Ariadne Arrais Cruz

Projeto e Implementação de um Framework para WebLabs Baseado em Ajax e Padrões de Projeto

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Telecomunicações e Telemática.

Orientador: Dalton S. Arantes

Campinas, SP
Junho, 2007

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

C889p

Cruz, Ariadne Arrais
Projeto e Implementação de um *Framework* para *WebLabs* Baseado em Ajax e Padrões de Projeto
Ariadne Arrais Cruz. – Campinas, SP:
[s.n.], 2007.

Orientador: Dalton Soares Arantes.
Dissertação (Mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Framework (Programa de Computador). 2. Serviços na Web. I. Arantes, Dalton. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título

Título em Inglês:	Design and Implementation of a WebLabs Framework Based on Ajax and Design Patterns.
Palavras-chave em Inglês:	Framework (Computer Program), Web Services.
Área de concentração:	Telecomunicações e Telemática.
Titulação:	Mestre em Engenharia Elétrica.
Banca Examinadora:	Antônio Marcos Alberti, Eliane Gomes Guimarães, Leonardo de Souza Mendes e Max Henrique Machado Costa.
Data da defesa:	29/06/2007.

Ariadne Arrais Cruz

Projeto e Implementação de um Framework para WebLabs Baseado em Ajax e Padrões de Projeto

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Telecomunicações e Telemática.
Aprovação em 29/06/2007.

Banca Examinadora:
Prof. Dr. Dalton Soares Arantes (Orientador) - UNICAMP
Prof. Dr. Antônio Marcos Alberti - INATEL
Dra. Eliane Gomes Guimarães - CenPRA
Prof. Dr. Max Henrique Machado Costa - UNICAMP
Prof. Dr. Leonardo de Souza Mendes - UNICAMP

Campinas, SP
Junho, 2007

Resumo

O objetivo deste trabalho é apresentar um *framework* genérico e extensível para integração de *WebLabs* e outros Serviços via *Web*. Esta integração visa promover uma colaboração acadêmica em diferentes áreas da ciência, oferecendo aos desenvolvedores uma infra-estrutura básica de serviços, tais como uma comunicação cliente/servidor, serviços de persistência e facilidades para o desenvolvimento de interfaces de usuários. Infra-estruturas como esta viabilizam o compartilhamento de experimentos reais e remotos, explorando o novo paradigma de *WebLab*. O *framework* proposto é baseado em uma arquitetura modular e utiliza o conceito de *plugins* pré-instalados. Esta abordagem é bastante flexível, permitindo aos administradores do *framework* adicionar, remover ou atualizar seus serviços *WebLabs* sem a necessidade de recompilar o núcleo da plataforma ou de interromper os serviços. Isso garante um maior compartilhamento de código e facilidade de manutenção. O *framework* foi implementado utilizando metodologias do estado-da-arte, incluindo técnicas de programação avançadas como os Padrões de Projeto e o paradigma Ajax.

Palavras-chave: WebLabs, Framework Colaborativo, Ajax e Padrões de Projeto.

Abstract

The objective of this work is to present a generic and extensible Framework for WebLab integration. Such integration aims at fostering academic collaboration in different areas of science, since it offers a basic service infrastructure to developers, such as client/server communication, persistency service and facilities for the development of user-friendly interfaces. An infrastructure such as this one is of great importance for scientists to share virtual and real remote experiments through the web. The proposed Framework is based on a modular architecture that uses the concept of preinstalled plugins. This approach is quite flexible, since it allows the Framework administrator to add, remove or upgrade the WebLab services without recompilation of the platform kernel or disruption of services. This guarantees a better code-sharing and an easier Framework maintenance. The Framework was implemented using state-of-the-art methodology and other advanced programming techniques, such as Design Patterns and Ajax.

Keywords: WebLabs, Collaborative Framework, Ajax, Design Pattern.

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidata: Ariadne Arrais Cruz

Data da Defesa: 29 de junho de 2007

Título da Tese: "Projeto e Implementação de um Framework para Weblabs Baseado em Ajax e Padrões de Projeto"

Prof. Dr. Dalton Soares Arantes (Presidente): _____

Dr. Antônio Marcos Alberti: _____

Profa. Dra. Eliane Gomes Guimarães: _____

Prof. Dr. Max Henrique Machado Costa: _____

Prof. Dr. Leonardo de Souza Mendes: _____

Agradecimentos

A Deus, em primeiro lugar, pela vida, pela saúde, pela minha família maravilhosa, pela oportunidade de realizar este trabalho, pela sua presença constante em minha vida, sem que eu precise pedir, pelo auxílio nas minhas escolhas e por me confortar nas horas difíceis.

Ao meu orientador Prof. Dr. Dalton Arantes, pelas sugestões e ensinamentos, pela confiança e seriedade com que conduziu todo o período de orientação.

Ao meu "co-orientador" e amigo Fábio Lisboa, por acreditar em mim e neste trabalho. Pelo acompanhamento, companheirismo e interesse sempre demonstrados durante o meu mestrado. Agradeço, também, pelas correções, comentários e expressiva contribuição que só enriqueceram este trabalho. Muito Obrigada!

Agradeço, em especial, aos meus pais, Albino e Graça, pelo amor que foi meu alicerce. Obrigada por estarem sempre ao meu lado, mesmo quando estavam fisicamente distantes, me apoiando, me incentivando e nunca medindo esforços para que eu pudesse alcançar todos os meus objetivos. Obrigada por terem acreditado em mim e na minha escolha profissional. A educação de vocês foi fundamental para que eu chegasse até aqui. Agradeço a vocês, por tudo que sei, tudo que sou e tudo que faço. Amo vocês!

Aos meus irmãos, Albyno e Brenda, a quem devo mais do que agradecimentos. Obrigada pela atenção e preocupação. Obrigada por estarem sempre ao meu lado e por diversas vezes entenderem a ausência dos nossos pais do nosso lar. Muito obrigada!

À toda a minha família, em especial a minha Vó Margarida e a tia Fafá, que sempre souberam compreender a minha ausência nas férias, feriados e almoços de domingo.

Aos meus amigos pelo apoio e momentos de alegria! Especialmente aos amigos conquistados durante o mestrado, Alexandre, André, Barbosa (Babs), Fábio, Fabbryccio, Gina, Jahyr, Lígia (chatinha), Lívio, Márzio (primo), Mota, Tarciana, Vanessa, Veruska e Zé pelas festas "surpresas", pelos bate-papos, pelos almoços. Enfim... pelo companheirismo, pela sólida amizade que construímos e por terem se tornado parte da minha família.

Aos meus colegas do *ComLab* pelas ajudas constantes. Muito obrigada!

À Simone pelas revisões no texto e sugestões.

Ao meu amigo Márzio Geandre pelas sugestões e palavras amigas.

Ao Prof. Eleri Cardozo e à Dra. Eliane Guimarães pelo estímulo e apoio à equipe do *ComLab* - Laboratório KyaTera em Comunicações Digitais.

A todos os funcionários da Faculdade de Engenharia Elétrica e Computação da Unicamp.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo apoio financeiro.

À Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) e à Financiadora de Estudos e Projetos (FINEP) pelo apoio ao *ComLab*.

A todos aqueles que de alguma forma contribuíram para a conclusão deste trabalho. Obrigada!

Aos meus pais, que com amor e dedicação me ajudaram a realizar os meus sonhos.

Sumário

Lista de Figuras

Lista de Tabelas

Glossário

Ajax - *Asynchronous JavaScript And XML*

AOP - *Aspect Oriented Programming*

API - *Application Programming Interface*

BJS - Backbase Java *Server*

BXML - Backbase XML

CF - *ComLab Framework*

CGI - *Common Gateway Interface*

CIL - *ComLab Intermediate Language*

CRUD - *Create, Read, Update and Delete*

CSS - *Cascading Style Sheets*

CVM - *ComLab Virtual Machine*

DAO - *Data Access Object*

DIP - *Dependency Inversion Principle*

DOM - *Document Object Model*

DTML - *Document Template Markup Language*

EJB - *Enterprise JavaBeans*

EO - *Extension Object*

FAPESP - Fundação de Amparo à Pesquisa do Estado de São Paulo

FEEC - Faculdade de Engenharia Elétrica e Computação

GUI - *Graphical User Interface*

HTML - *HyperText Markup Language*

HTTP - *HyperText Transfer Protocol*

I-Toolarr - *Integration of Tools for Accessing Resource Remotely*

IDE - *Integrated Development Environment*

IoC - *Inversion of Control*

J2EE - Java 2 *Enterprise Edition*

J2ME - Java 2 *Micro Edition*

J2SE - Java 2 *Standard Edition*

JDBC - Java *Database Connectivity*

JSF - *JavaServer Faces*

JSP - *JavaServer Pages*

MV - *Model-View*

MVC - *Model-View-Controller*

OODB - Banco de Dados Orientado a Objetos

PHP - PHP: *Hypertext Preprocessor*

RPC - *Remote Procedure Call*

RV - Realidade Virtual

TIDIA - Tecnologia da Informação no Desenvolvimento da Internet Avançada

Tidia-Ae - Tidia - *Aprendizado Eletrônico*

UI - *User Interface*

UML - *Unified Modeling Language*

VI - *Virtual Instrument*

XHTML - *Extensible HyperText Markup Language*

XML - *Extensible Markup Language*

XSLT - *Extensible Stylesheet Language Transformations*

ZODB - Z *Object DataBase*

Trabalhos Publicados Pela Autora

1. Ariadne A. Cruz, Fábio A. L. Gomes, Fabbryccio A. C. M. Cardoso, Ernesto B. Martin, Dalton S. Arantes, "Development of a Robust and Flexible Weblab Framework based on AJAX and Design Patterns", *Proceedings of 7th IEEE International Symposium on Cluster Computing and the Grid* (CCGrid'07), Rio de Janeiro, RJ, Brasil, pg. 811-816, Maio 2007.

Capítulo 1

Introdução

Este capítulo apresenta algumas motivações para o desenvolvimento de um *framework* flexível e robusto, baseado em Ajax (*Asynchronous JavaScript And XML*) e Padrões de Projeto, para a integração de *WebLabs*. Entre os benefícios obtidos com este *framework*, destacam-se uma maior usabilidade, um maior reuso de código e facilidade de integração de um novo *WebLab* à plataforma, sem a necessidade de recompilar o *framework*. Posteriormente, são discutidos os objetivos, as contribuições e alguns trabalhos relacionados.

1.1 Visão Geral

A demanda por aplicações e serviços baseados em *Web* [6][7] encontra-se em contínua expansão. Um *WebLab* é um exemplo típico de um serviço para *Web*, cujo objetivo é prover acesso remoto a experimentos de laboratório, ou seja, esses serviços oferecem acesso comum a dados geograficamente distribuídos, permitindo que os usuários possam utilizar todos os recursos de um equipamento remoto, sem a necessidade de investir grande quantidade de recursos na aquisição de equipamentos laboratoriais.

O desenvolvimento de uma plataforma para a integração de *WebLabs*, baseada em *frameworks*, provê uma implementação rápida e de baixo custo, resultando em uma estrutura básica eficiente para a integração de equipamentos remotos, sensores e qualquer tipo de dispositivo eletrônico. Os *frameworks* têm como objetivo otimizar a generalização e a especificação e diminuir, consideravelmente, o número de linhas de código para a implementação de aplicações similares. Agilidade no desenvolvimento é um dos grandes benefícios obtidos ao se desenvolver um *framework*. Além disso, ele também provê um maior reuso de componentes de *software*, que podem ou não ser implementados por diferentes desenvolvedores, e um maior atendimento aos requisitos não funcionais[1], permitindo ao desenvolvedor se dedicar mais aos requisitos funcionais[2] do modelo de negócio[3].

[1] São requisitos que especificam critérios que podem ser utilizados para avaliar a operação de um sistema, ao invés de especificar o seu funcionamento.

[2] São requisitos que especificam determinados comportamentos ou funções.

[3] É uma ferramenta conceitual que contém um conjunto de objetos, conceitos e seus relacionamentos com o objetivo

As vantagens da utilização de um *framework* são inúmeras, tais como a modularidade, reusabilidade, adaptabilidade, utilização de componentes otimizados e testados, facilidade de depuração, produção de código mais estável e eficiente, além de rapidez no desenvolvimento de aplicações. Em relação à aplicações de grande porte, um *framework* já provê ganhos imediatos quanto a modularidade e facilidade no desenvolvimento em equipe.

Este trabalho é parte de um projeto ambicioso no Estado de São Paulo, visando o desenvolvimento de uma rede acadêmica avançada de altíssima velocidade para pesquisa, educação e desenvolvimento. Este projeto é financiado pela FAPESP, Fundação de Amparo à Pesquisa do Estado de São Paulo, sendo descrito em mais detalhes na seção seguinte (Seção 1.2). Esta dissertação foi desenvolvida no âmbito do *ComLab* - Laboratório *KyaTera* em Comunicações Digitais, junto ao Departamento de Comunicações da FEEC (Faculdade de Engenharia Elétrica e Computação) - UNICAMP, que participa no desenvolvimento dessa rede.

Atualmente, há um grande número de propostas para *WebLabs* [8][9][10][11][12][13] na rede *KyaTera* da FAPESP [14], e o *framework* discutido neste trabalho visa uma integração eficiente desses *WebLabs*. O acesso comum a recursos geograficamente distribuídos é de fundamental importância para contribuir com o crescimento de trabalhos colaborativos em áreas estratégicas da ciência. O objetivo deste trabalho é desenvolver um *framework* para prover uma colaboração acadêmica em diferentes disciplinas e oferecer aos desenvolvedores uma infra-estrutura básica de serviços, tais como comunicação cliente/servidor e serviços persistentes, além de facilitar o desenvolvimento das interfaces gráficas do usuário.

O *framework* aqui apresentado é baseado em metodologias do estado-da-arte e tecnologias avançadas, tais como Padrões de Projeto e paradigma Ajax [15][16][17]. O Padrão de Projeto é uma solução para um problema em um contexto [18], ou seja, oferece soluções prontas, para serem utilizadas em problemas específicos que podem surgir durante o desenvolvimento de um *software* orientado a objetos. Entre os Padrões de Projeto utilizados no desenvolvimento deste trabalho, os padrões DIP (*Dependency Inversion Principle*)[19] e EO (*Extension Object*)[20] são os responsáveis por criar uma arquitetura de *plugins*, como detalhado na Seção 2.2.

1.2 Projeto TIDIA-*KyaTera*

O programa TIDIA (Tecnologia da Informação no Desenvolvimento da Internet Avançada) foi lançado em São Paulo, Brasil, no segundo semestre de 2003. Esse programa é financiado pela FAPESP, que também é responsável pelo gerenciamento dos endereços da Internet no Brasil [21]. Ele inclui três programas colaborativos: *KyaTera*, *Aprendizado Eletrônico* e *Incubadora Virtual*. O projeto *KyaTera* visa uma infraestrutura de rede e o desenvolvimento de *WebLabs*, enquanto que o *Aprendizado Eletrônico* e a *Incubadora Virtual* são responsáveis por projetos de desenvolvimento

de expressar a lógica do negócio.

colaborativo.

De acordo com a FAPESP [14], o projeto *KyaTera* é um projeto cooperativo consistindo de uma rede de fibras ópticas projetada para a pesquisa e desenvolvimento de conexões de alta velocidade, integrando laboratórios de pesquisa que focam o estudo, desenvolvimento e demonstração de tecnologia e aplicações na internet avançada. O projeto consiste de um *testbed*[4] usando tecnologia de multiplexação em comprimento de onda.

A rede *KyaTera* é uma rede óptica de alta velocidade, baseada no conceito de fibras apagadas, que chegam diretamente até os laboratórios. Em poucos anos, esta rede deverá evoluir para uma rede estável conectada ao mundo acadêmico na Internet, provendo alta qualidade de serviço, sem limitação da largura de banda. A rede *KyaTera* estará disponível para educação e ensino e para pesquisa e desenvolvimento no Estado de São Paulo.

1.3 Objetivos e Motivação

O objetivo deste trabalho consiste em projetar e implementar um *framework* colaborativo virtual, tornando possível utilizar uma única linguagem (Java) para a implementação tanto no lado cliente quanto no lado servidor. Esta característica viabiliza a reutilização de um mesmo código em ambos os lados, cliente e servidor. Isto é uma grande vantagem, quando comparado a técnicas clássicas de desenvolvimento *Web* baseados em Java, que utilizam JSP (*JavaServer Pages*), Java, *JavaScript*, HTML (*HyperText Markup Language*), onde o reaproveitamento de código é mínimo, exigindo uma manutenabilidade mais complexa. Nas Figuras 1.1 e 1.2 é mostrado um comparativo entre as técnicas de desenvolvimento necessárias para a implementação de uma plataforma *Web* baseada em Java e a implementação da plataforma proposta.

Além disso, este trabalho também tem como objetivo criar uma *framework* para *Web* totalmente baseado no conceito de *plugins*, semelhante à idéia do ambiente de desenvolvimento Eclise que será descrito na Seção 1.6.1. A idéia é poder acoplar diferentes tipos de *plugins* (serviços) ao *framework*, sem que para isso haja a necessidade de interromper a aplicação, ou seja, recompilar a plataforma inteira toda vez que um *plugin* for adicionado. Entretanto, a atualização só terá efeito após a reinicialização da plataforma, ou seja, após o usuário clicar no botão de atualização (*refresh*) do navegador ou na reinicialização do Servidor de Aplicações. Desta forma, nem a plataforma e nem os serviços precisam ser interrompidos quando ocorrer uma atualização. O desenvolvedor não perderá tempo recompilando todo o sistema e nem precisará ter um conhecimento sobre como a plataforma inteira foi implementada para realizar uma alteração nos serviços. Outra vantagem está no fato de que ao recompilar um sistema, especialmente de grande porte, alterações erradas podem levar a eventos imprevisíveis. O *framework* também oferece ganhos significativos em relação a sua interface, pela utilização do paradigma Ajax, que utiliza técnicas já conhecidas, como *JavaScript* e XML (*Extensible Markup Language*). O Ajax utiliza essas tecnologias de maneira eficiente, provendo interfaces ricas

[4] É uma rede experimental que se concentrará em desenvolver projetos de fotônica e engenharia de rede [14].

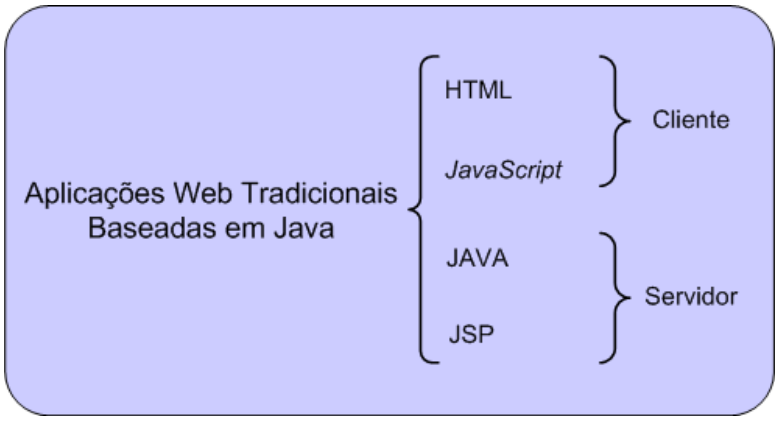


Fig. 1.1: Desenvolvimento *Web* Tradicional Baseado em Java.

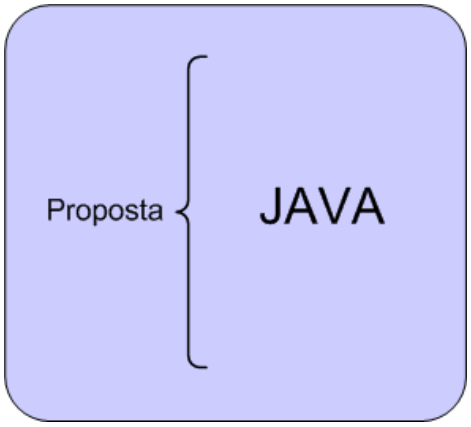


Fig. 1.2: Desenvolvimento *Web* Proposto.

e amigáveis, e que cada vez mais se parecem com aplicações *desktop*.

Atualmente, existem alguns *frameworks* para *Web* baseados em *plugins*, como o Joomla [22], Mambo [23] e Plone [24]. Os dois primeiros *frameworks* são baseados em PHP (PHP: *Hypertext Preprocessor*) e o último é baseado em Zope e escrito em Python, com o seu próprio servidor *Web* (Zserver) e banco de dados (ZODB - Z *Object DataBase*). Ao invés do PHP, Zope utiliza uma linguagem baseada em marcação, chamada DTML (*Document Template Markup Language*), que é bastante similar ao PHP, mas só é utilizada para a camada de apresentação dos dados. Todos esses *frameworks* apresentam IDEs (*Integrated Development Environment*) muito limitadas quando comparados aos existentes para Java, além de utilizar várias linguagens durante o seu desenvolvimento. Desta forma, concluiu-se que o desenvolvimento de uma ferramenta *Web* baseada em *plugins*, que possibilita a sua implementação em Java, provendo códigos de alta qualidade e com baixa propensão a erros, e que permite a utilização de ambientes de desenvolvimento maduros e com um grande número de ferramentas de auxílio a programadores, seria um excelente diferencial do trabalho proposto.

1.4 Premissas

A plataforma de integração de *WebLabs* descrita neste trabalho foi desenvolvida de acordo com os princípios de extensibilidade, robustez e usabilidade.

1.4.1 Extensibilidade

A diversidade de *WebLabs* ora em desenvolvimento no projeto *KyaTera* requer que o *framework* deve ser capaz de fornecer diferentes tipos de serviços para diferentes públicos. Para engenheiros, os *WebLabs* poderão disponibilizar acesso remoto a osciloscópios, analisadores espectrais, algoritmos

de compressão de vídeo, dentre muitas outras possibilidades. Para estatísticos, poderão ser fornecidos serviços de análise de dados, reconhecimento de padrões, classificação, entre outros. Como é impossível antecipar-se a todos os serviços que poderão ser fornecidos pelos *WebLabs*, a solução encontrada foi utilizar o Padrão de Projeto *Plugin* (combinação dos padrões DIP e EO), descrito mais detalhadamente nas Seções 2.2.2 e 2.2.3.

1.4.2 Robustez

Devido ao grande número de serviços que poderão ser providos pelos *WebLabs*, é importante garantir que a inclusão, remoção ou alteração de serviços não faça com que uma parte do sistema pare de funcionar, garantindo assim a robustez do mesmo, que é a habilidade de detectar falhas de modo que os danos possam ser mantidos em um patamar aceitável. Essa robustez é alcançada, novamente, com a utilização do padrão *Plugin*, que permite um bom nível de isolamento entre os serviços e também entre os serviços e o *framework*.

1.4.3 Usabilidade

Usabilidade é a rapidez e a facilidade com que as pessoas podem empregar uma ferramenta ou objeto a fim de realizar uma tarefa [25]. A usabilidade é aplicada a todos os aspectos do sistema com o qual o ser humano pode interagir. Segundo [26], a usabilidade é composta por Aprendizagem (um usuário pode rapidamente aprender a utilizar o sistema), Eficiência (alto nível de produtividade para usuários mais experientes), Memorização (facilidade de lembrar como funciona a interface nas utilizações seguintes), Erros (deve ter uma pequena taxa de erros, sendo que erros catastróficos nunca devem ocorrer) e Satisfação (o sistema deve ser agradável de usar). É muito importante que um *framework* desenvolvido para *Web* seja projetado seguindo esse princípio, visto que normalmente os usuários não têm tempo ou disposição para aprender a usar um aplicativo *Web*. Portanto, para que este aplicativo seja prático as pessoas têm que rapidamente entender como ele funciona.

Os aplicativos *Web* tradicionais utilizam uma comunicação síncrona entre o cliente e servidor. Neste paradigma, o cliente (navegador) envia uma solicitação ao servidor e fica aguardando a sua resposta. Enquanto a resposta não vem, a página *Web* fica bloqueada, impedindo o fluxo normal de trabalho do usuário. Além dessa deficiência, aplicativos *Web* tradicionais também possuem uma interface pobre, que limita a criação de aplicativos mais sofisticados e com melhor tempo de respostas às ações provocadas pelos usuários. Como solução para esses problemas, as interfaces do usuário foram construídas seguindo o paradigma Ajax (Seção 2.1), que proporciona interfaces mais amigáveis e com boa usabilidade, além de prover comunicação assíncrona entre cliente e servidor.

1.5 Contribuições do Trabalho Proposto

Dentro do contexto descrito nas seções anteriores, o propósito deste trabalho é apresentar uma plataforma de integração de *WebLabs* baseada em Padrões de Projeto e Ajax. A plataforma favorece a inclusão de novos serviços, sem a necessidade de recompilação do *framework*, além de disponibilizá-lo através de uma interface amigável.

A principal contribuição desse trabalho consiste no desenvolvimento de um *framework* para diferentes áreas da ciência, trazendo benefícios ao projeto *KyaTera*. O trabalho provê, também, flexibilização e reutilização de *software* através do uso de Padrões de Projeto. Como contribuições conseqüentes da metodologia, tem-se ganhos quanto as interfaces, que são mais sofisticadas, e que proporcionam melhor desempenho para as aplicações, já que a idéia básica do Ajax é explorar a assincronicidade, proporcionando ao cliente trabalhar de forma independente do servidor, otimizando-se, assim, a capacidade do lado cliente. Finalmente, considerando os poucos trabalhos na área que fazem uso do paradigma Ajax, tem-se um contribuição na melhora das respostas das aplicações *Web*, com menor custo e complexidade.

1.6 Trabalhos com Temas Relacionados

Os trabalhos correlatos aqui citados foram os que motivaram o desenvolvimento desta dissertação. Dentre eles temos a Plataforma Eclipse[27], o Framework ViRAL[28], o I-Toolarr (*Integration of Tools for Accessing Resource Remotely*)[29] o Laboratório Virtual REAL[30][31] e a Plataforma Google[5].

1.6.1 Plataforma Eclipse

A Plataforma Eclipse é projetada para a construção de ambientes de desenvolvimento integrado. Ela é construída sob um mecanismo para descobrir, integrar e executar módulos de *plugins*. Qualquer *plugin* é livre para definir novos pontos de extensão e prover novas APIs (*Application Programming Interface*) para outros *plugins*. Alguns *plugins* qualificados podem estender a funcionalidade de outros *plugins* tão bem quanto estender o *kernel*. Isto provê flexibilidade para criar configurações mais complexas.

Com a análise desta plataforma, foi observado que os padrões DIP e EO (Seções 2.2.2 e 2.2.3) seriam uma solução interessante para a implementação dos serviços, pois eles permitem a possibilidade de adicionar, remover e atualizar um serviço sem a necessidade de recompilar o núcleo do *framework*.

1.6.2 Framework ViRAL

O ViRAL é um *framework* baseado em *plugins* para o desenvolvimento de aplicações de Realidade Virtual (RV). A interface gráfica de uma aplicação ViRAL é dinâmica, construída em tempo de execução, e reflete os componentes ativos e *plugins*. Note que o conceito de *plugins* introduz novos sistemas e componentes para as aplicações e também permite que o desenvolvedor defina livremente novos tipos de aplicações para estender componentes e sistemas. O ViRAL tira vantagens dessa importante flexibilidade [28][32].

O ViRAL usa uma arquitetura *grey-box*[5] baseada em componentes. Uma vez programado, um componente deve ser usado em qualquer aplicação ViRAL pelo operador, procedimento este que permite a criação dinâmica de novas aplicações. O *framework* ViRAL é uma aplicação *desktop*, enquanto que a proposta deste trabalho é uma aplicação *Web* que não oferece apenas aplicações de RV, mas muitas outras. Apesar dessas grandes diferenças, ambos usam uma interface baseada em componentes.

1.6.3 I-Toolarr

A ferramenta I-Toolarr permite a integração de experimentos ao ambiente Tidia-Ae (Tidia - *Aprendizado Eletrônico*). Para que um *WebLab* possa integrar um experimento ao ambiente é necessário que o *WebLab* esteja registrado no sistema. Dessa forma, o administrador precisa preencher alguns dados no momento do cadastro do *WebLab*, tais como o nome do *WebLab*, a sua descrição, a URL (uma URL para acesso ao site do Laboratório, caso exista) ou uma foto [29][33].

Após feito o cadastro do *WebLab*, o administrador poderá cadastrar alguns recursos, como por exemplo, a presença de um professor no local do experimento durante a sua execução. Depois que o *WebLab* e os recursos foram cadastrados, o *WebLab* ainda não estará disponível. O administrador ainda precisará cadastrar um Toolarr, preenchendo dados como o nome da ferramenta e uma breve descrição, a URL de acesso *Web* à ferramenta, tempo mínimo e máximo da sessão, entre outros. Note que diferente do trabalho proposto, o I-Toolarr não inclui o conceito de *plugins*, e os *WebLabs* só podem ser integrados ao ambiente Tidia-Ae se forem incluídos através de URLs de acesso *Web*.

1.6.4 Laboratório Virtual REAL

O Laboratório Virtual REAL [30], que pode ser acessado via Internet, foi implementado como um serviço de telemática. O principal objetivo dessa proposta é prover aos pesquisadores e estudantes, localizados em qualquer lugar, acesso a um robô móvel XR4000, onde eles podem testar e validar os controles do robô e seus métodos de navegação.

Para usar a plataforma de desenvolvimento do Laboratório Virtual REAL, o usuário necessita de um ambiente computacional que suporte um navegador *Web* usando a Máquina Virtual Java 2 com

[5] Quando as alterações são feitas via configuração de parâmetros.

capacidade de executar *applets*. A infraestrutura de suporte aos serviços precisa ser instalada no ambiente do usuário, sempre que este decida utilizá-lo. No caso do trabalho proposto, como a proposta visa uma instalação mínima, a tecnologia Ajax foi escolhida para o desenvolvimentos das páginas, pois não há necessidade do usuário fazer *downloads* de *plugins* ou *runtimes* (por exemplo, Java, Flash, ActiveX, etc.). A abordagem da proposta deste trabalho permite ainda, a possibilidade de combinar a interação do usuário e as solicitações ao servidor de maneira mais integrada, permitindo que as suas interfaces sejam bem similares a programas *desktop* sem perder as vantagens de uma aplicação para *Web*.

