
```

[11] http://www.w3.org/XML/Schema

```
16      </m:GetFlightInfo>
17    </SOAP:Body>
18  </SOAP:Envelope>
```

A Listagem 2.4 mostra a resposta à requisição SOAP enviada anteriormente. Neste caso, a resposta contém um valor estruturado, com os campos *gate* (número do portão de embarque) e *status* (situação do voo).

Implementações de SOAP existem para diversas linguagens, como C/C++, Java, Perl e Lua[12], que automaticamente geram e processam mensagens SOAP. Assumindo que as mensagens estão em conformidade com as especificações do protocolo SOAP, elas podem ser trocadas por serviços implementados em diferentes linguagens e plataformas, permitindo uma total interoperabilidade entre sistemas heterogêneos, algo que não é sempre verdade para sistemas que utilizam a arquitetura CORBA, que possui objetivos semelhantes ao SOAP.

Listagem 2.4: Resposta de requisição SOAP empacotada em HTTP [Curbera et al. 2002]

```
1  HTTP/1.1 200 OK
2  Content-Type: text/xml; charset="utf-8"
3  Content-Length: nnnn
4
5  <SOAP:Envelope
6    xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/">
7    <SOAP:Body>
8      <m:GetFlightInfoResponse
9        xmlns:m="http://www.acme-travel.com/flightinfo"
10       SOAP:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
11       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
12       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
13          <flightInfo>
14            <gate xsi:type="xsd:int">10</gate>
15            <status xsi:type="xsd:string">ON TIME</status>
16          </flightInfo>
17      </m:GetFlightInfoResponse>
18    </SOAP:Body>
19  </SOAP:Envelope>
```

2.3.2 Descrição de Serviços

O protocolo SOAP define uma linguagem padrão para uso de WSs, mas ele não informa quais mensagens devem ser trocadas para que tenhamos sucesso no uso de um determinado serviço. Para isto, existe a *Web Service Description Language* (WSDL), uma linguagem também baseada em XML. Um documento WSDL descreve a interface de um WS, permitindo que as aplicações que consumirão o serviço disponibilizado saibam quais informações

[12]Uma implementação do protocolo SOAP foi feita para uso em aplicações de TV digital desenvolvidas em NCL e Lua, e será apresentada no Capítulo 5.

devem incluir dentro de um envelope SOAP, para chamar um determinado procedimento remoto, disponibilizado pelo WS. O WSDL descreve um WS como uma coleção de *end points* que podem trocar certas mensagens [Curbera et al. 2002]. Um *end point* é um elemento no WSDL que define a ligação entre a definição de um determinado serviço e o seu endereço de rede, permitindo o acesso ao mesmo.

As mensagens SOAP anteriormente mostradas não poderiam ter sido construídas se não conhecêssemos o documento WSDL que descreve o serviço que desejamos utilizar. Informações como o nome do método a ser acessado e o nome e tipos dos parâmetros de entrada e saída foram obtidos a partir do WSDL do serviço [Curbera et al. 2002].

Um serviço completo de descrição WSDL provê dois pedaços de informação: uma descrição do serviço em nível de aplicação, ou interface abstrata, e detalhes específicos dependentes do protocolo, que os usuários devem seguir para acessar o serviço em *end points* concretos [Curbera et al. 2002].

2.3.2.1 Descrição de um *Web Service*

Um documento WSDL define uma descrição abstrata de um serviço, em termos de troca de mensagens em uma interação com o mesmo. Existem três principais componentes desta interface abstrata: o vocabulário, a mensagem e a interação. Acordo para uso de um determinado vocabulário é a fundação para qualquer tipo de comunicação. Um WSDL usa sistemas de tipos externos para prover definição de tipos de dados para troca de informações. Embora o WSDL possa suportar qualquer sistema de tipos, a maioria dos serviços usa o XML *Schema Definition* (XSD). Na Listagem 2.5, podemos ver o uso de dois tipos de dados definidos no XSD (*int* e *string*), e dois tipos de dados definidos em um *schema* externo (*FlightInfoType* e *Ticket*). O WSDL pode importar tais definições XSD externas usando um elemento *import*, especificando sua localização [Curbera et al. 2002].

O WSDL define elementos *message* contendo partes, cada qual descrita por tipos XSD ou elementos de um vocabulário pré-definido. Cada *message* provê uma definição de dados tipados e abstratos enviados e recebidos pelos serviços. Uma mensagem pode ser de entrada (*input*), definindo o tipo do(s) parâmetro(s) que uma determinada função deve receber; ou de saída (*output*), indicando o(s) tipo(s) de valor(es) a ser(em) retornado(s) pela função, como pode ser visto na Listagem 2.5 [Curbera et al. 2002].

Os elementos *portType* e *operation* combinam mensagens para definir iterações. Cada *operation* representa um padrão de troca de mensagens suportado pelo WS. Cada *operation* é uma combinação de mensagens rotuladas como *input* (entrada), *output* (saída) ou *fault* (falha), para indicar o papel de cada mensagem em uma iteração [Curbera et al. 2002].

Um *portType* é uma coleção de operações que são coletivamente suportadas por um *end point* (*port*). No exemplo da Listagem 2.5, *AirportServicePortType* descreve duas operações: uma única operação no estilo *request-response*, *GetFlightInfo*, que espera uma mensa-

gem *GetFlightInfoInput* como entrada e retorna uma mensagem *GetFlightInfoOutput* como resposta; e uma operação uni direcional, *CheckIn*, que apenas recebe uma mensagem *CheckInInput* como entrada [Curbera et al. 2002].

Listagem 2.5: Fragmento de uma descrição de WSDL [Curbera et al. 2002]

```
<message name="GetFlightInfoInput">
  <part name="airlineName" type="xsd:string"/>
  <part name="flightNumber" type="xsd:int"/>
</message>

<message name="GetFlightInfoOutput">
  <part name="flightInfo" type="fixsd:FlightInfoType"/>
</message>

<message name="CheckInInput">
  <part name="body" element="eticketxsd:Ticket"/>
</message>

<portType name="AirportServicePortType">
  <operation name="GetFlightInfo">
    <input message="tns:GetFlightInfoInput"/>
    <output message="tns:GetFlightInfoOutput"/>
  </operation>
  <operation name="CheckIn">
    <input message="tns:CheckInInput"/>
</operation>
</portType>
```

2.3.2.2 Informações de *Binding*

A especificação do WSDL apresenta três tipos de informações sobre o serviço[Sommerville 2011]:

- quais operações o serviço suporta (chamado de interface);
- como mapear as interfaces abstratas para um conjunto de protocolos concretos (*binding*), e
- onde está a implementação dos métodos suportados pelo serviço (o endereço de rede).

Por meio do *binding* temos a resposta para o "como", incluindo o protocolo de comunicação e especificação de formato de dados para um completo elemento *portType*. Sucintamente, o elemento *binding* diz como dada interação ocorre sobre o protocolo especificado. A Listagem 2.6 mostra um fragmento para o exemplo citado. O *binding* descreve como usar SOAP para acessar o serviço *travelservice*. Em particular, o documento WSDL mostra que:

- *GetFlightInfo* usará interações em estilo SOAP-RPC, em que todas as trocas de mensagens usam codificação padrão SOAP, e
- *CheckIn* é uma interação de mensagem pura (chamada de orientada a documento em termos de WSDL), em que o corpo do envelope SOAP contém a mensagem codificada, sem nenhuma codificação de tipo adicional; isto é, ele usa XSD para literalmente descrever o XML transmitido.

O restante do documento descreve o "onde", para ligar a interface abstrata, protocolo e detalhes de *marshalling*[13], promovendo a ligação (*binding*) desses elementos para permitir o acesso aos métodos do serviço. Um elemento *port* descreve um único *end point* como uma combinação de um *binding* e um endereço de rede. Consequentemente, um elemento *service* pode agrupar um conjunto de *ports* relacionados.

Listagem 2.6: Informações concretas de ligação em um WSDL [Curbera et al. 2002]

```
<binding name="AirportServiceSoapBinding"
  type="tns:AirportServicePortType">
  <soap:binding
     transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetFlightInfo">
    <soap:operation style="rpc"
          soapAction="http://acme-travel/flightinfo"/>
    <input>
      <soap:body use="encoded"
          namespace="http://acme-travel.com/flightinfo"
          encodingStyle=
           "http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>
    <output>
      <soap:body use="encoded"
          namespace="http://acme-travel.com/flightinfo"
          encodingStyle=
           "http://schemas.xmlsoap.org/soap/encoding/"/>
    </output>
  </operation>

  <operation name="CheckIn">
    <soap:operation style="document"
          soapAction="http://acme-travel.com/checkin"/>
    <input>
      <soap:body use="literal"/>
    </input>
  </operation>
</binding>

```

[13]Processo de organizar dados em computador, de forma padronizada, para permitir que diferentes aplicações possam utilizar tais dados, garantindo a interoperabilidade.