1.6.5 Aplicativos Google

Depois que o Google iniciou o desenvolvimento de algumas novas aplicação usando Ajax, a atenção do público se voltou para esta nova tecnologia. Alguns dos maiores produtos do Google foram apresentados nos últimos anos usando essa tecnologia, como o Google Groups, Google Suggests, Google Maps e Google Mail [34].

A proposta deste trabalho é similar aos Aplicativos Google em muitos aspectos, especialmente pela interface amigável que favorece a usabilidade inerente à tecnologia Ajax. Com essa tecnologia as possibilidades para o desenvolvimento *Web* aumentam significativamente. Por exemplo, é possível uma interação maior com o usuário usando Ajax, desde que os agentes da interface sejam independentes das ações do usuário.

Em um dos aplicativos Google, a página de buscas personalizada, incorpora o conceito de *plugins*, que podem ser *feeds*[6] ou *gadgets*[7]. Ao incluir um *feed* basta informar a URL de acesso, já para incluir um *gadget* é necessário que alguns campos sejam preenchidos, como título, descrição, autor, email, etc. Esses dados são incluídos em um arquivo XML. Abaixo é mostrado o código fonte de um *gadget*.

```
<?xml version="1.0"encoding="UTF-8"?>
  <Module>
    <ModulePrefs title="MSN Messenger 2.0" description="Chat with
    your MSN buddies anywhere, anytime."  author="Whizz"
    author_email="grenardus@gmail.com" author_location="The
    Netherlands" author_link="http://whizzgadgets.googlepages.com"
    author_quote="I reject your reality and subsitute my own."
    thumbnail="http://img426.imageshack.us/img426/5339/
    msnthumbia5.png" screenshot="http://img426.imageshack.us/
    img426/5985/ msnhk7.png" title_url="http://whizzgadgets.
    googlepages.com/msn" height="400" />

    <Content type="html">
```

[6] É um padrão para denominar arquivos de conteúdo, ou seja, são listas de atualização de determinados sites.
[7] Proporcionam um site rápido e fácil de personalizar com uma interface de usuário limpa.

```
        <![CDATA[
        <iframe src="http://grenardus.googlepages.com/msnmessenger.htm" frameborder="0"
        scrolling="yes" width="280" height="400"/>
        ]]>
    </Content>
</Module>
```

Observe, neste exemplo, que dentro da *tag* `<Content type="html">`, está incluído um *iframe*, mas poderiam ser incluídos códigos *JavaScript*, HTML e XML. Entretanto, os *gadgets* e os *feeds* são totalmente independentes dos Aplicativos Google. Caso seja necessário incluir um serviço que precise de processamento no lado servidor, é necessário ter um Servidor de Aplicações. Desta forma, não há nenhuma garantia de que, quando uma nova funcionalidade é adiciona ao servidor, (que não é o Servidor do Google), a plataforma não seja recompilada. Contudo a plataforma do Google garante que quando a interface desse serviço é adicionada, seja ela uma URL ou código XML, JavaScript e HTML, a plataforma não é recompilada.

1.7 Organização da Dissertação

Este trabalho contém cinco capítulos e está estruturado da forma descrita a seguir. O Capítulo 2 apresenta as principais metodologias utilizadas para o desenvolvimento da plataforma. O Capítulo 3 apresenta a arquitetura do *framework*, onde é feita a descrição do projeto, especificando seus módulos, o detalhamento do modelo de dados e o diagrama de casos de uso, classes, seqüências e pacotes. O Capítulo 4 aborda as tecnologias utilizadas para a implementação dos serviços, e descreve alguns dos serviços que atualmente integram a plataforma. O Capítulo 5 conclui este trabalho tecendo considerações finais com base nos resultados obtidos durante o desenvolvimento e sugerindo possíveis extensões ao trabalho aqui apresentado.

Capítulo 2

Fundamentação Teórica

Neste capítulo apresentamos uma descrição sobre as principais tecnologias usadas no desenvolvimento do trabalho proposto e as suas vantagens em relação ao modo de desenvolvimento *Web* tradicional. Além disso, são detalhados os *frameworks* Ajax estudados durante o desenvolvimento deste trabalho, os principais Padrões de Projeto utilizados, o *framework* Spring e a linguagem intermediária, que foram utilizados para a criação de uma plataforma baseada em *plugins*.

2.1 *Asynchronous JavaScript And XML* (Ajax)

O Ajax não é uma nova tecnologia, ele incorpora uma apresentação baseada em padrões usando XHTML (*Extensible HyperText Markup Language*) e CSS (*Cascading Style Sheets*), interface dinâmica e interativa usando DOM (*Document Object Model*), troca e manipulação de dados usando XML e XSLT (*Extensible Stylesheet Language Transformations*), busca de dados assíncrona utilizando XMLHttpRequest e *JavaScript* ligando tudo isso [15][16][35].

A comunicação entre o lado cliente (navegador) e o servidor (PHP, Java, Perl, etc.) sempre foi um problema nas aplicações *Web*, pois esta comunicação se estabelece de forma síncrona. Ou seja, a cada solicitação a página inteira é atualizada, mesmo que a atualização seja apenas de uma pequena parte da página. Contudo, algumas estratégias eram usadas para contornar o problema da comunicação, mas elas nunca conseguiram resolver o problema de maneira adequada. Os dados eram apenas escondidos e mostrados conforme fossem necessários, dando a falsa impressão de assincronicidade.

O Ajax surgiu para enfatizar a comunicação assíncrona entre o cliente e o servidor. Ele garante que o cliente poderá utilizar a aplicação de forma contínua enquanto os dados são enviados ou recebidos em segundo plano. Somente os dados realmente necessários são baixados para a máquina cliente, o que propicia uma interação melhor e mais rápida. Na Figura 2.1, é feita uma comparação entre o modelo tradicional de aplicações *Web* e o modelo Ajax [36][37]. A principal diferença entre os dois modelos é a presença do mecanismo Ajax que é responsável pela serialização[1] dos dados,

[1] É o processo de transmitir um objeto através de uma conexão de rede, no formato binário ou no formato de texto, como arquivo XML.

assincronicidade e o estabelecimento da conexão entre o cliente e o servidor.

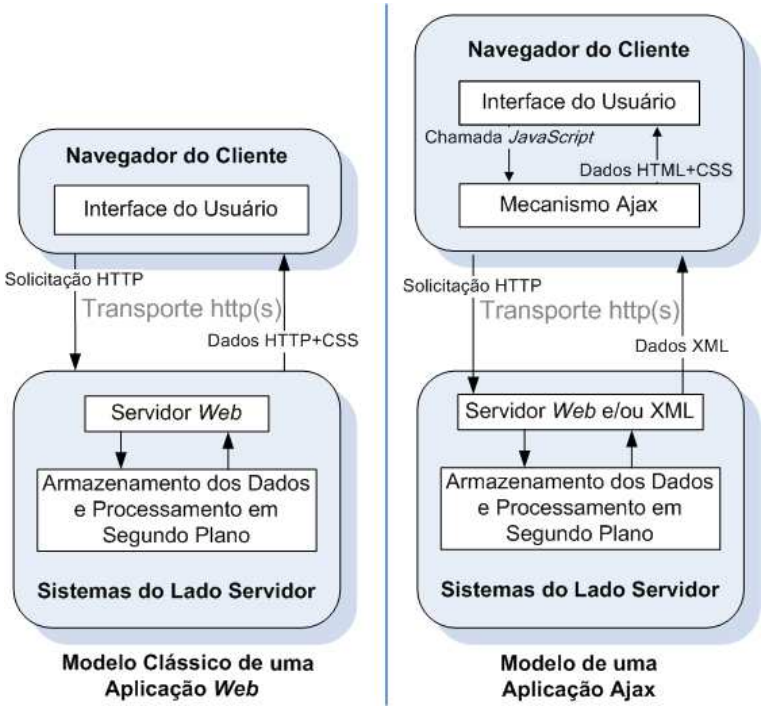


Fig. 2.1: Comparação entre o modelo tradicional de aplicação *Web* e o modelo Ajax [1].

No momento em que o usuário entra em um site que utiliza o paradigma Ajax, o tráfego é alto, já que é entregue ao cliente uma infra-estrutura relativamente grande e complexa em rajada. Porém, as comunicações posteriores com o servidor são muito mais eficientes. Além disso, à medida que o tempo de interação com a aplicação aumenta, o consumo de largura de banda é relativamente menor quando comparado a uma aplicação *Web* tradicional [15].

Este paradigma possui uma grande interatividade, baixo consumo da largura de banda, uma alta portabilidade, além de não ser solução proprietária. Dessa forma, ele não se restringe a utilização de um navegador ou plataforma específicos, e não requer a instalação de qualquer *plugin* no navegador ou *software* cliente. Antes mesmo do termo Ajax ser conhecido, aplicações *Web* como Google *Maps* e Google *Suggest* evidenciaram toda a força deste novo paradigma.

Os maiores problemas na utilização do Ajax são a complexidade de se programar em *JavaScript*, a depuração do código e a incompatibilidade do código com os diversos tipos de navegadores existentes. Felizmente, há várias maneiras de contornar esses problemas, como a utilização de *frameworks* (descritos nas seções seguintes) e bibliotecas de funções.

Atualmente, os usuários buscam aplicações *Web* com mais interatividade, que sejam mais agradáveis e ágeis. Normalmente, essas aplicações se adaptam ao modo de vida de usuário, o qual pode escolher que informações devem aparecer na sua página principal, como é o caso da página de busca

personalizada do Google (Figura 2.2). Em meados de 2008, novas versões dos navegadores existentes atualmente, lançarão versões que irão converter o código *JavaScript* para código nativo, ou seja, o código será executado diretamente no computador e não mais interpretado no navegador do cliente. Isto irá permitir que o tempo de execução de aplicativos Ajax sejam comparáveis aos de aplicativos nativos (em código de máquina). Com este cenário favorável, é natural que o paradigma Ajax torne-se um padrão de desenvolvimento para aplicações *Web*, criando aplicações com mais interatividade, usabilidade e agilidade.

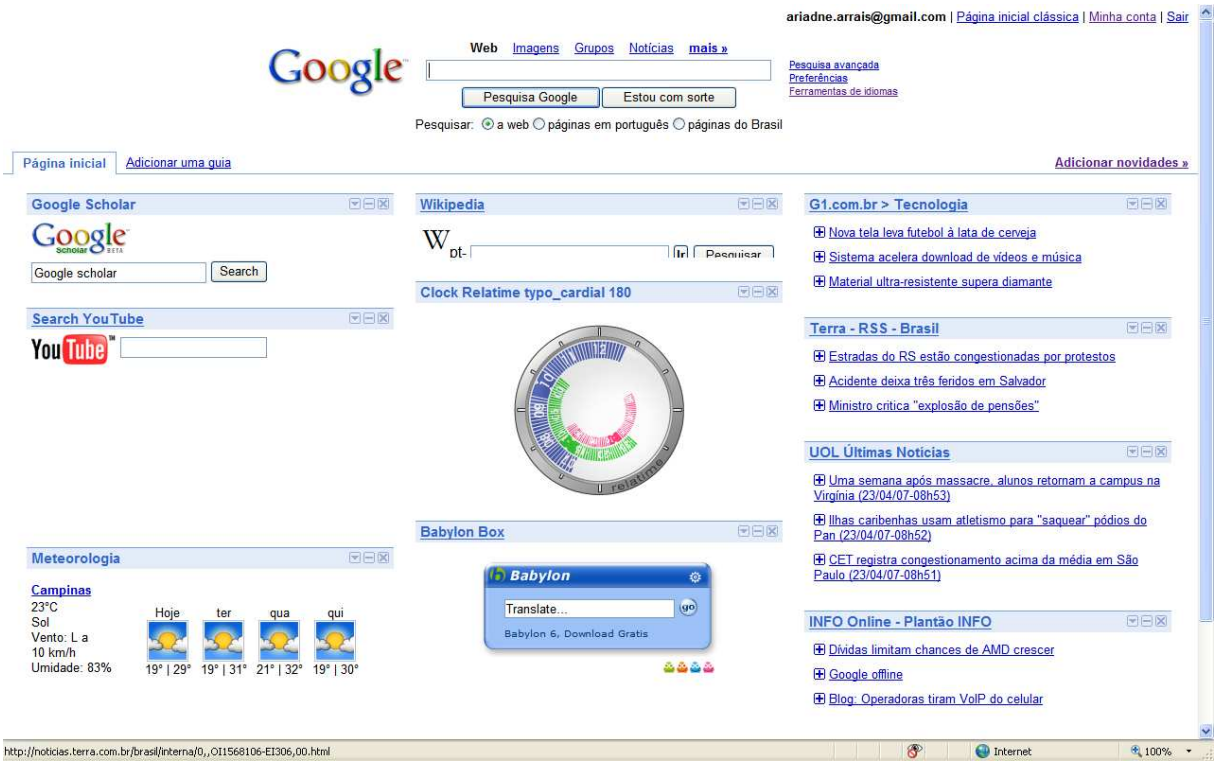


Fig. 2.2: Página de Busca Personalizada do Google.

2.1.1 *Frameworks* Ajax

Atualmente, devido a força que o Ajax ganhou, uma grande diversidade de *frameworks* Ajax estão sendo desenvolvidos. A cada dia um novo *framework* é lançado, e a escolha de qual *framework* usar depende de muitos fatores, como o tipo de linguagem e se o desenvolvedor da aplicação é um programador ou um *Web Designer*.

Durante o desenvolvimento deste trabalho, foram estudados quatro diferentes *frameworks* e implementados protótipos usando três destes, que serão analisados nas seções seguintes. Entre os *frameworks* estudados, tem-se o Qooxdoo [38], um framework de código aberto; o pacote comercial desenvolvido pela Backbase [39]; o *framework* fornecido pelo Google, chamado GWT [2] e o Echo2

[3].

Qooxdoo

É uma *framework* Ajax em código aberto (distribuído sob a licença EPL/LGPL)[2]. Inclui suporte para o desenvolvimento *JavaScript* profissional, um *kit* de ferramentas GUI (*Graphical User Interface*) e comunicação cliente/servidor de alto nível. Ele é inteiramente baseado em classes e tenta elevar as características do *JavaScript* a orientação a objetos [38].

O Qooxdoo traz com ele todas as vantagens do uso do paradigma Ajax, como uma interface agradável, a não necessidade de instalações no computador do cliente, além da não necessidade de conhecimento de CSS e HTML pelos desenvolvedores. Outra vantagem é que ele possui uma camada de abstração do navegador, que é responsável por abstrair todos os navegadores, especificando um padrão comum. A camada de abstração possui funções básicas que são usadas com freqüência para a criação de GUI.

O Qooxdoo está na sua versão 0.6.6, sendo ideal para programadores que não têm a habilidade de um *Web Designer*, porém o programador deve ter boas noções de programação *JavaScript*. Ele utiliza a mesma filosofia das linguagens visuais como Delphi [40]. Porém, não possui um editor visual tão maduro quanto os existentes para Java, o que torna a implementação um pouco mais complexa. Todo o desenvolvimento precisa ser feito em *JavaScript*, que é uma linguagem fracamente tipada. A seguir é mostrado um código Qooxdoo, que é responsável por construir a interface da Figura 2.3.

```
<script type="text/javascript">

qx.core.Init.getInstance().defineMain(function()()
{

  var d = qx.ui.core.ClientDocument.getInstance();
  var chooser = new qx.ui.component.DateChooser;
  chooser.setLocation(10,50);
  chooser.setWidth("auto");
  chooser.setHeight("auto");
  d.add(chooser);
}
</script>
```

Fig. 2.3: Exemplo de uma Interface Qooxdoo.

Uma implementação foi desenvolvida usando este *framework* Ajax, porém a sua plataforma é relativamente grande (mais de 1MB) e o tráfego é muito alto no início. Dessa forma, a aplicação

[2]http://qooxdoo.org

demora a iniciar, o que torna o seu uso inviável para a Internet nos lugares onde a banda disponível ainda é muito baixa, como por exemplo, aqui no Brasil. Juntamente com o fato de não haver ferramentas sofisticadas para o seu desenvolvimento, e a necessidade de programar em *JavaScript*, outro *framework* Ajax foi escolhido para a implementação do trabalho proposto.

Backbase

Backbase é uma companhia alemã que disponibilizou um dos primeiros *frameworks* comerciais Ajax[3]. Ele está na sua versão 3.3.1 e possui uma versão chamada de *Desenvolvimento* e uma versão chamada de *Produção*. A versão de *Desenvolvimento* é composta por um cliente de apresentação necessário para executar a aplicação, documentação, recursos para auxiliar o desenvolvedor e alguns exemplos. Por outro lado, a versão de *Produção* contém apenas os recursos necessários para executar a aplicação, ou seja, o cliente de apresentação e as *skins* do Backbase. O elemento chave do *framework* Backbase é o cliente de apresentação. Ele é todo escrito em *JavaScript* que executa em um navegador *Web*, sendo programado via uma linguagem de interface do usuário, chamada BXML (Backbase XML). Esta linguagem oferece uma biblioteca de controles de UI (*User Interface*), um mecanismo para anexar ações a elas, assim como facilitar conexões assíncronas com o servidor [39].

O lado servidor do Backbase é formado por BJS (Backbase Java *Server*) que é construído no topo do *JavaServer Faces* (JSF)[4], a nova arquitetura de apresentação J2EE (Java 2 *Enterprise Edition*). O BJS provê seu próprio conjunto de componentes UI e estende o *framework* JSF para prover a implementação de uma interface única. Assim como o Qooxdoo e os demais *frameworks* Ajax, o Backbase traz vantagens como esconder a complexidade do desenvolvimento de aplicações Ajax, a incompatibilidade entre os diferentes tipos de navegadores e plataformas, a complexidade da comunicação cliente/servidor, além de alcançar uma interatividade rica e portabilidade para os usuários finais [34].

Um outro problema relacionado ao Ajax, é que por ser um paradigma novo, não há muitos trabalhos na literatura. Assim, o Backbase surge como uma boa alternativa, já que possui uma excelente documentação e uma interface bem agradável. Este *framework* é ferramenta ideal para *Web Designer*, já que diferentemente do Qooxdoo, pode-se desenvolver aplicações *Web* em Ajax sem que seja necessário qualquer programação para isso, apenas a utilização de *tags*. A seguir é mostrado um pequeno exemplo de um código Backbase, que é responsável por construir uma estrutura de árvores, como na Figura 2.4.

Este *framework* é uma solução proprietária e só pode ser usada gratuitamente para fins não comerciais. Além disso, mudar o padrão da interface (cores e *layouts*) é uma tarefa trabalhosa. Essas características aliadas a preferência em se programar em Java, propiciou a migração da plataforma para o *framework* GWT, descrito em mais detalhes na seção seguinte.

[3]http://www.backbase.com

[4]Especificação JavaServer Faces: http://java.sun.com/j2ee/javaserverfaces.

```
<b:tree b:label="Base de Dados" b:open="true"
style="width:250px;">
  <b:tree b:label="Atores">
    <b:tree b:label="George Clooney"/>
    <b:tree b:label="Brad Pitt"/>
    <b:tree b:label="Matt Damon"/>
  </b:tree>
  <b:tree b:label="Atrizes">
    <b:tree b:label="Alicia Silverston"/>
    <b:tree b:label="Cate Blanchett"/>
    <b:tree b:label="Kate Beckinsale"/>
  </b:tree>
</b:tree>
```

Fig. 2.4: Exemplo de uma Interface Backbase.

Google Web Toolkit (GWT)

Quando se decidiu pela utilização do Backbase para a implementação da plataforma de integração de *WebLabs*, o GWT ainda não estava disponível. Ele é um *framework* Java para o desenvolvimento de aplicações *Web* dinâmicas (Ajax)[5] e está na sua versão 1.4. O GWT permite desenvolver sites como o GMail e o Google *Maps*.

O GWT é uma *framework* bastante promisor, e tem vantagens significativas para a construção de aplicações Ajax. Talvez, uma das grandes vantagens do GWT seja a possibilidade de implementar toda a aplicação em Java, tanto no lado cliente, quanto no servidor. Java tem a vantagem de ser fortemente tipado [6], quando comparado ao *JavaScript*. Dessa forma, oferece aos programadores a possibilidade de utilizarem ferramentas sofisticadas para o desenvolvimento, checagem de erros, checagem de tipo, além de análise de qualidade para a geração de código. A Figura 2.5 mostra um pequeno exemplo de um código GWT, responsável por construir uma estrutura de menus.

O GWT interage com códigos *JavaScripts* existentes, viabilizando a utilização de bibliotecas populares, como o Scriptaculous [41] e TinyMCE [42]. Ele também possui ferramentas de comunicação, que permitem que serviços no lado servidor, escritos em qualquer linguagem, sejam integrados com *frameworks* como o JSF [43], o Spring [4], o Struts [44] e o EJB (*Enterprise JavaBeans*) [45]. Esta flexibilidade garante que as ferramentas utilizadas para a implementação no lado servidor sejam mantidas [2].

Ele também oferece suporte ao *framework* de teste JUnit[7] e a um navegador especial que permite o desenvolvimento e o *debug* da aplicação em Java, sem a necessidade de incluir seu código no servidor. Além disso, ele provê um rico conjunto de ferramentas, que inclui componentes gráficos, interpretador XML, várias ferramentas para comunicação com o servidor, internacionalização,

[5] http://code.google.com/webtoolkit/

[6] Uma linguagem de computador é fortemente tipada se cada elemento dado tem um único tipo de dado e a sintaxe da linguagem torna impossível visualizar aquele elemento como um tipo diferente.

[7] É um *framework* livre para testes automatizados escrito em Java.

```
public Menus() {
  MenuBar subMenu = new MenuBar(true);
  subMenu.addItem("<code>Code</code>", true, this);
  subMenu.addItem("<strike>Strikethrough</strike>", true, this);
  subMenu.addItem("<u>Underlined</u>", true, this);

  MenuBar menu0 = new MenuBar(true);
  menu0.addItem("<b>Bold</b>", true, this);
  menu0.addItem("<i>Italicized</i>", true, this);
  menu0.addItem("More ", true, subMenu);

  MenuBar menu1 = new MenuBar(true);
  menu1.addItem("<font color='#FF0000'><b>Apple</b></font>",true, this);
  menu1.addItem("<font color='#FFFF00'><b>Banana</b></font>",true, this);
  menu1.addItem("<font color='#FFFFFF'><b>Coconut</b></font>",true, this);
  menu1.addItem("<font color='#8B4513'><b>Donut</b></font>",true, this);

  menu.addItem(new MenuItem("Style", menu0));
  menu.addItem(new MenuItem("Fruit", menu1));
  menu.setWidth("100%");
  initWidget(menu);
}
```

Fig. 2.5: Exemplo de uma Interface GWT.

ferramentas de configuração e sistemas de gerenciamento do histórico do navegador. Na Figura 2.6, observa-se como as ferramentas estão divididas e as bibliotecas que compõem o GWT.

Além de todas as vantagens citadas anteriormente, o GWT, diferentemente dos demais *frameworks* Ajax existentes que constroem uma camada de abstração, provê um código otimizado para os diferentes tipos de navegadores existentes, o que resulta em códigos menores e mais rápidos do que os da maioria dos outros *frameworks*. Esta característica, que produz um código otimizado para cada navegador, é obtida gerando-se tantos *scripts* quanto forem o número de navegadores suportados pelo GWT. Isso mostra uma grande preocupação dos projetistas do GWT, não apenas com questões de desempenho, mas também com o tamanho do código produzido. Para se ter um comparativo, um código básico implementado no Qooxdoo não é inferior a 1 MB, enquanto um aplicativo inteiro no GWT pode ser construído com menos de 100 KB de código. O GWT também possui uma excelente documentação e um fórum de desenvolvedores bastante ativo.

Cada novo *plugin* adicionado à plataforma necessita de uma interface por meio da qual os usuários possam acessá-lo. Isto requer que essas interfaces sejam construídas em tempo de execução, ou seja, em tempo real, à medida que os plugins forem sendo adicionados à plataforma. Uma vez que na versão corrente (Versão 1.4), o GWT não é capaz de gerar códigos em tempo real, foi necessária uma abordagem alternativa, que consistiu na implementação de uma linguagem intermediária, e respectivo interpretador, responsáveis então por construir essas interfaces dinamicamente (descrita na Seção 2.4).

Por todas as vantagens citadas, o GWT é utilizado para a implementação da plataforma de *Web-Labs* proposta neste trabalho. Em conjunto é utilizado o Scriptaculous, que é uma biblioteca extensível, que incorpora efeitos como o de "arrasta e puxa", e o Spring, que é um *framework* utilizado para

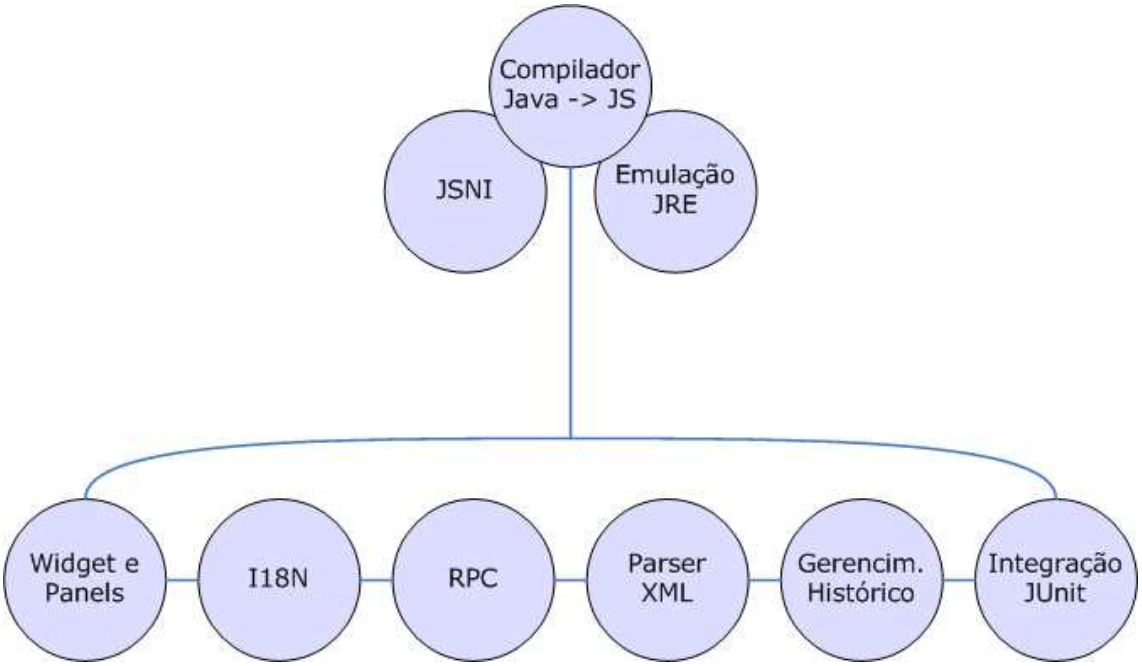


Fig. 2.6: Conjunto de Ferramentas do GWT [2].

auxiliar na implementação do Padrão de Projeto *Plugin*, que será descrito em mais detalhes na Seção 2.3.

Echo2

Echo2 [3] é um *framework* de código aberto[8] que está na sua versão 2.0.0, sendo distribuído sob a licença pública do Mozilla. Ele possibilita aos desenvolvedores criarem aplicações *Web* usando orientação a objeto, utilizando a linguagem Java, tanto no cliente quanto no servidor, além de prover APIs para o gerenciamento do estado de uma aplicação e de sua interface. Este *framework* é dividido em três módulos distintos: *Framework* de Aplicação, Mecanismo de Renderização, e *Container* da Aplicação, como é mostrado na Figura 2.7.

O módulo *Framework* de Aplicação provê uma API que representa e gerencia o estado de uma aplicação e sua interface. O módulo do Mecanismo de Renderização é responsável pela comunicação cliente/servidor. Já o *Container* de Aplicação é uma extensão do Mecanismo de Renderização que serve para renderizar e sincronizar o estado da interface construída com o *Framework* de Aplicação. Neste *framework*, todo o código é executado no servidor, e só depois da execução a resposta é enviada ao cliente. Desta forma, o número de conexões entre cliente e servidor é muito grande. Por outro lado, no GWT o processamento é dividido entre o cliente e o servidor, garantindo um número menor de conexões.

[8] http://www.nextapp.com/platform/echo2/echo/

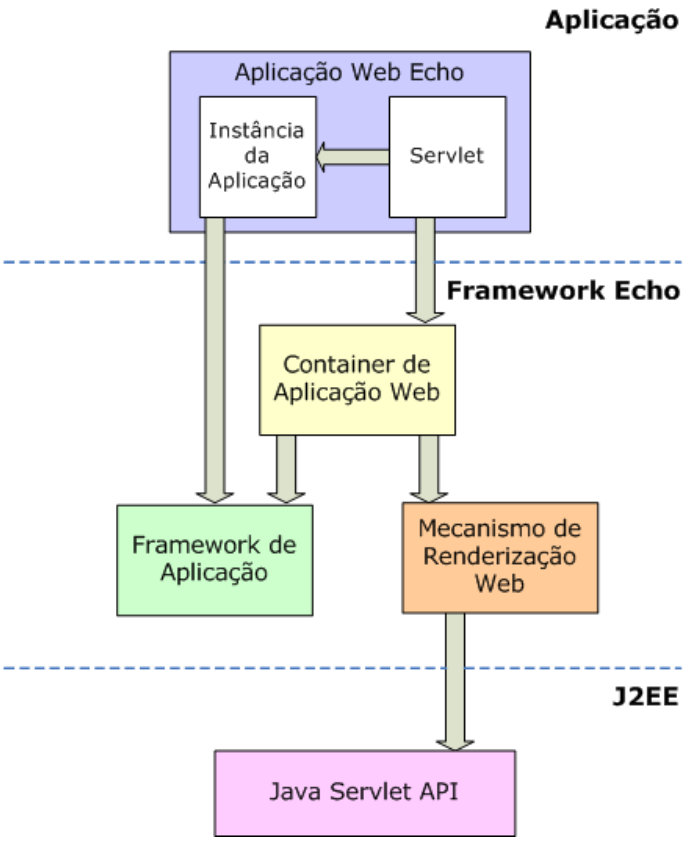


Fig. 2.7: Módulo do Framework Echo2 [3].