```
31 <service name="travelservice">
32   <port name="travelservicePort"
33       binding="tns:AirportServiceSoapBinding">
34   <soap:address
35       location="http://acmetravel.com/travelservice"/>
36   </port>
37 </service>
```

A Figura 2.3 mostra como a especificação do WSDL é organizada. A seção *Introdução* do WSDL contém a declaração dos *Namespaces* XML contendo as definições de tipos padrões usados pelo *Web Service*. A seção *Interface Abstrata* apresenta os tipos definidos pelo *Web Service*, as declarações de interfaces (os *portTypes*) e as mensagens suportadas (como mensagens com os parâmetros de entrada a serem enviadas a um método no *Web Service* e as mensagens de saída com o retorno do método). A seção *Implementação Concreta* contém as informações de *binding* (como o protocolo a ser utilizado na troca de mensagens) e os *end points* (as interfaces concretas para realização da vinculação da aplicação cliente com o endereço de rede do *Web Service*.

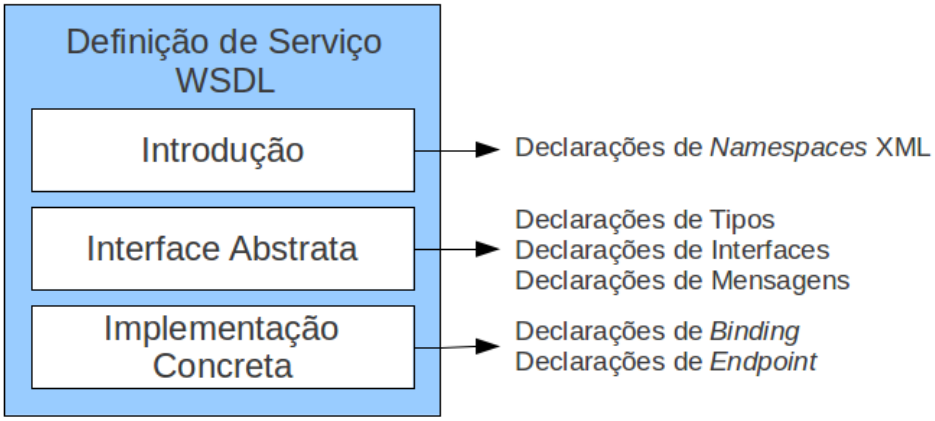


Figura 2.3: Organização da Especificação do WSDL[Sommerville 2011].

2.3.2.3 Usando o WSDL

Para usuários e desenvolvedores, o WSDL provê uma descrição formal das interações cliente/servidor. Desenvolvedores podem usar o documento WSDL como entrada para uma aplicação que gere funções *proxy*[14], utilizadas no cliente, para acesso aos métodos do WS. Tais funções *proxy* também podem ser geradas dinamicamente, em tempo de execução, por rotinas implementadas no cliente que interpretam o WSDL e geram as chamadas SOAP necessárias. Em ambos os casos, as funções *proxy* têm a finalidade de esconder do cliente toda a complexidade envolvida no processo de chamada dos métodos remotos.

[14]Tais funções *proxy* são escritas ou geradas na linguagem utilizada pela aplicação cliente (que consome o *Web Service*) e possuem a mesma assinatura dos métodos remotos, tornando transparente para o cliente a chamada aos métodos no WS, sendo responsáveis por gerar o envelope SOAP para enviar ao servidor onde o WS é hospedado.

2.3.3 Descoberta de Serviços com UDDI

As especificações do *Universal Description, Discovery and Integration* (UDDI) oferecem aos usuários uma forma sistemática e unificada para encontrar provedores de serviços através de um registro centralizado de serviços, que é, grosseiramente, equivalente a uma "lista telefônica"automatizada *online* de WSs [Curbera et al. 2002]. O *UDDI Business Registry* (UBR), acessível por meio de um *browser*, foi disponibilizado pouco tempo após a especificação se tornar pública, no entanto, o mesmo foi descontinuado em 2005[15] [Treiber e Dustdar 2007].

Segundo [Sommerville 2011], o padrão UDDI não foi amplamente adotado. UDDI provê dois tipos básicos de especificação que definem a estrutura e operação do registro de serviços [Curbera et al. 2002]:

- uma definição da informação para prover sobre cada serviço, e como codificá-la; e
- uma API de busca e atualização para o registro, que descreve como esta informação pode ser acessada e atualizada.

O acesso ao registro é realizado usando uma API SOAP para busca e atualização.

2.3.4 *Web Services* REST

Como visto anteriormente, SOAP é o padrão W3C para *Web Services*, amplamente utilizado na integração de aplicações heterogêneas. No entanto, existe uma outra vertente para construção de *Web Services* denominada REST. O REST é o acrônimo para *Representational State Transfer*. Ele é um estilo arquitetural para construção de aplicações distribuídas que também permite a integração de aplicações heterogêneas, mas não é um padrão W3C.

O estilo REST foi apresentado pela primeira vez no trabalho de dissertação de Roy Fielding[Fielding 2000], sendo baseado em padrões *Web* como HTTP e URL. Ele consiste em prover serviços *Web* a partir de um recurso *Web* acessível por meio de uma URL. A interação com tal serviço é feita por meio de uso dos métodos HTTP como *GET*, *POST* e *DELETE* para passagem de parâmetros ao serviço. Tal estilo é bastante simplificado em relação ao SOAP, pois todos os dados a serem passados ao serviço vão diretamente na URL do mesmo, por meio de uma requisição HTTP. Desta forma, o tamanho de uma requisição REST é bastante reduzido em relação ao SOAP.

REST não define um padrão de formato das mensagens de resposta, no entanto, pode-se perfeitamente utilizar XML para isto. Atualmente outros padrões de formato de dados são bastante utilizados como o *JavaScript Object Notation* (JSON). Desta forma, pode-se utilizar qualquer padrão que se desejar, como por exemplo, o formato de dados definido pela

[15] `http://uddi.microsoft.com/about/FAQshutdown.htm`

linguagem Lua, sendo bastante útil em aplicações para o Sistema Brasileiro de TV Digital, desenvolvidas em tal linguagem, por utilizar um formato nativo da mesma.

O estilo REST atualmente se tornou um padrão de fato e é amplamente utilizado para integração de serviços *Web* com aplicações de diferentes tipos, como é o caso dos serviços disponibilizados pelo *microblog Twitter*. Este permite que desenvolvedores construam aplicações nas mais diferentes plataformas (*Web, desktop, mobile*) que se integrem com o serviço de *microblog*.

O REST também segue a arquitetura cliente/servidor usada no protocolo SOAP. Segundo [Welling 2006], o estilo é denominado REST (em português, Transferência de Estado Representacional), pois o cliente obtém uma representação de um recurso através de uma URL, o que faz com que a aplicação cliente entre em um determinado estado. O cliente pode selecionar um link que foi incluído no primeiro recurso para, por exemplo, obter informações detalhadas. Como o link direciona para outro recurso, a aplicação cliente obtém uma nova representação de um recurso e entra em um novo estado.

Como citado, a grande vantagem do REST é sua simplicidade, no entanto, por não haver um padrão de passagem de parâmetros e formato de dados para a respostas das requisições, tal estilo pode não ser adequado para a construção de aplicações seguindo o padrão de arquitetura orientada a serviços, onde uma aplicação pode utilizar serviços de diferentes provedores. Usando REST, cada provedor de serviços pode definir sua forma de passar os parâmetros e o formato de dados da mensagem de retorno, assim a aplicação que integra com todos os provedores precisará implementar diversos formatos de entrada e saída de dados. Com o SOAP isto não ocorre, pois todos os provedores seguem um padrão bem definido e a aplicação cliente implementa apenas tal padrão.

2.3.5 Arquitetura Orientada a Serviços

Segundo [Sommerville 2011], uma arquitetura orientada a serviços (*Service Oriented Architecture* - SOA) é uma forma de desenvolvimento de sistemas distribuídos onde os componentes de tal sistema são serviços *stand-alone* (autônomos), executando em computadores geograficamente distribuídos. Em tal arquitetura, ainda segundo [Sommerville 2011], usar *Web Services* é uma forma de as organizações tornarem suas informações acessíveis. Com isto, é possível haver a integração entre empresas, mesmo que por meio de sistemas heterogênos completamente diferentes.

Capítulo 3

Arquitetura de comércio eletrônico para a TV Digital

Neste capítulo é apresentada uma arquitetura para provimento de comércio eletrônico para o Sistema Brasileiro de TV Digital. A mesma é uma arquitetura distribuída, baseada em componentes reutilizáveis, os *Web Services*, conhecida como Arquitetura Orientada a Serviços.

Desta forma, a arquitetura proposta foi definida, incluindo a implementação de um *framework* de comunicação (baseado nos protocolos HTTP e SOAP) que é apresentado sucintamente neste capítulo, e em mais detalhes no Capítulo 5.

3.1 Requisitos da Arquitetura

A arquitetura deverá atender aos requisitos apresentados na Tabela 3.1 a seguir.

Requisito	Funcional	Não funcional
T-01) Estar preparada para adaptação em qualquer equipamento com qualquer implementação do *middleware* Ginga, seja para dispositivos fixos (como conversores digitais e TV's com conversores integrados), móveis (como *Notebooks*) ou portáteis (como telefones celulares).		x
T-02) Utilizar, preferencialmente, serviços gratuitos da *Web*.		x
T-03) Facilitar o desenvolvimento de aplicações interativas para o SBTVD, inclusive com relação ao tratamento de recursos de interface gráfica.	x	
T-04) Tornar transparente para a aplicação de TVD os provedores de serviços utilizados, permitindo a substituição por outros apenas estendendo-se classes na aplicação cliente e implementando a comunicação SOAP/HTTP com os provedores desejados.	x	
T-05) Facilitar a extensão e a manutenção das aplicações com uso de orientação a objetos e padrões de projetos.	x	
T-06) Fazer interoperação com serviços de forma amigável a *firewalls*.		x
T-07) Integrar diferentes serviços para compor as funcionalidades a serem disponibilizadas aos usuários das aplicações de TVDi.	x	
T-08) Ser compatível com padrões W3C e ABNT (Ginga-NCL)		x

Tabela 3.1: Requisitos da Arquitetura de *T-Commerce*

3.2 Apresentação Geral

A arquitetura proposta é baseada no conceito de *Service Oriented Architecture* (SOA), ou Arquitetura Orientada a Serviços, onde a aplicação é construída de forma distribuída, utilizando diferentes *Web Services*, podendo haver diferentes provedores de serviços, que é o caso da proposta implementada.

Uma arquitetura de comércio eletrônico considera, comumente, o conceito de loja virtual, que precisa integrar diferentes serviços para prover as funcionalidades necessárias aos usuários e ao gerenciamento da mesma. Em uma aplicação de *T-Commerce* não é diferente, o que muda é apenas a plataforma, tendo-se uma interface gráfica de usuário específica para a TV Digital, considerando-se todas as questões de usabilidade como uso de fontes maiores

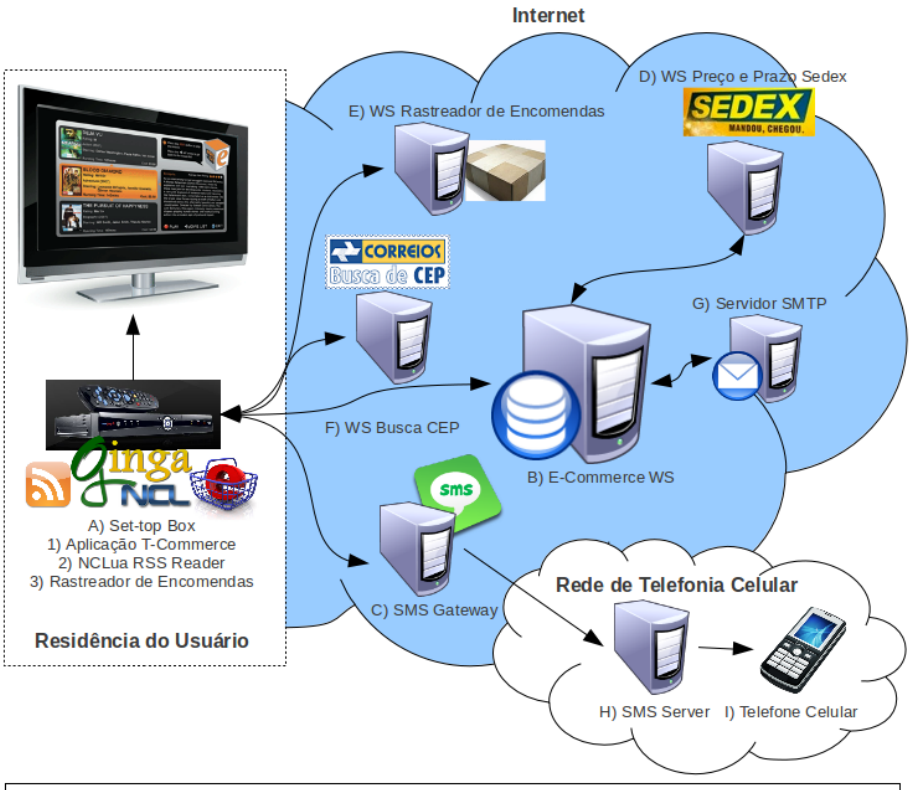
devido ao usuário ficar afastado do aparelho de TV e de integração *Web*-TV referentes a esta plataforma. Todos os serviços que a loja virtual já disponibiliza precisam ser acessados pela interface de TV Digital. Desta forma, a arquitetura orientada a serviços vem ao encontro dessa necessidade, permitindo a integração de uma aplicação cliente em uma plataforma diferente, possuindo uma interface gráfica para TV Digital, com os serviços já implementados pela loja.

SOA permite a integração de sistemas hetererogêneos por utilizar a tecnologia de *Web Services* SOAP que, diferentemente do modelo REST, define um contrato único que todos os provedores e consumidores de serviços devem seguir (o protocolo SOAP), o que facilita a integração de tais sistemas.

Assim, para atender aos requisitos citados, a Figura 3.1 apresenta a arquitetura proposta. Cada item da mesma é apresentado nas subseções a seguir, em que se adotam as seguintes convenções: as letras representam os elementos de *hardware* ou provedores de serviço, e os números representam os elementos de *software* (apenas os *softwares* aplicativos são apresentados em tal figura).

Adicionalmente, embora não constando da Figura 3.1 (para dar uma visão mais simples inicialmente), os seguintes *softwares* são empregados (os quais são apresentados nos capítulos seguintes):

- *Framework* LuaOnTV 2.0;
- e *Framework* de Comunicação de Dados.



T) Arquitetura Proposta

Figura 3.1: Visão Geral da Arquitetura SOA proposta para *T-Commerce*

A) Conversor Digital/*Set-top Box* (STB)

Conversor digital, televisor com conversor integrado contendo o *middleware* Ginga, onde serão executadas as aplicações de TV Digital Interativa. Tais aplicações são conhecidas como *Thin Client* (Clientes Leves/Magros)[Sommerville 2011] por apenas conterem uma camada de apresentação, todas (ou grande parte) das regras de negócio são processadas no lado servidor.

Visando experimentos futuros, considera-se o uso de *notebooks* ou telefones celulares com implementações de Ginga-NCL para teste das aplicações desenvolvidas.

Tais equipamentos serão denominados daqui por diante como receptor/equipamento de TVD.

B) *E-Commerce WS*

Web Services contendo as rotinas utilizadas em uma loja virtual convencional na *Internet*, que serão utilizadas pela aplicação de TV Digital para prover comércio eletrônico pela TV. Tais *Web Services* encapsulam todas as regras de negócio para o gerencimento da loja virtual, como rotinas de cadastro de clientes, produtos, pedidos, etc. A aplicação de TV Digital consumirá tal *Web Service* para prover muitas das funcionalidades disponíveis aos usuários.

O uso de tais *Web Services* é essencial para centralizar as regras de negócio utilizadas tanto pela aplicação de *E-Commerce* (normalmente acessada a partir de computadores ou dispositivos móveis como aparelhos celulares) como pela aplicação de *T-Commerce* (acessada a partir de receptores de TVD).

C) *SMS Gateway*

O *Gateway* SMS é utilizado como intermediário para comunicação via SMS da loja com o cliente. Devido à arquitetura proposta ser baseada em *Web Services*, o uso do *Gateway* facilita tal integração com a rede de telefonia celular.

Um serviço implementado na aplicação de *T-Commerce* que requer o uso de SMS é a recuperação de senha, permitindo que o usuário que esqueceu a senha, receba a mesma via mensagem SMS no telefone celular que estiver cadastrado na loja. Tais serviços não são gratuitos e cobram por cada SMS enviado.

O uso do *Gateway* SMS é essencial por tornar transparente para a aplicação qual a operadora sendo utilizada para enviar as mensagens SMS. Um das formas de envio de SMS é utilizando *softwares* específicos para determinados aparelhos celulares. No entanto, para automatizar tal processo seria necessário fazer a integração com tal software por meio de

alguma API, normalmente específica do *software*. O uso do *Gateway* dispensa tal complexidade.

D) *WS* Preço e Prazo Sedex

Na arquitetura proposta, considerou-se que as entregas da loja sejam feitas pela Empresa Brasileira de Correios e Telégrafos (ECT), denominada daqui em diante simplesmente como Correios. Desta forma, a obtenção de preços, prazos de rastreamento das encomendas utiliza recursos dos Correios. Desta forma, o uso de um *Web Service* dos Correios facilita a obtenção de tais dados a partir de qualquer aplicação.

E) *WS* Rastreador de Encomendas

Após o cliente ter decidido realizar a compra e finalizá-la, apesar de já saber previamente qual a previsão para a entrega do produto, um recurso muito utilizado é o rastreamento *online* da encomenda. Para tal funcionalidade, pelo fato de estarem sendo usados os serviços de logística dos Correios, o rastreador de encomendas para TV Digital realiza o rastreamento *online* de encomendas postadas por tal empresa.

F) *WS* Busca CEP

A busca de endereço a partir de um Código de Endereçamento Postal (CEP) é um requisito fundamental para a aplicação de TVD, devido ao fato de o usuário normalmente possuir apenas o controle remoto para entrada de dados, utilizando o mesmo da mesma forma como entra dados em um telefone celular. Desta forma, a aplicação de *T-Commerce* precisa reduzir ao máximo a quantidade de dados que o usuário deve digitar, para agilizar tal processo tedioso de entrada de dados. Na arquitetura proposta foi utilizado um *Web Service* para prover tal funcionalidade.

G) Servidor SMTP

Como a aplicação de *T-Commerce* possui recurso de recuperação de senha também por *e-mail*, a arquitetura proposta inclui um servidor de *Simple Mail Transfer Protocol* (SMTP). Tal servidor é também conhecido como *Mail Transfer Agent* (MTA), que é responsável por transferir mensagens de correio eletrônico entre um computador e outro. Por meio de tal servidor, a aplicação de TVD pode, por exemplo enviar a senha do usuário para seu *e-mail* ou confirmar a realização de uma compra.

H) SMS *Server*

Os *Gateways* SMS fazem a integração com o SMS *Server* de alguma(s) operadora(s) para permitir o envio das mensagens SMS. Logo, o SMS *Server* é o responsável direto pelo envio das mensagens SMS pela rede de telefonia celular. Desta forma, o *Gateway* apenas se encarrega de fazer a integração com o(s) servidor(es) de SMS, tornando tranparente para a aplicação esta comunicação com o terminal celular do usuário.

3.2.1 Casos de Uso das Funcionalidades da Arquitetura

Para dar uma visão geral das funcionalidades providas pela arquitetura, tanto para os desenvolvedores de aplicações de TV Digital Interativa (TVDi), quanto para os usuários finais, são apresentados os diagramas de casos de uso a seguir.

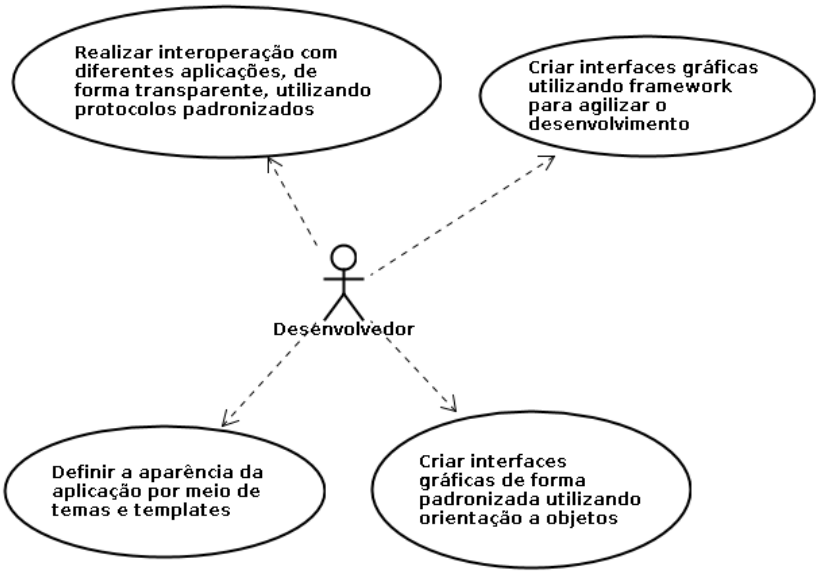


Figura 3.2: Diagrama de Casos de Uso das Funcionalidades providas a desenvolvedores de aplicações TVDi

A Figura 3.2 apresenta as funcionalidades que a arquitetura provê aos desenvolvedores de aplicações de TVDi, por meio dos *frameworks* utilizados. Tais funcionalidades permitem um grande aumento de produtividade no desenvolvimento de tais aplicações, reduzindo a quantidade de código que os desenvolvedores precisam escrever e, consequentemente, o total de *bugs*[1].

[1]Erro no funcionamento de um *software*

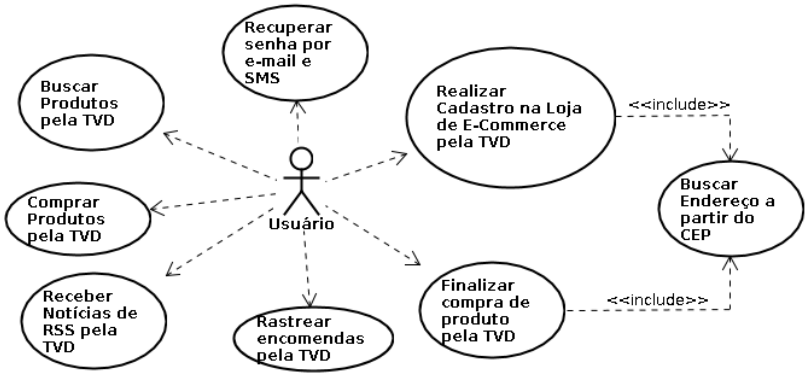


Figura 3.3: Diagrama de Casos de Uso das Funcionalidades providas aos usuários das aplicações de TVDi desenvolvidas

A Figura 3.3 apresenta as funcionalidades que a arquitetura provê aos usuários de aplicações de TVDi. As funcionalidades implementadas são as consideradas mais comuns em sistemas de *E-Commerce* tradicionais.

3.2.2 Tecnologias *Core* da Arquitetura Proposta

Em [Chu et al. 2007] são tratadas as tecnologias *core* para o provimento de comércio eletrônico, divididas em quatro camadas: comunicação, apresentação e representação de informações, linguagens, e armazenamento e recuperação de dados.

A camada de comunicação conta com as tecnologias necessárias para estabelecer a comunicação entre as partes envolvidas nos sistemas de comércio eletrônico, assim como os protocolos utilizados para garantir a interoperabilidade entre as aplicações que compõem tais sistemas. Utilizam-se, na arquitetura aqui proposta, as tecnologias *Hyper Text Transfer Protocol* (HTTP), *Simple Object Access Protocol* (SOAP), *Simple Mail Transfer Protocol* (SMTP) e *Short Message Service* (SMS).

A camada de apresentação e representação de informações contém as tecnologias que definem o formato das informações, utilizado tanto para apresentação como para garantir a troca de informações entre sistemas heterogênos. Utilizam-se, na arquitetura aqui proposta, as tecnologias *eXtensible Markup Language* (XML), *Really Simple Syndication* (RSS), Lua, e imagens *Joint Photographic Experts Group* (JPEG) e *Portable Network Graphics* (PNG).

As linguagens de programação são utilizadas para a construção das regras de negócio das aplicações, definindo toda a lógica das rotinas a serem executadas por elas. São utilizadas as linguagens Java, *Nested Context Language* (NCL) e Lua.

A camada de armazenamento e recuperação de dados é responsável pelo armazenamento local ou remoto dos dados das aplicações, como Sistemas Gerenciadores de Bancos de Dados (SGBD's). São utilizadas as tecnologias MySQL *DataBase Management System*, *Java Database Connectivity* (JDBC) e *Structured Query Language* (SQL).

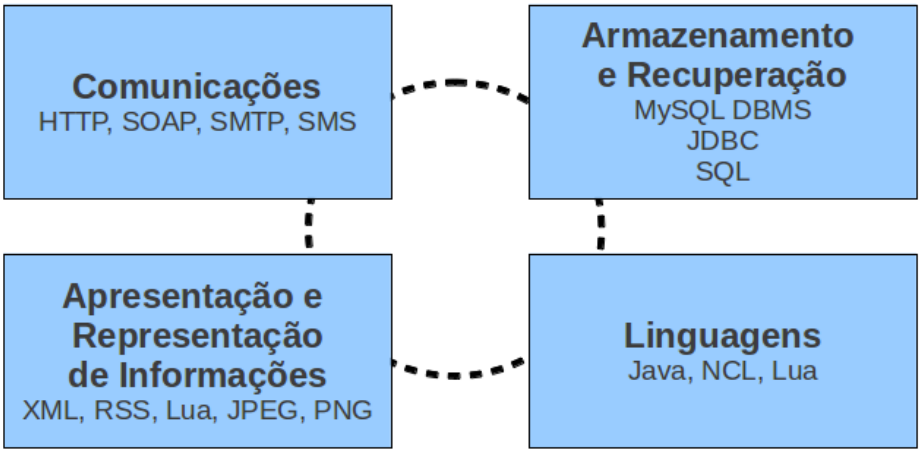


Figura 3.4: Tecnologias *Core* da arquitetura de *T-Commerce* (adaptada de [Chu et al. 2007]).

A Figura 3.4 apresenta as tecnologias empregadas na arquitetura proposta, em cada uma das camadas supracitadas.

3.3 Descrição dos Componentes de *Software* da Arquitetura

Conforme a Figura 3.1 exibida anteriormente, nesta seção são apresentados os *softwares* aplicativos que compõem a arquitetura.

A) *Set-top Box*

A.1) Aplicação de *T-Commerce*

A aplicação é a interface gráfica pela qual o usuário pode realizar compras por meio da TV Digital.

Os requisitos funcionais da aplicação foram baseados em avaliação empírica das funcionalidades existentes na maioria dos sistemas de *E-Commerce* disponíveis no Brasil e amplamente conhecidos e utilizados. A Tabela 3.2 apresenta os requisitos funcionais e não funcionais atendidas pela aplicação.

Requisito	Funcional	Não Funcional
A.1-01) Permitir a exibição de produtos em destaque ao iniciar a aplicação	X	
A.1-02) Realizar a busca de produtos pelo título parcial	X	
A.1-03) Adicionar produtos ao carrinho de compras	X	
A.1-04) Remover produtos do carrinho de compras	X	
A.1-05) Cadastrar vários endereços, permitindo ao usuário escolher um endereço de entrega a partir da lista de endereços cadastrados	X	
A.1-06) Permitir múltiplas formas de pagamento, possibilitando ao usuário selecionar uma das formas suportadas	X	
A.1-07) Cadastrar usuário	X	
A.1-08) Realizar login de usuário utilizando *e-mail* ou CPF	X	
A.1-09) Buscar endereço a partir do CEP	X	
A.1-10) Recuperar senha por *e-mail* e SMS	X	
A.1-11) Rastrear automatizadamente encomendas postadas pelos Correios, exibindo ao usuário informações sempre que a situação da encomenda mudar	X	
A.1-12) Exibir preço e prazo de entrega, baseado em *Web Services* dos Correios	X	
A.1-13) Realizar processo de compra em etapas, permitindo que o usuário volte a uma etapa anterior, antes de finalizar a compra, para alterar algum dado (como mudar a forma de pagamento)	X	
A.1-14) Utilizar botões coloridos do controle remoto para o acionamento da maioria das funções da aplicação	X	
A.1-15) Processar grande parte das regras de negócio nos servidores *Web* que integram a arquitetura, desonerando o conversor digital da maior parte da carga de processamento, considerando que o mesmo é um dispositivo de recursos restritos		X
A.1-16) Armazenar dos dados do carrinho de compras em memória RAM até a finalização da compra, quando estes dados são enviados ao *Web Service* para registro da mesma		X
A.1-17) Carregar dinâmicamente a lista de produtos a partir do *Web Service*		X
A.1-18) Permitir a exibição de comunicados, ofertas de produtos e promoções	X	

Tabela 3.2: Requisitos funcionais e não funcionais da aplicação de *T-Commerce*

A aplicação é apenas um cliente que consome os *Web Services* utilizados na arquitetura. Assim, a maior parte da carga de processamento e regras de negócio é feita nos servidores *Web* que hospedam os serviços. Quase todas as telas da aplicação (que podemos chamar também de formulários ou páginas) fazem acesso a algum *Web Service* para obtenção ou registro de dados. Desta forma, a aplicação é totalmente dependente de um canal de interatividade (também conhecido como canal de retorno) para conexão à *Internet* e acesso aos servidores *Web* que hospedam os serviços.

Como o telespectador pode visualizar e comprar qualquer produto existente na loja, e não somente aqueles que estejam em destaque, para a exibição dos produtos sempre é feita uma consulta ao *Web Service* de *E-Commerce* da loja. Em outros modelos de aplicação, como o apresentado em [Síntese 2010], que exibem apenas os produtos em destaque, o telespectador não precisa ter a TV conectada à *Internet*, pois ele só visualiza os produtos enviados pela emissora, e não pode finalizar o processo de compra pela TV. Desta forma, não há necessidade de conexão com a *Internet*. A arquitetura proposta vai além deste modelo, permitindo que o usuário possa comprar qualquer produto e finalizar a compra direto da TV, o que exige a existência de um canal de interatividade.

A interface gráfica da aplicação foi desenvolvida utilizando-se uma nova versão do *framework* LuaOnTV[Júnior e Gondim 2009], cujas melhorias foram desenvolvidas no presente trabalho de dissertação e são apresentadas em detalhes no Capítulo 4. O LuaOnTV é a primeira e, até o momento em que esta dissertação foi desenvolvida, a única biblioteca de

componentes NCLua para criação de interfaces gráficas em aplicações de TV Digital para o Ginga-NCL que se tem conhecimento.

Um *screenshot*[2] da aplicação é apresentado na Figura 3.5. Tal figura mostra a tela inicial da aplicação, com os produtos em destaque na loja virtual (definidos no banco de dados da loja). O usuário pode localizar um produto a partir de uma palavra contida no título, inserindo tal valor por meio do controle remoto. O apêndice no final da dissertação mostra *screenshots* de todas as telas da aplicação.

Figura 3.5: Aplicação de *T-Commerce*: Tela inicial mostrando os produtos em destaque

Toda a comunicação com os serviços que compõem a arquitetura é feita por meio do protocolo SOAP, utilizando-se a implementação de SOAP para o Ginga-NCL denominada NCLua SOAP, construída neste trabalho de dissertação e apresentada em detalhes no Capítulo 5.

O processo de compra na aplicação de *T-Commerce* desenvolvida é bastante simples e direto. O diagrama de sequência apresentado na Figura 3.6 mostra como tal processo ocorre.

[2]Imagem capturada de uma tela de determinada aplicação em execução

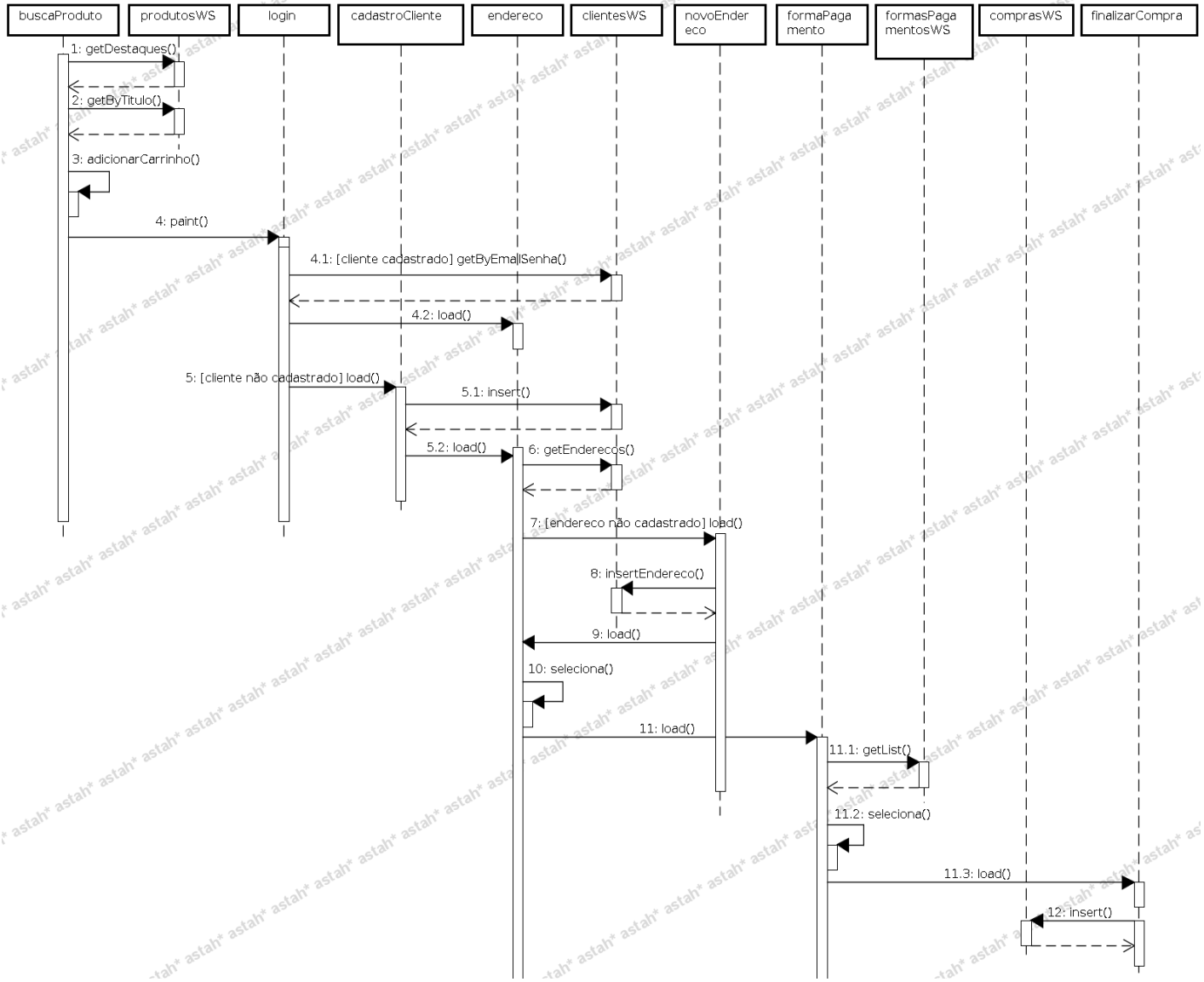


Figura 3.6: Aplicação de *T-Commerce*: Diagrama de sequência do processo de compra

A.2) NCLua RSS *Reader*

Como pode ser visto na Figura 3.1, o *Set-top Box* (TV, notebook ou celular com o *middleware* Ginga) armazena a aplicação de *T-Commerce* além de um leitor de RSS desenvolvidos com as linguagens NCL e Lua.

Um arquivo RSS possui um formato XML e é utilizado para divulgação de notícias na *Internet*, permitindo aos usuários assinarem os chamados *feeds* RSS para obterem as notícias de um determinado provedor desejado. Os *feeds* são os arquivos RSS que contêm as notícias em formato XML. Com o RSS, a notícia vai até o usuário, no lugar de ele ir até a notícia. Assim, tendo um leitor de RSS, o usuário pode adicionar vários *feeds* e receber várias notícias de diferentes provedores.

O leitor de RSS implementado para a TV Digital nesta dissertação é denominado *NCLua*

RSS Reader[3]. Ele implementa o padrão RSS 2.0[Board 2009] e compõe a arquitetura de *T-Commerce* para permitir que, quando o usuário não estiver acessando a aplicação da loja virtual pela TV, ele possa ficar atualizado com as promoções que a loja esteja divulgando.

A aplicação utiliza o módulo LuaXML[4] (que foi estendido para funcionar com Lua 5). Este é um parser XML escrito completamente em Lua, que permite obter as notícias do *feed* RSS e exibir em um equipamento de TVD. A aplicação utiliza o módulo NCLua HTTP para fazer o *download* do arquivo RSS diretamente do site do provedor de conteúdo, neste caso o servidor *Web* da loja virtual. O NCLua HTTP será apresentado no Capítulo 5. O *NCLua RSS Reader* ainda utiliza o módulo *canvas* de NCLua para desenho da interface gráfica, além do módulo *event* para tratamento de eventos.

A Figura 3.7 apresenta um *screenshot* da aplicação de leitura de RSS em execução, mostrando a mesma sendo exibida sobre o vídeo principal da emissora (em um ambiente simulado, o *Ginga Virtual Set-top Box*).

Figura 3.7: *NCLua RSS Reader* - Leitor de Notícias para TV Digital

[3] `http://ncluarss.manoelcampos.com` e `http://labtvdi.unb.br`
[4] `http://lua-users.org/wiki/LuaXml`

A.3) Rastreador de Encomendas para TVD

O rastreador de encomendas para a TV Digital faz acesso a este *Web Service* desenvolvido para o Ginga-NCL, utilizando as linguagens NCL e Lua e o módulo NCLua SOAP, este a ser apresentado no Capítulo 5.

As Figuras 3.8 e 3.9 apresentam *screenshots* da aplicação em execução. Tal aplicação, juntamente com o módulo NCLua SOAP, foi ganhadora do Concurso Latino-Americano de Conteúdo para TV Digital Interativa [5], na categoria *Widgets*, realizado pela PUC-Rio em 2010.

Figura 3.8: Rastreamento de encomendas pela TVD: Inserção do código de rastreamento

[5] http://clube.ncl.org.br/node/82

Figura 3.9: Rastreamento de encomendas pela TVD: Situação da entrega da encomenda (atualizada automaticamente a cada 5 minutos)

B) *E-Commerce WS*

O *E-Commerce WS* contém um conjunto de *Web Services* desenvolvidos utilizando-se o IDE NetBeans 6.7.1 e a *Java API for XML Web Services* (JAX-WS) com o servidor de aplicações *Glass Fish* 2.1 para execução dos *Web Services*. Tais ferramentas foram escolhidas por serem livres e multiplataforma, amplamente utilizadas no mercado para o desenvolvimento de aplicações, e pela familiaridade com as mesmas. O servidor de aplicações pode ser qualquer outro (como *Tomcat, Jetty ou JBoss*) que implemente a especificação de *Servlet* 2.5. Um *Servlet* é uma classe Java *server side*, ou seja, que executa do lado do servidor *Web*, sendo capaz de responder a requisições HTTP com resposta em formato (X)HTML, XML e outros. A escolha pelo *Glass Fish* veio devido ao fato de ele ser a atual implementação de referência para a plataforma *Java Enterprise Edition* (Java EE), que provê as tecnologias para desenvolvimento de aplicações *Web* em Java.

Os métodos desenvolvidos em cada *Web Service* foram implementados, baseado em um levantamento de requisitos do que deve ter uma loja virtual, como já apresentado na Tabela 3.2. Utilizou-se a linguagem UML para modelar as classes e métodos que comporiam os serviços.

A Figura 3.10 apresenta um diagrama de classes com os *Web Services* implementados e seus respectivos métodos.

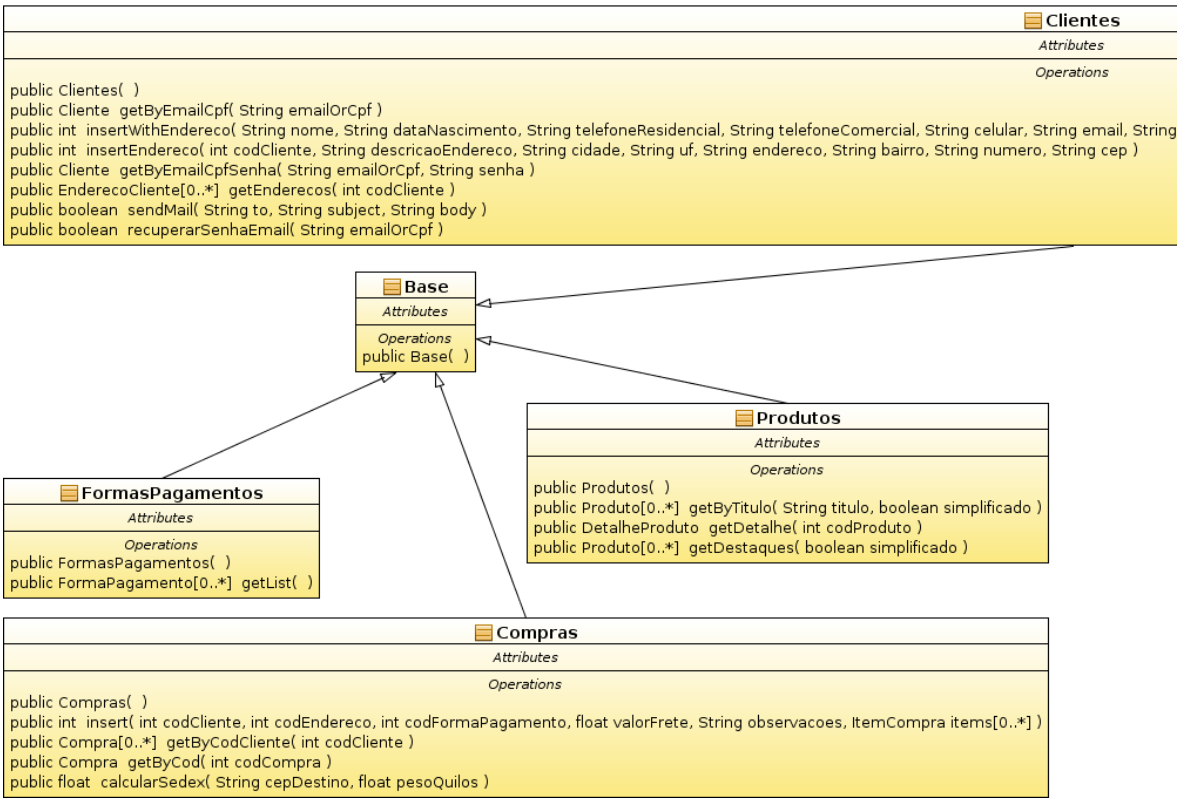


Figura 3.10: Diagrama de Classes dos *Web Services* implementados

Cada *Web Service* agrupa funcionalidades comuns. O "Clientes", por exemplo, provê todas as funcionalidades para gerenciamento do cadastro do cliente, o "Compras", todo o processo de compra, o "Produtos", todo o cadastro de produtos.

Estes *Web Services* publicam os métodos a serem acessados pela aplicação de *T-Commerce*. Os mesmos são responsáveis pelo gerenciamento de todos os dados armazenados pela loja (como produtos, clientes e pedidos de compra). Eles utilizam as classes Java apresentadas na Figura 3.11 como base para seus métodos.

Tais classes são responsáveis pelo gerenciamento do cadastro do cliente, cadastro de produtos, formas de pagamento, compras e tipo de frete. Para o armazenamento destes dados foi utilizado o Sistema Gerenciador de Banco de Dados (SGBD) *MySQL 5.1* por ser um sistema multiplataforma, livre, bastante leve, que não requer muitos recursos de hardware e amplamente utilizado no mercado, além da familiaridade com o mesmo. No entanto, a arquitetura pode utilizar qualquer outro banco de dados que for desejado, pois o acesso aos dados é feito por meio da API *Java Database Connectivity* (JDBC) que é compatível com diversos SGBD's existentes no mercado. Além disto, foi utilizado o padrão de projeto *Data Access Objects* (DAO) que permite fazer uma separação total entre as classes de negócio e as instruções SQL para acesso ao banco de dados.

Figura 3.11: Diagrama das classes utilizadas internamente pelos WS's implementados

Os *Web Services* não utilizam nenhum *framework* de persistência por falta de familiaridade com tais tecnologias e restrições de tempo. No entanto, a utilização de algum *framework* como *Hibernate* ou *Java Persistent API* (JPA) não altera em nada a comunicação entre a aplicação de TVD e os *Web Services*. Tal mudança arquitetural pode ser feita para

aumentar o nível de abstração do acesso ao banco de dados e automatizar a geração das instruções SQL necessárias para isto, sob pena de aumentar o *overhead* de acesso aos dados.

C) *SMS Gateway*

Para a arquitetura proposta, foi utilizado o *Gateway* MyTalk[6], que permite o acesso via *Web Services* SOAP. Contudo, tal componente pode ser substituído na arquitetura por qualquer outro *Gateway* SMS. Todos os *gateways* encontrados e testados, disponibilizam acesso via SOAP ou REST. A escolha de tal serviço foi feita por ter sido o primeiro encontrado, mas reitera-se que qualquer outro pode ser utilizado, levando-se em consideração custo, resiliência, QoS (como tempo de resposta), etc.

Caso o usuário esqueça sua senha, ele pode informar seu *e-mail* ou CPF na aplicação de TVD e recuperá-la via *e-mail* ou SMS. Como ele normalmente estará na frente da TV, sair deste local para ir até um computador e entrar no seu *e-mail* para pegar uma nova senha é bastante incômodo. Como o celular é algo pessoal e que geralmente as pessoas o mantêm por perto o tempo todo, o usuário pode receber a nova senha sem precisar sair da frente da TV, facilitando o processo de compra.

Neste caso, optou-se por a aplicação de TV enviar a requisição diretamente ao *Gateway* SMS, no lugar de requisitar ao *Web Service* da loja (*E-Commerce* WS) para depois este despachar a requisição ao *Gateway* SMS, no intuito de reduzir o tempo de resposta para a aplicação de *T-Commerce*. No entanto, esta abordagem tem a desvantagem de vincular a aplicação a um determinado *Gateway* SMS. Caso escolha-se utilizar um *Gateway* diferente, a aplicação precisará ser atualizada. Centralizando tal regra de negócio no *Web Service* da loja, não haverá tal problema, mas aumentará o tempo de resposta da requisição que precisará ser encaminhada ao *Gateway*.

D) *WS* Preço e Prazo Sedex

Um dos recursos básicos que toda loja virtual deve prover a seus usuários é informar o preço e prazo para entrega do produto. Tais informações podem ser decisivas para a concretização ou não da compra.

Os Correios são responsáveis por grande parte das entregas de produtos e correspondências em todo o país. Desta forma, a arquitetura de *T-Commerce* proposta faz integração com o *Web Service* dos Correios[7] que permite calcular o preço e prazo de entrega de uma correspondência/produto de/para qualquer lugar do país. Logo, a aplicação de *T-Commerce* consegue exibir o preço e prazo para entrega do(s) produto(s) que o usuário deseja comprar, acessando tal *Web Service*.

[6] `http://www.mytalk.com.br`
[7] `http://www.correios.com.br/webservices/`

A atual versão da proposta permite apenas calcular preço e prazo de entrega de encomendas utilizando o serviço de entregas rápidas dos Correios, denominado Sedex, mas a obtenção de informações de entrega para outros serviços dos Correios pode ser feita facilmente a partir da implementação atual.

A Figura 3.12 apresenta quais informações podem ser obtidas do *Web Service* dos Correios.

Figura 3.12: *Web Service* dos Correios: Calculando preço e prazo de entrega de encomenda (fonte: `http://www.correios.com.br/webservices/`).

O acesso ao serviço é feito a partir do *E-Commerce WS* apresentado na Seção 3.2. Assim, a aplicação de *T-Commerce* em NCLua envia uma requisição ao *Web Service* "Compras"que possui o método "calcularSedex"que retorna as informações sobre preço e prazo de entrega, como apresentado no diagrama de classes da Figura 3.10.

E) *WS* Rastreador de Encomendas

Os Correios possuem um serviço na *Internet* que permite ao usuário saber a situação da entrega de uma determinada encomenda, a partir do código de rastreamento da mesma.

No entanto, o usuário, para ficar atualizado da situação, precisa periodicamente acessar tal página, gastando tempo que ele poderia utilizar em outras atividades mais importantes.

Assim, na arquitetura proposta, decidiu-se implementar um serviço automatizado para rastreamento de encomendas, onde o usuário possa acessar o serviço e deixar a tela do mesmo aberta, sem precisar interagir com ela ou ficar voltando lá para verificar se a situação da encomenda mudou. Uma vez tendo aberta a tela do serviço, o usuário será automaticamente notificado por meio de uma mensagem sonora, que a situação da encomenda mudou.

O serviço está disponível via *Web*[8] e foi também implementado em uma aplicação de TV Digital para o Ginga-NCL.

Como os Correios não disponibilizam tal serviço de rastreamento por meio de um *Web Service*, para facilitar a integração do serviço em diferentes arquiteturas, foi desenvolvido um *Web Service* para o rastreador. Este é responsável por fazer o *parse* do código HTML da página do serviço dos Correios e obter as informações sobre o rastreamento da encomenda. Tais informações são extraídas e retornadas pelo *Web Service* de forma estruturada, permitindo a exibição delas facilmente em qualquer aplicação em diferentes plataformas. Um exemplo é a aplicação iComenda[9], desenvolvida por Maurício Júnior[10], para os dispositivos móveis iPhone, iPod e iPad, a partir do *Web Service* disponibilizado.

F) *WS* Busca CEP

Infelizmente os Correios não disponibilizam um *Web Service* para a consulta de endereços a partir de um número de CEP. Só há disponível uma página *Web* para usuários finais[11] que retorna os dados do endereço como uma imagem, para impedir (ou pelo menos dificultar bastante) que aplicações consigam acessar a página e extrair tais dados. Tal atitude provavelmente é devida ao fato de os Correios terem um *software* comercial, denominado Guia Postal Brasileiro Eletrônico[12], contendo um banco de dados de CEP's, que é vendido na sua loja virtual e nas agências.

No entanto, existem alguns *Web Services* de terceiros que permitem obter o endereço a partir de um CEP. Qualquer um deste serviços pode ser utilizado na arquitetura implementada para permitir a obtenção de endereços.

Para a implementação realizada, foi escolhido o *Web Service* disponível em `http://www.maniezo.com.br/webservice/soap-server.php`, por ter sido o primeiro a ser encontrado. Tal serviço requer a realização de um cadastro prévio para poder acessá-lo, e com os poucos testes realizados, retornou os endereços corretos e atualizados.

A aplicação de *T-Commerce* faz acesso direto a tal serviço para obter um endereço, utilizando-se o módulo NCLua SOAP.

[8] `http://rastreador.manoelcampos.com` e `http://labtvdi.unb.br`
[9] `http://itunes.apple.com/br/app/icomenda/id412358490?mt=8`
[10] `http://mauriciojunior.org`
[11] `http://www.buscacep.correios.com.br`
[12] `http://www.correios.com.br/servicos/cep/gpbe.cfm`

G) Servidor SMTP

Para envio de *e-mail*, a aplicação de *T-Commerce* acessa o método *sendMail* do *Web Service* "Clientes"do *E-Commerce WS*, apresentado na Seção 3.2 e na Figura 3.10, para enviar *e-mail* ao cliente. Desta forma, a aplicação cliente não acessa diretamente o servidor de *e-mail*. O *E-Commerce WS* é que faz esta integração, despachando a requisição para o servidor SMTP.

O *E-Commerce WS* utiliza a API Java *Mail*, funcionando como um cliente SMTP para despachar a mensagem de *e-mail*. Ele torna transparente para a aplicação, o servidor de *e-mail* utilizado e as configurações para acesso ao mesmo.

Na implementação feita, foi utilizado um servidor de *e-mail* disponibilizado por um plano de hospedagem profissional, logo é um serviço com custo mensal. No entanto, pode-se utilizar qualquer servidor de *e-mail* que desejar, como por exemplo um que seja disponibilizado por serviços de *WebMail* gratuitos, amplamente difundidos na *Internet* e usados no cotidiano.

3.4 Diagrama de Distribuição/Implantação

A Figura 3.13 apresenta um diagrama de distribuição/implantação que mostra como os componentes da arquitetura, apresentados na Seção 3.2 são distribuídos em diversos *hardwares*, mostrando a arquitetura distribuída montada e a relação de dependência entre cada componente. Tal diagrama apresenta todos os componentes de *hardware* e *software* da arquitetura proposta, incluindo os *softwares* aplicativos e os *frameworks* implementados.

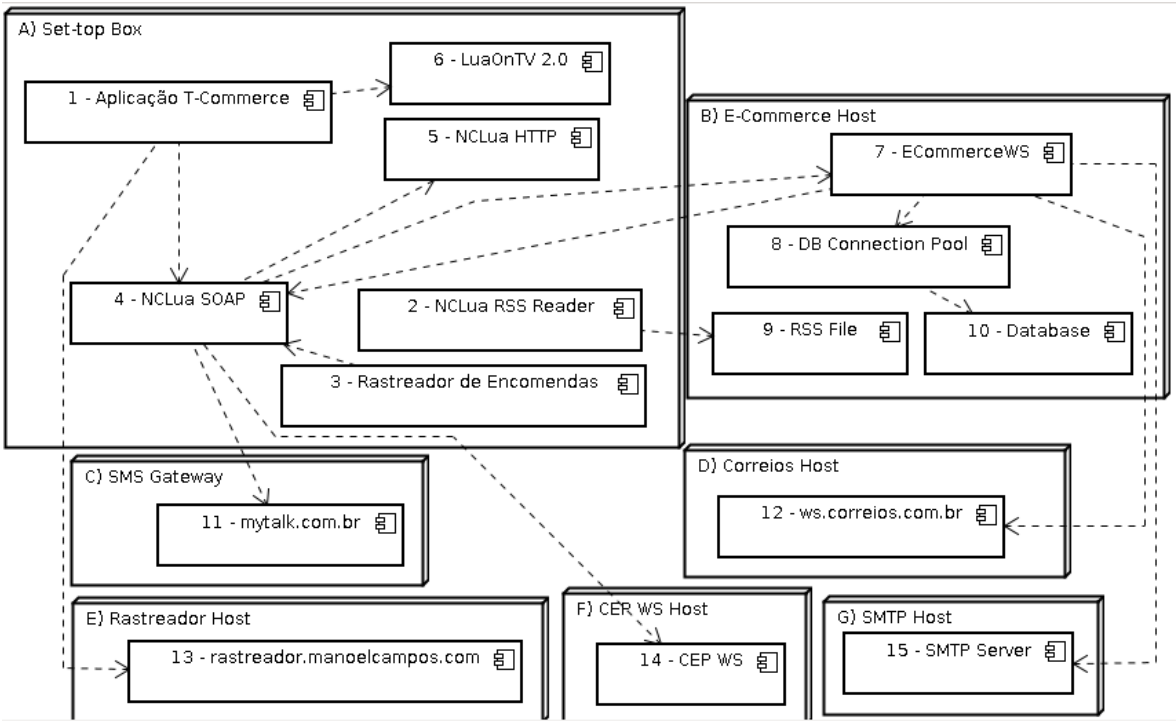


Figura 3.13: Arquitetura de *T-Commerce*: Diagrama de Distribuição/Implantação

As funcionalidades de cada componente são resumidas a seguir:

A) *Set-top Box* - Conversor de TV Digital, TV com conversor integrado, notebook ou celular que executa as aplicações de TVD desenvolvidas:

- 1 - Aplicação *T-Commerce*: aplicação de comércio eletrônico para TVD (aplicação cliente);
- 2 - *NCLua RSS Reader*: leitor de notícias para TVD, utilizado para ver notícias e ofertas de produtos; da loja virtual;
- 3 - Rastreador de Encomendas para TVD: permite ao usuário rastrear a entrega do(s) produto(s) comprado(s) por meio da TV Digital, sendo notificado sempre que a situação da encomenda mudar;
- 4 - NCLua SOAP: módulo para acesso a *Web Services* SOAP em aplicações NCLua para TVD;
- 5 - NCLua HTTP: módulo que implementa o protocolo HTTP, utilizado pelo NCLua SOAP e por aplicações como o *NCLua RSS Reader*. O módulo é apresentado no Capítulo 5;
- 6 - LuaOnTV 2.0: *framework* para construção da interface gráfica da aplicação de *T-Commerce*.

B) *E-Commerce Host* - Servidor *Web* que hospeda os *Web Services* a serem utilizados pela aplicação de *T-Commerce*:

- 7 - *ECommerceWS*: *Web Services* da loja virtual, implementados em Java com a API JAX-WS;
- 8 - *DB Connection Pool*: *pool* de conexões ao banco de dados, implementado utilizando a API JDBC, que permite que conexões ao banco de dados sejam compartilhadas entre diferentes requisição ao *Web Service*, possibilitando um aumento de performance no acesso aos dados;
- 9 - *RSS File*: arquivo RSS contendo as notícias e ofertas de produtos a serem divulgados pela loja virtual;
- 10 - *Database*: banco de dados *MySQL Server* 5.1 que armazena todos os dados da loja virtual, como cadastro de clientes, produtos e pedidos de compra.

C) SMS *Gateway* - *Gateway* para envio de mensagens SMS aos clientes:

- 11 - mytalk.com.br: *Host* que hospeda o serviço *MyTalk* SMS, um *Gateway* para envio de mensagens SMS.

D) Correios *Host* - *Host* dos correios que hospeda *Web Services* como os para consulta de preço e prazo de entrega de encomendas:

- 12 - ws.correios.com.br: *Web Services* dos Correios para consulta de preço e prazo de entrega de encomendas.

E) Rastreador *Host* - *Host* que armazena o serviço de rastreamento de encomendas postadas pelos Correios:

- 13 - rastreador.manoelcampos.com: *Web Service* de rastreamento de encomendas.

F) CEP WS *Host* - *Host* que hospeda um dos serviços de busca de endereço a partir do CEP:

- 14 - CEP WS: *Web Service* maniezo.com.br que fornece o endereço a partir de um CEP

G) SMTP *Host* - *Host* que hospeda servidor de *e-mail*:

- 15 - SMTP *Server*: Servidor de *e-mail* para envio de mensagens eletrônicas aos clientes da loja virtual.

H) SMS *Server* - *Host* de uma operadora de telefonia celular, responsável pela comunicação via SMS com o terminal celular do usuário.

3.5 Ambiente de desenvolvimento

Para o desenvolvimento do projeto foram utilizados:

- sistema operacional GNU/Linux Ubuntu 10.10 como sistema *desktop* para realização das tarefas de desenvolvimento;
- *Astah Community* 6.1 (antigo *Jude*) para modelagem UML;
- *soapUI* 3.5.1 para testes de consumo de *Web Services* SOAP;
- IDE Eclipse 3.6 para desenvolvimento de aplicação Ginga-NCL:
  - *plugin* NCLEclipse 1.5.1;
  - *plugin* LuaEclipse 1.3.1;
- ferramenta LuaDoc[13] para geração de documentação;
- implementação de referência do Ginga-NCL (Ginga Virtual Set-top Box 0.11.2);
- interpretador Lua para execução do *scripts* fora do Ginga *Virtual Set-top Box*;
- *XVidCap* 1.1.7 para captura de *screencasts*[14] da aplicação em execução;

[13] `http://luadoc.luaforge.net`
[14] Vídeo mostrando a execução de determinada aplicação

- IDE *NetBeans* 6.7.1 para modelagem UML e desenvolvimento dos *Web Services* da loja virtual, utilizando API JAX-WS; servidor de aplicações *Glass Fish* 2.1 para execução dos *Web Services* da loja virtual [15];
- *Java Development Kit* (JDK) 1.6.0.24, o kit de desenvolvimento Java, contendo API's e ferramentas para desenvolvimento de aplicações;
- *Java Runtime Environment* (JRE) 1.6.0.24, o ambiente de execução Java para execução das aplicações Java como os *Web Services* criados, os IDE's e outras ferramentas desenvolvidas em Java como o *soapUI*;
- *MySQL Server* 5.1 como Sistema Gerenciador de Banco de Dados (SGBD) para os *Web Services* da loja virtual;
- *PhpMyAdmin* para administração do banco de dados;