2.2 Padrões de Projeto

A idéia de Padrões de Projeto surgiu com o engenheiro civil Christopher Alexander, que escreveu sobre a experiência em resolver problemas de projeto que ele encontrava em construções e cidades. Anos atrás, os desenvolvedores de *software* começaram a incorporar os conceitos de Christopher na primeira documentação de Padrões de Projeto para desenvolvedores iniciantes [18][46][47][48].

Os Padrões de Projeto não são uma invenção, são soluções já testadas para problemas recorrentes. Um Padrão de Projeto tem como objetivo melhorar a qualidade do *software* em termos de reusabilidade, manutenção, extensibilidade, além de reduzir o tempo de desenvolvimento. Eles são representados como um relacionamento entre classes e objetos com responsabilidades definidas, que agem em harmonia a fim de levar a uma solução.

A descrição dos padrões é independente da linguagem de programação ou detalhes de implementação e podem ser usados em quase todos os tipos de aplicações. No entanto, eles não existem na forma de componentes de *software*, necessitam ser implementados cada vez que forem usados. Os padrões provêem uma maneira de implementar um "bom" projeto e são bastante usados no desenvolvimento de *frameworks*.

Os principais objetivos da utilização de Padrões de Projeto são o compartilhamento de experiências e agilidade no desenvolvimento. A experiência tem mostrado que é difícil conseguir um projeto flexível e reutilizável na primeira tentativa. Antes que um projeto seja terminado, programadores experientes tentam reutilizá-lo por diversas vezes, e se necessário o modificam.

A intenção e motivação de se criar um Padrão de Projeto é poder abstrair um problema recorrente e criar uma solução viável, além de poder compartilhar este conhecimento para que outras pessoas possam se beneficiar dele. A documentação de um padrão é feita numa forma muito bem definida. Segundo [19], a forma geral para se documentar um padrão consiste em quatro elementos essenciais: Nome, Problema, Solução e Conseqüência.

Nome é uma referência que pode ser usada para descrever um problema de projeto, suas soluções e conseqüências, através de uma ou duas palavras. Isso permite que se trabalhe com mais abstração, já que se pode descrever uma solução inteira utilizando apenas uma expressão, ao invés de uma lista de objetos e seus relacionamentos.

Problema descreve quando aplicar o padrão. Explica o problema e seu contexto. O problema também poderá incluir uma lista de condições que devem ser satisfeitas antes de aplicar o padrão. A Solução descreve os elementos que compõem o projeto, seus relacionamentos, responsabilidades e colaborações. Finalmente, as Conseqüências são os resultados, as vantagens e as desvantagens da aplicação do padrão.

De acordo com [49], os padrões são classificados quanto ao propósito em três tipos: criação, estrutural e comportamental. Quanto ao escopo podem ser de classe ou de objeto. Os padrões com propósito de criação preocupam-se com o processo de instanciação dos objetos. Os estruturais tratam das composições de classes e objetos. Já os comportamentais caracterizam a maneira pela qual as classes e objetos interagem e distribuem responsabilidades.

Os padrões com escopo de classe tratam relacionamentos entre classe e subclasses. Esses relacionamentos são estabelecidos através de herança, ou seja, são estáticos - definidos em tempo de compilação. Já os padrões com escopo de objeto tratam relacionamento entre objetos, os quais podem ser mudados em tempo de execução e são mais dinâmicos.

Os Padrões de Projeto possuem dois benefícios principais. Primeiro, eles fornecem uma forma de resolver problemas relacionados com o desenvolvimento de *software* utilizando soluções privadas. A solução facilita o desenvolvimento de módulos altamente coesos com mínimo acoplamento. Eles isolam a variabilidade que pode existir nos requisitos do sistema, tornando todo o sistema mais fácil de se entender e manter. Segundo, tornam a comunicação entre projetistas mais eficiente. Profissionais de *software* podem imediatamente visualizar o projeto de alto nível em suas mentes quando eles sabem o nome do padrão utilizado para resolver um problema particular quando se discute o projeto do sistema.

A utilização dos Padrões de Projeto proporciona de forma otimizada e clara a comunicação entre os desenvolvedores, documentação e maiores possibilidades de exploração para soluções alternativas

no desenvolvimento. Os Padrões de Projeto também melhoram a qualidade geral do programa e reduzem o tempo de aprendizado de novas bibliotecas e funções. Contudo, existem algumas desvantagens óbvias de sua utilização. Uma adoção tardia dos padrões tornará a implementação mais oneroza tanto em relação a prazos, quanto a custo. Outro problema, é quanto a implementação desses padrões, que nem sempre é fácil, e em muitos casos é necessária a adoção de outro padrão, todavia os resultados são sempre expressivos em relação à qualidade e à versatilidade.

Os principais padrões utilizados pelo *framework* são o MV (*Model-View*), DIP, EO, DAO (*Data Access Object*), *Factory* e *Singleton* que serão descritos com mais detalhes nas próximas seções.

2.2.1 Modelo Visão (MV)

Este padrão é uma versão simplificada do padrão MVC (*Model-View-Controller*), sendo uma variação específica do padrão *Observer*. Tem uma abordagem mais dinâmica, combinando as funcionalidades da Visão e do Controlador do paradigma MVC dentro da Visão do paradigma MV. Ele se caracteriza por separar muito bem as classes em dois grupos, de modo a facilitar o desenvolvimento como um todo. Uma alteração na camada de interface com o usuário, por exemplo, não irá alterar em nada o modo como os dados são buscados no banco ou a lógica da aplicação. As classes da aplicação são classificadas em Modelo e Visão. A Figura 2.8 mostra a interação entre essas classes.

O Modelo controla os dados da aplicação, responde às consultas da Visão a respeito do estado e atualiza seu estado quando solicitado pelas Visões. Ele também notifica as Visões quando o estado dos dados foi alterado.

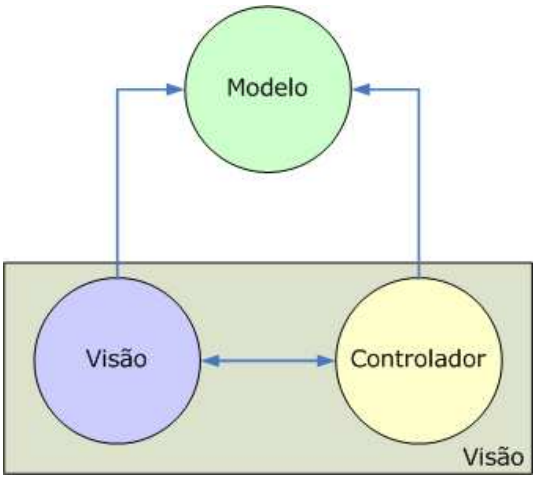


Fig. 2.8: Padrão de Projeto Modelo Visão.

A Visão apresenta a visão do modelo de dados e responde às entradas dos usuários, instruindo o Modelo para atualizar seus dados. Em uma notificação de troca do modelo de dados, ele recupera o novo estado do modelo e provê uma visão do último estado dos dados. Este padrão é usado no lado

cliente do *framework* proposto neste trabalho, como mostrado na Figura 2.1.

2.2.2 Princípio da Inversão de Dependências (DIP)

Também chamado de Inversão de Controle (IoC)[50], o objetivo desse padrão é remover o controle de dependências de dentro das classes. Ele simplesmente recebe a referência, ao invés de instanciar cada objeto internamente. Os objetos necessários são passados a classe, por exemplo, como parâmetro de construtor.

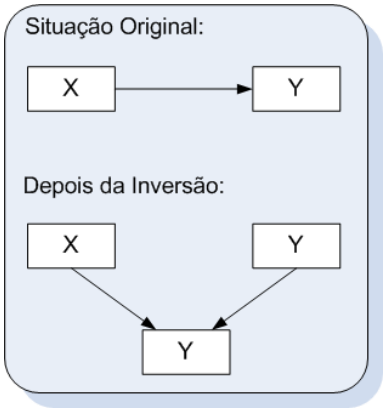


Fig. 2.9: Quebra de dependência com a inversão de controle.

Nesse padrão, o programador não precisa criar seus objetos, mas apenas descrever como eles devem ser criados. Não há necessidade de conectar os componentes e serviços diretamente no código, basta descrever que serviços são requeridos por cada componente. O *container* é responsável por realizar as tarefas restantes.

Muitos *frameworks* e *containers* como o PicoContainer, HiveMind e o Spring oferecem suporte para o gerenciamento de inversão de dependência, ou seja, os componentes não são responsáveis por instanciar suas dependências em outros componentes ou recursos. Ao invés disso, o *container* fornece essas dependências ao componente em uma das várias maneiras possíveis.

Foram desenvolvidas três implementações usando os *containers* citados anteriormente. Com o PicoContainer não existe a possibilidade de incluir um arquivo de configuração. Então, foi desenvolvida uma outra implementação usando o HiveMind, pois em seu conceito ele já utiliza um arquivo de configuração XML, o que facilita o desenvolvimento. Entretanto, o *container* Spring se integra mais facilmente ao GWT (Seção 2.1.1), além de utilizar arquivos de configuração XML. Pelas vantagens em relação aos outros *containers*, o Spring foi escolhido para a implementação do padrão DIP, sendo descrito em mais detalhes na Seção 2.3.

2.2.3 Objeto de Extensão (EO)

Também conhecido como Interface de Extensão, este padrão permite que um componente possa exportar múltiplas interfaces, prevenindo a quebra do código do cliente e o "inchaço" de interfaces quando desenvolvedores estendem ou modificam a funcionalidade do componente. Múltiplas extensões podem ser anexadas ao mesmo componente [20].

O padrão EO é necessário quando os usuários desejam modificar ou estender as funcionalidades dos clientes. Entretanto, os componentes devem suportar uma evolução, e o código do cliente nunca deve ser quebrado, quando os desenvolvedores adicionarem novas funcionalidades ao componente. Outro ponto importante, é que as funcionalidades de um componente modificado ou estendido não devem "inchar" as interfaces existentes. O padrão deve também suportar a remoção de componentes.

Em conjunto com o DIP, esses dois padrões formam o núcleo do *framework* proposto, pois são os responsáveis por criar uma arquitetura baseada em *plugins*. Essa arquitetura torna possível inserir, remover e atualizar serviços, sem a necessidade de recompilar o núcleo do *framework*. Assim como o padrão DIP, este padrão também é implementado com a utilização do *container* Spring. (Seção 2.3).

2.2.4 Objeto de Acesso aos Dados (DAO)

O Padrão de Projeto DAO (*Data Access Object*) foi utilizado para permitir flexibilidade ao projeto, praticidade de programação e facilidade de manutenção em relação à persistência dos dados. Ele encapsula a lógica de acesso a dados. Assim, se for necessário a alteração dos parâmetros do banco de dados (login, senha, localização do banco de dados, porta de acesso, parâmetros de otimização) ou mesmo trocar o banco de dados utilizado na aplicação, não será necessário alterar todo sistema, mas somente os DAOs. Dentro do DAO são realizadas as operações CRUD (*Create, Read, Update and Delete*) típicas de um banco de dados [19][51].

Este padrão oferece uma interface comum de acesso a dados e esconde as características de uma implementação específica. Ele define uma interface que pode ser implementada para cada nova fonte de dados utilizada, viabilizando a substituição de uma implementação por outra. Além disso, gerencia a conexão com a fonte de dados para obter e armazenar os dados. Isso permite, por exemplo, a utilização de um *pool* de conexões gerenciado internamente pelo DAO, sendo, portanto, transparente para as outras partes do sistema.

Para se obter tal resultado, utiliza-se uma interface com os métodos necessários (tais como, `add`, `delete`, `findAll`). Quando deseja-se efetuar alguma dessas operações de persistência de dados, basta invocar o "DAOFactory" com o método e, em seguida, chamar a operação de persistência desejada. Nota-se aqui que, neste momento, não se sabe como a implementação da interface foi feita e nem qual método de persistência será usado.

Por exemplo, para listar todos os serviços incluídos na plataforma, invoca-se o método responsável por distribuir as implementações DAO da aplicação. Em seguida, utiliza-se o método `listAll`

e tem-se uma linha de código como `dao.listAll()`. Observe que não é necessário saber como este método foi implementado e que banco de dados é utilizado.

A utilização deste padrão tem como conseqüências: transparência quanto à fonte de dados; facilidade de migração para outras implementações (basta implementar um DAO com a mesma interface); redução da complexidade do código nos objetos de negócio e centralização de todo o acesso aos dados em camada separada (qualquer componente pode usar os dados, além de facilidade de manutenção de acesso aos dados).

2.2.5 *Factory*

O padrão *Factory* oferece uma interface para a criação de famílias de objetos relacionados ou dependentes, sem especificar suas classes concretas [52]. Ele é utilizado para definir e manter relacionamentos entre objetos. A idéia deste padrão consiste no cliente chamar um método abstrato especificado em alguma classe abstrata (ou interface) e a subclasse concreta vai decidir que tipo exato de objeto criar e retornar.

Este padrão é utilizado quando uma classe não pode antecipar a classe dos objetos que deve criar ou quando uma classe quer que suas subclasses especifiquem os objetos criados. O padrão *Factory* elimina a necessidade de colocar classes específicas no código, já que pode-se criar objetos dinamicamente sem conhecer a classe da implementação, somente sua interface. Isto é possível, pois o padrão *Factory* estabelece uma forma de desenvolver objetos que são responsáveis pela criação de outros objetos.

2.2.6 *Singleton*

O padrão *Singleton* é responsável por assegurar que uma classe tenha uma única instância e por prover um ponto de acesso global a esta instância. Os clientes obtém um objeto da classe *Singleton* solicitando-o à classe. A classe *Singleton* faz o controle do número de instâncias. Quando a solicitação de uma instância é feita, a classe *Singleton* cria uma instância ou retorna a já existente. A instância deverá ser única para todo o sistema [19].

Este padrão é utilizado quando houver uma única instância de classe e esta instância precise ser acessada a partir de um ponto de acesso bem conhecido. É usado também uma instância única precise ser extensível através de subclasses e os clientes possam usar instâncias diferentes polimorficamente, sem modificação do código. Dessa forma, o *Singleton* tem o controle sobre como e quando os clientes acessam a instância.

O uso abusivo de *Singletons* leva a soluções onde a dependência entre objetos é muito forte e a testabilidade é fraca. Além disso, os *Singletons* são difíceis de trabalhar em grande escala, pois é difícil garantir uma única instância de um *Singleton* em um *cluster*, onde existem várias máquinas

virtuais Java.

2.3 Spring

Para entender o *framework* Spring, é necessário compreender o Padrão de Projeto DIP, que foi discutido na Seção 2.2.2. O Spring é um *framework* aberto e foi criado com o objetivo específico de tornar o desenvolvimento das aplicações J2EE mais fácil. Qualquer aplicativo pode se beneficiar do uso deste *framework*, em termos de simplicidade, testabilidade e agrupamento. A Figura 2.10 apresenta a estrutura do *framework* Spring [4].

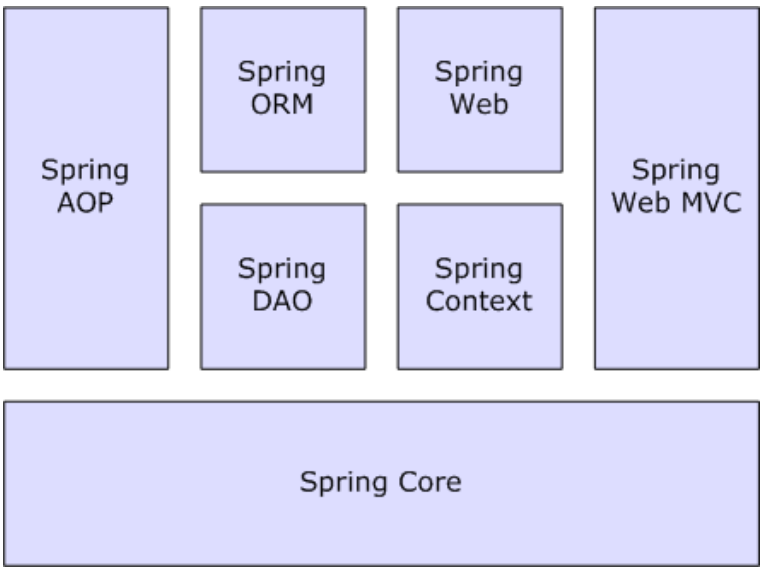


Fig. 2.10: Estrutura do *Framework* Spring [4].

O módulo Spring Core representa as principais funcionalidades do Spring, no qual o principal elemento é o BeanFactory, que é a implementação do padrão Factory (descrito na Seção 2.2.5). Este módulo garante um baixo acoplamento entre a configuração e a especificação de dependências. O módulo Spring DAO provê uma camada de abstração para JDBC (Java *Database Connectivity*), eliminando grande parte da codificação necessária para interagir com um banco de dados. Entretanto, o módulo ORM provê integração do Spring com outros *frameworks* para persistência de objetos, como Hibernate [53] e iBatis [54]. Para prover uma implementação de Orientação a Aspectos que permite a definição de *pointcuts* e métodos interceptores, existe o módulo Spring AOP (*Aspect Oriented Programming*).

O módulo Spring *Web* provê funcionalidade específicas para o desenvolvimento de aplicações *Web*, como componentes para envio de arquivos e suporte para a utilização de inversão de controle. Já o módulo Spring MVC provê a implementação de um *framework Web*, similar ao Struts, que é um *framework* de código aberto que auxilia na construção de aplicações *Web*. Ele também favorece o

desenvolvimento de aplicações que seguem o padrão MVC.

O Spring é um *framework* eficiente, porém leve, pois ele não é intrusivo e as classes da aplicação não possuem dependência com o Spring. Além disso, a sua estrutura inteira é distribuída através de um único arquivo `.jar`, inferior a 1MB, e o processamento de *overhead* é insignificante. O Spring também reduz significativamente a complexidade do uso de interfaces e agiliza e simplifica o desenvolvimento de uma aplicação.

2.3.1 *Container* de Inversão de Dependências

No Spring, os objetos que formam sua aplicação e são controlados por ela, são conhecidos como *beans*, que são objetos que são instanciados, montados e gerenciados pelo *container* de inversão de controle do Spring. As *beans* e suas dependências são mostradas em um arquivo metadados gerenciado pelo *container*.

A interface responsável por controlar e gerenciar essas dependências é a BeanFactory, que é a interface principal do *container*. Ela é responsável por montar as dependências entre os objetos. Há diversas implementações para a BeanFactory, sendo a XMLBeanFactory a mais comum. Nela toda a configuração de dependência entre os objetos é definida em um arquivo XML.

A configuração do Spring consiste basicamente na definição de uma *bean*, a qual o *container* deve gerenciar. Quando são usados os arquivos de configuração XML, as *beans* são configuradas como elementos `<bean />` dentro do elemento raiz `<beans />`. Abaixo é mostrado a estrutura básica de um arquivo de configuração XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=
"http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

   <bean id="..."  class="...">
      <!- Configuração deste bean aqui ->
   </bean>

   <bean id="..."  class="...">
      <!- Configuração deste bean aqui ->
   </bean>

      <!- Mais definições ->

</beans>
```

2.3.2 Injetando Dependências

O princípio básico da injeção de dependências é que os objetos definem suas dependências somente através de argumentos construtores, argumentos para o método *factory* ou propriedades. É trabalho do *container* injetar as dependências quando uma *bean* é criada. Assim, a própria *bean* está no controle da instanciação e da localização de suas dependências.

Quando o padrão DIP é aplicado, é gerado um código mais limpo e um baixo nível de acoplamento é alcançado. Há duas maneiras de implementar o DIP através do Spring, que são o método Construtor e o *Setter*, que são descritos a seguir.

Método Construtor

Neste método, as classes especificam, nos seus construtores, as suas classes dependentes. No momento da instanciação da classe, o *container* passa as referências para as instâncias das classes dependentes através do construtor. A seguir é mostrado um exemplo de um arquivo de configuração XML que especifica argumentos do construtor [4].

```
<bean id="foo" class="com.springinaction.Foo">
   <constructor-arg>
      <ref bean="bar"/>
   </constructor-arg>
</bean>
```

Método *Setter*

Neste método, as instâncias das classes dependentes são referenciadas através de propriedades Java *Beans*. Após a instanciação da classe, o *container* passa as referências para as classes dependentes através de métodos do tipo `setXXX()`. A seguir é mostrado um exemplo de arquivo de configuração XML que utiliza o método *Setter*.

```
<bean id="foo" class="com.springinaction.Foo">
   <property name="name"><value>Foo McFoo</value>
   </property>
</bean>
```

2.3.3 Camada Web

O coração do Spring MVC é o DispatcherServlet, um *servlet* que funciona como um controlador do Spring MVC. Como qualquer *servlet*, o DispatcherServlet deve ser configurado no arquivo `web.xml` da sua aplicação *Web*. O `<servlet-name>` dado é importante, pois por padrão quando o *servlet* Dispatcher é carregado, ele carregará o contexto da aplicação Spring a partir do arquivo XML

baseado no nome da *servlet* (`<servletname>-servlet.xml`). No caso do exemplo abaixo, ele tentará carregar a partir do nome `comlabfmwspring-servlet.xml`. Além disso, é necessário indicar que URL será requisitada pelo DispatcherServlet. Para isso, será adicionado o elemento `<servlet-mapping>` no arquivo `web.xml`. O código abaixo mostra a declaração do controlador Spring MVC (web.xml).

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="Comlab_Framework" version="2.4"
   xmlns="http://java.sun.com/xml/ns/j2ee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
      http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

   <display-name>Comlab Framework</display-name>

   <servlet>
      <servlet-name>comlabfmwspring</servlet-name>
      <servlet-class>
         org.springframework.web.servlet.DispatcherServlet
      </servlet-class>
      <load-on-startup>1</load-on-startup>
   </servlet>

   <servlet-mapping>
      <servlet-name>comlabfmwspring</servlet-name>
      <url-pattern>/service</url-pattern>
   </servlet-mapping>

   <listener>
      <listener-class>
         org.springframework.web.context.ContextLoaderListener
      </listener-class>
   </listener>

   <servlet>
      <servlet-name>local_default</servlet-name>
      <servlet-class>
         org.apache.catalina.servlets.DefaultServlet
      </servlet-class>
      <init-param>
         <param-name>listings</param-name>
         <param-value>true</param-value>
      </init-param>
      <load-on-startup>1</load-on-startup>
```

```
   </servlet>

   <servlet-mapping>
      <servlet-name>local_default</servlet-name>
      <url-pattern>/</url-pattern>
   </servlet-mapping>
</web-app>
```

2.3.4 Integração com o GWT

A possibilidade de integrar o GWT como diversas tecnologias, como o Struts, Spring, Hibernate e o Ant, o tornam ainda mais interessante. Para integrar uma aplicação GWT com o Spring, basta registrar a *servlet* GWT no arquivo `web.xml`, mostrado na seção anterior. A biblioteca GWT Widget contém o GWTSpringController, um controlador Spring abstrato que assume as funcionalidades do GWT-RPC dentro do ambiente Spring. Não há alterações no lado cliente, porém no lado servidor o seu serviço não é mais declarado, e sim o DispatcherServlet, como foi observado na seção anterior [55].

O serviço RPC (*Remote Procedure Call*) é instanciado no arquivo `<servletname>-servlet.xml` como mostra o exemplo abaixo. Obviamente, é possível criar outros serviços instanciando como controladores e mapeando-os em diferentes URLs.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<beans>

   <bean id="urlMapping"
   class="org.springframework.web.servlet.
   handler.SimpleUrlHandlerMapping">
   <property name="mappings">
      <props>
         <prop key="/service">ServiceImpl</prop>
      </props>
   </property>

   </bean>

   <bean id="ServiceImpl"
   class="com.google.gwt.sample.comlabfmw.impl.ServiceImpl"/>
```

```
</beans>
```

2.4 Linguagem Intermediária do ComLab

A Linguagem Intermediária do ComLab (CIL - *ComLab Intermediate Language*) é uma linguagem de *script* que foi concebida para ser compacta e permitir uma rápida e fácil interpretação de seus comandos. Por meio desta linguagem, códigos de programa podem ser facilmente enviados de um servidor para diversos tipos de clientes, tais como navegadores, *palmtops*, celulares, dentre outros. Outra característica da CIL, que a torna diferente de outras linguagens, é que ela foi projetada tendo-se em mente a possibilidade de integrá-la à plataforma proposta neste trabalho, descrita no Capítulo 3.

Na prática, essa forte integração com uma plataforma de serviços permite que operações complexas como o envio (recebimento) de *stream* de vídeos, som ou mesmo de arquivos simples possam ser executadas com uma instrução tão simples quanto SEND_STREAM (LOAD_STREAM); ou que RPC possam ser realizadas por meio de uma única instrução, RPC_INVOKE. Nenhuma pré-configuração ou preparação é necessária para executar estas operações, pois elas já farão parte da plataforma proposta neste trabalho.

É muito importante enfatizar que a CIL não pretende ser uma linguagem de uso genérico. Ao invés disso, seu objetivo principal é permitir o envio do código que criará a interface de usuário nas máquinas clientes (navegadores, *palmtops*, *pocket PC*, celulares, etc.). Mais especificamente, ao se adicionar um novo serviço à plataforma aqui proposta (por exemplo, um novo *WebLab*), é necessário que a interface, por meio da qual os usuários poderão interagir com o novo serviço, seja enviada para a máquina onde o usuário se encontra. É nesse contexto que a CIL está inserida.

2.4.1 Tipos de Comandos

A CIL foi concebida com o propósito de atender as necessidades de envio de interfaces de usuários do servidor para os clientes. Para atender essas necessidades, ela é composta de comandos de interfaces, estruturas de controle, declaração de variáveis e comandos utilitários, que serão descritos a seguir.

Comandos de Interface: São os comandos que permitem a criação das interfaces de usuários. Em um navegador, serão criados objetos Ajax tais como caixas de texto, listbox, gerenciadores de *layout*, botões e muitos outros. Nos dispositivos móveis, poderão ser criados objetos gráficos suportados nativamente pelo dispositivo móvel ou pelo J2ME (Java 2 *Micro Edition*). Isso dependerá da implementação da Máquina Virtual do *ComLab* (CVM - *ComLab Virtual Machine*). Todos os objetos, incluindo os objetos da UI são armazenados em um *heap* (implementado por uma lista).

Estruturas de controle: São os comandos de desvio que permitem a implementação das estruturas *if*, *for* e *while*.

Declaração de Variáveis: São os comandos que permitem a criação de variáveis numéricas, booleanas e do tipo caractere. Essas variáveis não precisam do operador NEW para serem instanciadas. Estas variáveis, por serem estáticas, são implementadas por um *array* dinâmico, aquele cujo tamanho é definido durante sua instanciação.

Comandos utilitários: São comandos existentes em função da estreita integração da CIL com a plataforma aqui proposta. Como exemplos, têm-se comandos RPC, que são comandos responsáveis pela comunicação entre a UI e a plataforma, e comandos para envio de arquivos, *stream* de videos e de voz, entre vários outros.

2.4.2 Arquitetura da Máquina Virtual da CIL

A CVM é a máquina virtual responsável pela interpretação dos comandos CIL. Esta máquina possui uma estrutura bastante modular, o que lhe permite adaptar-se às necessidades particulares de cada *Host*, dispositivo no qual a CVM estará instalada. A Figura 2.11 apresenta a arquitetura da CVM, que é composta por três elementos: Tabela de Comandos, Núcleo e o Hospedeiro.

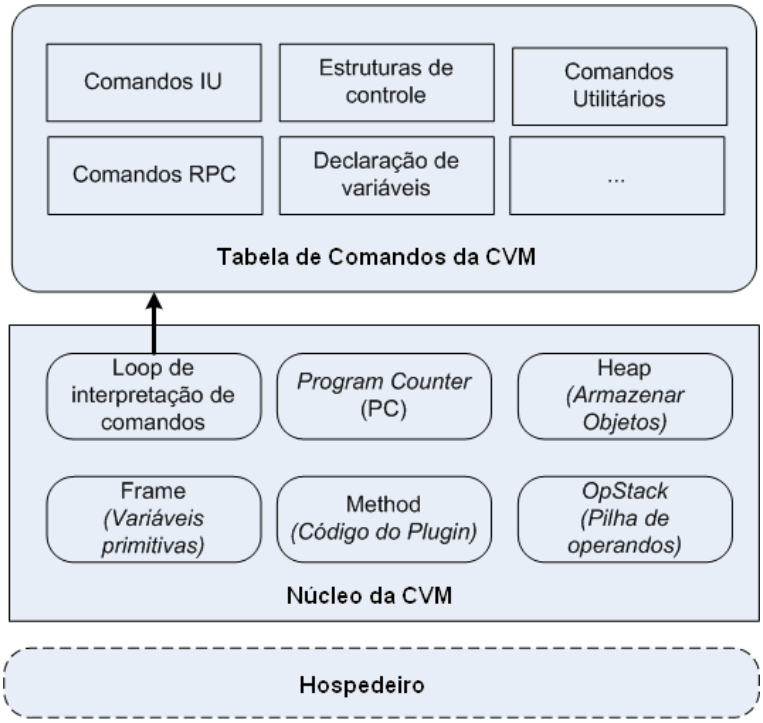


Fig. 2.11: Arquitetura da Máquina Virtual da CIL.

A Tabela de Comandos é um módulo que implementa os comandos suportados pela CVM. Esta tabela é implementada em uma estrutura do Java do tipo *Hashmap*. Cada entrada deste *Hashmap*

é um objeto que implementa um comando específico da CIL. Essa estrutura permite um alto grau de flexibilidade, pois novos comandos podem ser incorporados à tabela sem que o núcleo da CVM seja modificado. Tudo que se precisa fazer é cadastrar os novos comandos na Tabela de Comandos e recompilar a CVM. Outro benefício desta arquitetura é que ela permite a existência de diferentes Tabelas de Comandos para diferentes hospedeiros. Por exemplo, se o hospedeiro for um navegador, os Comandos UI criarão interfaces Ajax, baseadas nos objetos fornecidos pelo GWT (também poderia gerar objetos em HTML puro, ou em outra tecnologia desejada). Se, por outro lado, o hospedeiro for um dispositivo móvel, os Comandos UI criarão interfaces baseadas nos objetos gráficos do J2ME. Além de permitir que blocos de comandos sejam trocados conforme o tipo de hospedeiro, é possível gerar versões da CVM bastantes compactas, incluindo-se apenas alguns dos blocos de comandos. Por exemplo, em uma aplicação particular, pode-se desejar que apenas os Comandos UI estejam disponíveis. Isto pode ser facilmente obtido desabilitando-se os comandos desnecessários e recompilando-se uma nova versão da CVM, contendo apenas os comandos desejados. Por fim, é importante observar que todos os comandos suportados pela CVM podem ser desativados, até mesmo as Estruturas de Controle e Declaração de Variáveis. No caso mais extremo, é possível termos uma CVM sem nenhum comando definido, o que obviamente não seria muito útil.

O Núcleo é o coração da CVM. Ele gerencia:

- O *Heap*, onde ficam armazenados os objetos criados por meio do operador NEW.
- O *Frame*, onde ficam armazenadas as variáveis primitivas, isto é, variáveis inteiras, reais e lógicas.
- O *Method*, onde fica armazenado o código do *plugin*.
- O *OpStack*, que contém os operandos de cada operação.
- O *Contador de Programa*, inteiro que indica a linha correntemente sendo executada.
- O *Loop* de execução do interpretador de comandos.

O Hospedeiro representa o ambiente no qual a CVM executa. Pode ser um navegador, uma máquina virtual Java ou um sistema operacional. O código fonte da CVM está escrito em Java, o que simplifica bastante seu porte para diferentes hospedeiros. Por exemplo, por meio do GWT, pode-se compilar a CVM para *JavaScript*, permitindo-lhe executar em um navegador (que é a versão atualmente disponível). Pode-se também executá-la em dispositivo móvel, compilando-se o código para o formato compatível com J2ME. A execução direta em um sistema operacional, em código nativo, requer seu porte manual, pois não foi encontrada uma ferramenta que compile código J2ME diretamente para código nativo. Entretanto, isso pode ser realizado com relativa facilidade.

2.4.3 Princípios de Projeto

O desenvolvimento da CVM seguiu algumas diretrizes, citadas a seguir.

1. Do ponto de vista do programador, a CVM é vista como uma classe Java normal, que recebe um código escrito em CIL, executa este código e retorna um objeto gráfico a ser exibido. Este objeto gráfico, no caso de navegadores, podem ser comandos HTML puros ou, na versão implementada atualmente, objetos Ajax. No caso de celulares e *palmtops*, podem ser objetos gráficos definidos na plataforma J2ME, e assim por diante.

2. Cada instância da CVM executa um único *plugin* (um bloco de código escrito em CIL). Caberá ao aplicativo principal, a responsabilidade por instanciar e controlar as diversas instâncias da CVM. Essa arquitetura permite que diversos *plugins* sejam executados ao mesmo tempo sem que seja preciso a CVM gerenciar *threads*. Além disso, cada *plugin* terá um espaço próprio de endereçamento e respectivos controles internos, o que simplifica o trabalho do núcleo.

3. É permitido ao aplicativo principal acessar e modificar o conteúdo de qualquer uma das instâncias da CVM, permitindo que os diversos *plugins* troquem dados entre si. De fato, isso é o esperado uma vez que, do ponto de vista do programador, a CVM é apenas uma classe, cujas propriedades estão acessíveis externamente.

```
01: {"VERSION", "0.3"}
02: {"STARTUP", "20", "0", "20"}

//IU: Dados de Entrada
03: {"TEXTBOX", "NEW", "0"}
04: {"TEXTBOX", "NEW", "0"}
05: {"BUTTON", "NEW", "Add"}
06: {"VERTICALPANEL", "NEW"}

//Retorno do RPC
07: {"TEXTBOX", "NEW", "0"}

//Adiciona os objetos no VerticalPanel
08: {"VERTICALPANEL", "ADD", "3", "0"}
09: {"VERTICALPANEL", "ADD", "3", "1"}
10: {"VERTICALPANEL", "ADD", "3", "2"}
11: {"VERTICALPANEL", "ADD", "3", "4"}

12: {"BUTTON", "ONCLICK", "2", "13"}
13: {"RETURN", "3"}
```

```
//Código do Botão
14: {"TEXTBOX", "GETTEXT", "0"}
15: {"TEXTBOX", "GETTEXT", "1"}
16: {"DCONST", "2"}
17: {"SCONST", "ADD"}
18: {"GET_PLUGINID"}

19: {"RPC_INVOKE", "20", "24"}
20: {"RETURN"}

//Em caso de sucesso
21: {"D2S"}
22: {"TEXTBOX", "SETTEXT", "4"}
23: {"TEXTBOX", "SETTEXT", "4"}
24: {"RETURN"}

//Em caso de falha
25: {"TEXTBOX", "SETTEXT", "4"}
26: {"RETURN"}
```

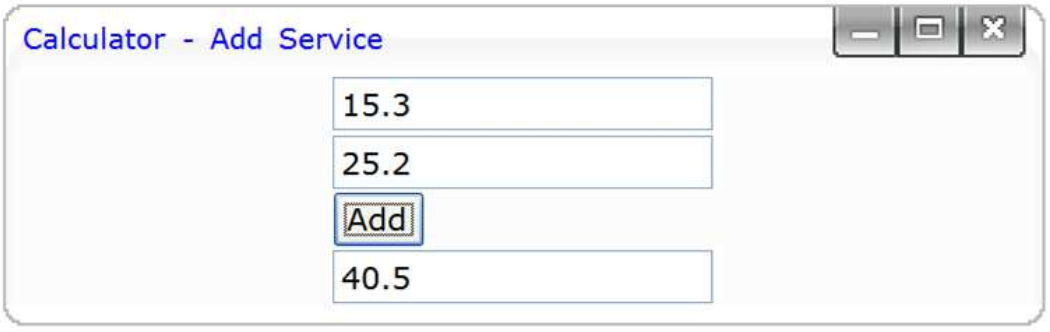


Fig. 2.12: Código de uma Interface Criada através da CIL.

2.4.4 Exemplo de um Código em CIL

A Figura 2.12 mostra um código de uma interface criada através da CIL. No código mostrado na Figura 2.12, a Linha 02 indica o tamanho da *heap* (20), tamanho do *Frame* (0) e da pilha de operandos (20) que serão utilizados pela CVM. Nas Linhas 03, 04, 05 e 07, quatro *widgets* (objetos gráficos como caixa de textos e botões) são criados. A Linha 06 cria um *panel* onde os *widgets* serão adicionados (Linhas 08 a 11). Da Linha 14 a 20 consiste no código que deverá ser executado quando o usuário clicar sobre o botão (Linha 12). Na Linha 19 é feita a chamada RPC.

É importante enfatizar que como trabalho futuro, será implementado um tradutor que converterá o código Java para CIL, o que possibilitará aos desenvolvedores implementar toda a plataforma, incluindo a interface dos serviços em Java.

2.5 Considerações Finais

Este capítulo forneceu uma visão geral sobre as principais tecnologias e metodologias utilizadas na implementação do trabalho proposto. Foram apresentadas as definições do paradigma Ajax e de Padrões de Projeto, descrevendo os principais padrões utilizados durante o desenvolvimento deste trabalho. Neste capítulo também foi apresentado o Spring, o *framework* responsável pela implementação do padrão *Plugin*, e a CIL, uma linguagem responsável pela criação da interface dos novos serviços, garantindo desta forma a não necessidade de recompilar o núcleo do *framework*, devido a adição, alteração ou remoção de serviços.

Capítulo 3

Arquitetura Proposta

Neste capítulo apresentamos uma visão geral dos serviços para *Web*, bem como descrevemos a arquitetura do *framework* proposto e a forma como novos serviços e suas respectivas interfaces poderão ser incluídas. Serão apresentados também os diagramas UML (*Unified Modeling Language*) elaborados durante o desenvolvimento deste projeto e a situação atual do desenvolvimento.

3.1 Visão Geral

Antes da popularização da Internet, grande parte das aplicações desenvolvidas eram projetadas para serem executadas em computadores pessoais ou em redes locais. Com a popularização da Internet e larguras de banda cada vez maiores, as plataformas *Web* ganharam grande impulso com o surgimento de tecnologias como o Ajax, que permitem a construção de aplicativos *Webs* com a mesma usabilidade e responsividade de aplicativos *desktop*. Dessa forma, os aplicativos foram migrando para plataformas *Web*, a fim de tornar as aplicações globalmente distribuídas através da rede. Assim, os usuários agora podem compartilhar as mesmas informações armazenadas num só lugar, aumentando assim a manutenabilidade [36][7][56][57].

A tecnologia dos serviços para *Web* surgiu com a finalidade de integrar diferentes tipos de sistemas e solucionar os possíveis problemas da integração destes sistemas. Com esta tecnologia é possível que estas aplicações interajam com aplicações já existentes e que sistemas desenvolvidos em diferentes plataformas sejam compatíveis. Os serviços para *Web* permitem que a integração ocorra de forma reutilizável e padronizada, facilitando o seu desenvolvimento, já que a Internet é cercada por uma grande variedade de aplicativos e plataformas.

Diante deste cenário, onde o compartilhamento de informações e a integração das aplicações tornam-se cada vez mais necessários, surge o problema de interoperabilidade. Dessa forma, a tecnologia de serviços para *Web* vem para solucionar esse tipo de problema, fazendo com que uma aplicação possa utilizar as funcionalidades de outras sem restrição. Sendo assim, o uso destes serviços no ambiente de aprendizagem, através dos *WebLabs*, torna-se cada vez mais comum. O desenvolvimento destes *WebLabs* permite que a informação seja compartilhada e os experimentos realizados utilizando

todos os recursos da rede.

3.2 Descrição do Projeto

O protótipo desenvolvido neste trabalho visa integrar os diversos serviços que estão sendo desenvolvidos na rede *KyaTera* à plataforma, sem a necessidade de qualquer interrupção nos serviços ou na plataforma. Esta integração permite não apenas a agregação de novas funcionalidades aos serviços já instalados, mas também possibilita que os serviços utilizem recursos uns dos outros.

O núcleo da aplicação é implementado em Java, sendo dividido em 7 módulos, como os Serviços Interface do Usuário (UI - User Interface), o Gerenciador de *Plugins*, o Gerenciador de Usuários, Serviços de Persistência, Modelo de Comunicação, Ferramentas de Monitoramente e *Log* e a Camada de Segurança (Figura 3.1). Os serviços que integram atualmente a plataforma são desenvolvidos em Java e Labview [58][59], que é um ambiente de desenvolvimento da *National Instruments* (Veja Seção 4.1.2).

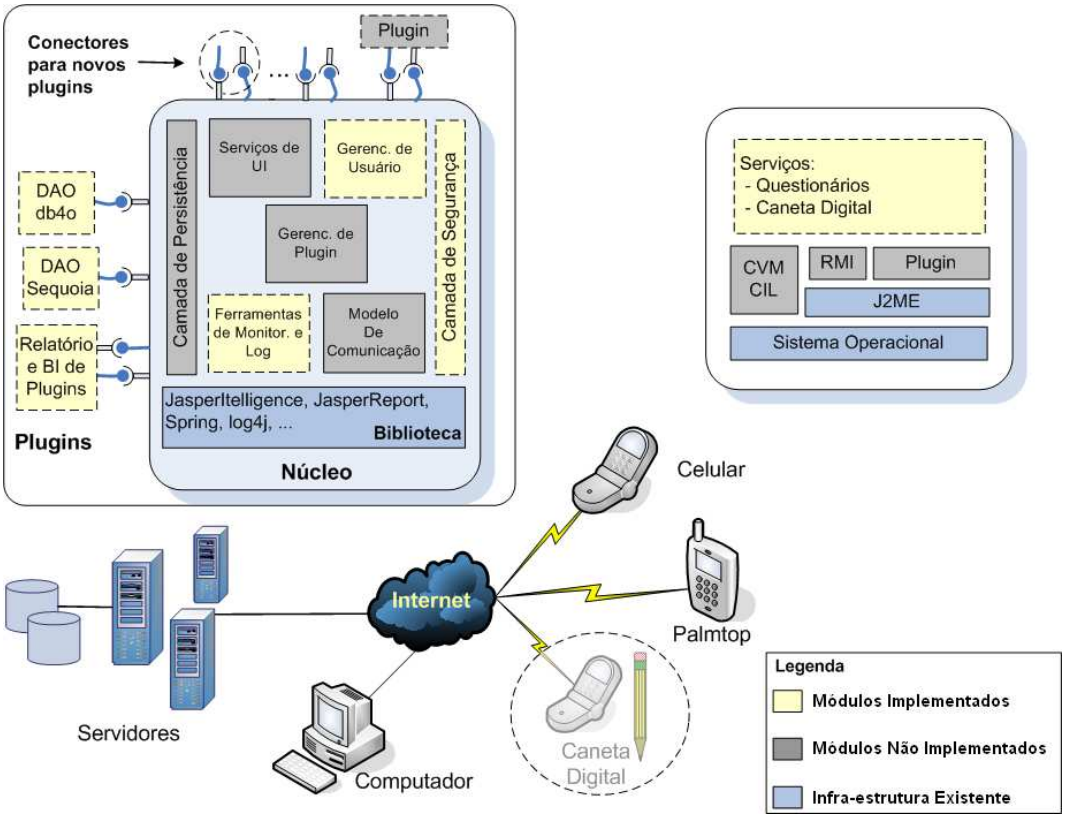


Fig. 3.1: Arquitetura da Plataforma Proposta.

Cada tipo de dispositivo pode instalar uma CVM, permitindo que o serviço instalado na plataforma possa, automaticamente, ser executado em vários tipos de dispositivos diferentes. A CVM

também poderá se integrar a diferentes tecnologias existentes como o *Appolo*, da Adobe; o *JavaX*, da Sun ou o *Siverlight*, da Microsoft. Esta facilidade de integração com diferentes tecnologias mostra que a plataforma está preparada para evoluir conforme novas tecnologias vão surgindo. A seguir será feita a descrição dos módulos atualmente implementados.

3.2.1 Serviços de Interface do Usuário

Os Serviços Interface do Usuário são uma aplicação Ajax que provê avançadas funcionalidades para o usuário. Ele também é responsável por oferecer serviços aos *plugins*. As interfaces dos serviços serão construídas em tempo de execução por meio da CIL (ver Seção 2.4). Quando executada em um navegador, a interface gerada pela CIL utilizará os objetos gráficos disponibilizados pelo GWT (ListBox, Botões, Menus, entre outros). Na Figura 3.2 na página seguinte são mostradas algumas interfaces criadas através do GWT. Os objetos visuais utilizados para a criação das interfaces são organizados através de uma hierarquia, mostrada na Figura 3.3 na página 39. Embora os *panels*, também, descendam dos *widgets*, eles são mostrados na Figura 3.4 na página 40, devido ao número de objetos que o compõem.

O módulo Serviços UI se comunica com a camada Modelo de Comunicação para fazer as atualizações necessárias a sua interface. A interface explora toda a interatividade oferecida pelas aplicações Ajax, e foi desenvolvida usando GWT, que foi descrito na Seção 2.1.1. Como mencionado anteriormente, todos os serviços integrados a plataforma têm sua interface desenvolvida com o auxílio da CIL, o que possibilita a geração de códigos em tempo de execução. Na Figura 3.5 é mostrado um código desenvolvido com a CIL (descrita em detalhes na Seção 2.4) e a sua respectiva interface.

3.2.2 Gerenciador de *Plugins*

Esta camada é responsável pelo controle de todos os *plugins* instalados. A arquitetura é baseada nos padrões DIP e EO, que foram descritos nas Seções 2.2.2 e 2.2.3, respectivamente. Os *plugins* podem ser inseridos na plataforma, sem a necessidade de recompilá-la. Os *plugins* e o núcleo da plataforma foram desenvolvidos baseado numa arquitetura em camadas, provendo um baixo acoplamento e uma alta coesão. Os *plugins* podem utilizar uma interface já existente ou alguma outra específica para serem integrados à plataforma.

Todos os serviços integrados à plataforma estão armazenados em um arquivo de configuração XML. Como foi descrito na Seção 2.2.4, o acesso a base de dados (e aos arquivos de configuração) é encapsulado. Se for necessário mudar a forma de acesso, não será preciso alterar o código do cliente, basta criar uma nova DAO. A seguir, é mostrado o arquivo responsável pelo cadastro dos serviços.

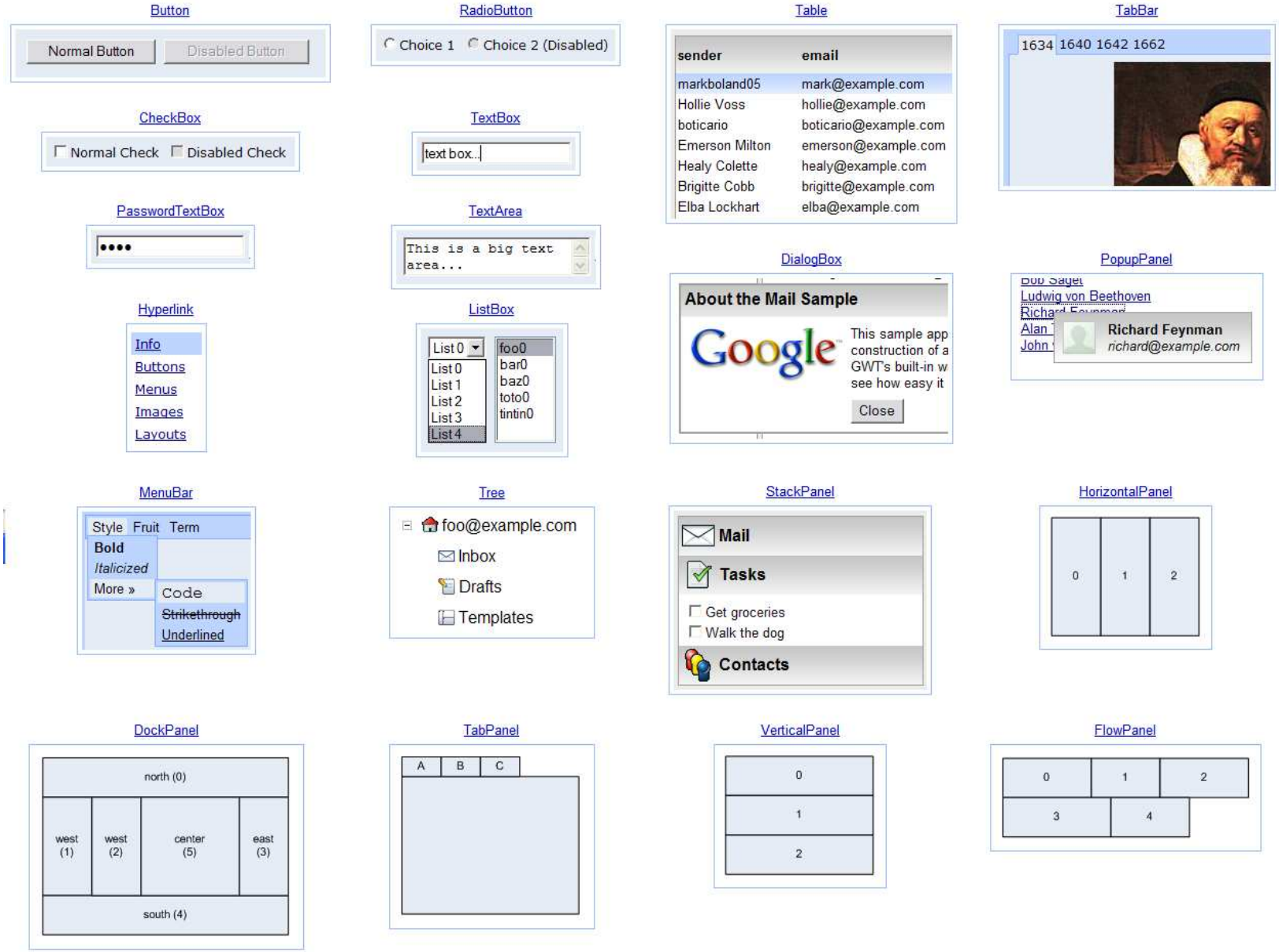


Fig. 3.2: Interfaces do GWT [5], acessíveis dinamicamente por meio da CIL.

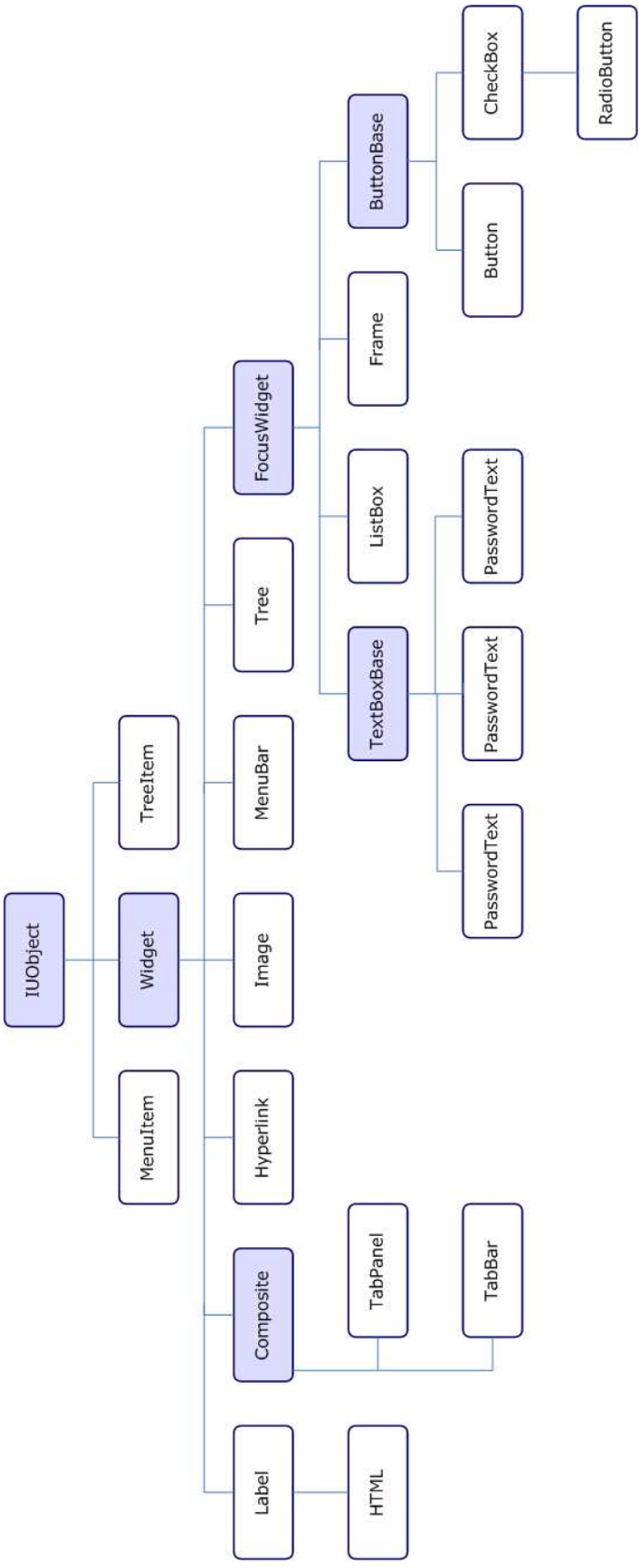


Fig. 3.3: Hierarquia dos Objetos Visuais do GWT [2].

Fig. 3.4: Hierarquia dos Panels [2].

```
01: VERSION 0.1
02: STARTUP 6 0 10
03: TEXTBOX NEW
04: TEXTBOX NEW
05: TEXTBOX NEW
06: BUTTON NEW "Click-me.."
07: VERTICALPANEL NEW
08: VERTICALPANEL ADD 4 0
09: VERTICALPANEL ADD 4 1
10: VERTICALPANEL ADD 4 2
11: VERTICALPANEL ADD 4 3
12: BUTTON ONCLICK 3 13
13: RETURN 4

14: TEXTBOX GETTEXT 0
15: S2D
16: TEXTBOX GETTEXT 1
17: S2D
18: ADD
19: D2S
20: TEXTBOX SETTEXT 2
21: RETURN
```

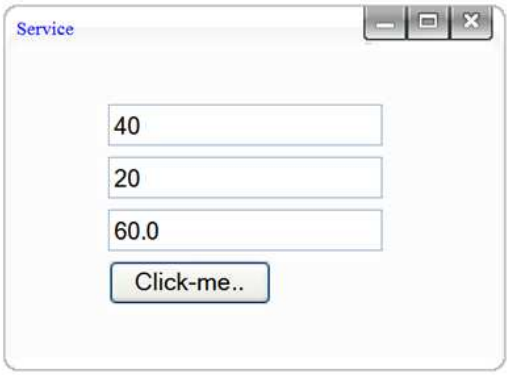


Fig. 3.5: Interface Desenvolvida através da CIL.

```
<plugins>
  <plugin id="1">
    <nome>Criptografia</nome>
    <descricao>Descrição do Serviço</descricao>
    <dtinclusao>26/02/2007</dtinclusao>
    <link></link>
    <autor>Ariadne Cruz</autor>
    <laboratorio>ComLab</laboratorio>
    <tpservico>Aplicação Java</tpservico>
    <hrinicio>00:00</hrinicio>
    <hrfim>24:00</hrfim>
    <hrlimite>2</hrlimite>
    <icone>cryptography.png</icone>
    <ultimaatual></ultimaatual>
    <free>true</free>
    <enable>true</enable>
    <interface>CryptoImpl</interface>
  </plugin>
</plugins>
```

Ao armazenar um serviço é necessário indicar as informações básicas de um serviço, tais como, o nome do serviço, a sua descrição, uma URL de acesso, o autor (desenvolvedor de serviço), o laboratório responsável, o tipo de serviço, a hora em que o serviço inicia e até que horas ele ficará disponível para o agendamento, o tempo máximo em que o usuário poderá executar o serviço, uma imagem para o serviço, a data da última atualização, uma indicação se o serviço é livre ou não, uma indicação se o serviço está disponível e a classe da implementação do serviço.

Um serviço é livre (*free*) quando ele não precisa de agendamento para ser executado, porém só poderá ser executado se estiver com o *status* de disponível, caso contrário o usuário não poderá executá-lo. Na Seção 3.3.4 será mostrado o diagrama de seqüência do que acontece no instante em que o usuário acessa o site da plataforma.

Inclusão de Novos Serviços

Para incluir novos serviços à plataforma foi utilizado o *framework* Spring no lado servidor, que inclui a inversão de dependências, e a CIL no lado cliente, que é responsável pela criação da interface dos serviços, já que o GWT não permite a criação de código em tempo de execução. A integração entre o Spring e a CIL garante que a plataforma seja extensível, flexível e robusta.

Para incluir um serviço é necessário:

1. Criar as classes Java que implementam o novo serviço (*plugin*). A classe principal deve estender a interface `Services`.
2. Copiar os arquivos `.class` das classes acima para a pasta com/google/gwt/sample/comlabfmw/impl/.
3. Inserir uma nova entrada no arquivo de configuração do Spring. Esta nova entrada apontará para a classe principal do novo serviço (aquela que estende a interface `Services`).
4. Inserir uma nova entrada no arquivo de configuração da plataforma (plugins.xml).

Para que a inserção de novos serviços ocorra de forma satisfatória, os arquivos responsáveis pela funcionalidade do novo serviço devem ser copiados para o endereço com\google\gwt\sample\comlabfmw\impl. Na Figura 3.6 mostra-se a estrutura de pastas da plataforma aqui apresentada. Atualmente, a cópia destes arquivos é realizada manualmente pelo administrador do sistema. Posteriormente, serão desenvolvidas ferramentas que automatizem este processo. Com relação aos novos serviços incluídos na plataforma proposta, existem duas políticas que podem ser adotadas para que estas atualizações tornem-se disponíveis. A primeira é recriar o serviço (isto é, instanciar o objeto que o implementa) a cada nova solicitação feita pelo usuário. Neste caso, as atualizações ficarão disponíveis quando o usuário clicar sobre o botão Atualizar(*refresh*) do navegador. A segunda política é criar o serviço somente na primeira solicitação, e reutilizá-lo nas próximas solicitações. A escolha por qual política adotar tem como um dos fatores determinante, a quantidade de acessos a plataforma. Por exemplo, no caso de milhões de solicitações diárias, criar e destruir um serviço a cada nova solicitação poderá comprometer o desempenho do servidor.

A classe principal responsável pela criação do serviço deve implementar a interface `Services`. Esta interface possui dois métodos. O primeiro deles é o método `String[][] getMethod ()`, que é responsável por retornar o código da CIL (por enquanto, o código é retornado como um *array* de strings). O segundo, é o método `String[] execute(String[] strings)`, que é o ponto

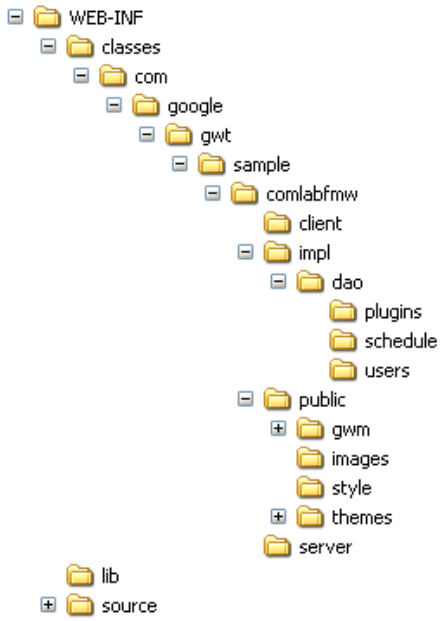


Fig. 3.6: Estrutura das Pastas do Projeto.

de acesso às funcionalidades do novo serviço. Este método recebe os parâmetros através de um *array* de String e retorna o resultado em um outro *array* de Strings.

Exemplo de Inclusão de um Novo Serviço

Para ilustrar o processo de inclusão de um novo *plugin* (serviço), serão mostrados os passos necessários para inclusão do serviço "Calculadora" (Figura 3.7 na próxima página). Trata-se de uma calculadora bastante simples, que tem por objetivo apresentar a soma de dois números.

Passo 1: Implementar as classes do novo serviço.

Neste exemplo, o serviço é composto por uma única classe, que se chama `SomaImpl`. Ela é responsável por implementar os métodos `GetMethod()` e `execute()` requeridos pela interface `Service`. A Figura 3.7 na página seguinte mostra o código desta classe.

Como uma observação adicional, as classes de um novo serviço devem pertencer ao pacote com.google.gwt.sample.comlabfmw.impl. Futuramente, esta limitação será removida, permitindo-se que o programador organize livremente sua estrutura de pacotes.

Passo 2: Copiar os arquivos `.class` do serviço.

O administrador, que não necessariamente é o desenvolvedor do novo serviço, precisará copiar os arquivos `.class` para a pasta <install-CF>/WEB-INF/classes/com/google/gwt/sample/comlabfmw/impl. Onde <install-CF> representa o diretório que contém os arquivos da plataforma.