- PHP 5.3 e Apache 2 para execução do *PhpMyAdmin*;
- *VirtualBox* 4.0 e *RemasterSys* para criação de distribuição Linux contendo todo o ambiente de desenvolvimento, com todas as ferramentas apresentadas acima.

3.6 Distribuição GNU/Linux para desenvolvimento, execução e teste de aplicações de TV Digital

Com base no trabalho apresentado em [Monteiro et al. 2010], foi gerada uma distribuição GNU/Linux contendo todo o ambiente de desenvolvimento necessário para a construção das aplicações apresentadas ao longo deste trabalho. O ambiente contém todas as ferramentas apresentadas em 3.5. Tal distribuição GNU/Linux foi elaborada com o intuito de facilitar a montagem do ambiente de desenvolvimento necessário para a construção de aplicações NCL/Lua para a TV Digital.

A comunidade Ginga no Portal do Software Público disponibiliza uma implementação de referência do sub-sistema Ginga-NCL em forma de uma máquina virtual já com tal implementação compilada e instalada. Tal máquina virtual facilita bastante a montagem do ambiente de desenvolvimento, uma vez que o processo de compilação de tal implementação é bastante extenso, dependendo de diversos softwares e bibliotecas, não sendo um processo trivial de ser executado, principalmente para usuários menos experientes com distribuições GNU/Linux e ferramentas de compilação de linha de comando. No entanto, o uso de uma máquina virtual adiciona um *overhead* de tempo no processo de teste das aplicações, uma vez que os arquivos das mesmas precisam ser enviados via SSH para a máquina virtual, apesar de tal processo ser automatizado com o *plugin* NCL Eclipse.

[15] `http://java.net/projects/openesb`

Desta forma, a instalação da implementação de referência do Ginga-NCL diretamente no sistema operacional utilizado pelo desenvolvedor para as suas tarefas rotineiras (como envio de *e-mail's*, elaboração de documentos em *suites* de escritórios, etc) e de desenvolvimento de sistemas agiliza bastante o processo de execução e teste das aplicações interativas, pois, como o Ginga é instalado localmente na máquina real, não há processo de transferência de arquivos para poder executar as aplicações. Com isto, a execução das aplicações é praticamente instantânea, além de obter-se melhor desempenho executando o Ginga nativamente no sistema operacional da máquina real, uma vez que o uso de uma máquina virtual obviamente requer o consumo de mais memória RAM e processador que executar o Ginga nativamente em uma máquina real.

A distribuição desenvolvida foi baseada na versão 10.10 do Ubuntu e permite o uso do ambiente de desenvolvimento sem a necessidade de instalação do mesmo na máquina do desenvolvedor, podendo ser dado *boot* na máquina por meio de um CD contendo tal distribuição, conhecido como *Live CD*. Ela pode ser instalada em uma máquina real, onde o usuário poderá utilizar tal distribuição como seu sistema operacional *Desktop* para a realização de suas tarefas rotineiras e de desenvolvimento e ainda pode ser instalada em uma máquina virtual, já com o ambiente gráfico e todas as ferramentas de desenvolvimento necessárias.

A versão da implementação de referência do Ginga-NCL embarcada na distribuição é a última (até a data de entrega de tal dissertação), a 0.11.2 revisão 23, disponível na Comunidade Ginga do Portal do Software Público[16].

O processo de criação da distribuição GNU/Linux consistiu basicamente em:

- instalar distribuição Ubuntu 10.10 em uma máquina virtual utilizando a ferramenta Virtual Box 4.0;
- baixar e compilar a implementação de referência do Ginga-NCL em tal máquina virtual, juntamente com todas suas dependêncais;
- instalar ferramentas de desenvolvimento e suas dependências (como JDK e JRE);
- configurar ferramentas de desenvolvimento para permitir a execução local de aplicações NCL/Lua;
- gerar uma nova distribuição a partir do ambiente criado, já com todas as ferramentas instaladas, utilizando o *software RemasterSys*[17].

Tal distribuição pode ser obtida pelo *site* do Laboratório de TV Digital Interativa da Universidade de Brasília[18].

[16] `http://svn.softwarepublico.gov.br/trac/ginga/wiki/Building_Wiki_GingaNCL`

[17] `http://remastersys.sourceforge.net`

[18] `http://labtvdi.unb.br`

Capítulo 4

Framework LuaOnTV 2.0

Atendendo ao requisito "T3) Facilitar o desenvolvimento de aplicações interativas"apresentado na Seção 3.1, foi utilizado o *framework* LuaOnTV, como ponto de partida, no qual foram realizadas melhorias a serem apresentadas neste capítulo.

O LuaOnTV[Júnior e Gondim 2009] é um *framework* para facilitar o desenvolvimento de aplicações procedurais para o Sistema Brasileiro de TV Digital, utilizando a linguagem Lua, por meio de *scripts* NCLua (*scripts* Lua embutidos em documentos NCL). O mesmo foi resultado de trabalho de projeto de pesquisa desenvolvido por equipe do Laboratório de TV Digital Interativa da Universidade de Brasília, sob orientação do professor Paulo Roberto de Lira Gondim. Ele foi um projeto pioneiro para o SBTVD, desenvolvido inteiramente em linguagem Lua, sob o paradigma de programação orientada a objetos, que disponibiliza um conjunto de classes e componentes visuais e não visuais para o desenvolvimento de aplicações interativas para TVD.

Os componentes do LuaOnTV utilizam os módulos *canvas* e *event* existentes em NCLua. Tais módulos estendem as funcionalidades da linguagem Lua para o contexto de TV Digital. Desta forma, o *framework* está totalmente em conformidade com as normas do Ginga-NCL. O *framework* tem como principal objetivo facilitar a construção de interfaces gráficas para aplicações interativas, disponibilizando componentes para entrada e saída de dados, além de automatizar o processo de controle de foco.

Os componentes visuais foram desenvolvidos com base nos componentes existentes em diferentes linguagens de programação e ambientes integrados de desenvolvimento (IDE's) como *NetBeans*, Eclipse, Adobe *Dreamweaver*, *Delphi*, *Visual Studio* e outros; além de ferramentas específicas para TV Digital como *Jame Author* e *Cardinal Studio*.

[Júnior e Gondim 2009] compararam uma aplicação desenvolvida utilizando a NCL como linguagem principal e outra utilizando o LuaOnTV, que tem a linguagem Lua como principal, e constataram que:

> "uma aplicação com códigos NCL e imagens que simulam os componentes gráficos do LuaOnTV chegou a quase 1,5MB. A mesma aplicação desenvolvida

com LuaOnTV chegou a pouco mais de 60 KB."

Eles também comentam que o *framework* tem como um dos requisitos não funcionais a diminuição do tamanho do código das aplicações interativas, considerando as capacidades restritas de memória e armazenamento dos conversores de TV Digital. Tal requisito foi alcançado devido à utilização do paradigma de orientação a objetos, que permite uma alta reutilização de código.

4.1 Delimitação do Problema

Devido à linguagem NCL ser uma linguagem apenas declarativa, funcionando como uma linguagem de cola para diferentes tipos de mídias (como GIF, JPEG, MPEG, PNG, XHTML, e outros, não restringindo nem prescrevendo nenhum tipo de mídia)[ABNT 2008], ela é uma linguagem de maior nível de abstração, não decompondo o resultado em uma implementação algorítmica[Barbosa e Soares 2008]. No entanto, tal linguagem não permite realização de tarefas específicas como operações aritméticas, utilização direta de protocolos de comunicação como TCP, HTTP e SOAP, manipulação de arquivos, muito menos a definição de procedimentos (rotinas/funções). Desta forma, a realização de tais tarefas só é possível por meio de uma linguagem procedural como a linguagem Lua.

No caso de entrada de dados, apesar do sub-sistema Ginga-NCL do *middleware* Ginga especificar um recurso de teclado virtual a ser utilizado por aplicações NCL[ABNT 2008], a implementação de referência do mesmo[PUC-Rio 2010] ainda não inclui tal recurso. Além disto, a manipulação de dados (armazenamento e obtenção) em documentos NCL não é tão trivial como a inclusão de mídias como imagens e vídeos.

Em NCL é possível até mesmo definir o controle de foco entre campos (por meio dos atributos *moveLeft, moveRight, moveUp, moveDown*, da *tag descriptor*[ABNT 2008]) permitindo que o usuário navegue de um campo a outro utilizando as setas do controle remoto. No entanto, a definição da navegação entre os campos é feita de forma completamente manual, e a inclusão ou remoção de um novo campo no meio dos campos existentes requer a redefinição da ordem de navegação entre os mesmos, o que pode ser um trabalho cansativo e suscetível a erros.

Os módulos de NCLua, utilizados pelo LuaOnTV, disponibilizam apenas um conjunto de funções básicas específicas. Existem 5 módulos, como apresentados a seguir[ABNT 2008]:

- ***canvas***: oferece uma API para desenhar primitivas gráficas e manipular imagens;
- ***event***: permite que aplicações NCLua comuniquem-se com o *middleware* através de eventos (eventos NCL e de teclas);
- ***settings***: exporta uma tabela com variáveis definidas pelo autor do documento NCL e variáveis de ambiente reservadas em um nó *"application/x-ginga-settings"*;

- ***persistent***: exporta uma tabela com variáveis persistentes, que estão disponíveis para manipulação apenas por objetos procedurais.

A utilização direta das funções destes módulos requer um conhecimento mais profundo dos mesmos, como temos provado durante a pesquisa e estudos de tais módulos, para o desenvolvimento de aplicações. As funcionalidades de captura de teclas (providas pelo módulo *event*) para entrada de dados alfabéticos e numéricos, da mesma forma como em um teclado de celular (devido o controle remoto só possuir teclas numéricas) não é trivial. O controle de foco a partir da captura do pressionamento das teclas direcionais do controle também é complicado e requer a inclusão de muito código. Tais dificuldades têm se provado verdade pelos diversos relatos de usuários, como no Fórum da Comunidade Ginga no Portal do *Software* Público[PUC-Rio 2010], além de outros grupos de discussão acompanhados.

Com todas as dificuldades apresentadas, o LuaOnTV se mostrou ser uma solução ideal para a redução da quantidade de código para a criação de aplicações procedurais com interface gráfica para a TVD, permitindo encapsular toda a complexidade envolvida na utilização direta dos módulos de NCLua para chegar ao mesmo fim.

4.2 LuaOnTV 2.0: Nova versão implementada

Da mesma forma que todo projeto de *software*, foi implementada e liberada uma primeira versão do LuaOnTV. Como extensão do trabalho de mestrado onde foi originado o mesmo, verificou-se que eram necessárias algumas melhorias e correções de alguns bugs encontrados. Nesta seção será apresentada a versão 2.0 do LuaOnTV, que traz uma série de importantes novos recursos.

A seguir são listados os requisitos funcionais e não funcionais, inicialmente identificados e implementados.

Requisito	**Funcional**	**Não Funcional**
A.6-01) Correção de bugs e melhoria de desempenho na entrada de dados e desenho da interface de usuário		x
A.6-02) Redução da quantidade de código para incluir e configurar um componente na tela		x
A.6-03) Criação de exemplos com os novos recursos implementados		x
A.6-04) Adaptação de interface para dispositivos portáteis	x	
A.6-05) Adaptação de interface para diferentes definições de tela	x	
A.6-06) Temas para os componentes gráficos	x	
A.6-07) Posicionamento e dimensões de componentes e fontes em percentual	x	
A.6-08) Centralização da interface na tela do dispositivo em casos onde a definição da tela for maior do que a da aplicação	x	

Tabela 4.1: Requisitos funcionais e não funcionais implementados na nova versão do LuaOnTV

4.2.1 Melhoria de Desempenho

Um dos principais problemas do LuaOnTV em sua versão 1.0 estava relacionado ao desempenho. Durante a utilização do mesmo, foi detectada uma lentidão na resposta aos eventos disparados pelo usuário (como a mudança de foco e entrada de caracteres pelo controle remoto). Foi realizada uma investigação, estudando-se a arquitetura do *framework*, onde observou-se que a causa de tal lentidão era devida ao redesenho de todos os componentes na tela, a cada botão que o usuário pressionava no controle remoto, ou a cada mudança de foco ocorrida. O módulo *canvas* de NCLua, utilizado para desenhar os campos e imagens na tela, permite trabalhar com camadas, no entanto, as camadas não são totalmente independentes. Assim, quando uma camada é composta sobre outra, esta passa a ser parte da segunda camada, não permitindo mais a separação entre elas, o que avalia-se que foi o principal motivo para os autores do framework realizarem o total redesenho da tela a cada evento ocorrido, causando a lentidão bastante perceptível.

O problema acima descrito foi corrigido: assim, a tela só é totalmente desenhada no momento em que a aplicação é inicializada. A cada evento ocorrido, somente a parte afetada da tela é redesenhada. Por exemplo, quando um componente não está em foco, sua borda padrão é de cor preta. Quando ele recebe foco, a borda é alterada para vermelha. Desta forma, neste evento, implementou-se a alteração apenas na borda dos componentes anterior e atual (que perdeu o foco e que recebeu o foco, respectivamente). Da mesma forma no evento de pressionamento de teclas, somente o componente atual tem sua interface redesenhada para refletir as alterações realizadas pelo usuário por meio do controle remoto.

4.2.2 Temas

A criação de interfaces gráficas com o LuaOnTV não permitia a definição da aparência dos componentes de forma centralizada, a não ser alterando o código nas classes do *framework*. Caso o desenvolvedor desejasse definir características visuais diferentes das especificadas pelo *framework*, uma alternativa seria definir em cada componente inserido, as características desejadas (como estilo e tamanho de fonte, cores, dimensões e posicionamento). Tal abordagem torna bastante trabalhoso o processo de mudar a aparência atual, precisando-se definir tais características para todos os objetos instanciados.

Tendo em vista tal deficiência, foi implementado no *framework* um recurso de temas. Este recurso é bastante conhecido em aplicações de diferentes plataformas, como é o caso do recurso denominado *look-and-feel* da biblioteca de componentes *Swing* da plataforma Java[Fowler 2000]. A *Swing* é uma biblioteca bastante conhecida pelos desenvolvedores Java. Sua versão atual é bastante madura e estável e seu modelo de componentes é fortemente definido em uma arquitetura orientada a objetos, seguindos padrões de projetos consolidados pela academica e pela indústria de *software*. O recurso de *look-and-feel* possibilita ao *Swing* ser executado em diferentes plataformas de *hardware* e sistemas operacionais, adaptando a

interface dos componentes visuais de acordo com os temas suportados pela plataforma.

O recurso de temas do LuaOnTV seguiu esse exemplo de larga utilização e amplo sucesso do *Swing*. Desta forma, foi implementada uma classe *Theme* que é a classe base para implementação de qualquer tema. A partir desta classe, foram criadas as classes *MasterTheme* e *DefaultTheme*. A *MasterTheme* define as características padrões para todos os temas. A *DefaultTheme* implementa o tema padrão a ser utilizado pelas aplicações caso nenhum seja definido pelo programador.

A classe *ThemeManager* é responsável por carregar um determinado tema para os componentes da aplicação.

A classe do tema (como *DefaultTheme*) contém as características que serão herdadas por todos os componentes utilizando o tema. Para cada componente do LuaOnTV pode-se definir uma classe de tema associada a um tema específico. Tal classe define as características específicas de um componente, e pode sobrescrever as características para as propriedades comuns definidas na classe principal do tema. O desenvolvedor pode ainda sobrescrever as características do tema atual, definindo novos valores para as propriedades de um componente no momento de instanciá-lo ou setando propriedades específicas posteriormente, por meio dos métodos *setters*[1] do componente.

O conjunto de classes de temas permitem que sejam criados temas para diferentes definições de tela (como *Low Definition, Standard Definition, High Definition* e *Full High Definition*), permitindo a adaptação automática da interface da aplicação, de acordo com a resolução da tela do dispositivo onde a mesma vai executar, seja uma TV CRT conectada a um *Set-top Box*, uma TV LCD/Plasma/LED HD ou Full HD, um notebook ou até mesmo um celular com o *middleware* Ginga.

Considerando que apenas o sub-sistema Ginga-NCL é obrigatório para dispositivos portáteis com TV Digital Interativa (como celulares), e que o Ginga-NCL e Ginga-J devem estar presentes em todas as implementações de Ginga para receptores fixos e móveis, o desenvolvimento de aplicações com garantia de execução em todos os tipos de dispositivos com Ginga só é possível se desenvolvidas para o Ginga-NCL. Desta forma, o desenvolvimento de aplicações em Ginga-J não garante que poderão ser executadas também em dispositivos portáteis.

Assim, para que o desenvolvedor não tenha que implementar duas aplicações (uma em Ginga-J para receptores fixos e móveis e outra em Ginga-NCL para receptores portáteis), é preciso implementar a aplicação em Ginga-NCL. No entanto, mesmo em Ginga-NCL, utilizando as bibliotecas de NCLua citadas anteriormente, pode haver um grande trabalho para adaptar a interface da aplicação para diferentes dispositivos e definições de tela. O recurso de temas implementado no LuaOnTV permite que todos esses detalhes sejam abstraídos pela criação de temas diferentes para dispositivos e definições de telas diferentes.

[1] Métodos responsáveis por alterar o conteúdo de um determinado atributo de uma classe

Um dos recursos implementados, utilizados pelos temas, que permitem essa independência de definição de tela foi o de informar posições, dimensões e tamanho de fonte de componentes por meio de valores percentuais. O módulo *canvas* de NCLua só permite o uso de valores absolutos em suas funções. Tais funcionalidades foram estendidas pelo LuaOnTV, tendo por base os recursos das Folhas de Estilo em Cascata (*Cascade Style Sheets*, CSS)[W3C 2009], amplamente utilizadas em páginas HTML.

Tal recurso permite a adaptação automática da fonte e das dimensões do componente, de acordo com o tema definido, adicionando recursos de acessibilidade nas aplicações interativas, pois o tamanho da fonte e dos componentes pode ser alterado dinamicamente. Desta forma, o desenvolvedor pode incluir os conhecidos botões "A+"e "A-"na aplicação, para dinamicamente aumentar ou diminuir o tamanho da fonte da aplicação.

Alguns outros recursos implementados no LuaOnTV foram:

- posicionamento automático de campos na tela: caso o desenvolvedor não informe posições (*left* e *top*) para um componente, ele será automaticamente posicionado na tela, com base nas coordenadas do último componente inserido, permitindo um ajuste automático da interface em telas de definições diferentes;
- centralização de *layout* automático na tela de diferentes dispositivos: permite centralizar a interface da aplicação na tela do dispositivo, em casos de a aplicação estar executando em telas de grandes definições, o que pode fazer com que sobre espaço nos quatro lados da tela;
- simplificação do conjunto de classes e componentes: criação de um único componente para permitir a entrada de texto, de números e de senhas. Propriedades foram adicionadas para definir o comportamento de entrada de dados no componente;
- simplificação do modelo de construção de aplicações: foi reduzida a quantidade de código necessária para incluir um componente na tela, implementando, por exemplo, o controle automático de foco. Assim, a cada campo incluído, ele automaticamente recebe uma numeração que define sua ordem na tela. Desta forma, o *framework* conhece automaticamente a ordem dos componentes.

A Figura 4.1 apresenta o novo diagrama de classes dos componentes do LuaOnTV.

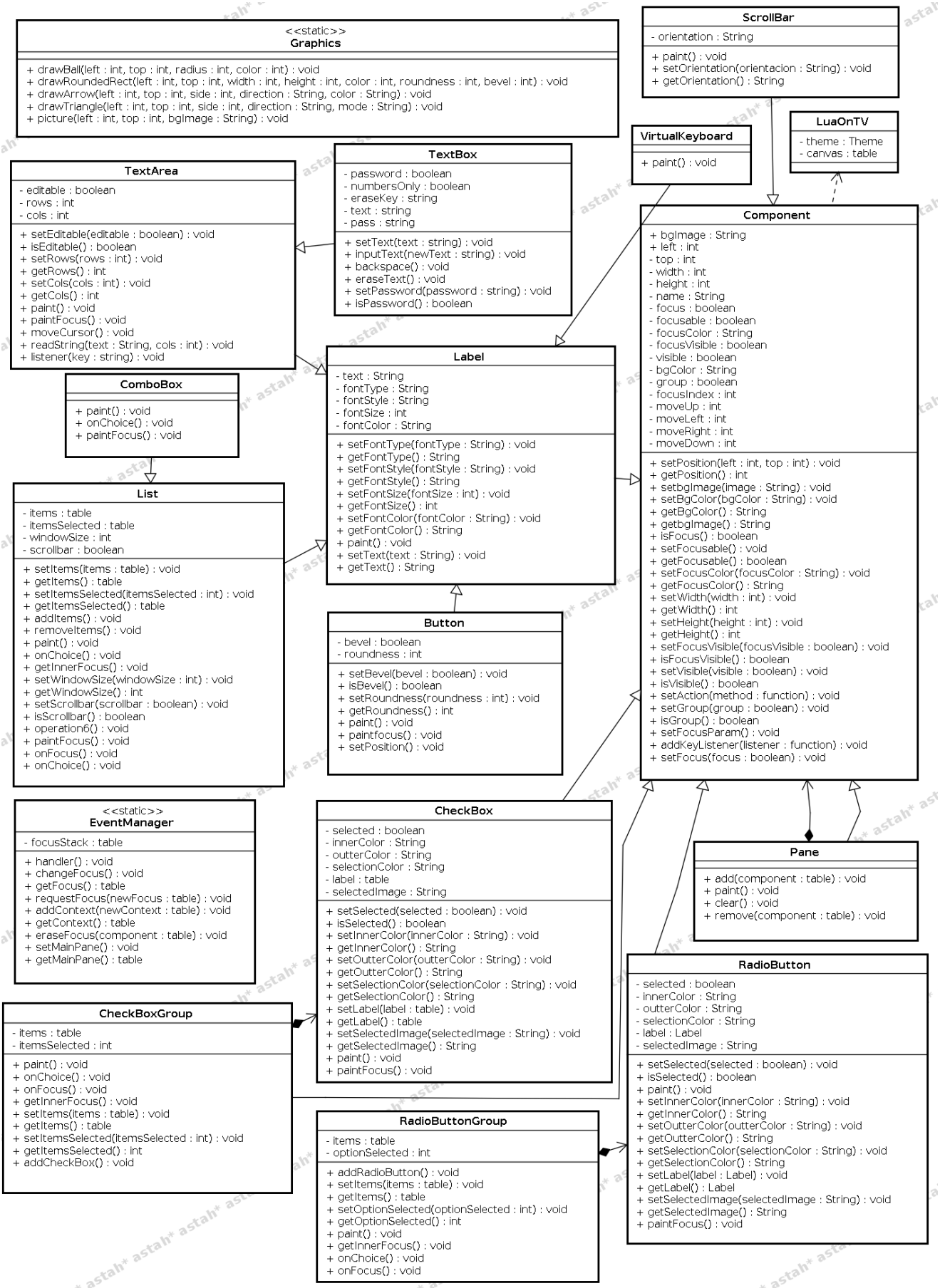


Figura 4.1: Novo diagrama de classes do LuaOnTV

Neste diagrama, a classe *EventManager* tem um papel fundamental no *framework*, pois ela é responsável pelo tratamento dos eventos ocorridos na aplicação, principalmente os eventos de pressionamento de teclas, controlando a entrada de dados na tela e a navegação entre os campos por meio das teclas direcionais do controle remoto. Desta forma, a Figura 4.2 apresenta um gráfico de máquinas estados desta classe.

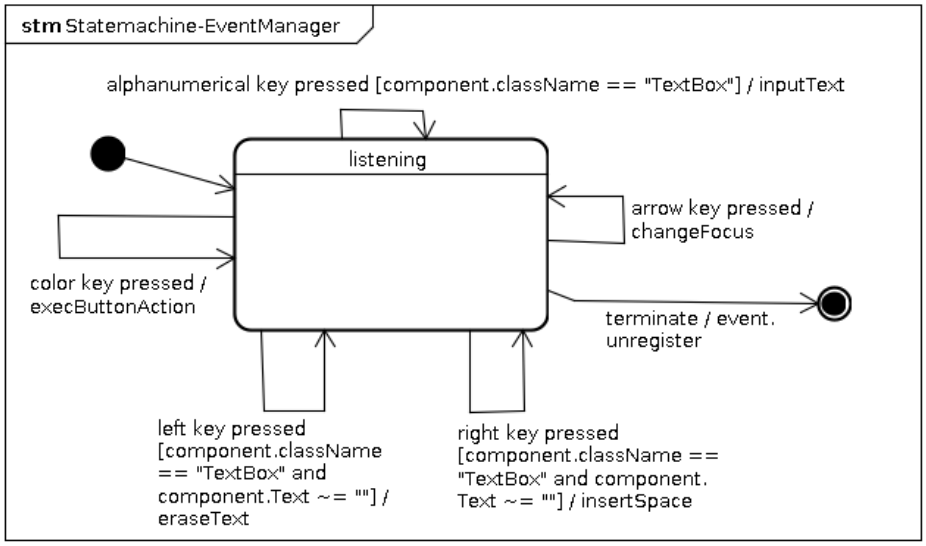


Figura 4.2: Gráfico de Máquinas de Estados da classe *EventManager*

Um dos principais novos recursos implementados no LuaOnTV foi o suporte a temas. A Figura 4.3 apresenta as classes relacionadas a tal recurso.

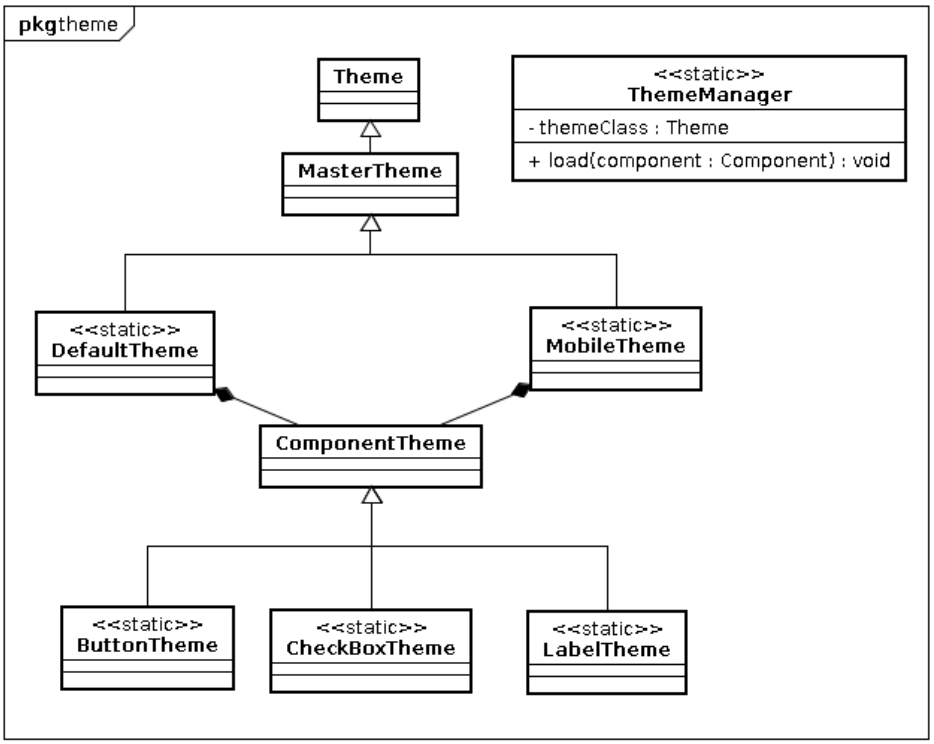


Figura 4.3: Classes relacionadas ao novo recurso de temas do LuaOnTV

Como mencionado anteriormente, o LuaOnTV utiliza os módulos *canvas* e *event* do Ginga-NCL. Assim, a Figura 4.4 apresenta um diagrama de componentes, mostrando como os elementos do LuaOnTV e do Ginga-NCL se relacionam. O LuaOnTV possui dois pacotes principais: *Components*, contendo as classes que implementam os componentes visuais e não visuais; e *Themes*, contendo as classes que implementam os temas. O LuaOnTV cria uma camada de abstração para os módulos *canvas* e *event* do Ginga-NCL, provendo funcionalidades de mais alto nível.

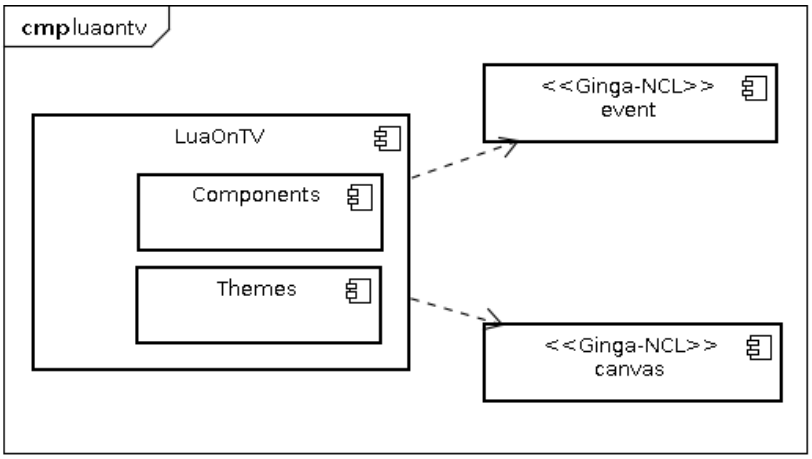


Figura 4.4: Diagrama de Componentes do LuaOnTV

Capítulo 5

Framework de comunicação de dados

A arquitetura apresentada no Capítulo 3 utiliza protocolos de comunicação HTTP e SOAP. Tais protocolos foram implementados em um *framework* de comunicação a ser apresentado neste capítulo.

5.1 Integração entre *Web* e TV