```
package com.google.gwt.sample.comlabfmw.impl;

public class SomaImpl implements Services{
      public String[][] getMethod (){

     String [][] teste = {{ "VERSION", "0.3" },   //00, Versão da CIL
               { "STARTUP", "20", "0", "20" },      //01

               //UI: Dados de entrada
               { "TEXTBOX", "NEW", "0" },          //02 heap[0]
               { "TEXTBOX", "NEW", "0" },          //03 heap[1]
               { "BUTTON", "NEW", "Add" },         //04 heap[2]
               { "VERTICALPANEL", "NEW" },         //05 heap[3]

               //Retorno do RPC
               {"TEXTBOX", "NEW", ""},             //06 heap[4]

               //Adiciona os TEXTBOX, BUTTON, e VERTICALPANEL
               {"VERTICALPANEL", "ADD", "3", "0"},   //07
               {"VERTICALPANEL", "ADD", "3", "1"},   //08
               {"VERTICALPANEL", "ADD", "3", "2"},   //09
               {"VERTICALPANEL", "ADD", "3", "4"},   //10

               {"BUTTON", "ONCLICK", "2", "13"},     //11
               { "RETURN", "3"},                    //12

               //Código do Botão
               {"TEXTBOX", "GETTEXT", "0"},         //13
               {"TEXTBOX", "GETTEXT", "1"},         //14
               {"DCONST", "2"},                    //15
               {"SCONST", "ADD"},                   //16
               {"GET_PLUGINID"},                    //17

               //Chamada RPC
               {"RPC_INVOKE", "20", "24"},          //18
               { "RETURN"},                        //19

               //Em caso de Sucesso
               {"D2S"},                            //20
               {"TEXTBOX", "SETTEXT", "4"},         //21
               {"TEXTBOX", "SETTEXT", "4"},         //22
               { "RETURN"},                        //23

               //Em caso de Falha
               {"TEXTBOX", "SETTEXT", "4"},         //24
               { "RETURN"},                        //25
               };
     return teste;
     }

       public String[] execute(String[] param) {
               String[] result = new String[2];
               Double d = Double.valueOf(param[0]) + Double.valueOf(param[1]);
               result[0] = d.toString();
               return result;
       }
}
```

Fig. 3.7: Exemplo de Implementação de um Serviço de Adição (Java + CIL).

Observe que os arquivos `.class` do novo serviço poderiam ser implementados por terceiros (outras instituições de pesquisa) e entregues ao administrador para inclusão na plataforma aqui apre-

sentada. Isso permite que instituições compartilhem código entre si de uma maneira bastante simples.

Passo 3: Atualizar arquivos de configuração do Spring.

O administrador deverá incluir um elemento `bean` no arquivo de configuração do Spring, que está localizado na pasta <install-CF>/WEB-INF/.

A seguir é mostrado o elemento bean que deverá ser incluído neste exemplo.

```
<bean id="SomaImpl"
     class="com.google.gwt.sample.comlabfmw.impl.SomaImpl"/>
```

Passo 4: Atualizar arquivos de configuração da plataforma.

Em seguida, o administrador precisará cadastrar o novo serviço no arquivo de configuração da plataforma proposta, plugins.XML. Este arquivo é responsável por armazenar todos os serviços integrados à plataforma, e encontra-se no diretório <install-CF>/WEB-INF/.

A seguir é mostrado o trecho que precisará ser incluído a este arquivo.

```
<plugins>
   <plugin id="2">
      <nome>Calculator - Add</nome>
      <descricao>Add</descricao>
      <dtinclusao>26/02/2007</dtinclusao>
      <link></link>
      <autor>Ariadne Cruz</autor>
      <laboratorio>ComLab</laboratorio>
      <tpservico>Aplicação Java</tpservico>
      <hrinicio>00:00</hrinicio>
      <hrfim>24:00</hrfim>
      <hrlimite>2</hrlimite>
      <icone>add.png</icone>
      <ultimaatual></ultimaatual>
      <free>true</free>
      <enable>true</enable>
      <interface>SomaImpl</interface>
   </plugin>
</plugins>
```

Passo 5: Disponibilizar o serviço.

Dependendo da política adotada pelo administrador, o serviço estará disponível tão logo que o usuário clique no botão Atualizar (*refresh*) do navegador ou somente quando o Servidor de Aplica-

ções (Tomcat, JBoss, etc.) for reiniciado.

3.2.3 Serviços de Persistência

Os Serviços de Persistência provêem a habilidade de armazenar objetos em um banco de dados relacional. Contudo, o uso do Padrão de Projeto DAO, descrito na Seção 2.2.4, permite que o uso de outros mecanismos de armazenamento, como Banco de Dado Orientado a Objetos (OODB), arquivos XML, ou outras formas.

A Figura 3.8 apresenta o modelo Entidade Relacionamento da estrutura de dados da plataforma proposta. Os retângulos representam as entidades, ou seja, "objetos" que precisam armazenar informações. Há quatro entidades (usuários, agenda, serviços e serviços restritos), cada uma dessas entidades possui um conjunto de atributos que a caracterizam. Para os usuários os atributos são: o identificador do usuário, nome, sobrenome, afiliação, departamento, endereço, país, estado, cidade, CEP, email, telefone, status, tipo, senha e habilitado. Para a entidade agenda são: o identificador do usuário, identificador do serviço, data de utilização, horário de início do experimento e tempo estimado de duração. Para os serviços e os serviços restritos, os atributos são os mesmo, pois os serviços restritos são uma generalização dos serviços, e dessa forma herdam todos os seus atributos.

Os atributos para os serviços são: o identificador do serviço, nome, descrição, data de inclusão, URL, autor, hora de inicial para disponibilização do experimento, hora final para disponibilização do experimento, tempo limite para a execução de um experimento, ícone, data da última atualização, livre, habilitado e interface. Além dos atributos, as entidades estabelecem relacionamentos entre si. A Figura 3.8 apresenta os seguintes relacionamentos:

1. Um usuário pode agendar nenhum ou muitos serviços restritos.

2. Serviço Restrito é também um Serviço (Relacionamento de Herança), que pode ser agendado por nenhum ou por muitos usuários.

No trabalho proposto, os dados são armazenados na forma de arquivos XML. Contudo, caso haja a necessidade de trocar a base da dados, bastará acrescentar a implementação da nova base de dados, sem precisar fazer outras alterações na plataforma, já que o padrão DAO encapsula esses dados.

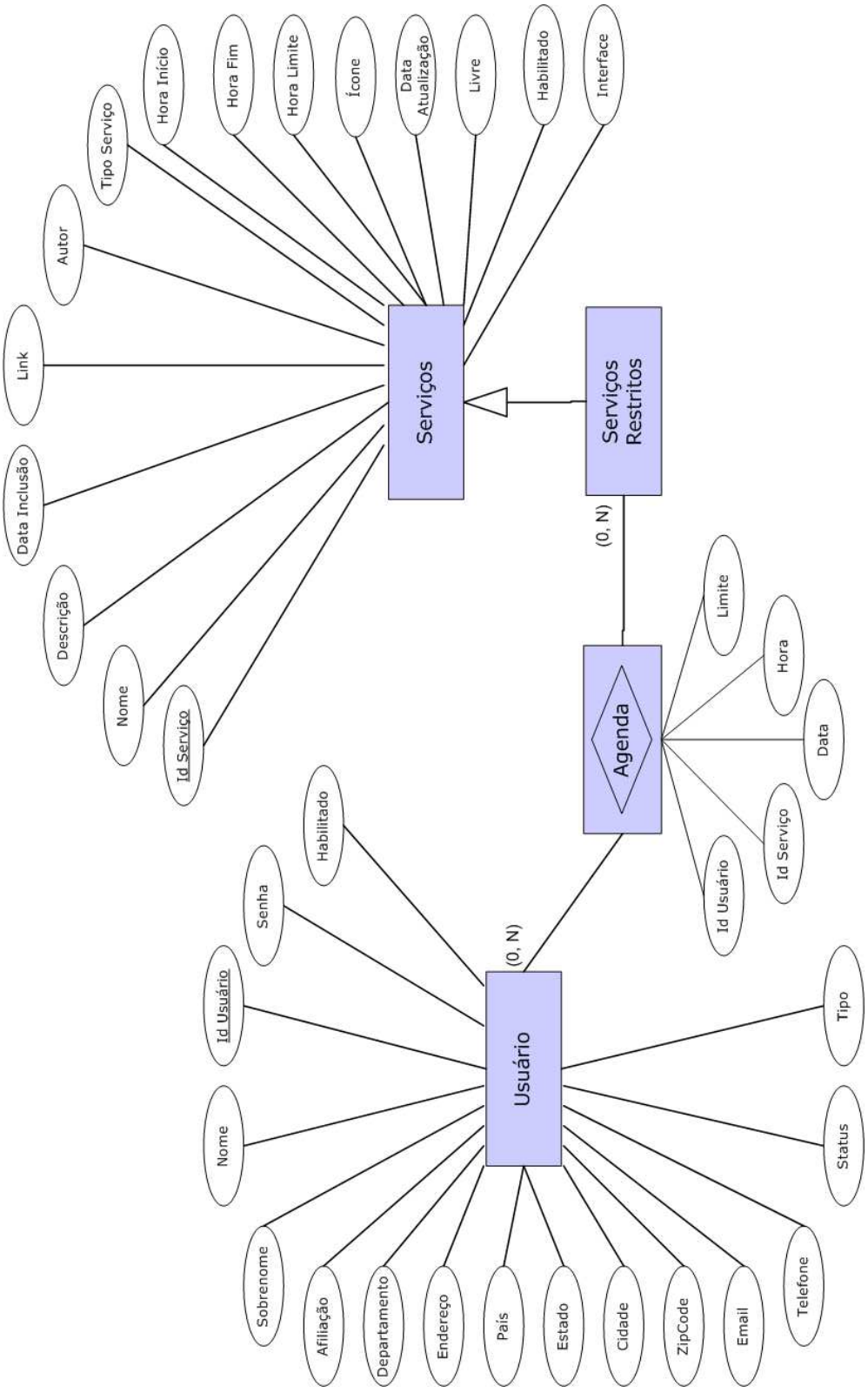


Fig. 3.8: Diagrama Entidade Relacionamento [2].

Os dados armazenados são acessados por meio de suas DAOs, que por sua vez são acessadas através da camada Modelo de Comunicação (descrita na seção seguinte). Na Figura 3.9 é mostrado o Diagrama de Classes do Padrão de Projeto DAO que foi implementado. A classe DAOFactory receberá a solicitação de que DAO precisará acessar (PluginDAOFactory, UserDAOFactory ou ScheduleDAOFactory) e o distribuirá para a classe indicada. Esta classe será responsável por chamar a implementação do banco da dados. Futuramente, a implementação do padrão DAO será integrada ao *framework* Spring dando uma maior flexibilidade à plataforma.

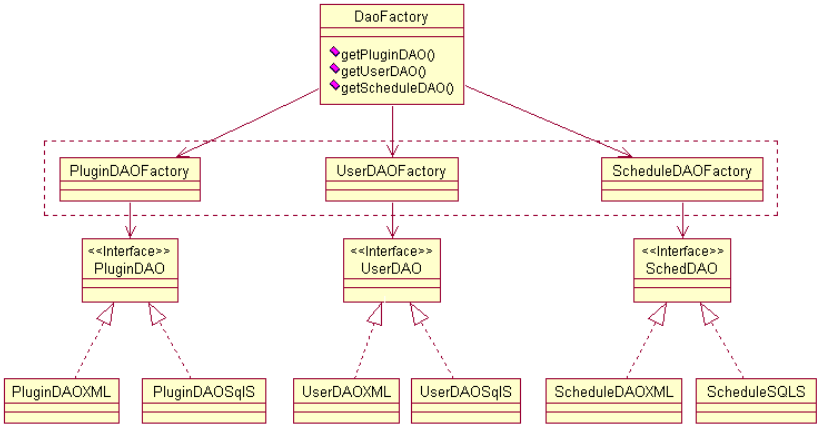


Fig. 3.9: Diagrama de Classe do DAO.

Atualmente, um usuário logado no sistema poderá realizar as operações de atualização de cadastro, alteração de senha e exclusão da sua conta, além de poder agendar um serviço e executá-lo. Todas essas operações acessam a base de dados e quando necessário fazem modificações na mesma. A seguir é mostrado um trecho de um código Java que lista todos os serviços integrados à plataforma.

```
PluginDAO dao = PluginDAOFactory.getPluginDAO(1);
List l = (List) dao.listAll();
String[][] plugin = new String[l.size()+1][16];

   if (l != null){
      for (int j=0; j < l.size(); j++) {

         Plugin p = (PluginImpl) l.get(j);

         plugin[j][0] = p.getId();
         plugin[j][1] = p.getName();
         plugin[j][2] = p.getDescription();
         plugin[j][3] = p.getInclusionDate();
         plugin[j][4] = p.getLink();
         plugin[j][5] = p.getAuthor();
         plugin[j][6] = p.getLab();
         plugin[j][7] = p.getServiceType();
```

```
        plugin[j][8] = p.getBeginHour();
        plugin[j][9] = p.getEndHour();
        plugin[j][10] = p.getLimit();
        plugin[j][11] = p.getIcon();
        plugin[j][12] = p.getLastUpdate();
        plugin[j][13] = p.getFree();
        plugin[j][14] = p.getEnable();
        plugin[j][15] = p.getInterface();

      }
    }
return plugin;
```

Observe que no código mostrado não é conhecido como foi feita a implementação do método listAll(), pois o padrão DAO encapsula a lógica dos dados. Isto facilita um futura migração da base de dados, além de separar muito bem o código de apresentação do código de acesso aos dados.

3.2.4 Modelo de Comunicação

Esta camada provê tipos para estrutura de dados dos elementos armazenados, sendo responsável pela comunicação entre o cliente *JavaScript* e o servidor, como mostra a Figura 3.10. A comunicação é feita usando o mecanismo primário de RPC que está embutido no *kit* de ferramentas do GWT. A referência a este mecanismo será feita como GWT-RPC, para diferenciá-los dos demais RPCs. A interface dos serviços também precisa se comunicar com o servidor, e faz isso através do protocolo GWT-RPC.

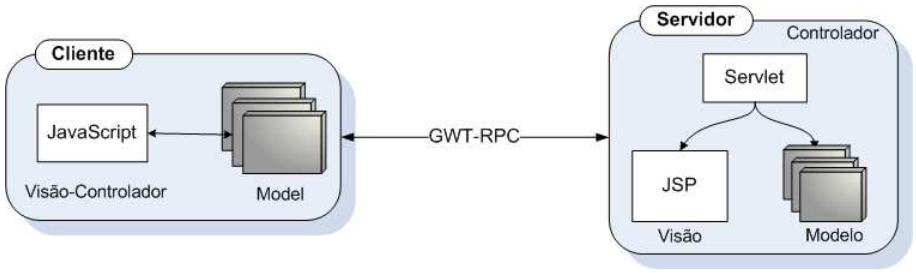


Fig. 3.10: Comunicação cliente/servidor [2].

Comunicação GWT-RPC

Há três partes fundamentais para uma aplicação GWT-RPC funcionar, como pode ser visto na Figura 3.11. O primeiro é o serviço que executa no servidor, o segundo é o cliente (navegador) que chama o serviço, e o terceiro são os dados que são transportados entre o cliente e o servidor. Tanto o lado cliente quanto o servidor são capazes de serializar e deserializar os dados que são trocados entre

eles.

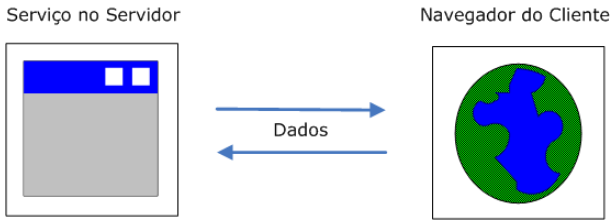


Fig. 3.11: As três partes da comunicação GWT-RPC [5].

A comunicação entre o cliente e o servidor na Figura 3.11 é iniciada pelo cliente e passada através de um objeto *proxy* fornecido pelo GWT. Este objeto serializa a requisição como um *stream* de texto e envia ao servidor. No servidor, a requisição é recebida por um *servlet* Java provido pelo GWT. O *servlet* então deserializa a requisição e a delega ao serviço adequado. Então, este serviço retorna o valor para a *servlet* GWT, e o objeto resultante é serializado e enviado de volta ao cliente. No lado cliente, ele é recebido pelo *proxy* GWT, que deserializa o objeto e retorna ao código da chamada. O GWT faz a serialização dos dados de forma automática.

Um conceito importante a ser entendido é a maneira assíncrona da comunicação GWT-RPC. Uma vez que o método do serviço remoto é chamado, o código continuará a executar. A comunicação assíncrona trabalha como um evento *handler*, que fornece uma rotina de *callback*, que é executada quando o evento ocorre. Neste caso, o evento é o retorno da chamada do serviço. A natureza assíncrona da comunicação traz algumas vantagens. A primeira é que a latência da rede juntamente com a execução de serviços complexos podem retardar a comunicação. Ao permitir que a comunicação ocorra de forma assíncrona, a chamada não ficará esperando a execução da aplicação terminar. Outra vantagem é poder emular um "*server-push*", que é um mecanismo onde o servidor envia dados para o cliente, sem que este tenha requisitado. Isto é usado em aplicação como o *chat*, por exemplo, onde o servidor precisa enviar mensagens para os seus clientes.

Cada serviço que precise fazer uma chamada GWT-RPC, é composto por um conjunto padrão de classes e interfaces. Entretanto, algumas dessas classes são geradas automaticamente e o desenvolvedor não precisa se preocupar com a sua implementação, e outras são importadas do *framework* GWT. O padrão dessas classes e interfaces é idêntico para todos os serviços que são implementados. Na Figura 3.12 é mostrado um diagrama genérico de todas as classes e interfaces que compõem um chamada GWT-RPC.

Como foi mencionado anteriormente, uma chamada remota é composta por três partes, e portanto a sua implementação também. A implementação de um serviço é dividida na criação do serviço, sua implementação e a chamada. A criação do serviço é responsável por criar uma interface Java no lado cliente, estendendo a interface `RemoteService`. Contudo, é necessário criar também uma interface assíncrona baseada na interface original do serviço. A seguir é mostrado um exemplo de um código de uma interface síncrona.

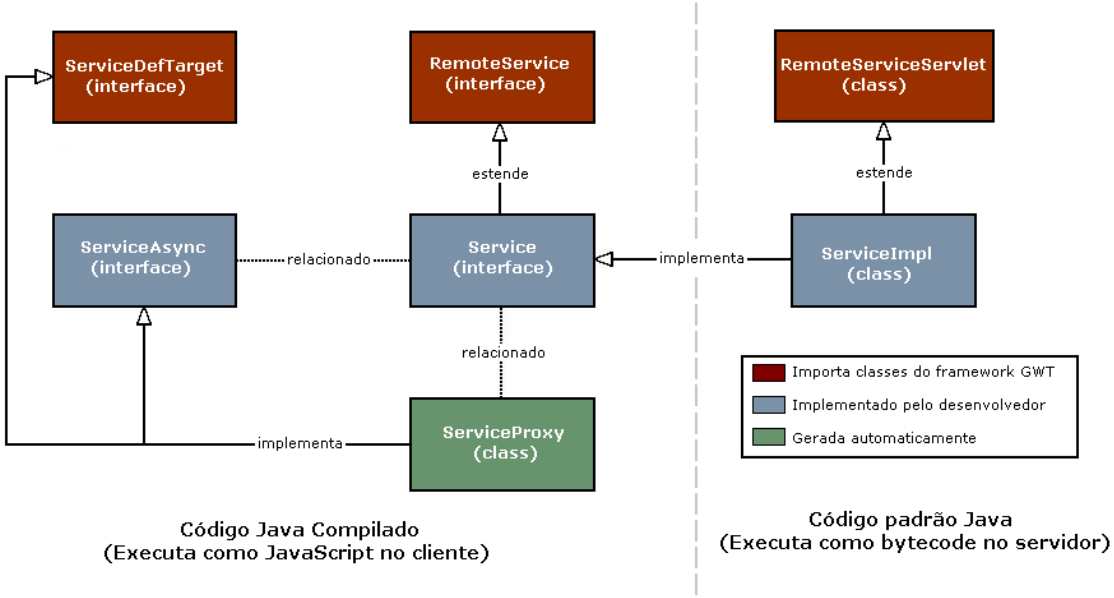


Fig. 3.12: Diagrama GWT-RPC [5].

```
public interface Service extends RemoteService
  public String[][] getPlugins ();
```

A natureza do método assíncrono requer que um objeto do tipo *callback* seja passado como parâmetro, que é responsável por notificar quando a chamada assíncrona termina. Por definição, o processo que invoca a chamada não pode ser bloqueado até que a chamada seja completada. Devido a isto, os métodos assíncronos não possuem retorno, ou seja, o retorno é sempre do tipo `void`. Quando a chamada termina, o retorno é passado ao processo que o chamou por meio do objeto de *callback*. Abaixo é mostrado um exemplo de um código de uma interface assíncrona.

```
public interface ServiceAsync extends RemoteService
  void getPlugin(AsyncCallback callback);
```

O relacionamento entre a implementação do serviço e suas interfaces síncrona e assíncrona é direto. Se a interface do serviço é chamada `com.google.gwt.sample.comlabfmw.impl.ServiceImpl`, então a sua interface assíncrona deve ser chamada `com.google.gwt.sample.comlabfmw.server.ServiceAsync`. As interfaces deverão estar localizadas no mesmo pacote e ter o mesmo nome, porém a interface assíncrona deverá ter o sufixo Async.

O processamento no lado servidor ocorre com a implementação do serviço, que é baseada na arquitetura de *servlets*. A implementação do serviço precisa estender a classe `GWTSpringController`, e deve implementar a interface síncrona do serviço associado. A interface assíncrona não é implementada. A classe `GWTSpringController` serializa o *handler* e invoca o método desejado.

O código da implementação de um serviço é mostrado abaixo.

```
public class ServiceImpl extends GWTSpringController
   implements Service {
        public String[][] getPlugin() {
           /** Lê o arquivo de configuração e
           *** retorna todos os serviços
           *** armazenados.
           ***/
           return (String[][]) plugins;
        }
     }
```

Para carregar a implementação do serviço é necessário criar uma *tag* `<servlet>` dentro do módulo XML da aplicação (`<servletname>.gwt.xml`). Além disso, é necessário assegurar-se de que o código do cliente esteja configurado para invocar o serviço, usando a URL que está mapeada no arquivo de configuração `web.xml`. O código cliente usa os seguintes passos para fazer uma chamada GWT-RPC:

1. Instancia a interface do serviço usando o método GWT.create().
2. Especifica um ponto de entrada para o *proxy* do serviço, usando ServiceDefTarget.
3. Cria um objeto *callback* assíncrono para ser notificado quando a chamada for completada.
4. Faz a chamada.

No código abaixo, é possível identificar cada um dos passos citados acima.

```
public void listPlugins () {
   ServiceAsync service =
      (ServiceAsync) GWT.create(Service.class);

   ServiceDefTarget target = (ServiceDefTarget) service;
   endpoint.setServiceEntryPoint("../../service");

   AsyncCallback callback = new AsyncCallback() {
     public void onSuccess(Object result) {
        answer = (String[][]) result;
     }

     public void onFailure(Throwable caught) {
        Window.alert("Error during invocation of the schedule
          service:"  + caught);
```

```
        }
    };