Um dos grandes atrativos da TV Digital (TVD) é com certeza a interatividade. Existem diversas categorias de aplicações interativas como jogos, informações (notícias, horóscopo, previsão do tempo, etc), educação (*T-Learning*), governo eletrônico (*T-Government*), comércio eletrônico (*T-Commerce*), saúde (*T-Health*), bancárias (*T-Banking*) e outras, como apresentado em [Fernández e Goldenberg 2008], [Teixeira 2006] e [Barbosa, Kutiishi e Lima 2010].

O nível de interatividade das aplicações vai desde a chamada interatividade local (onde os usuários/telespectadores podem acessar apenas os dados enviados pela emissora, por radiodifusão) até a interatividade plena (onde os usuários/telespectadores dispõem de um canal de retorno, permitindo enviar e receber dados em uma rede como a *Internet*) [Soares e Barbosa 2009].

Tais aplicações que permitem interatividade plena precisam utilizar protocolos de comunicação padrões e consolidados na *Internet* (como TCP, HTTP e SOAP) para garantir a interoperabilidade com outros sistemas.

Utilizando os protocolos citados, é possível garantir a convergência entre *Web* e TV. Com isto, as aplicações de TVDi podem ser enriquecidas com conteúdo proveniente da *Internet*, como é o caso da aplicação que busca conteúdo na Wikipédia, baseada nas *tags* do programa em exibição, obtidos a partir do Guia Eletrônico de Programação (cujas informações são enviadas pela emissora), como apresentado em [Ghisi, Lopes e Siqueira 2010].

A norma do sub-sistema Ginga-NCL do *middleware* Ginga define apenas a obrigatorie-

dade do protocolo TCP. Quaisquer protocolos acima da camada de transporte precisam ser implementados pelo desenvolvedor de aplicações.

Tendo em vista o cenário supracitado, são apresentados neste trabalho os projetos NCLua HTTP e NCLua SOAP: implementações dos protocolos HTTP e SOAP, respectivamente, para o sub-sistema Ginga-NCL, utilizando linguagem Lua, que permitem a convergência entre aplicações de TVDi e a *Web*.

A escolha de implementação de HTTP e SOAP partiu da inexistência de versões livres e de código aberto de tais protocolos para o Ginga-NCL. Até onde sabe-se, o NCLua HTTP e NCLua SOAP são as primeiras implementações livremente disponibilizadas.

5.2 Delimitação do Problema

Apesar de Lua ser uma linguagem extensível [Ierusalimschy, Figueiredo e Celes 2007], principalmente pela capacidade de utilizar módulos construídos em linguagem C, e de existirem vários destes módulos para as mais diversas finalidades, tais módulos binários não podem ser utilizados em aplicações interativas de TV Digital enviadas via *broadcast*, devido a questões de segurança, uma vez que um módulo escrito em linguagem C pode ter acesso a qualquer funcionalidade do sistema operacional (embarcado juntamente com o *middleware* no receptor de TV Digital).

Em [Braga e Restani 2010] são citadas algumas ameaças de segurança em sistemas de TVDi, como transação fraudulenta (perda/roubo de dados), pirataria de conteúdo, falsificação, violação ou corrupção das aplicações e uso ilegítimo de serviços do provedor.

Tais ameaças podem ser potencializadas com o uso de módulos binários. Além do mais, a compilação de módulos em C gera código nativo dependente de plataforma, o que não garante que a aplicação executará em qualquer receptor [Costa 2009]. Somente aplicações residentes podem ser desenvolvidas em linguagens compiladas como C.

Com isto, para haver, em aplicações NCLua de TVDi, algumas das funcionalidades dos módulos binários citados, é preciso implementar módulos inteiramente em linguagem Lua, cujo ciclo de vida é controlado pelo *middleware* Ginga[ABNT 2008].

5.3 Tecnologias Envolvidas e Trabalhos Relacionados

Nesta seção são apresentadas as tecnologias envolvidas no desenvolvimento da solução apresentada e os trabalhos relacionados.

5.3.1 Tecnologias de *Web Services*

SOAP é um protocolo padrão da *World Wide Web Consortium* (W3C) que permite a aplicações oferecerem seus serviços na *Internet*, em uma arquitetura distribuída no modelo cliente/servidor. O protocolo permite troca de mensagens e chamadas de procedimentos remotos. Tais serviços podem ser providos e consumidos por aplicações desenvolvidas em diferentes linguagens e plataformas. Isto permite a interoperabilidade entre diferentes aplicações, por meio da troca de documentos XML, usando algum protocolo de transporte como o HTTP [W3C 2007] [Curbera et al. 2002] [Newcomer 2002].

Um dos grandes benefícios do protocolo SOAP para a integração de aplicações é a sua linguagem para descrição dos serviços disponibilizados, a *Web Service Description Language* (WSDL). De forma padronizada, manual ou automatizadamente, uma aplicação cliente pode conhecer os serviços disponibilizados por um *Web Service* lendo o documento WSDL do serviço (também em formato XML)[W3C 2007].

Os *Web Services* permitem a construção de aplicações distribuídas pela *Internet*, garantindo a centralização de regras de negócios em servidores de aplicações, encarregados de toda a carga de processamento de tais regras. Considerando-se isto, *Web Services* são ideais para aplicações clientes executando em sistemas com restritos recursos de *hardware*, como celulares e *Set-top Boxes*, estes últimos sendo o foco do presente trabalho. Além de desonerar os clientes da carga de processamento, alterações nas regras de negócio não requerem a atualização das aplicações clientes.

5.3.2 Lua e os *scripts* NCLua

Lua é a linguagem imperativa utilizada pelo sub-sistema Ginga-NCL para o desenvolvimento de aplicações procedurais. Ela tem como grandes vantagens sua simplicidade, eficiência e portabilidade. Tais características são extremamente importantes em uma linguagem a ser utilizada em dispositivos com recursos de hardware restritos, como os conversores de TV Digital (*Set-top Boxes*). Além de tudo, Lua é livre de *royalties*, o que permite que a mesma seja embarcada em *Set-top Boxes* sem onerar o custo dos equipamentos. Este é um requisito muito importante, considerando as características sócio-econômicas do Brasil, a intenção do governo de utilização da TV como meio para inclusão digital e sua presença na grande maioria dos lares brasileiros.

Como a linguagem NCL é apenas declarativa, a inclusão de características imperativas a uma aplicação de TVDi para o Ginga-NCL é possibilitada por meio dos chamados NCLua, *scripts* Lua funcionando como nós de mídia dentro de um documento NCL [ABNT 2008] [Soares, Rodrigues e Moreno 2007].

Tais *scripts* aumentam o poder das aplicações de TVDi, possibilitando a construção de aplicações sofisticadas, seguindo paradigmas como Programação Estruturada e Programação

Orientada a Objetos, com definição de regras de negócio na aplicação, interoperabilidade com sistemas na *Internet* por meio de protocolos como HTTP e SOAP, entre outros recursos.

O que diferencia os *scripts* NCLua de *scripts* Lua convencionais é a possibilidade de comunicação bidirecional entre este e o documento NCL. Tal comunicação acontece por meio de eventos, como definido na norma do Ginga-NCL em [ABNT 2008]. Segundo [Sant'Anna, Cerqueira e Soares 2008] "essa integração deve seguir critérios que não afetem os princípios da linguagem declarativa, mantendo uma separação bem definida entre os dois ambientes". Para isto, nenhuma alteração nas linguagens NCL ou Lua foi necessária, o que garante a evolução independente das linguagens, como ressalta [Sant'Anna, Cerqueira e Soares 2008].

A integração entre as linguagens NCL e Lua foi feita para ser minimamente intrusiva, garantindo uma separação entre a forma declarativa e a procedural de desenvolver uma aplicação para o Ginga-NCL. Assim, o código NCLua deve ser escrito obrigatoriamente em um arquivo separado do NCL. Isto permite uma clara divisão de tarefas entre profissionais da área de *design* e da área de programação [Sant'Anna, Cerqueira e Soares 2008]. A integração entre outras linguagens como HTML e *JavaScript* é bastante intrusiva, onde, muitas vezes, código *JavaScript* é intercalado com código HTML.

O tratamento de eventos e as particularidades associadas a aplicações de TVDi, desenvolvidas em linguagem Lua, são implementadas por módulos adicionais à Linguagem, definidos na norma do Ginga-NCL [ABNT 2008]. Isto permite que a linguagem se mantenha inalterada para o contexto de TV Digital. Os módulos disponíveis em NCLua, utilizados neste trabalho são: ***event***, que permite a comunicação bidirecional entre um NCL e um NCLua; e ***canvas***, que disponibiliza uma API para desenhar imagens e primitivas gráficas.

5.3.3 Protocolo TCP no Ginga-NCL

A norma do Ginga-NCL [ABNT 2008] define a disponibilidade do protocolo TCP para ser utilizado pelas aplicações interativas. Como os objetos NCLua têm a característica de poderem ser orientados a eventos, o módulo *event* permite a captura e tratamento de eventos, possibilitando a comunicação assíncrona entre o formatador NCL e um objeto NCLua.

A implementação do protocolo TCP deve ser disponibilizada por meio do módulo *event*. A norma especifica uma classe de eventos denominada *tcp*. Assim, por meio das funções do módulo *event*, um objeto NCLua pode enviar requisições e receber respostas usando o protocolo TCP.

Devido à característica assíncrona do módulo *event*, o tratamento de requisições TCP em NCLua não é trivial. Para facilitar o uso de tal protocolo, pode-se recorrer ao recurso de co-rotinas da linguagem Lua. Segundo [Ierusalimschy 2006], uma co-rotina é similar a um *thread* (no sentido de *multithreading*). No entanto, co-rotinas são colaborativas, sendo executada apenas uma por vez.

A documentação de NCLua disponível em [Sant'Anna 2010], apresenta um módulo que utiliza co-rotinas para facilitar o tratamento das requisições assíncronas da classe de eventos *tcp*. No entanto, mesmo tal módulo ainda não permite encapsular todos os detalhes do envio da requisição em apenas uma função, para simplificar o uso para os desenvolvedores de aplicações interativas.

A Figura 5.1 apresenta um gráfico de máquinas de estados da realização de uma conexão TCP no Ginga-NCL, utilizando o módulo *tcp.lua*. Em um processo convencional, a aplicação estabelece uma conexão a um servidor. Após estabelecida a conexão ela envia uma requisição e fica aguardando o retorno. A função *tcp.receive* é executada até que não haja mais dados a serem recebidos, realizando a desconexão.

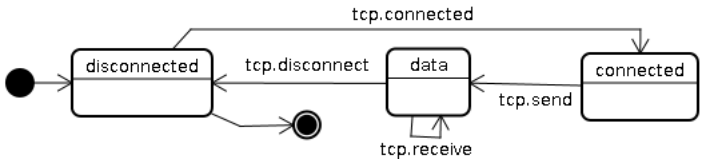


Figura 5.1: Diagrama de Máquinas de Estados do Módulo *tcp.lua*

A Figura 5.2 apresenta um gráfico de máquinas de estados do mesmo módulo, porém, do ponto de vista das co-rotinas em execução (que são semelhantes a *threads*, como já discutido anteriormente). O processo de uso do módulo é iniciado internamente com a criação de uma corotina (que é criada suspensa, não automaticamente iniciando sua execução). A função *coroutine.resume* inicia a execução da co-rotina. Quando é solicitada uma tentativa de conexão, a co-rotina é suspensa, ficando aguardando até que a conexão seja estabelecida, ocorrendo tudo de forma assíncrona. Neste momento, a co-rotina é novamente resumida (continuando a execução), permitindo que seja enviada uma requisição ao servidor (*tcp.send*). Tal função retorna imediatamente. Para a obtenção da resposta da requisição (que também será feita de forma assíncrona), é preciso usar a função *tcp.receive*, que faz com que a co-rotina seja suspensa novamente, até que algum dado da resposta seja obtido. A função *tcp.receive* pode ser chamada iterativamente até que não haja mais nenhum dado a ser retornado para a aplicação.

Todo este processo apresentado nos gráficos de máquinas de estado, mesmo utilizando as facilidades providas pelo módulo *tcp.lua*, não são encapsulados para facilitar o uso. O módulo citado apenas facilita o gerenciamento das chamadas assíncronas. A implementação do NCLua HTTP e NCLua SOAP encapsulam todas essas chamadas de funções, tornando o processo bem mais simples para o desenvolvedor, disponibilizando uma simples função para realização de uma requisição.

É importante lembrar que tal abordagem é útil em cenários de aplicações *stateless*, que não mantém estado entre uma requisição e outra, como o caso do HTTP/1.0 e das chamadas SOAP. Aplicações que precisem manter a conexão aberta, como mensageiros instantâneos, precisam utilizar diretamente as funções do módulo tcp.lua.

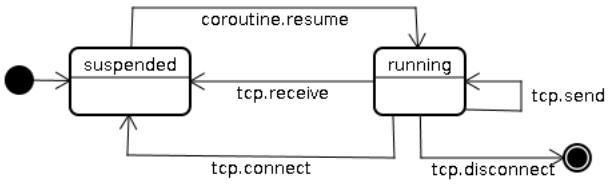


Figura 5.2: Diagrama de Máquinas de Estados do Módulo *tcp.lua* (co-rotinas em execução)

5.3.4 Implementações de SOAP

Existem diversos *toolkits* para provimento e consumo de *Web Services*, disponíveis em diferentes linguagens e plataformas. Nas sub-seções seguintes são apresentados alguns destes.

5.3.4.1 LuaSOAP

Para acesso a *Web Services* SOAP a partir de aplicações Lua, pode-se utilizar o módulo LuaSOAP [Ierusalimschy, Carregal e Guisasola 2004].

O protocolo SOAP envolve a troca de mensagens em formato XML, permitindo a interoperabilidade entre sistemas desenvolvidos em diferentes linguagens. No entanto, para fazer o *parse* de arquivos XML, o LuaSOAP depende da biblioteca *Expat* [Jr e Waclawek 2007][Cooper 1999], que é um *parser* XML desenvolvido em linguagem C, cujos problemas de uso em aplicações de TVDi foram apresentados na Seção 5.2. O projeto também depende da biblioteca *LuaSocket*, que também usa módulos em C.

O LuaSOAP não permite a geração automática de *stubs* para realizar a chamada aos métodos remotos do *Web Service*. Desta forma, o desenvolvedor precisa ler o documento WSDL e obter as informações sobre o método que deseja invocar. A última atualização do projeto foi em 2004, o que mostra que o mesmo não está acompanhando as novas versões de Lua, como a 5.1 utilizada na implementação de referência do Ginga.

Uma das vantagens do projeto é que as chamadas aos métodos remotos no *Web Service* são síncronas, o que facilita bastante o uso.

5.3.4.2 gSOAP

O gSOAP é classificado como um *Software Development Kit* (SDK) para permitir que aplicações legadas, sistemas embarcados e de tempo real, desenvolvidos em linguagens C, C++ e Fortran, possam consumir *Web Services* [Engelen e Gallivan 2005]. O projeto possui um utilitário capaz de gerar *stubs* em C/C++, a partir do documento WSDL do serviço. Em tal *stub* são incluídos métodos *proxies* para realizar chamadas aos métodos remotos do serviço, tornando-as transparentes para a aplicação cliente.

As chamadas realizadas com gSOAP também são síncronas, o que facilita muito o desenvolvimento das aplicações clientes, pois o envio da requisição e tratamento da resposta pode ser todo feito em uma única rotina.

Pelo fato do projeto ser desenvolvido em C, o mesmo só poderia ser utilizado em aplicações de TVD residentes no conversor digital, como já comentado na Seção 5.2. A utilização de linguagens compiladas como C/C++ permite que o *(un)marshalling* seja feito em tempo de compilação, o que garante que a aplicação terá maior velocidade na execução. No entanto, a necessidade de tal processo de compilação elimina a grande vantagem do dinamismo existente em linguagens interpretadas como Lua.

5.3.4.3 Apache Axis

Em [Davis e Parashar 2005] são apresentados alguns outros *toolkits* para provimento e consumo de *Web Services*. Um dos projetos citados é o Apache Axis, uma implementação de SOAP para Java e C, que tem, como uma das vantagens, o uso do *parser* XML SAX, o qual é completo e bastante eficiente. Ele também possui a vantagem de gerar *stubs* Java ou C, a partir do documento WSDL.

Mesmo o Ginga permitindo o uso de Java, o Apache Axis pode não ser uma solução ideal para aplicações de TVD, uma vez que inclui o SAX como *parser* XML. Considerando a capacidade de hardware restrita dos conversores de TV Digital, tal *parser* pode não ser ideal em tais equipamentos, além de poder não estar em conformidade com as normas do Sistema Brasileiro de TV Digital (SBTVD). *Parsers* mais simples como o NanoXML[1], que demandam menos capacidade de processamento, podem ser mais adequados neste cenário. Por fim, o Apache Axis não atende a um dos requisitos elicitados: ser implementado em Lua para uso direto por aplicações NCLua.

5.4 Proposta de Novas Implementações de HTTP e SOAP

O Ginga-NCL provê protocolos de rede até o TCP, assim, qualquer protocolo acima da camada de transporte do modelo OSI precisa ser implementado. Como o envio de envelopes SOAP é normalmente feito por HTTP, tal protocolo precisou ser implementado como um módulo NCLua, ao qual denominou-se NCLua HTTP, que será utilizado pelo NCLua SOAP.

5.4.1 Decisões de Projeto

Para o desenvolvimento do projeto, optou-se por seguir o padrão de módulos, amplamente utilizado na atual versão da linguagem Lua. Um módulo encapsula funções com objetivos correlatos e tal padrão é bastante conhecido dos programadores Lua.

[1] `http://nanoxml.sourceforge.net`

Tal modelo segue o paradigma de programação estruturada, assim, um módulo consiste de um conjunto de funções que são chamadas pelo desenvolvedor que for utilizá-lo.

Na geração das requisições SOAP, optou-se por fazer o *(un)marshalling* de XML para tabelas Lua por estas serem as estruturas de dados padrões disponibilizadas pela linguagem e por serem estruturas bastante flexíveis e fáceis de manipular, devendo ser de conhecimento de qualquer programador Lua.

Como o conteúdo das requisições HTTP e SOAP é apenas texto, formatado segundo o padrão de cada protocolo, a geração e formatação deste conteúdo foi bastante simples e direta. Devido a *strings* em Lua serem imutáveis[Ierusalimschy 2006], para economizar memória na concatenação das *strings* que compõe cada requisição, as tabelas de Lua foram utilizadas como um *buffer* de *strings* para otimizar o uso de memória RAM.

Na obtenção dos resultados, procurou-se facilitar ao máximo tal tarefa para o desenvolvedor, permitindo que ele tenha acesso direto aos dados retornados, sem precisar conhecer o caminho dentro do XML de retorno onde a resposta está armazenada, assim como fazem outros *toolkits* SOAP como a API JAX-WS.

Quanto ao *parser* XML escolhido, não haviam muitas opções implementadas inteiramente em Lua. Todas as implementações testadas foram encontradas em `http://lua-users.org` e em [Ierusalimschy 2006]. Optou-se pelo uso do LuaXML[2], pois foi a implementação que fazia o *unmarshalling* de XML para tabelas Lua com a estrutura mais simples e fácil de manipular. Além do mais, as outras implementações encontradas não funcionaram adequadamente para XML's mais complexos retornados por alguns *Web Services*.

5.4.2 NCLua HTTP

O NCLua HTTP[3] implementa alguns dos principais recursos do protocolo HTTP/1.0. Ele é um módulo escrito inteiramente em linguagem Lua para ser utilizado em *scripts* NCLua. O mesmo utiliza o protocolo TCP da forma como especificado na norma do Ginga-NCL em [ABNT 2008], por meio da classe de eventos *tcp* de NCLua. Pela simplicidade do protocolo HTTP, o módulo possui apenas algumas funções que permitem a geração de requisições e tratamento de respostas. Atualmente existem os seguintes recursos implementados:

- suporte à autenticação básica, uso de portas específicas e download de arquivos;
- suporte à requisições *GET* e *POST*;
- passagem de parâmetros em requisições *POST*;
- suporte à passagem de cabeçalhos HTTP e definição de *User-Agent*;

[2] `http://lua-users.org/wiki/LuaXml`
[3] `http://ncluahttp.manoelcampos.com` e `http://labtvdi.unb.br`

- suporte à separação automática dos dados do cabeçalho e do corpo da resposta de uma requisição.

5.4.3 NCLua SOAP

O NCLua SOAP[4] implementa as principais funcionalidades do protocolo SOAP 1.1 e 1.2. Ele também é um módulo inteiramente escrito em Lua, que faz o *parse* de arquivos XML, realizando o *marshalling* e *unmarshalling* de/para tabelas Lua, permitindo que o desenvolvedor Lua trabalhe com a estrutura de dados principal da linguagem: o tipo *table*.

O módulo utiliza o NCLua HTTP para transportar as mensagens SOAP. Assim, todos os detalhes do protocolo HTTP são encapsulados pelo respectivo módulo. Com isto, a implementação do SOAP fica bastante simplificada, tornando o código fácil de ser mantido. O NCLua SOAP encarrega-se apenas de gerar o XML da requisição SOAP, utilizando o NCLua HTTP para enviar tal XML no corpo da mensagem. O *parse* e *unmarshalling* do XML para uma tabela Lua é todo encapsulado pelo módulo LuaXML[5].

Um importante recurso, não disponível em implementações como o LuaSOAP (citada na Seção 5.3), e que facilita bastante a utilização do módulo, é a simplificação do XML retornando como resposta, que é convertido (*unmarshalling*) automaticamente para uma tabela Lua. Para demonstrar este recurso, utilizar-se-á o *Web Service* de consulta de endereço a partir de um CEP, disponível em `http://www.bronzebusiness.com.br/webservices/wscep.asmx`. Tal WS possui um método chamado "cep", que recebe um determinado CEP e retorna o endereço referente ao mesmo. O XML do retorno do método "cep", convertido para uma tabela Lua, é semelhante ao mostrado na Listagem 5.1.

A estrutura da tabela reflete o código XML retornado. Como pode ser visto, o elemento da tabela que contém de fato os dados do endereço retornado (tbCEP) está envolvido em várias outras tabelas que não contém dado algum, sendo estruturas completamente desnecessárias para a aplicação NCLua. Com isto, para o desenvolvedor poder acessar, por exemplo, a cidade do CEP indicado, precisará conhecer toda a estrutura retornada, utilizando uma instrução como *result.cepResult.diffgr.NewDataSet.tbCEP.cidade*.

Para esconder estes detalhes do desenvolvedor, o NCLua SOAP simplifica qualquer resultado que contenha estruturas desnecessárias, como o mostrado na Listagem 5.1.

Listagem 5.1: Exemplo de tabela Lua gerada a partir do XML de uma resposta SOAP

```
1 { cepResult = { diffgr = { NewDataSet = { tbCEP = {
2   nome="Cln 407", bairro="Asa Norte", UF="DF", cidade="Brasilia"
3 } } } } }
```

Desta forma, para o exemplo citado, a tabela Lua (gerada a partir do XML de retorno da requisição) ficará como apresentado na Listagem 5.2, o que simplifica o acesso aos ele-

[4] `http://ncluasoap.manoelcampos.com` e `http://labtvdi.unb.br`

[5] *Parser* XML escrito inteiramente em Lua, adaptado para funcionar com Lua 5

mentos da estrutura retornada, permitindo, por exemplo, que o campo cidade seja acessado utilizando-se apenas a instrução *result.cidade*.

Listagem 5.2: Exemplo de simplificação de retorno de resposta SOAP pelo NCLua SOAP

```
1 {    nome="Cln 407", bairro="Asa Norte", UF="DF", cidade="Brasilia"    }
```

O módulo ainda conta com um *script* (*wsdlparser.lua*), em fase inicial de implementação, que realiza o *parse* de um documento WSDL e obtém algumas das informações que precisa-se passar ao NCLua SOAP para que ele realize a requisição (como o *namespace* do serviço, o nome do método desejado e a lista de parâmetros de entrada). Atualmente o *script* apenas lê o WSDL e exibe algumas das informações citadas, cabendo ao desenvolvedor copiá-las e passá-las ao método *call* do NCLua SOAP para realizar a chamada a um determinado método remoto. No entanto, a extração de tais informações já ajuda de alguma forma, principalmente os usuários menos experientes na tecnologia de *Web Services* e no NCLua SOAP.

Para resumir, as características principais do NCLua SOAP são:

- suporte a SOAP 1.1 e 1.2;
- suporte a parâmetros de entrada e saída do tipo *struct* e *array* (sendo feito *marshalling* e *unmarshalling* de/para tabelas Lua automaticamente);
- facilidade para manipulação de chamadas assíncronas, característica do protocolo TCP disponível no Ginga-NCL;
- simplicidade na obtenção do retorno de uma requisição a um método remoto;
- suporte a SOAP *Fault* para captura de erros SOAP;
- suporte a SOAP *Header*[W3C 2007] possibilitando a passagem de parâmetros específicos da aplicação (como informações sobre autenticação, pagamento, etc);
- realização de testes com *Web Services* desenvolvidos em diferentes linguagens.

A Figura 5.3 apresenta um diagrama de componentes dos módulos implementados. O módulo *event* faz parte do Ginga-NCL e é responsável por tratar eventos, como requisições e obtenção de respostas por meio do protocolo TCP. Ele é a base para toda a implementação. O módulo *tcp.lua* facilita o gerenciamento das requisições TCP assíncronas, geradas por meio do módulo *event*. O módulo *ncluahttp.lua* implementa o protocolo HTTP, utilizando o TCP como camada de transporte. O módulo *ncluasoap.lua* implementa o protocolo SOAP, utilizando o *ncluahttp.lua* para enviar os envelopes SOAP por HTTP. Uma aplicação cliente, que queira utilizar o protocolo HTTP (*NCLua HTTP Client App*), pode fazer uso direto das funções do módulo *ncluahttp.lua*, abstraindo todos os outros módulos. Uma aplicação cliente, que queira consumir *Web Services* SOAP (*NCLua Web Service Client App*), pode fazer uso direto das funções do módulo ncluasoap.lua, também abstraindo todos os outros módulos.

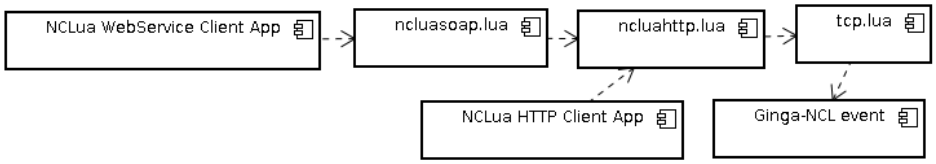


Figura 5.3: Diagrama de Componentes do NCLua SOAP e NCLua HTTP

5.5 Exemplos de Uso e Testes de Interoperabilidade

Nesta seção são apresentados os exemplos de utilização e aplicações desenvolvidas utilizando os módulos NCLua HTTP e NCLua SOAP, bem como testes de interoperabilidade realizados.

5.5.1 Exemplos de uso do NCLua HTTP

A utilização básica do módulo NCLua HTTP é por meio de uma chamada *GET* a uma determinada URI, utilizando o método *request* do módulo, como exemplificado no *script* NCLua da Listagem 5.3. Tal exemplo envia uma requisição HTTP *GET* para a página em `http://manoelcampos.com/votacao/votacao2.php`, enviando um parâmetro "voto"com valor igual a "sim", obtendo o resultado (um código HTML neste caso) e exibindo no terminal. Não existe interface gráfica neste exemplo, desta forma, o *script* contém apenas detalhes da requisição HTTP, mas é inteiramente funcional.

Na Listagem 5.3, a linha 1 adiciona o módulo NCLua HTTP. A linha 8 chama a função *http.request* que envia a requisição HTTP. Devido à particularidade assíncrona do protocolo TCP no Ginga-NCL (que é utilizado para transportar as mensagens HTTP), como explicado na Seção 5.3.3, para facilitar o envio de requisições e recebimento de respostas, o módulo NCLua HTTP utiliza co-rotinas de Lua. Por este motivo, a obtenção do retorno deve ser feita dentro de uma função, como a definida na linha 6, contendo os parâmetros header e body (comentados na definição da mesma). Tal função deve ser passada como parâmetro à função *http.request*, para que seja chamada por esta quando a resposta for obtida.

Listagem 5.3: Exemplo de envio de requisição GET com NCLua HTTP