    service.getPlugin(callback);
}
```

3.3 Diagramas UML

Nesta seção são apresentados os diagramas UML do *framework* desenvolvido. O sistema é apresentado através de um conjunto de diagramas, onde cada diagrama se refere a uma visão parcial do mesmo. Todo o sistema foi implementado seguindo as bases da Programação Orientada a Objetos (POO), que traz benefícios quanto a modularidade, encapsulamento, reuso, simplificação e redução dos custo de manutenção.

3.3.1 Diagramas de Casos de Uso

O diagrama de casos de uso representa graficamente a interação dos usuários com as ações que o sistema realizará. No trabalho desenvolvido, os usuários poderão ser de dois tipos: usuário comum ou administrador. O usuário comum pode entrar no sistema (caso de uso "Login"), acessar sua conta (caso de uso "Minha Conta") ou o catálogo de serviços (caso de uso "Catálogo de Serviços"). O Administrador é uma generalização do usuário comum, o que significa que além de poder realizar as operações deste último, ainda pode interagir com os casos de uso adicionais "Gerenciar Usuários" e "Gerenciar Serviços". A Figura 3.13 apresenta o diagrama de casos de uso de trabalho aqui proposto.

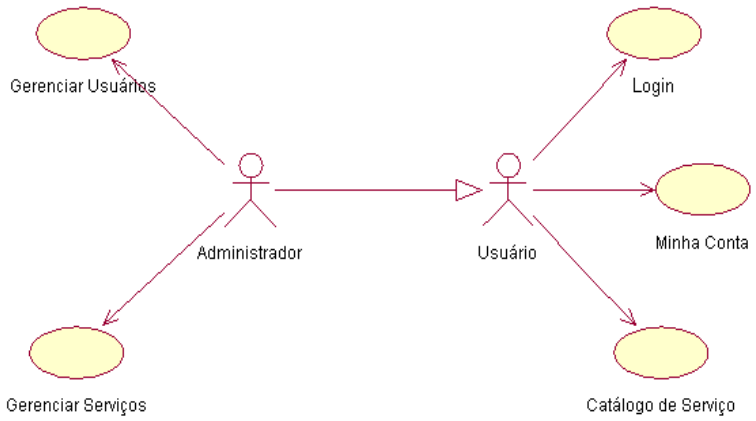


Fig. 3.13: Diagrama de Casos de Uso.

Cada Caso de Uso (funcionalidade do sistema), que foi apresentado no Diagrama será agora detalhado quanto as suas dependências funcionais e de negócio. No Apêndice A, estes casos de uso são

apresentados de modo sistematizado, seguindo-se um modelo padrão.

Login

Este Caso de Uso permite que o usuário possa entrar no sistema ou solicitar a criação de um novo usuário. Ele consiste dos subfluxos "Efetuar Login" e "Criar Conta" Se o usuário optar pelo primeiro subfluxo, o sistema solicitará o preenchimento do formulário (login e senha), e estando os dados corretos, o usuário estará logado ao sistema. Caso escolha o outro subfluxo, o usuário deverá preencher alguns dados cadastrais, que serão avaliados a *posteriori* pelo Administrador do Sistema que decidirá ou não pela cadastro do usuário. Nota-se que, para utilizar as demais funcionalidades do sistema, o usuário precisará estar logado.

A Figura 3.14 mostra a interface deste Caso de Uso, o usuário poderá entrar com o login e a senha, ou solicitar a criação de uma nova conta.

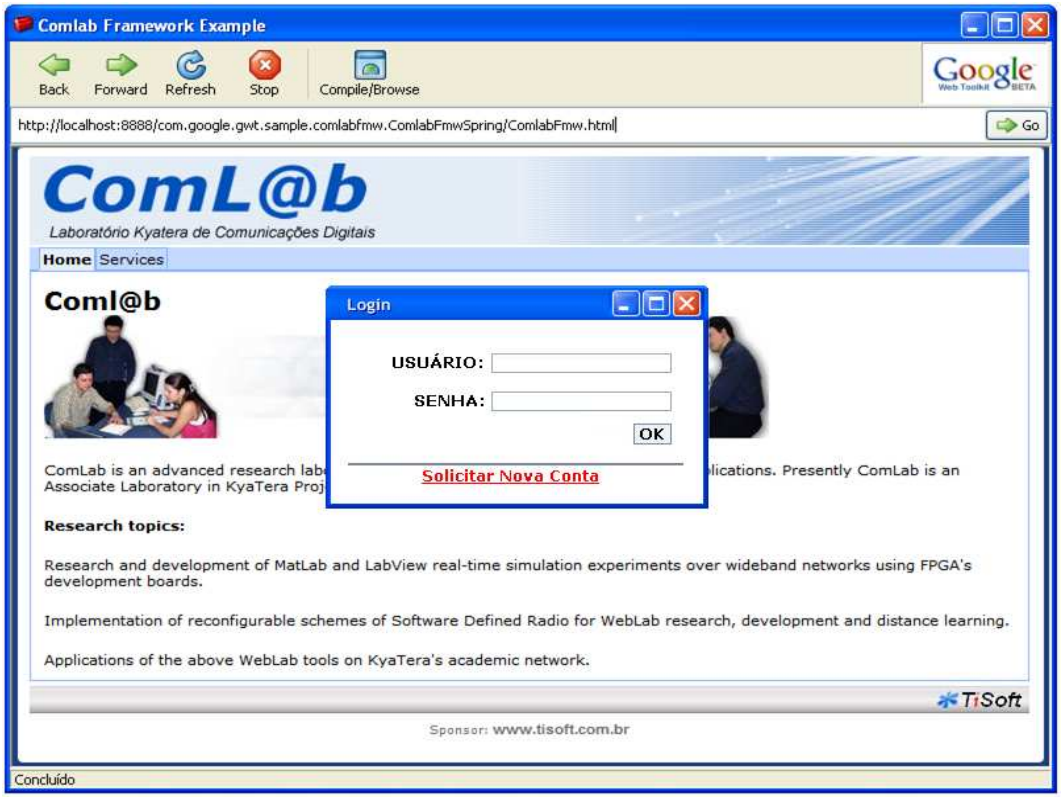


Fig. 3.14: Interface do Caso de Uso Login.

Minha Conta

Este Caso de Uso permite que o usuário possa editar (atualizar, trocar senha e/ou excluir) sua conta. Ele consiste dos subfluxos “Atualizar Conta”, “Alterar Senha” e “Excluir Conta”. Se o usuário solicitar a atualização da conta, o sistema irá exibir uma tela com as informação cadastradas (Figura 3.15), onde o mesmo poderá editá-la. Se os dados forem validados, o sistema irá exibir uma mensagem informando que as atualização ocorreram com sucesso, caso contrário o usuário será avisado de que campos estão com problemas.

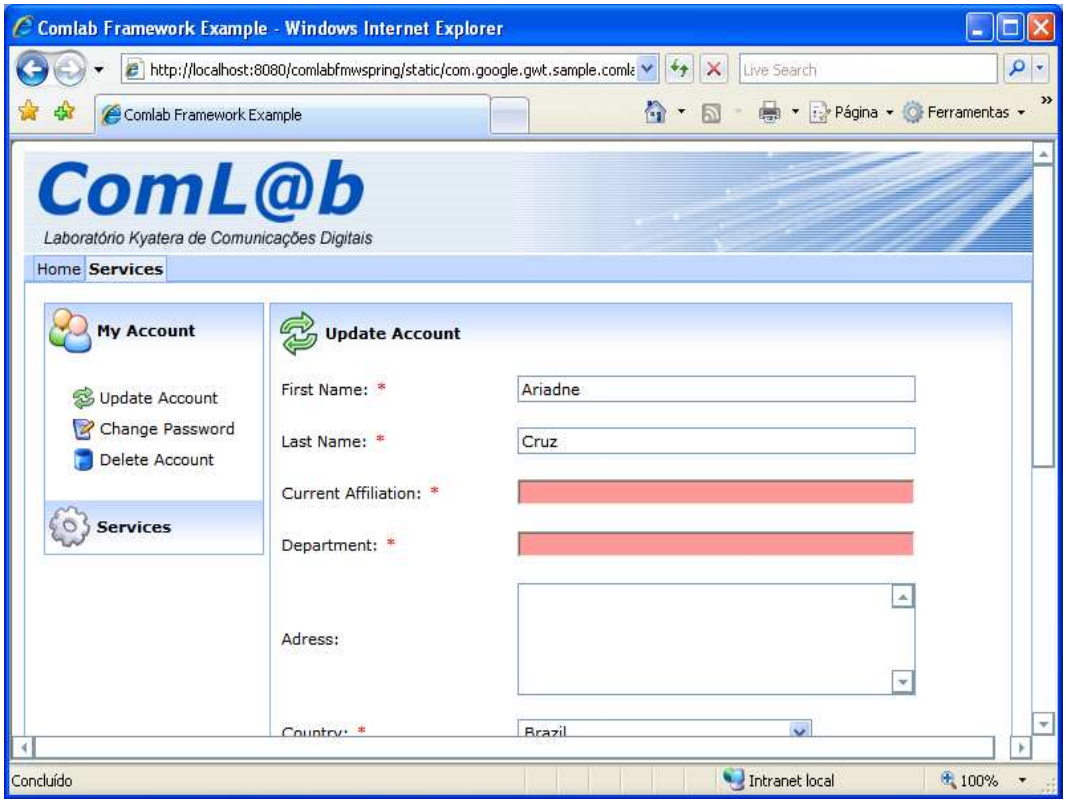


Fig. 3.15: Interface da Atualização da Conta.

Se o usuário escolher a opção de alterar senha, o sistema solicitará que o usuário informe a senha antiga e a nova senha (Figura 3.16). Durante a digitação da senha, o sistema informa ao usuário o nível de segurança da senha. Se as informações forem validadas, o sistema informa ao usuário que alteração ocorreu com sucesso, caso contrário os campos com problemas ficarão em evidência. Por último, o usuário poderá escolher a opção de excluir a conta (Figura 3.17 na próxima página). Antes de excluir a conta, o sistema pedirá a confirmação da exclusão, e caso afirmativo, a conta ficará com *status* de inativa. Além disso, todos os serviços agendados por este usuário serão excluídos da lista de agendamento.

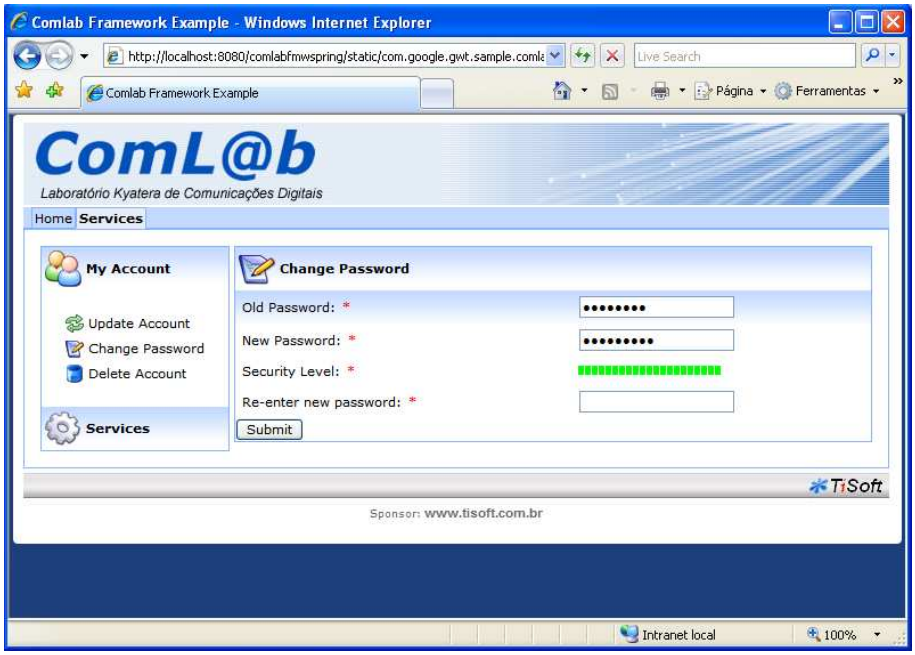


Fig. 3.16: Interface da Alteração de Senha.

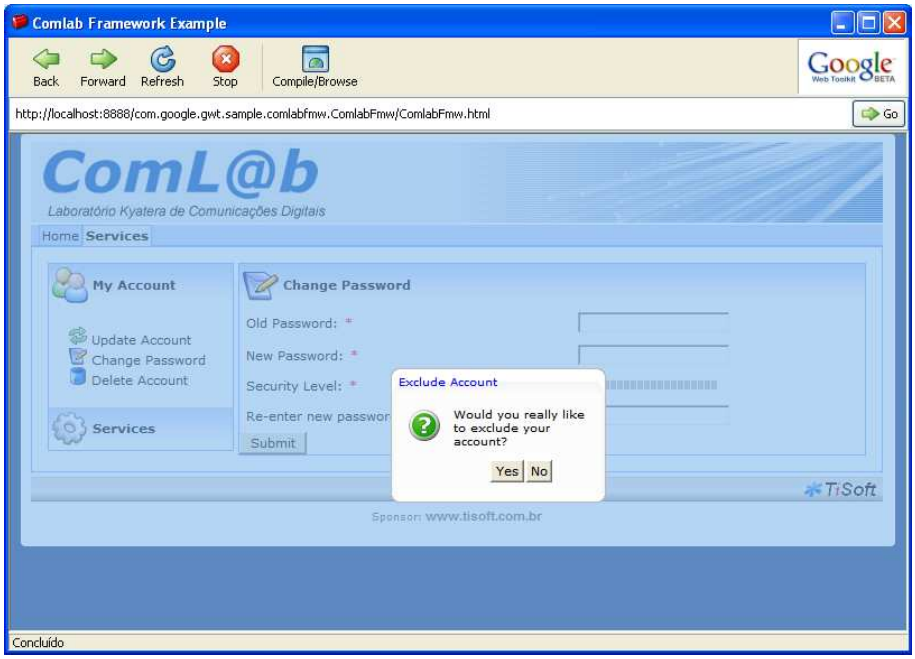


Fig. 3.17: Interface da Exclusão da Conta.

Catálogo de Serviços

O Caso de Uso Catálogo de Serviços é responsável por permitir que o usuário possa agendar os serviços e listar tanto os serviços agendados, quanto todos os serviços existentes no sistema. Ele

consiste dos casos de uso “Agendar Serviço” e “Executar Serviços”. O primeiro subfluxo exibe todos os serviços que podem ser agendados, ou seja, que estejam disponíveis e que precisem da agendamento (não sejam serviços livres) (Figura 3.18). Então, o usuário poderá incluir o serviço na sua lista de agendamento. Porém, antes ele poderá ver uma breve descrição do mesmo. Caso o usuário, realmente, opte pelo agendamento, o sistema irá solicitar o preenchimento de um formulário, que contém a data da execução, a hora de início da execução e o tempo estimado de duração. Se os dados forem validados, o usuário receberá uma notificação de que o agendamento foi efetivado e o serviço será incluído na lista de serviços agendados do usuário. Caso contrário, o sistema informará qual dos campos do formulário está com problemas.

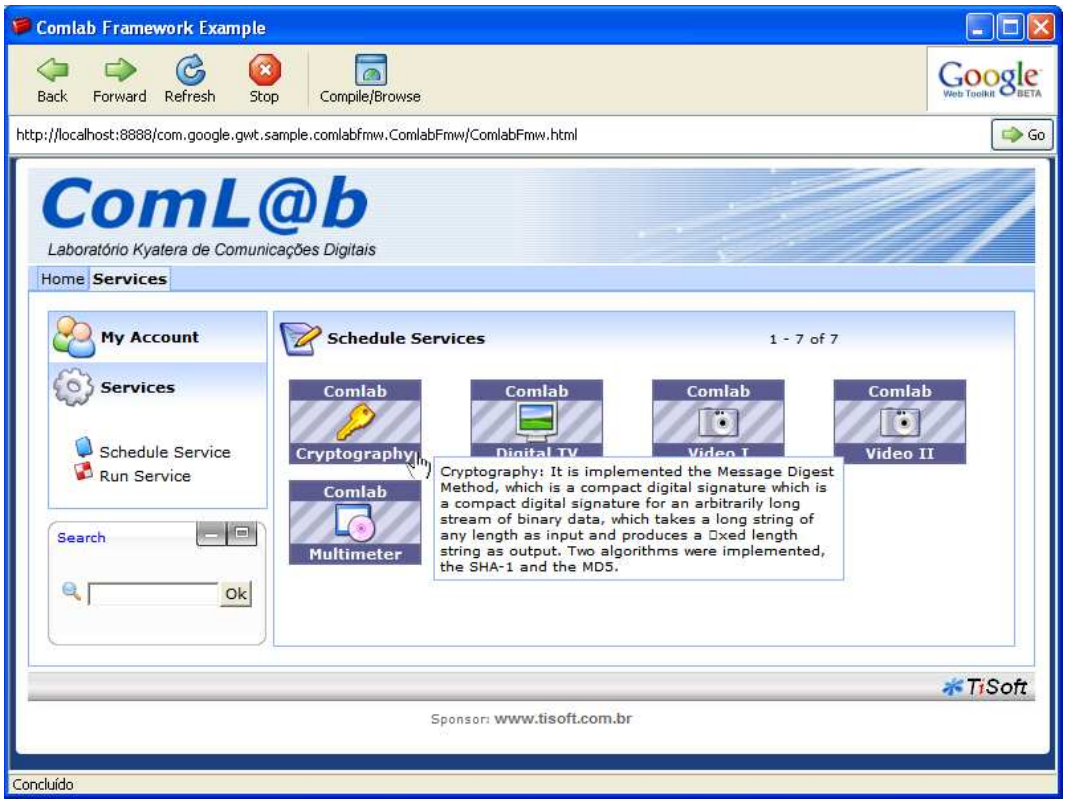


Fig. 3.18: Interface de Agendamento.

O outro subfluxo inicia-se quando o usuário desejar executar um serviço (Figura 3.19 na próxima página). Os serviços estão divididos em serviços e serviços restritos. Na lista de serviços estarão todos os serviços restritos que o usuário agendou, já na lista de serviços, estarão todos os serviços que não precisam de agendamento. Se os serviços estiverem disponíveis, o usuário poderá solicitar a execução e o sistema retornará a interface do serviço.

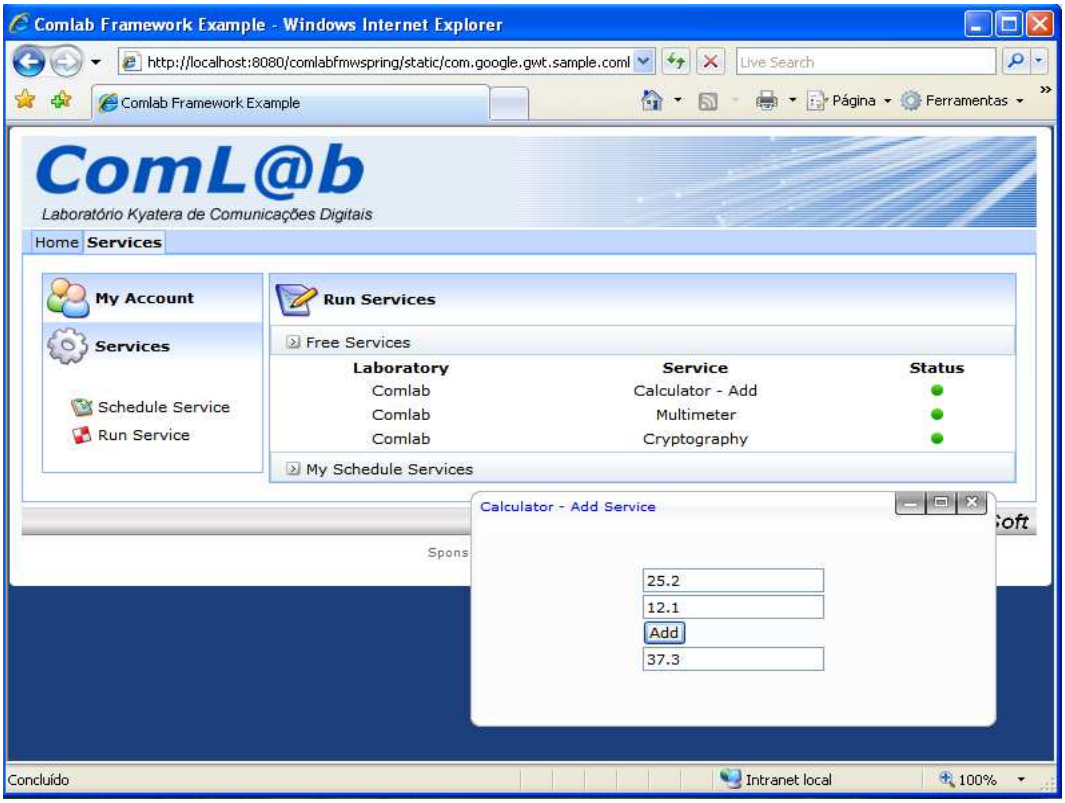


Fig. 3.19: Interface de Execução do Serviço.

Gerenciar Usuários

Este caso de uso só pode ser acessado por um usuário Administrador e permite que ele possa incluir, atualizar, excluir ou consultar um usuário cadastrado no sistema. Como pré-condições um usuário administrador precisa estar logado no sistema. Ele consiste dos subfluxos "Incluir Usuário" e "Editar Usuário". No primeiro subfluxo, quando o usuário Administrador solicitar a inclusão, o sistema exibirá um formulário (Figura 3.20 na página oposta), que deverá que ser preenchido pelo usuário (nome do usuário, afiliação, departamento, endereço, etc.). Após a confirmação, o sistema solicitará o *login* e a senha do Administrador, para então confirmar a inclusão, se houver qualquer problema em alguma etapa, o usuário será informado.

Já no subfluxo "Editar Usuário", o Administrador poderá escolher a opção atualizar, excluir ou consultar um usuário do sistema (Figura 3.21 na próxima página). Se a escolha for pela atualização, o sistema exibirá uma tela com os dados a serem alterados, que deverá ser preenchida. Se os dados forem validados, uma confirmação é enviada ao Administrador. Já se o Administrador escolher pela opção de excluir, o sistema informará se o usuário tem serviços agendados e solicitará a confirmação da exclusão. Caso afirmativo, o Administrador recebe uma notificação e todos os agendamentos feitos por aquele usuário são excluídos da lista de serviços agendados por usuário.

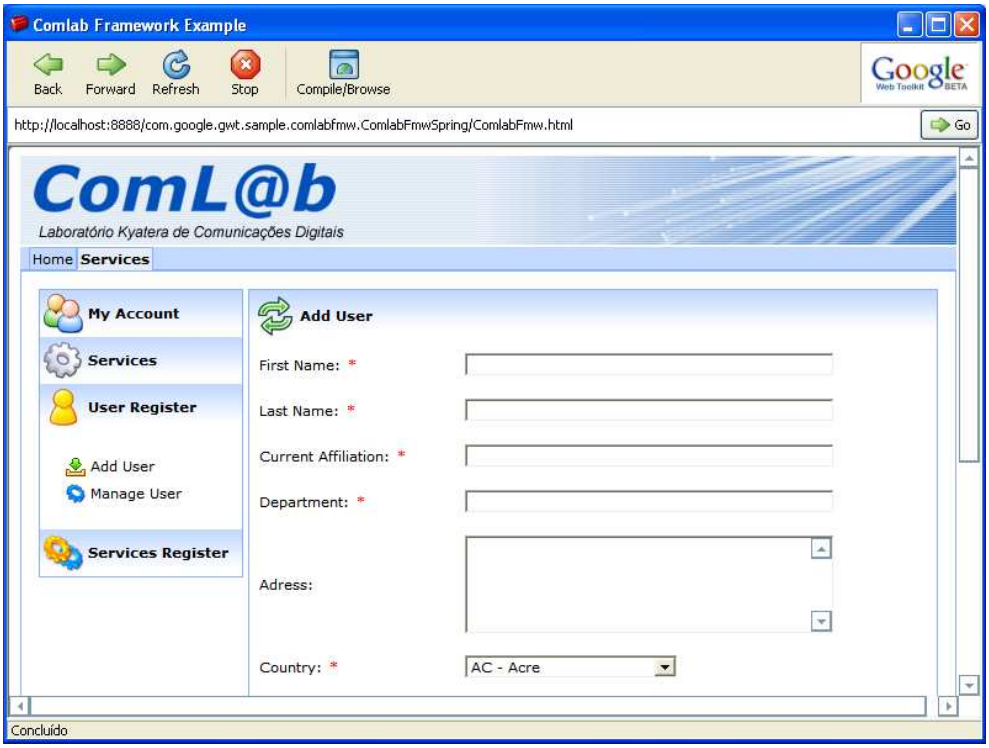


Fig. 3.20: Interface da Adição de um Usuário.

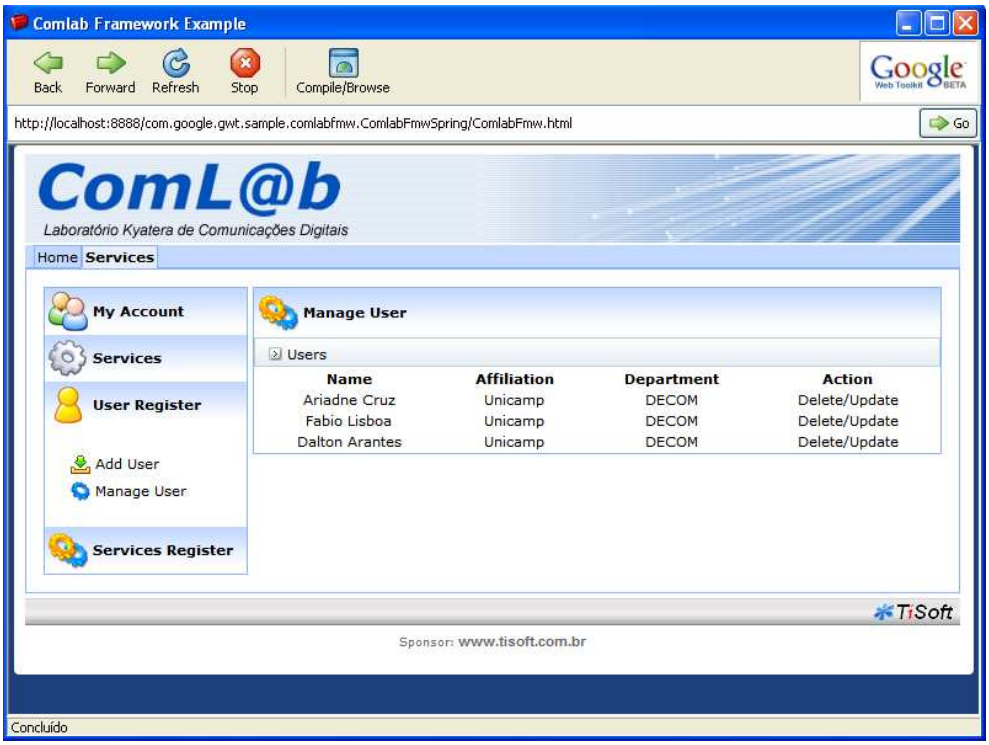


Fig. 3.21: Interface da Edição da Conta dos Usuários.

Gerenciar Serviços

Este caso de uso só pode ser acessado por um usuário Administrador e permite que ele possa incluir, atualizar, excluir ou consultar um serviço. Como pré-condições um usuário administrador precisa estar autorizado no sistema. Ele consiste dos subfluxos "Incluir Serviço" e "Editar Serviço". No primeiro subfluxo, quando o usuário Administrador solicitar a inclusão, o sistema exibirá um formulário (Figura 3.22), que deverá que ser preenchido pelo usuário (nome do serviços, descrição, data de inclusão, etc.). Após a confirmação, o sistema solicitará o *login* e a senha do Administrador, para então confirmar a inclusão. Se houver qualquer problema em alguma etapa, o usuário será informado.

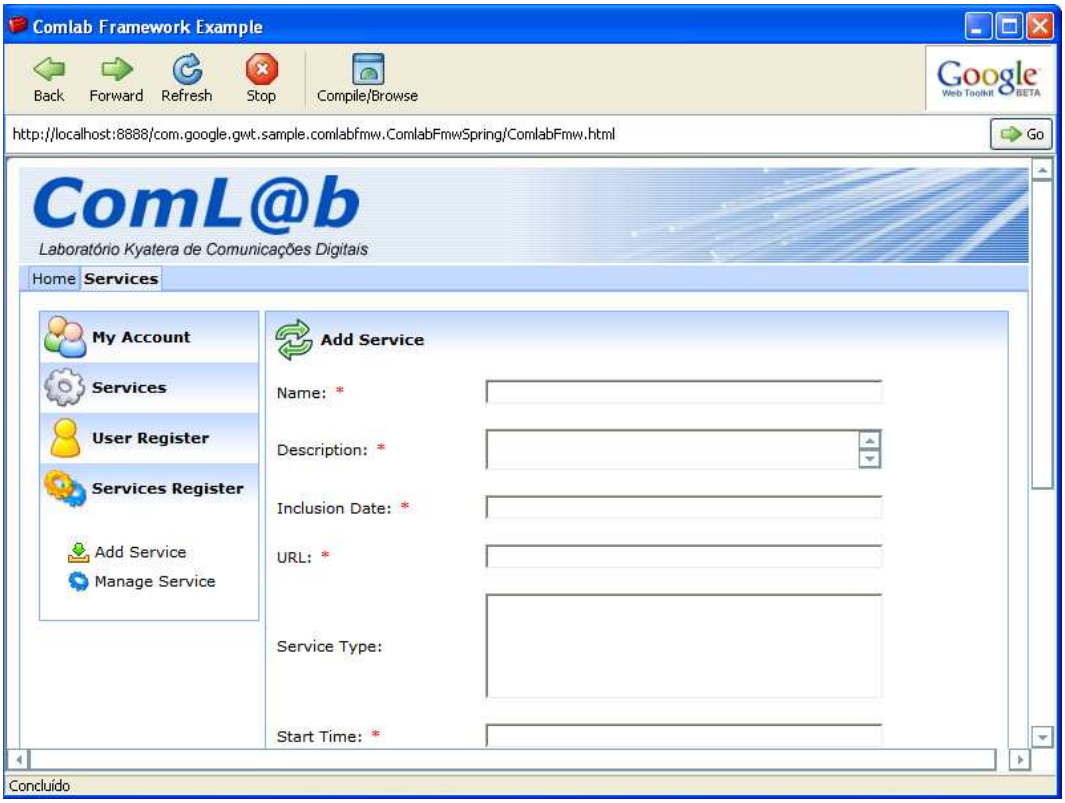


Fig. 3.22: Interface da Adição de um Serviço.

Já no subfluxo "Editar Serviço", o Administrador poderá escolher a opção atualizar, excluir ou consultar um serviço (Figura 3.23 na próxima página). Se a escolha for pela atualização, o sistema exibirá uma tela com os dados que podem ser alterados (nome, descrição, URL, ícone, disponibilidade), que deverá ser preenchida. Se os dados forem validados, uma confirmação é enviada ao Administrador. Já se o Administrador escolher pela opção de excluir, o sistema informará se existem ou não usuários que agendaram o serviço e solicitará a confirmação da exclusão, caso afirmativo, o usuário recebe uma notificação e o serviço é excluído da lista serviços agendados por usuário.

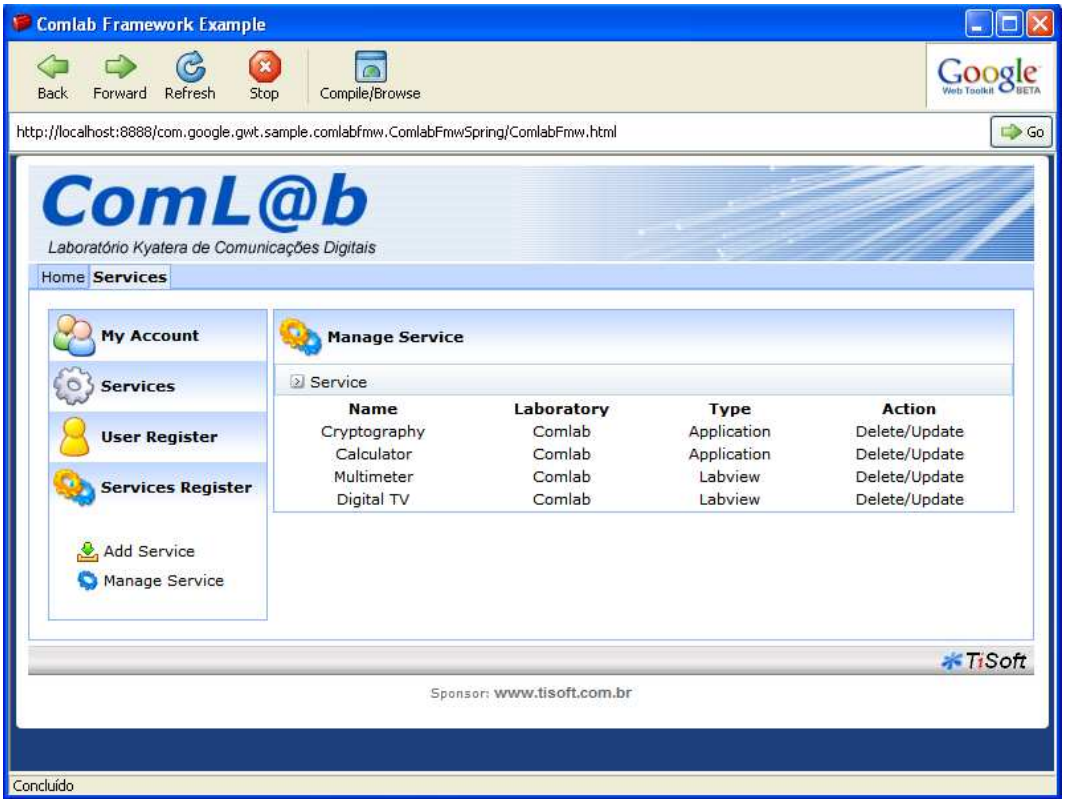


Fig. 3.23: Interface da Edição dos Serviço.

3.3.2 Diagramas de Classes

Os diagramas de classes são usados para descrever a estrutura do sistema e suas relações. É um diagrama muito importante, pois é nele que encontramos informações sobre métodos, atributos, nomes de funções e como as classes serão integradas. A maioria dos outros diagramas vai utilizar as classes definidas no diagrama de classes. De acordo com [60], classes são abstrações que especificam a estrutura comum e o comportamento de um conjunto de objetos, e os objetos são instâncias de classes que são criadas, modificadas ou destruídas durante a execução de um sistema.

Todas as classes implementadas no sistema estão representadas no diagrama da Figura 3.24 na página seguinte e são organizados em camadas, que foram descritas na Seção 3.2. A classe "ComlabFmw" é a principal classe da UI, que é responsável por chamar a classe "Shortcuts", que é uma classe menu, responsável por chamar as demais classes da interface, conforme a solicitação do usuário. Na camada Modelo de Comunicação, responsável pela comunicação entre cliente/servidor, temos as classes "Service" e "ServiceAsync". Essas classes têm unicamente operações abstratas, ou seja, nenhum método implementado e nenhum atributo. Dessa forma se for necessário mudar a implementação dessas classes, não será necessário modificar as camadas de UI, e nem a camada Modelo de Comunicação. A classe "ServiceAsync" responde pela assincronicidade na comunicação.

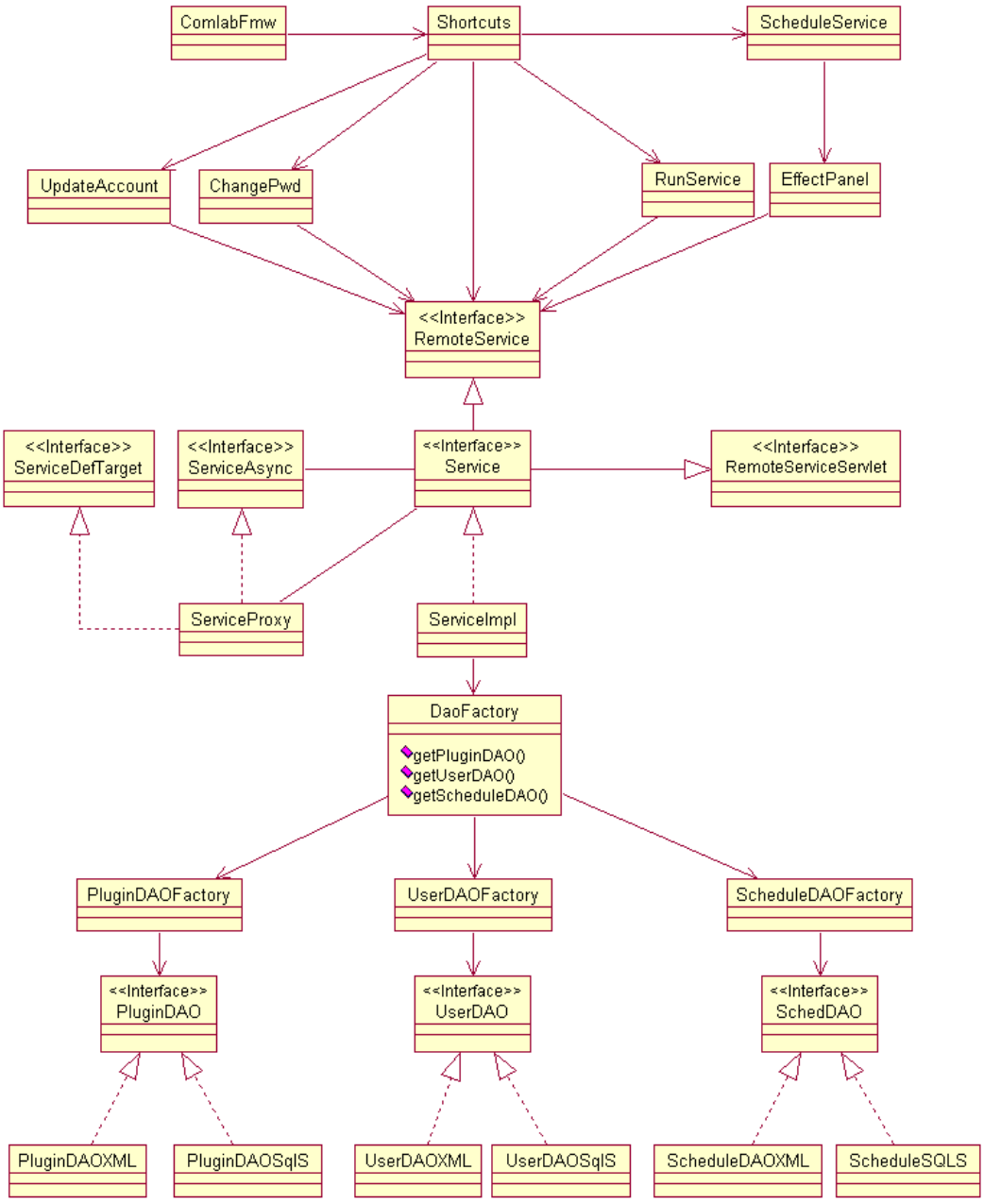


Fig. 3.24: Diagrama de Classes da Plataforma.

A implementação das classes de comunicação é feita na camada de Gerenciamento de *Plugins*. Inicialmente, o sistema tem a classe "ServiceImpl". Contudo, como o sistema é baseado no conceito de *plugins*, novas classes poderão ser adicionadas a esta camada e a camada Modelo de Comunicação.

A camada de persistência é responsável pela consulta a base de dados, sendo representada pela classe “DaoFactory”, que é responsável por instanciar os DAOs e passá-los às classes que os necessitem. A grande vantagem desta classe é a possibilidade de desacoplar a parte da aplicação responsável por tratar as operações de banco de dados das outras camadas da aplicação. A classe “DaoFactory” retorna um das DAO implementadas PluginDAOFactory, UserDAOFactory ou ScheduleDAOFactory. Essas classes são responsáveis pela consulta na base de dados dos serviços, usuários e serviços agendados por usuário, respectivamente. Cada uma dessas classes pode ter diferentes implementações para a consulta a base de dados, que podem ser arquivos XML, SQL Server, Oracle, etc. Com essa abstração é possível mudar a base de dados sem precisar fazer grandes alterações no código, basta modificar a chamada na classe “DaoFactory”. Como essa divisão em camadas, é garantida uma boa modularização pela utilização de POO e Padrões de Projeto.

3.3.3 Diagramas de Pacotes

O Diagrama de Pacotes representa os pacotes (conjuntos de classes agrupadas para formar uma unidade de execução concisa), classes e suas dependências na execução de tarefas do sistema. Seu principal objetivo é agrupar classes em pacotes. Como mencionado anteriormente, o sistema foi desenvolvido de forma modularizada, facilitando o desenvolvimento e futuras manutenções, como pode ser observado na Figura 3.25.

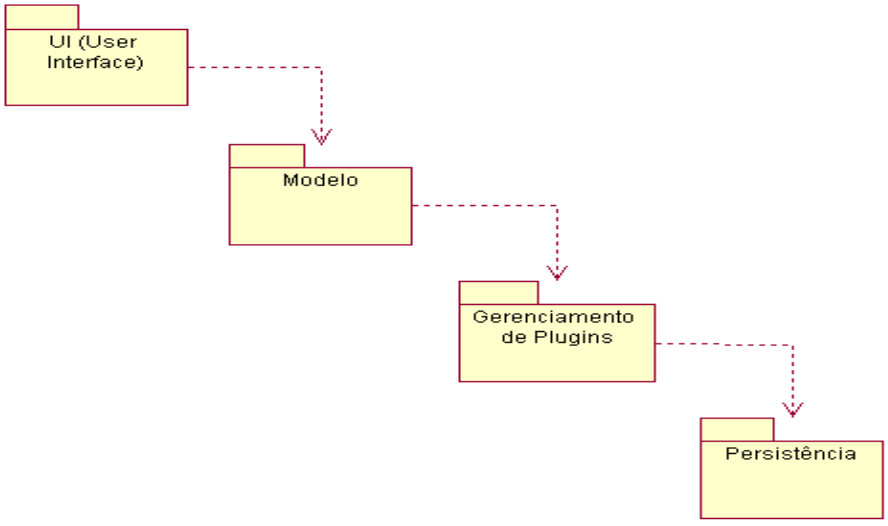


Fig. 3.25: Diagrama de Pacotes.

Cada pacote apresenta uma única dependência com a camada inferior. Ou seja, o pacotes de classes da UI que cuida da interface da aplicação depende funcionalmente da classes do Modelo de Comunicação para cumprirem suas atividades. Por outro lado, as classes do Modelo de Comunicação dependem da classe Gerenciamento de *Plugins*, que são responsáveis pela implementação. Já essas

dependem das classes da camada de Persistência.

3.3.4 Diagramas de Seqüência

O diagrama de seqüência mostra a colaboração dinâmica entre os vários objetos do sistema, apresentando a seqüência de ações executadas por ele e a seqüência de mensagens enviadas entre os objetos. Ele prioriza a ordem das mensagens no tempo. Cada objeto é apresentado com uma linha vertical que representa a "vida" do objeto. A caixa acima desta linha representa o objeto.

A linha de vida do objeto será representada por uma caixa, quando o objeto tem o controle ou está aguardando o retorno de uma operação. Caso contrário, ela é representada por uma linha tracejada. O diagrama de seqüência representa o tempo de cima para baixo e não há necessidade de representar o número de seqüência das mensagens, pois eles estão implícitos no eixo vertical. Na Figura 3.27 é mostrado o diagrama de seqüência de como a plataforma é baixada para a máquina do cliente. Ou seja, o que acontece no instante em que o usuário acessa o site da plataforma.

Os passos para carregar uma aplicação GWT iniciam-se quando o usuário acessa o site da aplicação. Neste momento, o arquivo HTML principal é baixado para o navegador do cliente, que processa o arquivo HTML e verifica se é necessário carregar e executar o arquivo *JavaScript* `gwt.js`. Este arquivo *JavaScript* processa o HTML procurando pela *tag* `gwt:module`, para determinar qual arquivo *JavaScript* precisa ser carregado.

Para cada *tag*, cujo o conteúdo seja `className`, o *script* `gwt.js` cria um novo elemento *IFrame* que tenta carregar o arquivo `className.nocache.html`. Um vez carregado, ele é responsável por identificar o tipo do navegador (IE, Firefox, Netscape, etc.) e baseado nisto, trocar o arquivo HTML para um específico para o tipo de navegador. Quando o arquivo específico é carregado para o navegador, o método `gwtOnLoad()` invoca a inicialização da aplicação. A partir deste ponto, o usuário pode interagir com a aplicação GWT. Na Figura 3.26 é mostrado o diagrama de seqüência de uma aplicação Ajax, e define passo-a-passo a seqüência dos eventos.

Uma interação Ajax começa com um objeto *JavaScript* chamado XMLHttpRequest, que permite um *script* no lado cliente executar uma solicitação HTTP (*HyperText Transfer Protocol*), e interpretar a resposta XML do servidor. A aplicação Ajax, então, instancia o objeto XMLHttpRequest, onde estão configurados a requisição (GET ou POST) e a URL de destino. Como a natureza do Ajax é assíncrona, uma função de *Callback* é registrada e disparada com o XMLHttpRequest. Então, o controle retorna para o navegador, e a função de *Callback* irá chamar quando a resposta do servidor chegar.

No Servidor de Aplicações, qualquer solicitação chega como um HttpServletRequest. Depois que os parâmetros da solicitação são interpretados, a *servlet* invoca a lógica de aplicação adequada, serializando sua resposta dentro de um documento XML e envia através do HttpServletResponse. De volta no lado cliente, a função de *callback* registrada no XMLHttpRequest é agora invocada para processar o documento XML retornado do servidor. Finalmente, a interface do usuário é atualizada a partir dos

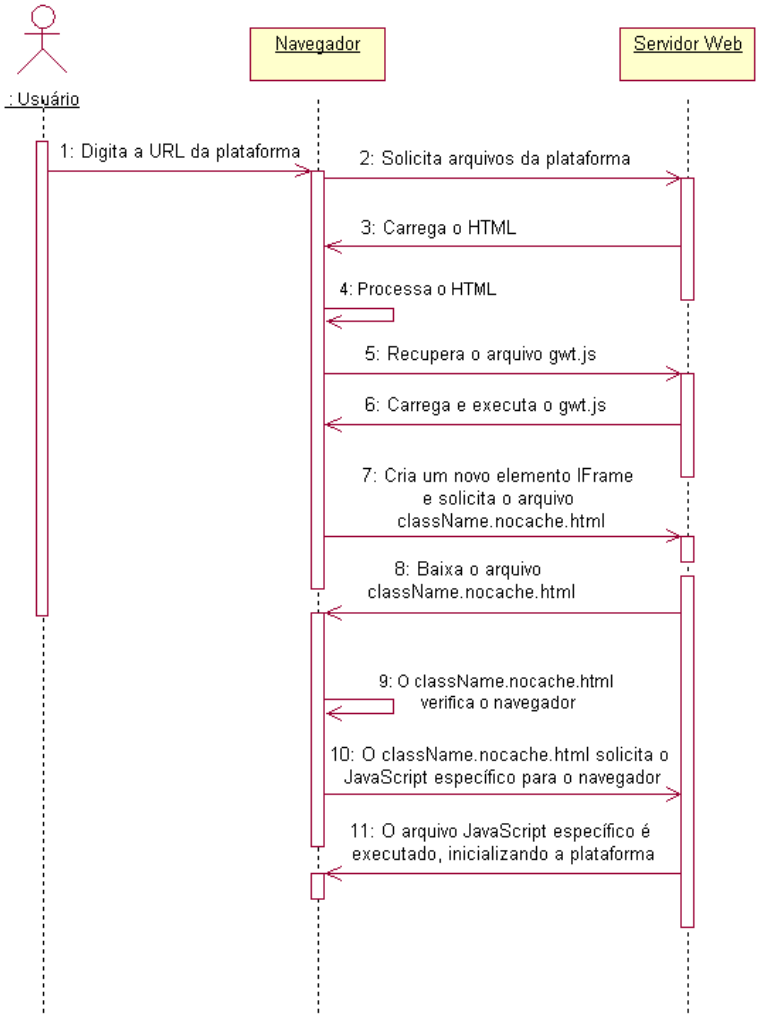


Fig. 3.26: Diagrama de Seqüência Inicial de uma Aplicação GWT.

dados enviados do servidor.

3.4 Implementação

Esta seção detalha o funcionamento de parte do sistema responsável pelo agendamento e execução de um determinado serviço pelo usuário. Na Figura 3.28 é mostrada a interface de agendamento, onde usuário acessa a plataforma e a seguir escolhe um dos serviços que deseja agendar.

Os serviços são exibidos em blocos de dezesseis serviços. Quando há mais de dezesseis serviços, a plataforma mostrará botões de avançar e retornar para facilitar a localização dos serviços. Além disso, a plataforma tem um campo de busca, onde o usuário poderá entrar com o nome do serviço

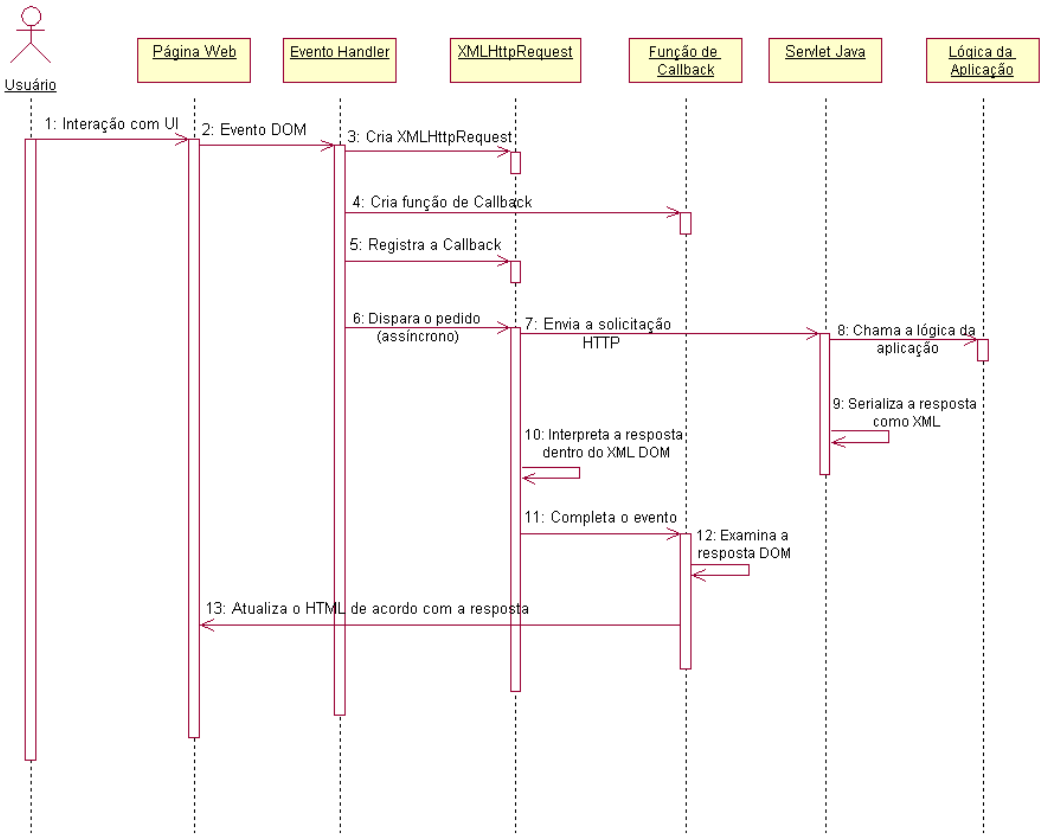


Fig. 3.27: Diagrama de Seqüência de uma aplicação Ajax.

que deseja encontrar e caso aquele serviço esteja cadastrado, o sistema retornará o serviço. O usuário poderá também visualizar a descrição do serviço. Basta passar o *mouse* sobre o ícone do serviço e aparecerá uma janela mostrando a descrição do mesmo.

Após feita a escolha de que serviço será agendado, é exibida uma janela onde o usuário deverá entrar com as informações referente a data de agendamento, horário de início e o tempo estimado para a execução (Figura 3.29).

A *servlet* responsável faz uma chamada ao serviço *Web* passando os parâmetros do agendamento. O serviço verifica se há disponibilidade, e caso positivo armazena o serviço na base de dados do usuário, enviando uma mensagem de que a operação ocorreu com sucesso. Caso contrário, o usuário será informado que a operação não ocorreu. Após isso, o serviço se encontra na lista de serviços agendados do usuário, e estará disponível pelo tempo solicitado.

Para executar o serviço, o usuário precisará selecionar a opção Executar no menu. Se o serviço já estiver disponível para execução, o usuário poderá clicar sobre botão de execução. Dessa forma, uma nova janela aparecerá com a tela do serviço selecionado. Esta tela foi desenvolvida com auxílio da CIL, evitando, assim, a recompilação da plataforma.

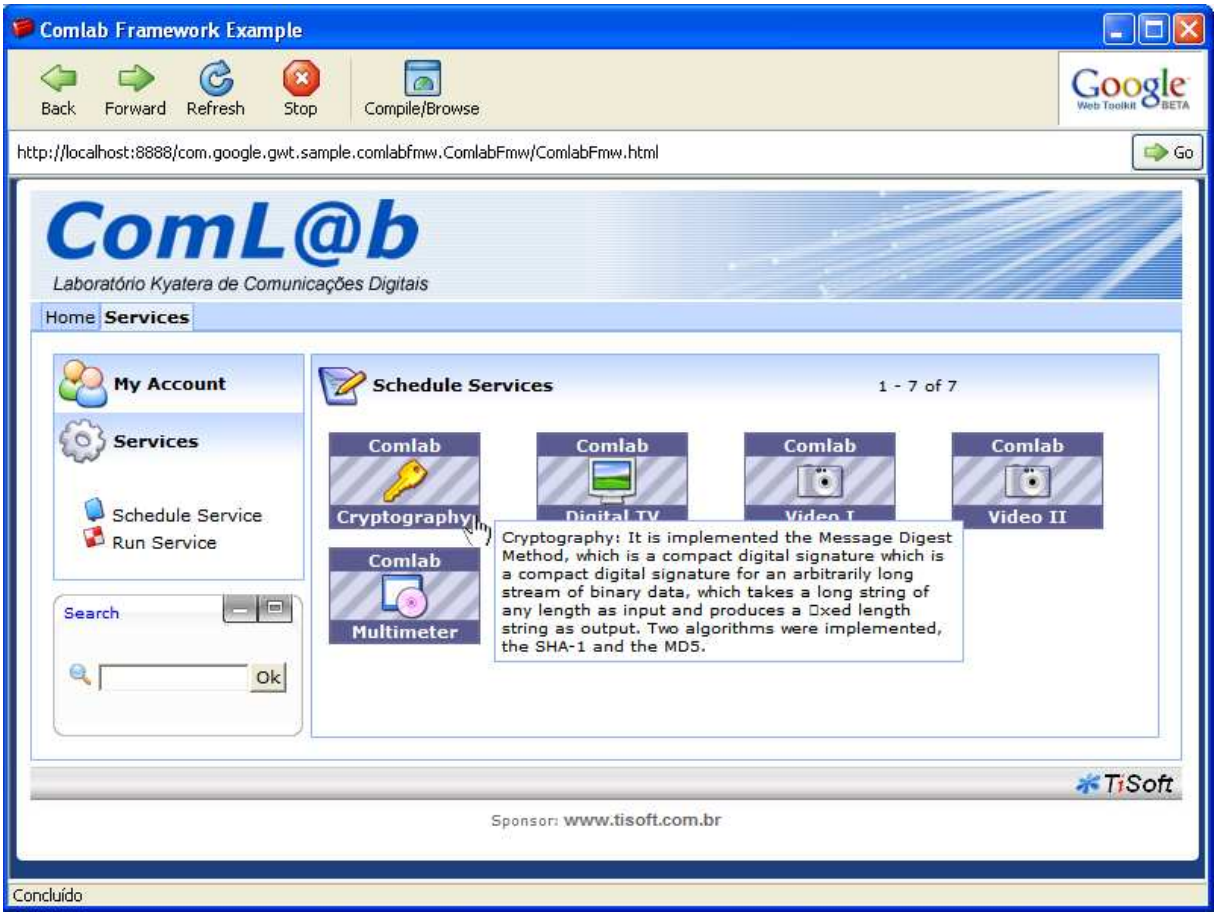


Fig. 3.28: Tela de Agendamento - Escolha do Serviço.

A seguir, é descrita a parte do código do cliente que chama o serviço. No código abaixo, a Linha 01 representa o código do *plugin*, ou seja, o código que foi criado com o auxílio da CIL. A Linha 02 é responsável por criar o interpretador para o *plugin*. A classe `Cvm_light` é criada para interpretar os comando escritos em CIL e o código na Linha 09 é responsável por inserir o objeto `button` na página HTML.