```
1 require "http"
2 ---Funcao de callback executada de forma assincrona pelo modulo,
3 --quando a resposta da requisicao eh obtida.
4 --@param header Headers HTTP enviados na resposta
5 --@param body Corpo da mensagem HTTP
6 function getResponse(header, body) print("Resposta obtida\n", body) end
7
8 http.request("http://manoelcampos.com/voto.php?voto=sim", getResponse)
```

O envio de requisições *POST* é bastante semelhante ao exemplo apresentado anteriormente. Neste caso, os parâmetros devem ser passados à função *http.request* por meio de

uma tabela Lua. A formatação destes valores, como exigido pelo protocolo HTTP, é feita automaticamente pelo NCLua HTTP.

Algumas aplicações foram desenvolvidas como prova de conceito do uso do módulo NCLua HTTP, as quais são apresentadas a seguir:

- **Enquete TVD**: enquete com registro de voto e apuração a partir de servidor *Web*;
- **NCLua RSS Reader**: leitor de notícias RSS de um provedor de conteúdo na *Web*;
- **NCLua Tweet**: envio e recebimento de mensagens pelo micro *blog Twitter*.

5.5.2 Exemplos de uso do NCLua SOAP

A seguir são apresentados exemplos de uso do NCLua SOAP para consumo de alguns *Web Services* disponíveis na *Internet*.

A Listagem 5.4 apresenta um exemplo de aplicação para exibir a previsão do tempo de uma cidade. A mesma não possui interface gráfica, mostrando o resultado no terminal, para simplificar o código. A linha 10 realiza a chamada ao método remoto. Os dados para realização da requisição (incluindo o endereço do serviço, nome do método remoto e parâmetros de entrada) devem ser informados em uma tabela Lua, como mostrado entre as linhas 4 a 8. A obtenção do retorno do método remoto não é direta, devido à característica assíncrona do protocolo TCP no Ginga-NCL, como já explanado nas Seções 5.3.3 e 5.4.2. Desta forma, é necessária a definição de uma função que deve receber apenas um parâmetro (normalmente nomeado de *result*), como a função *getResponse* exibida na linha 2. Tal função deve ser passada como parâmetro ao método *call* do módulo ncluasoap, como mostrado na linha 10. O envio da requisição será executado dentro de uma co-rotina Lua, criada pelo NCLua SOAP. A função *getResponse* será executada automaticamente quando o retorno for obtido. Desta forma, todo o controle das chamadas assíncronas é feito pelo NCLua SOAP, como já amplamente discutido na Seção 5.4.2.

Como o serviço consumido neste exemplo retorna apenas uma *string* com a previsão do tempo (chuvoso, nublado, ensolarado, etc), para exibir tal resultado basta imprimir o parâmetro *result* da função *getResponse*, como apresentado na linha 6.

Listagem 5.4: Exemplo de consumo de *Web Service* de previsão do tempo

```
1 require "ncluasoap"
2 function getResponse(result) print("Previsao do Tempo: ", result) end
3
4 local msg = {
5   address = "http://www.deeptraining.com/webservices/weather.asmx",
6   namespace = "http://litwinconsulting.com/webservices/",
7   operationName = "GetWeather", params = {  City = "New York"  }
8 }
```

```
9
10 ncluasoap.call(msg, getResponse)
```

O exemplo apresentado na Listagem 5.5 consume um serviço para obtenção de um endereço a partir do CEP. Observe que o que muda entre as linhas 6 e 10, em relação ao exemplo anterior, são apenas os valores da tabela "msg", que contém os dados para geração da requisição SOAP. Na função *getResponse*, entre as linhas 2 e 4, note que agora, o resultado retornado é um tipo composto, que é acessado como uma tabela Lua. Tal tabela conterá campos com os valores armazenados no XML enviado pelo *Web Service*.

Listagem 5.5: Exemplo de consumo de WS de consulta de endereço a partir do CEP

```
1  require "ncluasoap"
2  function getResponse(result)
3      print(result.nome, result.bairro, result.cidade, result.UF)
4  end
5
6  local msg = {
7    address = "http://www.bronzebusiness.com.br/webservices/wscep.asmx",
8    namespace = "http://tempuri.org/",
9    operationName = "cep", params = {  strcep = "70855530"  }
10 }
11
12 ncluasoap.call(msg, getResponse)
```

Algumas aplicações foram desenvolvidas como prova de conceito de utilização do módulo. Entre elas estão o rastreamento de encomendas postadas pelos Correios, consulta de cotação do Dólar, previsão do tempo, consulta de endereço a partir do CEP e outras.

5.5.2.1 Testes de Interoperabilidade

Os diversos serviços consumidos pelas aplicações apresentadas são desenvolvidos em diferentes linguagens e plataformas. Procurou-se realizar testes com estes diferentes serviços para verificar a interoperabilidade do NCLua SOAP (um dos requisitos fundamentais, se não o mais importante, para qualquer implementação SOAP). Com os testes realizados pôde-se comprovar a eficiência do módulo, uma vez que para todos os serviços testados, obteve-se o retorno esperado, tratando-o de forma padronizada.

No início, alguns usuários do fórum do projeto relataram problemas ao utilizar serviços desenvolvidos em linguagem Java. Os problemas encontrados foram devido aos *Web Services* criados com a API JAX-WS, padrão da plataforma Java, especificarem os tipos de dados utilizados pelo serviço em um arquivo XSD (*XML Schema Definition*) externo, no lugar de especificar diretamente dentro do documento WSDL. Tal característica requer pequenas mudanças no formato da mensagem SOAP a ser enviada para consumir tais *Web Services*. Assim, foi preciso adequar o módulo para entrar em conformidade com o padrão SOAP.

Particularidades encontradas em *Web Services* PHP também foram relatadas e o módulo

foi adequado para permitir a interoperabilidade com tais serviços. É importante ressaltar que, no caso de tais serviços PHP, estes foram desenvolvidos utilizando o *tookit* nuSOAP[6]. Tal *tookit* não leva em conta os nomes dos parâmetros de entrada e sim a ordem dos mesmos, não estando em conformidade com o padrão SOAP. Assim, o problema é específico da implementação do nuSOAP, mas que foi contornado pelo NCLua SOAP para evitar quaisquer transtornos ao consumir *Web Services* desenvolvidos com tal *tookit*.

5.6 Ambiente de desenvolvimento

Para o desenvolvimento do projeto foram utilizados:

- sistema operacional GNU/Linux Ubuntu 10.10 como sistema desktop para realização das tarefas de desenvolvimento:
  - ferramentas *telnet* e *curl* para teste de envio de requisições HTTP/SOAP geradas com os módulos implementados.
- *soapUI* 3.5.1 para testes de consumo de *Web Services* SOAP;
- *Astah Community* 6.1 (antigo *Jude*) para modelagem UML;
- IDE Eclipse 3.6:
  - *plugin* NCLEclipse 1.5.1;
  - *plugin* LuaEclipse 1.3.1.
- ferramenta LuaDoc[7] para geração de documentação;
- implementação de referência do Ginga-NCL (Ginga *Virtual Set-top Box* 0.11.2):
  - módulo *tcp.lua*[8] para envio de requisições TCP;
  - módulo LuaXML para tratamento de XML;
  - módulo *LuaProfiler*[9] para realização das análises de desempenho, descritas na Seção 5.7.
- interpretador Lua para execução do *script wsdlparser.lua* fora do Ginga *Virtual Set-top Box*.

[6] `http://sourceforge.net/projects/nusoap`
[7] `http://luadoc.luaforge.net`
[8] `http://www.lua.inf.puc-rio.br/~francisco/nclua/`
[9] `http://luaprofiler.luaforge.net`

5.7 Avaliação de Desempenho e Comparações

5.7.1 Avaliação de Desempenho do NCLua SOAP

Utilizando-se a ferramenta *LuaProfiler* (que precisou ter seu processo de compilação alterado para funcionar em aplicações NCLua[10]) foram feitas algumas análises do desempenho do NCLua SOAP para avaliar o tempo gasto para geração da requisição SOAP. Por meio da função *collectgarbage* da biblioteca padrão de Lua, pôde-se verificar o total de memória consumida pelo módulo.

O percentual de uso da CPU foi obtido por meio da ferramenta *top*, existente no sistema operacional do Ginga *Virtual Set-top Box*. Para a obtenção deste percentual, executou-se a aplicação NCL (que não inicia o *script* Lua automaticamente) e verificou-se o uso da CPU pelo processo de nome "ginga"(responsável pela execução das aplicações interativas) no momento em que este se tornava estável. Em seguida, por meio de um comando na aplicação NCL, o *script* Lua é então executado, gerando e enviando a requisição SOAP. Após isto, verificou-se o pico de uso do processador, considerando que os *scripts* Lua testados somente enviam a requisição SOAP e exibem o retorno no terminal. Com a diferença entre o pico de uso do processador depois da execução do *script* e o valor observado antes do início do mesmo, obtém-se o percentual de uso do processador pelo NCLua SOAP.

A Tabela 5.1 apresenta os resultados obtidos para algumas das aplicações testadas.

***Web Service* consumido**	**Parâmetros de entrada**	**Tempo de geração da requisição: chamada ao método *call* (em segundos)**	**Uso de Memória RAM (em KB)**	**% de Uso da CPU**
Situação do tempo `http://www.deeptraining.com/webservices/weather.asmx`	*City* = "Brasília"	0.13	121.78	0.3
Conversão de Moeda `http://www.webservicex.net/CurrencyConvertor.asmx`	*FromCurrency* = "USD" *ToCurrency* = "BRL"	0.13	121.65	0.3
Consulta de endereço a partir do CEP `http://www.maniezo.com.br/webservice/soap-server.php`	cep = "77021682"	0.13	133.17	0.3

Tabela 5.1: Avaliação de Desempenho do NCLua SOAP

Como pode ser observado na Tabela 5.1, o tempo para geração da requisição, ou seja, a chamada ao método *call* do módulo, na implementação de referência do Ginga (Ginga *Virtual Set-top Box*, versão 0.11.2) está em torno de 0.13 segundo. O método *call* inclui a conversão dos dados passados em uma tabela Lua para o formato XML para serem enviados ao *Web Service*. Tal método não aguarda a conexão ao servidor *Web* e nem o envio e resposta da requisição, pois tais operações são assíncronas no Ginga-NCL. Assim, o tempo de execução do método *call* reflete apenas o tempo de geração da requisição SOAP. Os tempos

[10] `http://goo.gl/42CQA`

obtidos nos testes não variaram muito, e a variação vai depender apenas do total de parâmetros passados.

O uso de memória RAM também ficou baixo, em torno de 120KB, que varia de acordo com os parâmetros de entrada e saída do método remoto.

Em todos os exemplos executados, pôde-se observar que o percentual de uso da CPU se manteve estável em 0.3%, sendo um valor consideravelmente baixo.

Foram feitas análises do uso direto do módulo *tcp.lua* para verificar o *overhead* causado pelo NCLua SOAP, que é uma camada sobre o *tcp.lua*. Nos três exemplos apresentados na Tabela 5.1, o tempo de uso da CPU se manteve também estável e igual aos valores obtidos com o NCLua SOAP, 0.3%. Não foram feitas medidas quanto ao tempo gasto (em segundos) para geração da requisição, pois, para uso direto do *tcp.lua*, o desenvolvedor precisa passar ao módulo uma *string* já com a requisição SOAP formatada. Assim, não há um *delay* para geração da requisição pois o *tcp.lua* recebe a mesma pronta para envio (o que é bastante complicado para o desenvolvedor gerar manualmente tal requisição HTTP/SOAP, além de ser totalmente inflexível).

Quanto ao consumo de memória RAM, o uso direto do *tcp.lua* permite uma otimização neste sentido, uma vez que a quantidade de varáveis e estruturas de dados declaradas é muito menor, considerando que o módulo deve receber a requisição pré-formatada. Para o primeiro exemplo da Tabela 5.1, a aplicação consumiu 76.02 KB de memória RAM, para o segundo consumiu 75.59 KB e para o terceiro exemplo consumiu 66.28 KB. Tais valores mostram que o consumo de memória está em torno de 50% do consumido pelo NCLua SOAP.

Foram feitas aplicações para consumir os mesmos *Web Services* apresentados na Tabela 5.1 utilizando o módulo LuaSOAP. No entanto, a aplicação funcionou apenas para o *Web Service* de obtenção de endereço a partir do CEP, que implementa o SOAP 1.1. Para os outros *Web Services* testados, que implementam SOAP 1.2, o LuaSOAP não enviou a requisição no formato esperado e os *Web Services* retornaram mensagens de erro. O LuaSOAP é um projeto que não tem atualizações desde 2004, assim, tais problemas eram esperados. Com o *Web Service* de busca de endereço, a aplicação consumiu 139.20KB de memória RAM, cerca de 6KB a mais que com o NCLua SOAP. O tempo de geração da requisição foi de apenas 0.01 segundo, bem inferior ao do NCLua SOAP, que levou 0.13 segundo. Presume-se que este tempo reduzido gasto pelo LuaSOAP é devido ao mesmo utilizar módulos escritos em C, que por serem compilados, têm este ganho de desempenho.

Quanto ao uso de CPU, a aplicação com o LuaSOAP teve pico de 1.0%, sendo que com o NCLua SOAP este percentual foi de apenas 0.3. Presume-se que tal valor elevado é devido ao fato de uso do *parser* XML *Expat*, que é uma implementação mais completa que o LuaXML usado no NCLua SOAP.

Com tais valores apresentados anteriormente, considera-se que o NCLua SOAP é bastante eficiente em questões de tempo de processamento e uso de memória. Presume-se que o tempo de processamento pode ser reduzido em um hardware dedicado como o dos *Set-top*

Boxes, pois os testes foram realizados em uma sistema operacional Linux com várias aplicações em execução e em uma máquina virtual (cujo desempenho é inferior a de uma máquina real com finalidades específicas).

5.7.2 Comparativo entre os módulos implementados e o uso direto do módulo tcp.lua

Nesta seção é feito um comparativo entre aplicações desenvolvidas utilizando os módulos implementados e o uso direto do módulo *tcp.lua*[Sant'Anna 2010].

O NCLua SOAP, juntamente como o NCLua HTTP, encapsulam toda a complexidade de envio de requisições SOAP por meio do protocolo HTTP, facilitando a utilização de tais protocolos em aplicações de TVDi. A quantidade de código necessária para a realização de requisições HTTP ou SOAP é bastante reduzida com os módulos implementados, permitindo uma maior agilidade no desenvolvimento de aplicações, além de minimizar a possibilidade de erros e reduzir a duplicação de código.

A Tabela 5.2 apresenta um comparativo do total de linhas existentes em aplicações utilizando os módulos desenvolvidos e outras que não os utilizam, mostrando como o código das aplicações é reduzido com o uso dos módulos implementados. A utilização dos módulos no desenvolvimento de aplicações, além de reduzir o código das mesmas, libera o desenvolvedor da necessidade de conhecer detalhes de implementação dos protocolos HTTP e SOAP. Sem a utilização do módulo *tcp.lua*, com o uso direto da classe de eventos *tcp* do Ginga-NCL, o código das aplicações aumentaria substancialmente, em torno de 100 linhas de código.

Aplicação/Protocolo	**Sem módulos implementados**	**Com módulos implementados**
Enquete/HTTP	14 linhas de código 5 funções utilizadas diretamente	5 linhas de código = 35% do anterior 1 função usada diretamente
Cotação do dólar/SOAP	64 linhas de código 5 funções utilizadas diretamente	15 linhas de código = 23% do anterior (9 são parâmetros do WS) 1 função usada diretamente

Tabela 5.2: Comparativo entre aplicações de TVD sem e com os módulos implementados

5.7.3 Comparativo entre o NCLua SOAP e outras implementações

A Tabela 5.3 apresenta um comparativo entre alguns *toolkits* SOAP e o NCLua SOAP.

Características	**Axis**	**Axis2**	**PHP**	**gSOAP**	**NCLua SOAP**
Linguagem	Java	Java	PHP 5	C++	NCLua
SOAP 1.1	Sim	Sim	Sim	Sim	Sim
SOAP 1.2	Sim	Sim	Sim	Sim	Sim
SOAP com Anexos	Sim	Sim	Sim	Sim	Ainda não
Geração de código cliente a partir do WSDL	Sim	Sim	Sim	Sim	Ainda não
Suporte para formato *document/literal*	Bom	Bom	Médio	Bom	Bom
Requisitos de *runtime*	JVM	JVM	PHP engine	Nenhum	Ginga-NCL
Documentação	Boa	Pequena	Média	Boa	Média

Tabela 5.3: Comparação entre NCLua SOAP e outros *toolkits* SOAP (adaptada de [Louridas 2006]).

Como pode ser observado na Tabela 5.3, o NCLua SOAP só não possui dois dos recursos encontrados nos outros *toolkits*: suporte a anexos e geração de código cliente a partir do WSDL. O uso de anexos não é algo comumente encontrado nos *Web Services* disponíveis na *Web* e com os quais foram feitos testes de interoperabilidade. A falta de geração automática de código cliente a partir do WSDL não inviabiliza o uso do módulo, pois tal código pode ser escrito pelo desenvolvedor que desejar consumir algum serviço, obtendo manualmente as informações necessárias no WSDL. No entanto, considera-se que tal recurso facilitará bastante o uso do módulo.

5.8 Limitações do módulo NCLua SOAP

A atual versão do NCLua SOAP possui algumas limitações, que entendemos não interferirem no uso do mesmo:

- necessidade de o desenvolvedor saber a versão do protocolo SOAP do serviço que ele deseja consumir;
- necessidade de extrair manualmente alguns dados como o *namespace* do serviço;
- falta de tratamento de erros retornados pelo protocolo HTTP, o que causará comportamentos inesperados na aplicação;
- necessidade de saber se o serviço utiliza um arquivo *XML Schema Definition* (XSD) externo, para poder indicar isto no momento da chamada ao método remoto;
- não permitir o envio de anexos em formato *Multipurpose Internet Mail Extensions* (MIME)[IETF 1996] dentro de uma mensagem SOAP.

Capítulo 6

Conclusões e Trabalhos Futuros

6.1 Conclusões

As implementações apresentadas nesta dissertação são todas originais, por não haver nenhuma outra implementação (pelo menos de código aberto) para o sub-sistema Ginga-NCL dos protocolos HTTP e SOAP, de um *framework* de componentes gráficos e de aplicações de *T-Commerce*. Tais implementações permitem o desenvolvimento de aplicações dos mais variados tipos para o Ginga-NCL, alavancando o desenvolvimento de aplicações interativas para o SBTVD (ISDB-TB).

Os módulos NCLua HTTP e NCLua SOAP facilitam a convergência entre *Web* e TV, escondendo os detalhes de implementação dos protocolos HTTP e SOAP do desenvolvedor de aplicações de TVDi, permitindo o surgimento de novas aplicações interativas que fazem uso de conteúdo da *Internet*. O NCLua SOAP permitiu o desenvolvimento de aplicações de *T-Government* como apresentado em [Barbosa, Kutiishi e Lima 2010], entre outras. Na página do projeto, em `http://ncluasoap.manoelcampos.com` e em `http://labtvdi.unb.br`, existe um *link* para o fórum de discussão do módulo, onde alguns usuários relatam a utilização do mesmo, por exemplo, em aplicações de recomendação de conteúdo e *T-Learning*.

Atualmente, o processo de obtenção dos dados para realizar a chamada a um método remoto em um *Web Service* ainda é praticamente todo manual, no entanto, após o desenvolvedor obter tais dados para o método remoto desejado, a realização da requisição é bastante simplificada, principalmente pelo fato de Lua ser uma linguagem de tipagem dinâmica, não obrigando a declaração de variáveis com seus respectivos tipos. O *feedback* dos usuários tem mostrado que o módulo está sendo bastante útil para a comunidade, além de permitir a evolução do mesmo.

A arquitetura proposta serve de base para o desenvolvimento de serviços de *T-Commerce* para o SBTVD, mostrando como diversos serviços podem ser integrados em uma única arquitetura para um propósito final. Tal arquitetura também serve como um estudo de caso

dos percalços enfrentados para a elaboração de um ambiente de desenvolvimento e testes de aplicações para o SBTVD, mostrando as ferramentas necessárias para isto.

As análises de desempenho apresentadas dos protocolos implementados mostraram como a solução proposta é eficiente em uso de processador e memória RAM, sendo uma solução ideal para uso em ambientes restritos de recursos, como os *Set-top Boxes*.

Os protocolos implementados são a base da integração entre *Web* e TV e estão sendo bastante úteis em diversos outros trabalhos, como apresentado no Capítulo 5, mostrando como os resultados alcançados se tornaram importantes para a comunidade de TV Digital no Brasil e na América Latina, onde quase todos os países já adotaram o ISDB-TB.

A aplicação de *T-Commerce* desenvolvida delineia o processo de desenvolvimento de aplicações que fogem do trivial, mostrando como aplicar recursos como orientação a objetos na linguagem Lua, uso do canal de retorno/interatividade no SBTVD, uso de arquivos XML e arquivos de dados em formato Lua, além de outros recursos. O uso do *framework* LuaOnTV para construção de interfaces gráficas de usuário é bastante flexível e dá suporte a múltiplos dispositivos, adaptação automática das dimensões dos componentes gráficos da aplicação, além do uso de *templates* para permitir uma definição centralizada da identidade visual da aplicação. Tal recurso de *templates* permite alterar a identidade visual da aplicação facilmente, podendo-se ter *templates* diferentes para tipos de dispositivos diferentes (como TV's e celulares).

Outras aplicações foram desenvolvidas, como o NCLua RSS *Reader* e o Rastreador de Encomendas que servem como prova de conceito dos protocolos como HTTP e SOAP, implementados nesta dissertação.

Os objetivos apresentados na Seção 1.1 foram todos alcançados, apresentando uma solução completa, desde o ambiente de desenvolvimento, até a realização de análises de desempenho das aplicações desenvolvidas.

A Tabela 6.1 apresenta uma descrição de como tais objetivos foram alcançados.

Objetivo Específico	Como foi alcançado
A	Os requisitos funcionais e não funcionais foram elicitados, tendo servido de guia para o desenvolvimento da arquitetura proposta;
B	a arquitetura foi proposta e desenvolvida, apresentando-se a mesma por meio de figuras e diagramas UML, além da montagem de uma distribuição GNU/-Linux contendo todo o ambiente de desenvolvimento elaborado;
C	um *framework* de comunicação de dados foi desenvolvido, composto pelos módulos NCLua HTTP e NCLua SOAP, que implementam os protocolos HTTP e SOAP, respectivamente, sendo a base de toda a interoperação das aplicações desenvolvidas com diferentes provedores de serviços *Web*;
D	as diferentes aplicações de TVDi propostas foram desenvolvidas e apresentadas, servindo como prova de conceito do uso dos *frameworks* desenvolvidos ou estendidos;
E	o *framework* LuaOnTV foi estendido, incluindo-se suporte a temas. O recurso de temas foi utilizado na aplicação de *T-Commerce* desenvolvida. A adaptação automática da interface da aplicação foi testada em ambiente virtual de TV Digital (o Ginga *Virtual Set-top Box*) com diferentes resoluções de tela. Em tal ambiente, a adaptação automática do tamanho da interface gráfica da aplicação ao tamanho da tela do televisor mostrou-se satisfatória, possibilitando que a aplicação de adapte automaticamente a diferentes dispositivos receptores de TVD;
F	um modelo de desenvolvimento de aplicações de TVDi baseado em *templates* foi elaborado e apresentado. Tal modelo foi utilizado na aplicação de *T-Commerce* desenvolvida, o que permitiu uma agilidade no desenvolvimento da mesma, definindo um conjunto de componentes gráficos básicos e uma identidade visual comum a todas as telas da aplicação desenvolvida.

Tabela 6.1: Objetivos Específicos Alcançados

6.2 Trabalhos Futuros

Como trabalhos futuros, pretende-se concluir a implementação do *parser* automático do documento WSDL e geração de *stubs* Lua, contendo funções *proxies* para realizar a chamada aos métodos remotos (semelhante a ferramentas como o *wsdl2java*[1], pretende-se transformar o *script wsdlparser* em um *wsdl2nclua*) e incluir tratamento de exceções para permitir que as aplicações de TVDi possam emitir mensagens amigáveis ao usuário quando uma requisição HTTP falhar.

O módulo NCLua HTTP permite que seja utilizado qualquer método HTTP em uma re-

[1] `http://ws.apache.org/axis/java/user-guide.html`

quisição, no entanto, apenas os método *GET* e *POST* foram testados. Desta forma, pretende-se realizar testes de conformidade utilizando-se os métodos HTTP *OPTIONS, HEAD, PUT* e *DELETE*. Pretende-se também implementar mais funcionalidades no módulo, como realizar redirecionamentos automaticamente a partir de respostas HTTP como 301 e 302, além de implementar alguns recursos do HTTP/1.1, como conexões persistentes.

A arquitetura também pode ser estendida nos pontos a seguir:

- incluir suporte a metadados na aplicação para que a mesma possa oferecer produtos vinculados a um programa televisivo, permitindo que a oferta do produto seja mostrada ao telespectador em momento determinado no arquivo de metadados, utilizando os recursos de sincronização de mídias existente na linguagem NCL;
- estudar a viabilidade e uso do protocolo de autorização *Open Authentication* (oAuth)[2], que é bastante utilizado atualmente em redes sociais, permitindo ao usuário ter um login e senha únicos para acesso a diferentes serviços, agilizando o processo de login (caso o usuário decida salvar localmente, de forma criptografada, as informações de autorização);
- estender a arquitetura para um modelo baseado em descoberta de serviços que possibilite a integração de diferentes lojas virtuais que implementem a arquitetura aqui proposta, utilizando recursos como a linguagem BPEL (*Business Process Execution Language*) para permitir a composição de serviços e tornar tal integração transparente para aplicação de TVDi;
- utilizar a extensão *WS-Security*[OASIS 2006] para permitir a segurança na comunicação entre os *Web Services* e as aplicações cliente;
- criar ferramenta, para execução do lado da emissora de TV, que permita o envio dos produtos em oferta via *broadcast*, possibilitando a usuários, sem conexão com a *Internet*, visualizarem tais produtos na tela de seu equipamento de TVD.

[2] `http://oauth.net`

Referências Bibliográficas

[ABNT 2008]ABNT, N. 15606-2. *Televisão digital terrestre–Codificação de dados e especificações de transmissão para radiodifusão digital Parte 2: Ginga-NCL para receptores fixos e móveis–Linguagem de aplicação XML para codificação de aplicações*, 2008.

[Barbosa, Kutiishi e Lima 2010]BARBOSA, R. de C.; KUTIISHI, S. M.; LIMA, V. de. Desenvolvendo serviços de governo eletrônico multiplataforma para TV interativa utilizando Web Services. *Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia 2010)*, 2010.

[Barbosa e Soares 2008]BARBOSA, S. D. J.; SOARES, L. F. G. TV digital interativa no Brasil se faz com Ginga: Fundamentos, Padrões, Autoria Declarativa e Usabilidade. *Atualizações em Informática*, p. 105–174, 2008.

[Board 2009]BOARD, R. A. RSS 2.0 Specification. Disponível em: `http://www.rssboard.org/rss-specification`. Acessado em: 20 mar. 2009., 2009.

[Braga e Restani 2010]BRAGA, A. M.; RESTANI, G. S. Introdução à segurança de aplicações para a TV digital interativa brasileira. *Simpósio Brasileiro de Segurança (SBSeg 2010)*, 2010.

[CETIC.br 2009]CETIC.BR. Tic Domicílios e Usuários - Pesquisa sobre o Uso das Tecnologias da Informação e da Comunicação no Brasil. Disponível em: `http://cetic.br/usuarios/tic/`. Acessado em: 04 mar. 2010., 2009.

[Chu et al. 2007]CHU, S. C. et al. Evolution of e-commerce Web sites: A conceptual framework and a longitudinal study. *Information & Management*, Elsevier, v. 44, n. 2, p. 154–164, 2007. ISSN 0378-7206.

[Cooper 1999]COOPER, C. Using expat. Disponível em: `http://www.xml.com/pub/a/1999/09/expat/`. Acessado em: 20 dez. 2010., 1999.

[Costa 2009]COSTA, L. C. de P. Segurança para o Sistema Brasileiro de Televisão Digital: Contribuições à Proteção de Direitos Autorais e à Autenticação de Aplicativos. Dissertação de Mestrado. *Escola Politécnica da Universidade de São Paulo*, 2009.

[Coulouris, Dollimore e Kindberg 2007]COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. *Sistemas Distribuídos: Conceitos e Projeto*. Porto Alegre, 4a. ed.: Bookman, 2007.

[Curbera et al. 2002]CURBERA, F. et al. Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI. *IEEE Internet computing*, v. 6, n. 2, p. 86–93, 2002. ISSN 1089-7801.

[Davis e Parashar 2005]DAVIS, D.; PARASHAR, M. Latency performance of SOAP implementations. In: IEEE. *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid.* [S.l.], 2005. ISBN 0769515827.

[Engelen e Gallivan 2005]ENGELEN, R. A. V.; GALLIVAN, K. A. The gSOAP toolkit for web services and peer-to-peer computing networks. In: *Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium on*. [S.l.: s.n.], 2005. ISBN 0769515827.

[Fernández e Goldenberg 2008]FERNÁNDEZ, F.; GOLDENBERG, S. Aplicaciones interactivas para la televisión digital en Chile. *Cuadernos de información*, I, n. 22, p. 12, 2008.

[Fielding 2000]FIELDING, R. T. Architectural styles and the design of network-based software architectures. PhD Thesis. *University of California, Irvine*, 2000.

[Fowler 2000]FOWLER, A. A Swing architecture overview. *Sun Microsystems*, Web-based Technical Report `http://java.sun.com/products/jfc/tsc/articles/architecture/index.html`, 2000.

[G1 2007]G1. TV digital estréia em São Paulo com transmissão de emissoras abertas. Disponível em: `http://g1.globo.com/Noticias/0,,MUL201387-15515,00-TV+DIGITAL+ESTREIA+EM+SAO+PAULO+COM+TRANSMISSAO+DE+EMISSORAS+ABERTAS.html`. Acessado em: 23 fev. 2011., 2007.

[Ghisi, Lopes e Siqueira 2010]GHISI, B. C.; LOPES, G. F.; SIQUEIRA, F. Integração de Aplicações para TV Digital Interativa com Redes Sociais. *Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia 2010)*, 2010.

[IBGE 2009]IBGE. Pesquisa Nacional por Amostra de Domicílios 2009 - PNAD. Disponível em: `http://www.ibge.gov.br/home/presidencia/noticias/noticia_visualiza.php?id_noticia=1708`. Acessado em 14 fev. 2011, 2009.

[IBGE 2010]IBGE. Censo Brasileiro 2010. Disponível em: `http://www.censo2010.ibge.gov.br/dados_divulgados/index.php`. Acessado em: 25 fev. 2010., 2010.

[IDGNow 2010]IDGNOW. Ginga, completo, vira padrão UIT. Disponível em: `http://idgnow.uol.com.br/blog/circuito/2010/03/24/ginga-completo-vira-padrao-uit/`. Acessado em: 23 fev. 2011., 2010.

[Ierusalimschy 2006]IERUSALIMSCHY, R. *Programming in Lua*. 2nd. ed. Rio de Janeiro: Lua.org, 2006. 328 p. ISBN 8590379825.

[Ierusalimschy, Carregal e Guisasola 2004]IERUSALIMSCHY, R.; CARREGAL, A.; GUISASOLA, T. LuaSOAP - SOAP interface to the Lua programming language. Disponível em: http://www.keplerproject.org/luasoap/. Acessado em: 28 out. 2010., 2004.

[Ierusalimschy, Figueiredo e Celes 2007]IERUSALIMSCHY, R.; FIGUEIREDO, L. H. de; CELES, W. The evolution of Lua. In: *Proceedings of the third ACM SIGPLAN conference on History of programming languages*. [S.l.: s.n.], 2007.

[IETF 1996]IETF. MIME. Disponível em: http://tools.ietf.org/html/rfc2045. Acessado em: 22 jan. 2011., 1996.

[InfoExame 2009]INFOEXAME. Ginga-NCL é aprovado como padrão mundial para IPTV. Disponível em: http://info.abril.com.br/professional/network/ginganc l-e-aprovado-como-padrao-mundial-para-iptv.shtml. Acessado em: 23 fev. 2011., 2009.

[InterSystems 2010]INTERSYSTEMS. TrackCare Solution Guide. Disponível em: http://www.intersystems.com/trakcare/solution/section-04.html. Acessado em: 09 mar. 2010., 2010.

[ITU-T 2009]ITU-T. J.201: CABLE NETWORKS AND TRANSMISSION OF TELEVISION, SOUND PROGRAMME AND OTHER MULTIMEDIA SIGNALS. Application for Interactive Digital Television - Harmonization of declarative content format for interactive television applications. Disponível em: http://www.itu.int/itu-t/recommendations/rec.aspx?id=9583. Acessado em: 25 fev. 2011., 2009.

[ITU-T 2010]ITU-T. J.200: CABLE NETWORKS AND TRANSMISSION OF TELEVISION, SOUND PROGRAMME AND OTHER MULTIMEDIA SIGNALS. Application for Interactive Digital Television - Worldwide common core – Application environment for digital interactive television services. Disponível em: http://www.itu.int/itu-t/recommendations/rec.aspx?id=10827. Acessado em: 25 fev. 2011., 2010.

[ITU-T 2010]ITU-T. J.202: CABLE NETWORKS AND TRANSMISSION OF TELEVISION, SOUND PROGRAMME AND OTHER MULTIMEDIA SIGNALS. Application for Interactive Digital Television - Harmonization of procedural content formats for interactive TV applications. Disponível em: http://www.itu.int/itu-t/recommendations/rec.aspx?id=8667. Acessado em: 25 fev. 2011., 2010.

[Johnson e Foote 1988]JOHNSON, R.; FOOTE, B. Designing reusable classes. *Journal of object-oriented programming*, Citeseer, v. 1, n. 2, p. 22–35, 1988.

[Jr e Waclawek 2007]JR, F. L. D.; WACLAWEK, K. The Expat XML Parser. Disponível em: http://www.libexpat.org. Acessado em: 04 fev. 2011., 2007.

[Júnior e Gondim 2009]JÚNIOR, P. J. S.; GONDIM, P. R. L. LUACOMP: ferramenta de autoria de aplicações para tv digital. Dissertação de Mestrado. *Universidade de Brasília, Brasil.*, 2009.

[KOPECKÝ e Simperl 2008]KOPECKÝ, J.; SIMPERL, E. Semantic web service offer discovery for e-commerce. In: *Proceedings of the 10th international conference on Electronic commerce*. [S.l.: s.n.], 2008.

[Lausen e Haselwanter 2007]LAUSEN, H.; HASELWANTER, T. Finding web services. In: *1st European Semantic Technology Conference*. [S.l.: s.n.], 2007.

[Louridas 2006]LOURIDAS, P. Soap and web services. *Software, IEEE*, IEEE, v. 23, n. 6, p. 62–67, 2006. ISSN 0740-7459.

[Monteiro et al. 2010]MONTEIRO, C. de C. et al. Implementação de um Set-top Box Virtual para Desenvolvimento e Testes de Aplicações para TV Digital Interativa. *International Information and Telecommunication and Technologies Conference (I2TS 2010)*, 2010.

[Newcomer 2002]NEWCOMER, E. Understanding Web Services: XML, WSDL, SOAP and UDDI. Addison-Wesley Professional. 1st ed., 2002.

[OASIS 2006]OASIS. OASIS Web Services Security (WSS) TC. Disponível em: `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss`. Acessado em: 04 mar. 2010., 2006.

[OASIS 2007]OASIS. Web Services Business Process Execution Language Version 2.0. Disponível em: `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf`. Acessado em 02 mar. 2011., 2007.

[Point 2010]POINT, T. SOAP Tutorial. Disponível em: `http://www.tutorialspoint.com/soap/`. Acessado em: 09 mar. 2010., 2010.

[PUC-Rio 2010]PUC-RIO. Comunidade Ginga-NCL. Disponível em: `http://www.softwarepublico.gov.br/dotlrn/clubs/ginga/`. Acessado em: 28 nov. 2010., 2010.

[Sant'Anna 2010]SANT'ANNA, F. Documentação NCLua. Disponível em: http://www.lua.inf.puc-rio.br/˜ francisco/nclua/. Acessado em: 10 fev. 2010., 2010.

[Sant'Anna, Cerqueira e Soares 2008]SANT'ANNA, F.; CERQUEIRA, R.; SOARES, L. F. G. NCLua: objetos imperativos lua na linguagem declarativa NCL. In: ACM. *Proceedings of the 14th Brazilian Symposium on Multimedia and the Web*. [S.l.], 2008. p. 83–90.

[SBTVD 2010]SBTVD, F. O que é o ISDB-TB. Disponível em: `http://www.forumsbtvd.org.br/materias.asp?id=20`. Acessado em: 25 fev. 2011., 2010.

[Soares e Barbosa 2009]SOARES, L. F. G.; BARBOSA, S. D. J. *Programando em NCL 3.0: Desenvolvimento de aplicações para o middleware Ginga*. [S.l.]: Campus, Rio de Janeiro, 1a. edição., 2009.

[Soares, Rodrigues e Moreno 2007]SOARES, L. F. G.; RODRIGUES, R. F.; MORENO, M. F. Ginga-NCL: the declarative environment of the Brazilian digital TV system. *Journal of the Brazilian Computer Society*, SciELO Brasil, v. 12, p. 37–46, 2007. ISSN 0104-6500.

[Sommerville 2011]SOMMERVILLE, I. *Software Engineering*. 9th. ed. [S.l.]: Pearson, 2011.

[Síntese 2010]SíNTESE, T. Extra inicia vendas por TV digital. Disponível em: `http://goo.gl/DuMzD`. Acessado em: 23 fev. 2011., 2010.

[Teixeira 2006]TEIXEIRA, L. H. de P. Usabilidade e Entretenimento na TV Digital Interativa. *UNIrevista*, v. 1, n. 3, 2006. ISSN 1809-4651.

[TeleTime 2010]TELETIME. Totvs lança plataforma de aplicativos para TV digital. Disponível em: `http://www.teletime.com.br/23/08/2010/totvs-lanca-plataforma-de-aplicativos-para-tv-digital/tt/196493/news.aspx`. Acessado em: 14 mar. 2011., 2010.

[Treiber e Dustdar 2007]TREIBER, M.; DUSTDAR, S. Active web service registries. *IEEE Internet Computing*, IEEE Computer Society, p. 66–71, 2007.

[Veijalainen, Terziyan e Tirri 2006]VEIJALAINEN, J.; TERZIYAN, V.; TIRRI, H. Transaction management for m-commerce at a mobile terminal. *Electronic Commerce Research and Applications*, Elsevier, v. 5, n. 3, p. 229–245, 2006. ISSN 1567-4223.

[Vilas et al. 2007]VILAS, A. F. et al. Providing Web Services over DVB-H: Mobile Virtual Web Services. *IEEE Transactions on Consumer Electronics*, IEEE; 1999, v. 53, n. 2, p. 644, 2007.

[W3C 2007]W3C. SOAP Specification. Disponível em: `http://www.w3.org/TR/soap/`. Acessado em: 04 out. 2009., 2007.

[W3C 2009]W3C. Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. Disponível em: `http://www.w3.org/TR/2009/CR-CSS2-20090908/`. Acessado em: 29 nov. 2010., 2009.

[Welling 2006]WELLING, M. Integration of Interactive Applications in Digital Television. Monografia de Bacharelado. *EVTEK University of Applied Sciences - Institute of Technology. Finlândia*, 2006.

Apêndice A

Screenshots da aplicação de *T-Commerce*

Figura 6.1: Aplicação de *T-Commerce*: Tela inicial com produtos em destaque

Figura 6.2: Aplicação de *T-Commerce*: Busca de produtos

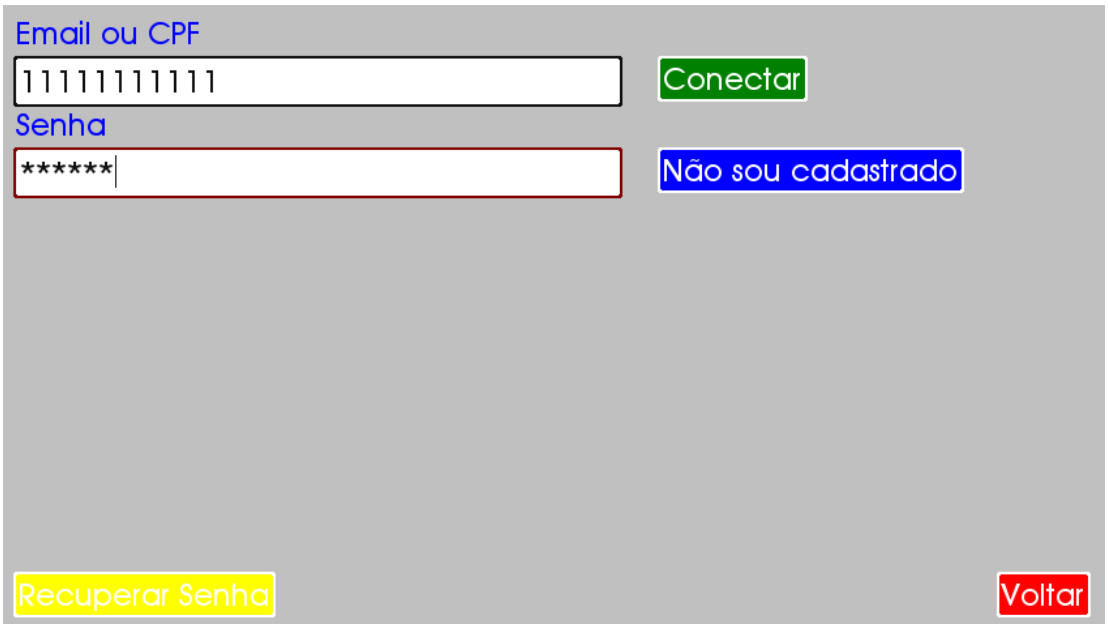


Figura 6.3: Aplicação de *T-Commerce*: Login (utilizando *e-mail* ou CPF)

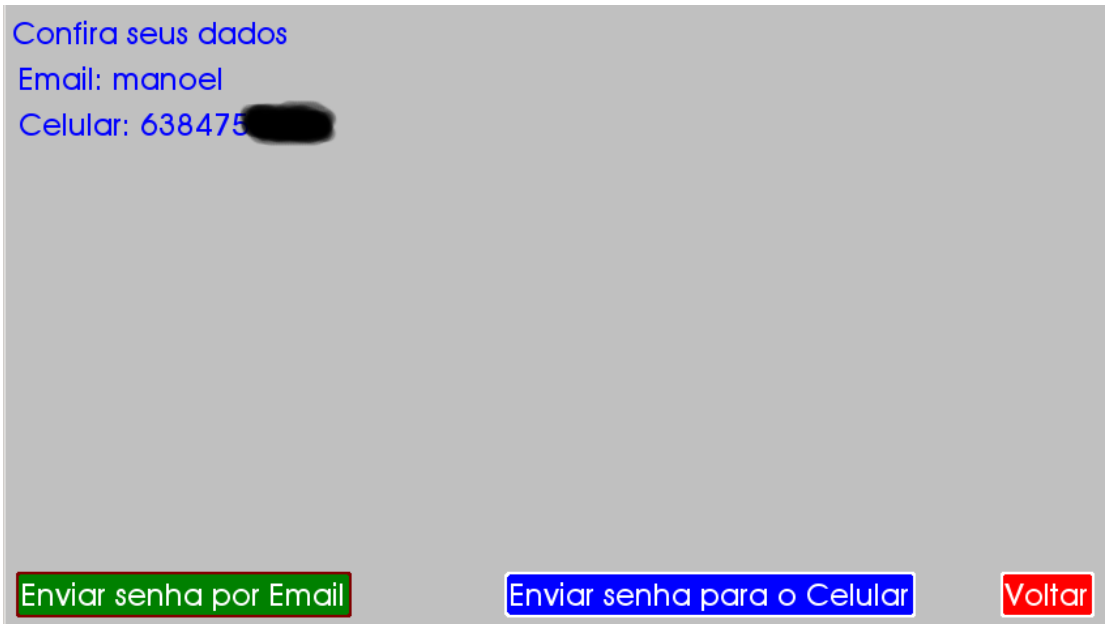


Figura 6.4: Aplicação de *T-Commerce*: Recuperar senha por *e-mail* ou SMS

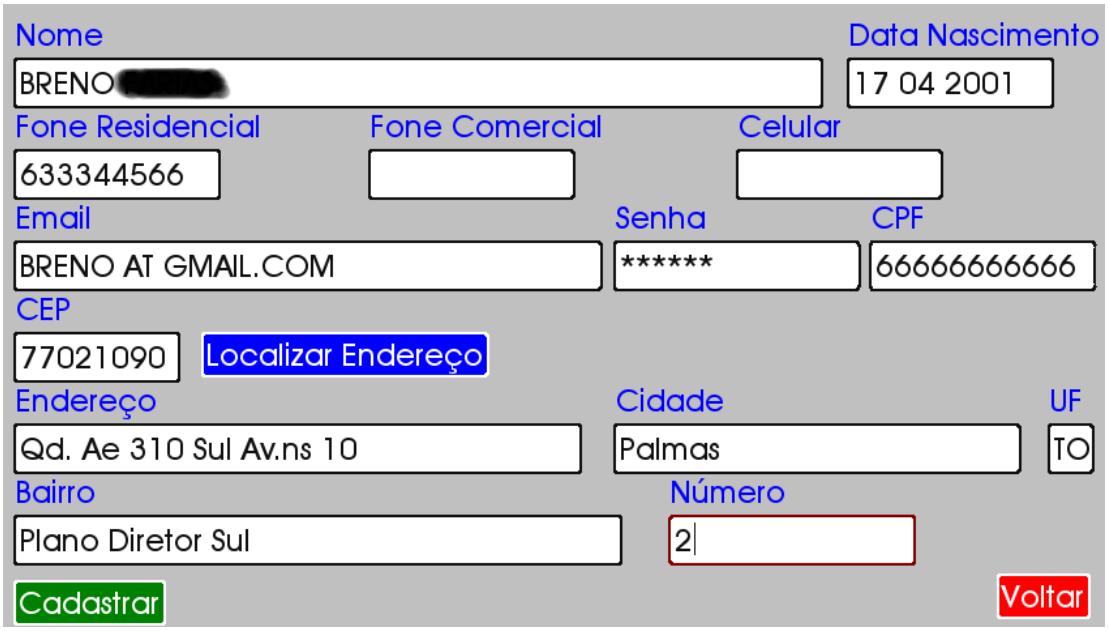


Figura 6.5: Aplicação de *T-Commerce*: Cadastro de Clientes (com busca de endereço a partir do CEP)

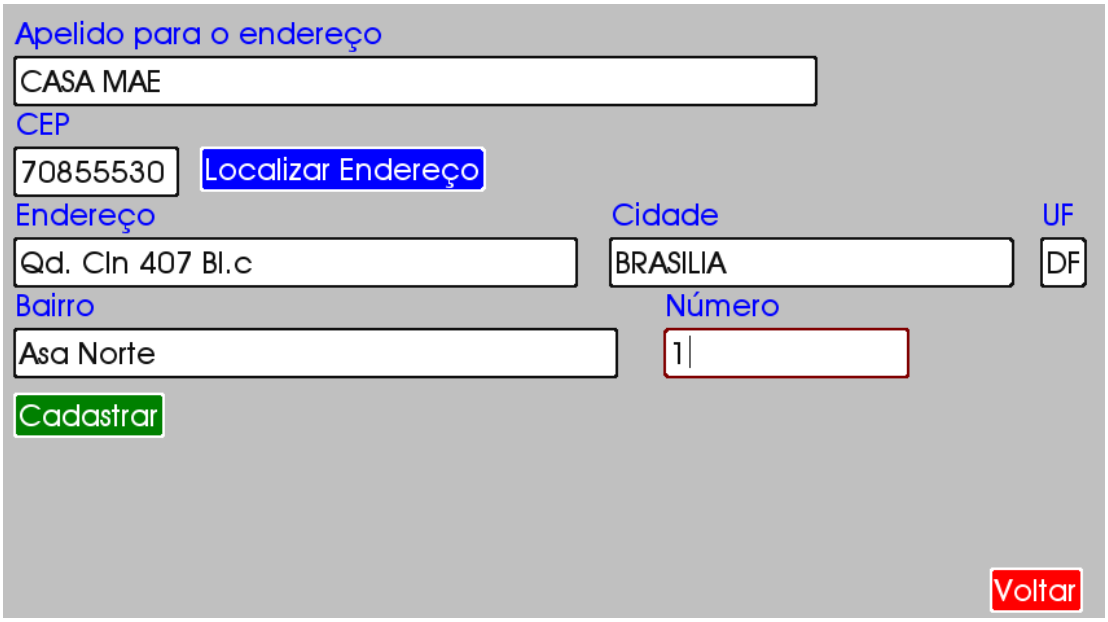


Figura 6.6: Aplicação de *T-Commerce*: Cadastro de Endereços (com busca de endereço a partir do CEP)

Figura 6.7: Aplicação de *T-Commerce*: Seleção de Endereços

Figura 6.8: Aplicação de *T-Commerce*: Formas de Pagamento

Figura 6.9: Aplicação de *T-Commerce*: Finalizar Compra