```
01:  final Cvm_light cvm = new Cvm_light(interfaces[pp][1]);
02:  Widget button = cvm.execute(0);
03:  FramesManager framesManager =
       new FramesManagerFactory().createFramesManager();
04:  GInternalFrame service =
       framesManager.newFrame(plugins[pp][1]+ " Service");
05:  service.setWidth(400);
06:  service.setHeight(400);
07:  service.setVisible(false);
08:  service.setTheme("alphacube");
```

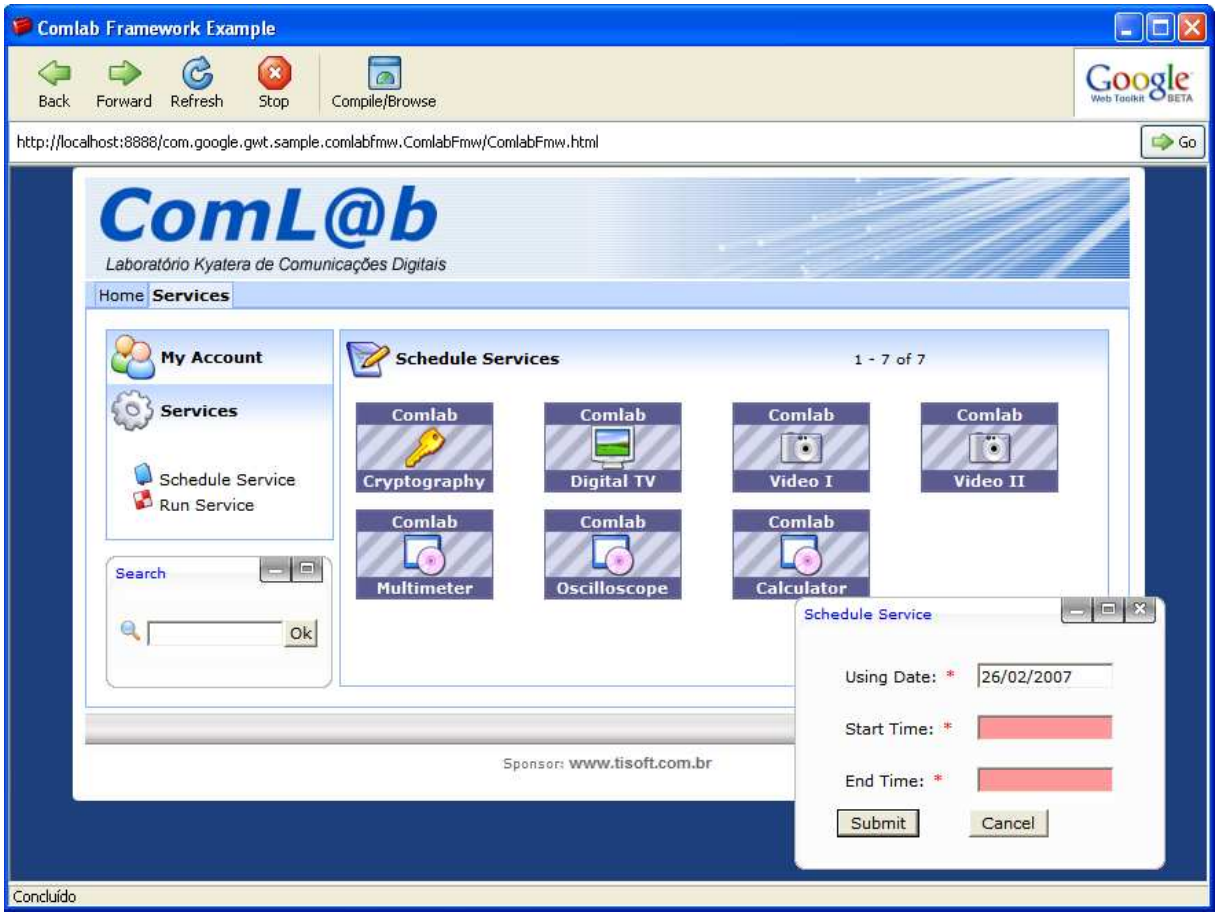


Fig. 3.29: Tela de Agendamento - Informação para o Agendamento.

```
09:   service.setContent(button);
10:   GwmUtilities.diplayAtScreenCenter(service);
```

O código acima é responsável apenas por chamar os serviços, porém é necessário que os mesmos tenham uma interface, que é criada através da CIL. Na Seção 3.2.1 foi mostrado um código simples de como uma interface é criada usando a CIL. Todo o sistema foi implementado em Java. Esta linguagem foi escolhida por ser orientada a objetos e ser bem eficiente na construção das páginas *Web*, através do *framework* GWT, viabilizando um maior reuso do código, com alta qualidade e baixa propensão a erros. O código gerado é multiplataforma e cria códigos otimizados para os diferentes tipos de navegadores existentes. O ambiente de desenvolvimento utilizado foi o Eclipse, que é um IDE de código aberto e tem uma excelente relação entre qualidade e facilidade de uso. Sua instalação é simples, basta descompactar o arquivo `.zip` em qualquer pasta e ele já estará pronto para utilização. Uma excelente referência para programação Java pode ser encontrada em [61].

3.5 Situação Atual do Desenvolvimento

A plataforma proposta foi desenvolvida em Java tanto no lado cliente quanto no servidor, e os serviços que a integram foram implementados utilizando uma linguagem intermediária para a criação da interface. A lógica dos serviços foi implementada usando as linguagens Java e Labview. A linguagem intermediária CIL é responsável por construir a interface dos serviços em tempo de execução, sem a necessidade de recompilar a plataforma. Ela encontra-se na sua versão 0.3 e oferece alguns objetos GWT para a criação de interfaces e suporte a RPC.

Atualmente, foram implementados completamente os módulos "Minha Conta" e "Catálogo de Serviços", descritos na Seção 3.3.1. A inclusão de novos serviços e usuários está sendo feita de forma manual, ou seja, alterando diretamente a base de dados. Como trabalhos futuros, sugere-se a implementação dos módulos Gerenciamento de Serviços e de Usuários, que irá automatizar a inclusão de novos serviços e usuários.

3.6 Considerações Finais

Este capítulo apresentou a arquitetura do trabalho proposto, descrevendo o projeto, que é dividido em quatro módulos principais (Serviços IU, Modelo de Comunicação, Gerenciamento de *Plugins* e Persistência), os diagramas UML, a sua implementação, a atual situação do desenvolvimento. O capítulo ainda descreve como novos serviços podem ser incluídos e como é feita a criação da interface dos mesmos utilizando a CIL. Os códigos da CIL e a chamada do serviço no lado cliente foram ilustrados para um melhor entendimento de como ocorre a inclusão de novos *plugins*. Com o detalhamento da arquitetura, fica claro que o sistema proposto é extensível, e que a inclusão de novos serviços a plataforma, não causará instabilidade ao sistema, o que propicia um sistema altamente robusto.

Capítulo 4

Serviços

Neste capítulo apresentamos a descrição das tecnologias usadas para a implementação dos serviços que são integrados a plataforma, entre elas destacam-se a linguagem Labview e o já conhecido Java. Após feita esta descrição, é feita uma apresentação dos serviços que atualmente integram a plataforma e que foram usados como prova de conceito.

4.1 Tecnologias

Com a crescente necessidade de expandir as aplicações *desktops* para *Web*, há um grande número de tecnologias para implementação desses serviços. Tradicionalmente, a programação no lado servidor tem sido executada usando Perl, Python, C++, ou alguma outra linguagem, para criar programas CGI (*Common Gateway Interface*), sistemas mais sofisticados têm aparecido. Estes incluem servidores *Web* baseados em Java que permitem a execução de toda a programação do lado servidor em Java escrevendo os assim chamados *servlets*.

A versatilidade de uma linguagem está intimamente relacionada com sua complexidade. Uma linguagem complicada em sua aprendizagem permite em geral realizar um espectro de tarefas mais amplo e mais profundo. A escolha da linguagem Java foi bastante natural por apresentar diversas vantagens, como ser uma linguagem portável, robusta, segura e possibilitar um maior reuso do código. Já que tanto o cliente quanto o servidor seriam implementados nessa linguagem. Além dos serviços desenvolvidos em Java, a plataforma agrega serviços implementados em outras linguagens e que possuam uma interface para *Web*, como é o caso da linguagem Labview [59].

4.1.1 Linguagem de Programação Java

Java é uma linguagem de programação orientada a objeto. Diferentemente das linguagens convencionais, que são compiladas para código nativo, a linguagem Java é compilada para um "*bytecode*" que é executado por uma máquina virtual. Esta linguagem tem como objetivo construir programas que possam ser utilizados em qualquer plataforma ou computador, independentes de sistema opera-

cional ou ambiente de desenvolvimento. A linguagem Java foi construída baseada nas técnicas de orientação a objetos, multitarefa e computação distribuída.

Esta linguagem de programação é apoiada por duas grandes empresas, a Sun e a IBM, que têm criado inúmeras melhorias na linguagem, além de criar ferramentas que agilizam o trabalho dos desenvolvedores. É uma linguagem portável, ou seja, pode ser utilizada em diversos ambientes, desde supercomputadores até celulares e computadores de mão. Com isso, a utilização de Java vem crescendo continuamente, já que devido a sua alta portabilidade, ela pode ser utilizada em diferentes computadores e dispositivos.

Além da portabilidade e o fato de ser uma linguagem orientada a objetos, o Java é uma linguagem robusta e segura, pois possui um controle sobre toda a área de memória do programa. Esta característica auxilia a evitar erros que poderiam ser ocasionados por uma programação não adequada, além de auxiliar na segurança da máquina. O Java possui um sistema de tratamento de exceções e permite a integração com códigos nativos implementados nas linguagens C/C++, além disso é uma linguagem dinâmica, *multithread* e distribuída.

Há duas opções para o desenvolvimento de aplicações em Java, o J2SE (Java 2 *Standard Edition*) e o J2EE, o primeiro é utilizado para desenvolvimento de aplicações *desktop*, enquanto o segundo é o ambiente de desenvolvimento para Servidores de Aplicações baseados na *Web*. Java possui ainda diversas ferramentas de desenvolvimento, cujo objetivo é agilizar o desenvolvimento e aumentar a produtividade dos programadores e analistas. Dentre as principais ferramentas de desenvolvimento para Java, o Eclipse é a única totalmente baseada em software livre.

4.1.2 Labview

O Labview é uma linguagem de programação gráfica desenvolvida pela *National Instruments* e está na sua versão 8.20. Com o desenvolvimento do Labview, foi introduzido o conceito de instrumentação virtual. Suas principais áreas de atuação são a medição e a automatização. A programação é feita através do modelo de fluxo de dados, que oferece a esta linguagem vantagens para a aquisição e manipulação dos dados.

Os programas Labview são chamados de instrumentos virtuais (VI - Virtual Instrument). Eles são compostos pelo painel frontal, que contém a interface, e o bloco de diagramas, que contém o código gráfico do programa. O programa não é lido por um interprete, mas sim compilado. Deste modo, a sua performance é comparável com de linguagens de alto nível. Uma grande vantagem da programação Labview é que usuários, que não sejam especialistas em programação, podem desenvolver aplicações sem qualquer dificuldade [58].

Linguagens de programação como C e Basic usam funções e subrotinas como elementos de programação, já o Labview usa uma VI. Várias VIs podem ser agrupadas para criar um aplicação de grande escala. Os dados podem ser adquiridos ao conectar a aplicação com vários dispositivos de *hardware*, e suas medições podem ser apresentadas por meio de interfaces gráficas, páginas *Web* ou

arquivos de banco de dados.

O Labview é um software proprietário e foi adquirido pelo projeto *KyaTera*. Foram adquiridas 200 licenças, que foram distribuídas a todos os laboratórios integrantes do projeto, que o solicitaram e tiveram o seu orçamento aprovado pela FAPESP [14].

4.2 Serviços Integrados à Plataforma

A robustez e a extensibilidade podem ser alcançadas pela integração de diferentes tipos de serviços à plataforma. Atualmente, há quatro serviços que integram a plataforma, e que fazem parte da prova de conceito do trabalho proposto, que são descritos a seguir.

4.2.1 Serviço de Criptografia

Com o crescimento dos serviços para *Web*, cresce, também, a necessidade de segurança com os dados que serão trocados entre o cliente e o servidor. Dessa forma, criptografar os dados antes de sua transferência torna-se uma necessidade. A linguagem Java, tem como uma de suas característica, a segurança (Ver Seção 4.1.1), ela oferece uma API para trabalhar com a criptografia dos dados, que utiliza os métodos, Message Digest e a Assinatura Digital.

O método Message Digest consiste de funções *hash* que geram códigos de tamanho fixo, em uma única direção, a partir de dados de tamanho arbitrário. A API Java implementa os algoritmos MD5[62] e SHA-1[63]. Já as Assinaturas Digitais servem para autenticar o remetente da informação, garantindo assim que o dado é confiável.

Na aplicação desenvolvida, foi implementado apenas o método Message Digest, incluindo os algoritmos MD5 e SHA-1. A aplicação foi desenvolvida em Java e sua interface foi construída através da CIL. Esta aplicação é um exemplo simples de criptografia de dados. O usuário entra com um texto, seleciona um dos algoritmos de criptografia e o sistema retorna o dado criptografado na forma de um *array* de bytes em representação hexadecimal. A interface do serviço de criptografia é mostrada na Figura 4.1.

4.2.2 Calculadora

Uma calculadora é um dispositivo utilizado para auxiliar na realização de cálculos numéricos. A implementação de uma calculadora é uma tarefa relativamente fácil e muito comum em ambientes acadêmicos, onde o aluno precisa se familiarizar com uma nova linguagem. Esta calculadora foi implementada em Java e realiza as quatro operações matemáticas básicas (adição, subtração, multiplicação e divisão).

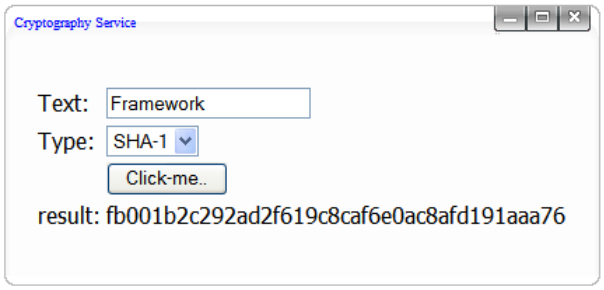


Fig. 4.1: Interface do Serviço de Criptografia.

A calculadora foi implementada para suportar operações com números flutuantes. O usuário entra com dois valores numéricos e escolhe uma das quatro opções, e o sistema retorna o resultado da operação. A figura 4.2 exibe a interface do serviço.

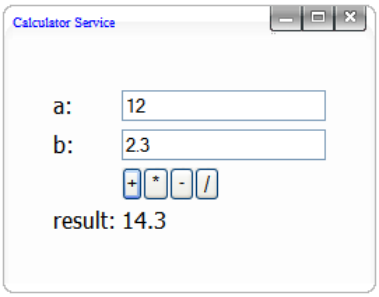


Fig. 4.2: Interface da Calculadora.

4.2.3 Multímetro

Um multímetro é um instrumento destinado a medir e avaliar grandezas elétricas. Ele incorpora diversos instrumentos de medida elétrica em um único aparelho, como o voltímetro, o amperímetro e o ohmímetro, por padrão. Porém dependendo do fabricantes outros opcionais podem ser adicionados, como o capacímetro, o frequencímetro e o termômetro, por exemplo. As funções são escolhidas através de uma chave seletora, que se encontra abaixo do painel principal do equipamento. Um multímetro é amplamente utilizado para pesquisar defeitos em aparelhos eletro-eletrônicos devido a sua simplicidade e portabilidade.

Este *WebLab* permite o controle de um Multímetro Digital Wavetek Metermann 235. Sua interface foi desenvolvida em Labview, e além de todas as funcionalidade de um multímetro convencional, ele inclui uma interface RS232 que permite ao multímetro se conectar com o computador e possui sensores que lêem a temperatura em tempo real. O monitoramento da temperatura é muito importante no ambiente industrial, mas aqui é especialmente interessante na integração de diferentes serviços ao *framework* proposto, para o desenvolvimento da *e-Science*. Na Figura 4.3 é mostrado o multímetro

remoto executando sob a plataforma.

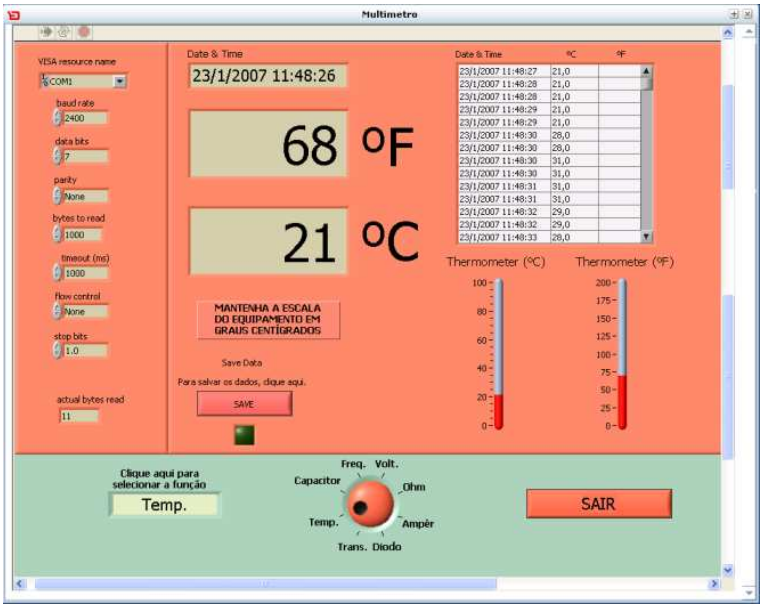


Fig. 4.3: Interface do Multímetro Digital.

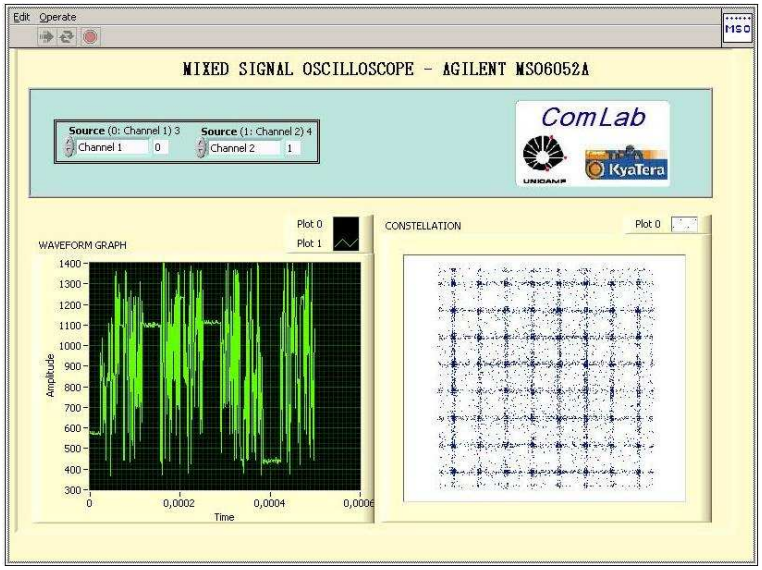


Fig. 4.4: Interface do Aplicativo de TV Digital.

4.2.4 Sistema de TV Digital

Uma plataforma computacional foi desenvolvida em Labview, permitindo o monitoramento, o controle remoto e a configuração de diferentes equipamentos conectados no Sistema Avançado de

TV Digital, que foi desenvolvido pelo *ComLab*, do qual a autora é uma das integrantes. O conjunto de equipamentos inclui um Analisador de Espectro, para estimação espectral em tempo real de um canal de TV Digital; um Osciloscópio Digital para análises e visualização da constelação de um sinal da TV Digital; um Gerador de Sinais Vetoriais para controle do clock e sincronização da transmissão e recepção de sistemas; e finalmente um Frequencímetro Digital para o monitoramento da freqüência do clock. A interface do Osciloscópio é mostrada na Figura 4.4.

Esta plataforma está atualmente integrada ao *framework* proposto e hoje faz parte dos serviços disponíveis pelo *ComLab*. Agora pretende-se tornar este experimento disponível para outras universidades no Brasil, por meio das redes de alta velocidade RNP/GIGA e *KyaTera*.

4.3 Considerações Finais

Este capítulo apresentou a descrição das tecnologias utilizadas para a implementação dos serviços e seus principais benefícios. O capítulo mostrou também uma descrição dos serviços que atualmente integram a plataforma e que foram usados como prova de conceito.

Capítulo 5

Conclusão

Este capítulo apresenta uma discussão sobre os principais pontos do trabalho realizado, através de uma retrospectiva do que foi desenvolvido, das contribuições dadas e dos resultados alcançados. O capítulo ainda contém sugestões de trabalhos futuros que poderão ser realizados para melhorar a infra-estrutura do trabalho proposto.

5.1 Retrospectiva

Neste trabalho foram descritos os princípios, metodologias e tecnologias usadas no desenvolvimento de uma plataforma de integração de *WebLabs*, alguns serviços utilizados como validação e a futura utilização de trabalhos colaborativos. Esta plataforma estará disponível para integração de *WebLabs* na rede *KyaTera* da FAPESP, o que permite o acesso remoto a equipamento laboratoriais. Como não é possível prever todos os serviços de *WebLabs*, foi adotada uma solução baseada em *plugins*, que torna possível inserir, remover e atualizar serviços sem recompilar a plataforma. Essas alterações, todavia, só terão efeito após a reinicialização da plataforma. Com o uso de *plugins* o *framework* torna-se muito mais extensível, robusto e flexível. Estas características são obtidas pela adoção dos Padrões de Projeto DIP e EO, que descrevem técnicas baseadas em *plugins*.

Outro importante ponto considerado no desenvolvimento do *framework* foi a usabilidade pobre das aplicações *Web* tradicionais. Por esta razão, o paradigma Ajax foi utilizado, permitindo o desenvolvimento de interfaces ricas. Estas interfaces se aproximam bastante das interfaces de aplicações *desktop*, porém sem a necessidade de qualquer instalação no navegador ou na máquina cliente. A plataforma foi implementada usando o *framework* GWT, que se mostrou o mais adequado entre os *frameworks* Ajax estudados para desenvolvimento deste trabalho. O GWT permite a implementação inteira da aplicação em Java, em ambos os lados cliente e servidor, possibilitando um maior reuso de código. Além disso, o GWT gera códigos otimizados para os diferentes tipos de navegadores existentes, propiciando códigos menores e mais rápidos que a maioria dos outros *frameworks*.

A criação de novos serviços pode ser dividida em duas partes: a implementação do serviço no lado servidor e a interface do serviço no lado cliente. No lado servidor, é utilizado o *framework*

Spring, que em seu conceito introduz a inversão de controle, evitando a recompilação da plataforma quando um serviço é adicionado ou atualizado. No lado cliente, é necessário criar a interface para o serviço. Entretanto, a versão corrente do GWT (Versão 1.4) não é capaz de gerar códigos em tempo de execução. Devido a isso, para a implementação da interface dos *plugins*, foi necessária a utilização de uma linguagem intermediária. Sendo assim, através da CIL e do *framework* Spring é possível criar a interface dos serviços no lado cliente e a implementação dos serviços no lado servidor sem a necessidade de recompilar a plataforma.

Além do GWT e da CIL, foram utilizadas as bibliotecas Scriptaculous e o GWM, responsáveis por efeitos como "arrasta e puxa" e o gerenciamento de janelas, respectivamente.

5.2 Trabalhos Futuros

Para trabalhos futuros, propõe-se a implementação dos módulos "Gerenciar Usuários" e "Gerenciar Serviços" descritos na Seção 3.3.1. Tal implementação permitirá que a adição e a manutenção de usuários e serviços seja feita de forma automatizada. Dessa forma, quando um novo serviço for adicionado ou atualizado, o administrador não precisará se preocupar com a localização exata dos arquivos, já que a plataforma se encarregará disto.

Propõe-se, também, o aprimoramento da estratégia de agendamento dos serviços, de forma que novos recursos possam ser incorporados ao agendamento, no momento em que o serviço é adicionado à plataforma. Com isso, ao adicionar um serviço, o administrador poderá escolher um agendamento padrão ou incluir novos recursos, tornando o *framework* ainda mais extensível. Esta característica possibilitará que o *framework* adapte-se aos mais diferentes tipos de serviço e agendamento. Como próximas etapas, pretende-se ainda realizar testes funcionais e de desempenho no sistema, fazer um estudo e uma análise sobre segurança em aplicativos *Web*, implementar objetos DAO para o postgreSQL e um mecanismo para fazer com que a plataforma tenha alta disponibilidade e escalabilidade, além de oferecer suporte ao gerenciamento de sessões do usuário.

Outro ponto importante é a implementação de novas versões da CIL, garantindo que grande parte dos objetos existentes no GWT estejam disponíveis através da CIL. Isto permitirá a criação de interfaces de serviços cada vez mais sofisticadas. Adicionalmente, será desenvolvido um tradutor que converterá o código Java para CIL, possibilitando que todo o código da plataforma seja implementado utilizando única linguagem (Java). Esta abordagem proporcionará uma maior reutilização de código e a simplificação do desenvolvimento e a manutenção do sistema.

Referências Bibliográficas

[1] J. J. Garrett. *Ajax: A New Approach to Web Applications*. Disponível em: http://www.adaptivepath.com/publications/essays/archives/000385.php, Fevereiro 2005.

[2] R. Hanson e A. Tacy. GWT in Action. Manning, 2007.

[3] Plataforma Echo2. Disponível em: http://www.nextapp.com/platform/echo2/echo/.

[4] C. Walls e R. Breidenbach. Spring in Action. Manning, 2005.

[5] Plataforma Google. Disponível em: http://code.google.com.

[6] D. Booth, H. Haas, F. McCabe, E. Newcomer; M. Champion, C. Ferris, e D. Orchard. *Web Service Architecture*. Disponível em: http://www.w3c.org/TR/ws-arch, Fevereiro 2004.

[7] A. Arsanjani, B. Hailpen, J. Martin e P. Tarr. Web Services: Promisses and Compromisses. ACM Queue - Vol 1, No. 1, 2003.

[8] G. Ceschini, G. Pereira, R. Cunha, C. Cugnasca e A. Saraiva. *The Greenhouse Web Lab: implementation of a remotely monitored and controlled greenhouse*. *III Workshop Tidia - FAPESP*, pages 198–200, Novembro 2006.

[9] C. Jesus, D. Ferreira, A. Cruz e R. Giordano. *WebLabs in Biochemical Engineering: Mass Transfer Experiments*. *III Workshop Tidia - FAPESP*, pages 201–203, Novembro 2006.

[10] H. Okajima, L. Ledel, H. Fragnito e H. da Rocha. *WebLab Development Using a LabView and Java Integrated Solution for Kyatera Network*. *III Workshop Tidia - FAPESP*, pages 204–206, Novembro 2006.

[11] G. Valença, E. Barrientos, K. Campos, M. Mori e D. Arantes. *Weblabs in Chemical Engineering: High- and Low-Pressure Chemical Processes*. *III Workshop Tidia - FAPESP*, pages 210–212, Novembro 2006.

[12] S. Szpigel, E. Souza, E. Antonio, J. Filho e F. Júnior. *WebLab for the Remote Measurement of an Optical Fiber Attenuation Coefficient*. *III Workshop Tidia - FAPESP*, pages 216–218, Novembro 2006.

[13] C. Giannini, E. Júnior, S. Hilario, S. Amancio, T. Cavalcante e A. Hirakawa. *A Weblab for Monitoring and Characterization of Stingless Bee (Meliponini)*. *III Workshop Tidia - FAPESP*, pages 225–227, Novembro 2006.

[14] Página da Rede KyaTera. Disponível em: http://www.kyatera.fapesp.br.

[15] D. Crane. *AJAX em Ação*. Manning Publication, 2005.

[16] C. Gross. AJAX Patterns and Best Practices. Apress, 2006.

[17] A. A. Cruz, F. A. L. Gomes, F. A. M. Cardoso, E. B. Martin e D. S. Arantes. *Development of a Robust and Flexible WebLab Framework based on AJAX and Design Patterns*. Proceedings of 7th IEEE International Symposium on Custer Computing and the Grid (CCGrid 07), pp. 811-816, Maio, 2007, Rio de Janeiro, Brasil., 2007.

[18] P. Kuchana. Software Architecture Design Pattern in Java. Auerbach Publishers Inc, 2004.

[19] E. Gamma et al. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.

[20] E. Gamma e K. Beck. Contributing to Eclipse Principles, Patterns, and Plug-ins. Addison Wesley Professional, 2004.

[21] Página Web da FAPESP. http://www.fapesp.br.

[22] Plataforma Joomla. Disponível em: http://www.joomla.org/.

[23] Plataforma Mambo. Disponível em: http://www.mamboserver.com/.

[24] Plataforma Plone. Disponível em: http://plone.org/.

[25] J. Nielsen. Usability Engineering. Morgan Kaufman, 1994.

[26] J. C. Redish e J. S. Dumas. A Practical Guide to Usability Testing. Intellect Books, 1999.

[27] *Eclipse Platform Technical Overview*. *Technical Report*, IBM, Disponível em: www.eclipse.org/whitepapers/eclipse-overview.pdf, Julho 2001.

[28] T. A. Bastos, R. J. M. Silva, A. B. Raposo e M. Gattass. ViRAL: Um Framework para o Desenvolvimento de Aplicações de Realidade Virtual . *In VII Symposium on Virtual Reality. São Paulo, SP, Brasil*, pages 51–62, 2004.

[29] C. Prazeres, M. G. Pimentel e C. A. C. Teixeira. *A Document-Based Approach for WebLab Configuration. III Workshop Tidia - FAPESP*, pages 125–127, Novembro 2006.

[30] E. G. Guimarães, A. T. Maffeis, J. L. Pereira, B. G. Russo, E. Cardozo e M. F. Magalhães. REAL: A Virtual Laboratory Built for Mobile Robot Experiments. *IEEE Transactions on Education*, pages 46(1):37–42, Fevereiro 2003.

[31] E. G. Guimarães. *Um Modelo de Componentes para Aplicações Temáticas e Ubíquas*. PhD thesis, Faculdade de Engenharia Elétrica e de Computação, Unicamp, Campinas, SP, Brasil, 2004.

[32] T. A. Bastos, A. B. Silva e M. Gattass. Um Framework para o Desenvolvimento de Aplicações de Realidade Virtual Baseadas em Componentes Gráficos. *XXV Congresso da Sociedade Brasileira de Computação. São Paulo, SP, Brasil*, 2005.

[33] A. R. Andrade, E. V. Munson e M. G. Pimentel. *A Document-Based Approach to the Generation of Web Applications*. *DocEng*, pages 45–47, 2004.

[34] A. Mesbah e A. Deursen. *An Architecture Style for AJAX*. Proceedings of the 6th Working IEEE/IFIP Conference on Software Architecture (WICSA'07). IEEE Computer Society, 2007.

[35] J. Garrett. AJAX: A new approach to web applications. Adaptive Path, 2005.

[36] J. Garret. *Ajax: A New Approach to Web Applications.* http://www.adaptivepath.com/publications/essays/archives/000385.php.

[37] A. Sayar, M. Pierce e G. Fox. *Integrating AJAX Approach into GIS Visualization Web Services*. Proceedings of IEEE International Conference on Internet and Web Applications and Services (ICIW'06). February, 23-25, 2006. Guadeloupe, French Caribbean., 2006.

[38] Plataforma Qooxdoo. Disponível em: http://qooxdoo.org.

[39] Backbase. Disponível em: http://www.backbase.com.

[40] Marco Cantu. *Dominando o Delphi 2005: a Bíblia*. Prentice-Hall, 2005.

[41] Scriptaculous. Disponível em: http://script.aculo.us/.

[42] TinyMCE. Disponível em: http://tinemce.moxiecode.com/.

[43] Cay Horstmann e David M. Geary. Core Java Server Faces. Alta Books, 2005.

[44] Ted Husted, Cedric Dumoulin, George Franciscus e David Winterfeldt. Struts in Action. Manning Publication, 2003.

[45] Martin Bond, Dan Haywood, Debbie Law, et al. *Aprenda J2EE: com EJB, JSP, Servlets, JNDI, JDBC e XML em 21 dias*. Makron Books, 2003.

[46] S. J. Metsker. Design Pattern Java Workbook. Addison-Wesley, 2002.

[47] F. Buschmann et al. Pattern-Oriented Software Architecture - A System of Pattern. John Wiley & Sons, 1996.

[48] D. Broemmmer. J2EE Best Practices: JavaTM Design Patterns, Automation and Performance. Jonh Willey and Sons Inc, 2002.

[49] S. Yacoub e H. Ammar. Pattern-Oriented Analysis and Design. Addison-Wesley, 2003.

[50] M. Fowler. *Inversion of Control Containers and the Dependency Injection Pattern.* http://www.itu.dk/courses/VOP/F2006/9_injection.pdf.

[51] J. Crupi et al. Core J2Ee Patterns. Prentice Hall PTR, 2003.

[52] *Factory Method Pattern*. Disponível em: http://en.wikipedia.org/wiki/Factory_method_pattern.

[53] Hibernate. Disponível em: http://hibernate.org/.

[54] Ibatis. Disponível em: http://ibatis.apache.org/.

[55] *Using the GWTSpringController*. http://g.georgovassilis.googlepages.com/usingthespring-gwtcontroller.

[56] G. Costa. *Modelo de Web Services - Como Desenvolver Aplicações em uma Nova Arquitetura de Software*. In: PBTR - Promon Tecnologia Business and Tecnology Review. Ano 02 - No. 4.

[57] M. Hendricks, B. Galbraith et al. Profissional Java Web Services. Rio de Janeiro: Alta Books, 2002.

[58] Plataforma Labview. Disponível em: http://www.ni.com/labview.

[59] Rick Bitter et al. LabVIEW Advanced Programming Techinques. Boca Raton: CRC Press LLC, 2001.

[60] B. Bruegge e A. Dutoit. Object-Oriented Software Engineering - Using UML, Patterns and Java^{TM}. Prentice Hall, 2004.

[61] H. M. Deitel e P. J. Deitel. *Java^{TM} Como Programar*. Bookman, 2003.

[62] *RFC 1321 - The MD5 Message-Digest Algorithm*. Disponível em: http://www.ietf.org/rfc/rfc1321.txt.

[63] *RFC 3174 - US Secure Hash Algorithm 1 (SHA1)*. Disponível em: http://www.ietf.org/rfc/rfc3174.txt.

Apêndice A

Apêndices

Cada Caso de Uso (funcionalidade do sistema), que foi apresentado no Diagrama mostrado na Seção 3.3.1 será agora detalhado quanto as suas dependências funcionais e de negócio. É importante que qualquer programador ou analista entenda os fluxos do sistema apresentados nessas descrições, por isso utilizamos os padrões UML para documentação dos casos de uso.

Descrição do Objetivo Breve descrição do objetivo do caso de uso;

Pré-Condições Eventos que são pré-requisitos para o acontecimento do caso de uso;

Pós-Condições Descrição dos estados do sistema gerados pelo caso de uso;

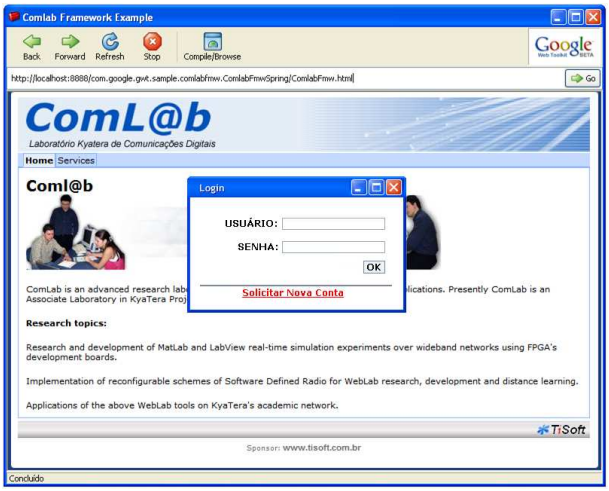
Regras de Negócio Associadas Regras de negócio que serão aplicadas naquela funcionalidade;

Regras Técnicas Regras associadas aos componentes visuais da funcionalidade;

Tabela de Mensagens Conjunto de mensagens que serão utilizadas no caso de uso;

Detalhamento dos Fluxos Documentação do fluxo principal e dos alternativos do caso de uso.

A.1 Login

Id do Caso de Uso:	00
Nome do Caso de Uso:	Login
Criador por:	Ariadne Cruz
Data de Criação:	03/08/2006
Última Atualização Realizada por:	Ariadne Cruz
Data da Última Atualização:	25/08/2006
Atores:	Usuário Comum e Usuário Administrador
Descrição:	Permitir que o usuário possa entrar no sistema ou Solicitar a criação de um novo usuário.
Pré-condições:	
Pós-condições:	1. Se o subfluxo "Efetuar Login" for executado, o usuário estará logado ao sistema; 2. Se o subfluxo "Solicitar Criação de Conta" for executado, os dados do novo usuário serão avaliados pelo administrador e armazenados ou não no sistema.
Interface:	

Tab. A.1: Identificação do Caso de Uso Login.

Tab. A.2: Caso de Uso Login

Continua na próxima página

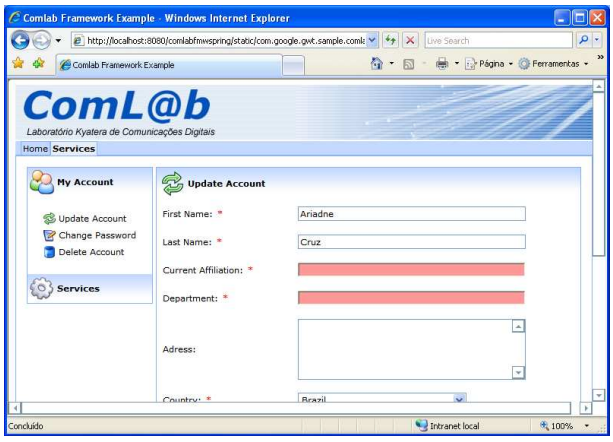
Cenário Principal
1. Este caso de uso inicia-se quando o usuário seleciona a opção “Efetuar Login” ou “Solicitar Criação de Conta”;
2. De acordo com a opção escolhida pelo usuário, um dos subfluxos será executado: a) Se o usuário desejar entrar no sistema, o subfluxo “Efetuar Login” é executado; b) Se o usuário desejar criar nova conta, o subfluxo “Solicitar Criação de Conta” é executado;
Subfluxo Efetuar Login
1. Este subfluxo inicia-se quando o usuário solicita entrar no sistema;
2. O sistema solicitará o preenchimento dos atributos login e senha;
3. Se os dados estiverem corretos o usuário estará logado no sistema, caso contrário o sistema reinicia o passo 2.
Subfluxo Nova Conta
1. Este subfluxo inicia-se quando o usuário solicitar incluir uma nova conta ao sistema;
2. O sistema solicitará ao usuário o preenchimento de alguns atributos: a) Login (*): 8 caracteres; b) Nome (*): 15 caracteres c) Sobrenome (*): 20 caracteres; d) Email (*): 25 caracteres; e) Endereço (*): 100 caracteres; f) País (*): Lista (Caso o nome do País não conste na lista, o usuário poderá adicionar um novo nome); g) Estado (*): Lista (Caso o nome do Estado não conste na lista, o usuário poderá adicionar um novo nome); h) Cidade (*): 20 caracteres; i) ZipCode (*): Máscara (Só aceita números); j) Afiliação (*): 20 caracteres; l) Departamento (*): 20 caracteres; m) Telefone (*): 20 caracteres; n) Status (*): Lista (Aluno, Professor, Outro).
3. Os dados serão enviados ao administrador do sistema, e podem ou não serem incluídos ao sistema.

Tab. A.2: Caso de Uso Login

Continua na próxima página

3.1. Caso os dados sejam incluídos no sistema, o usuário receberá um email com o login e a senha de acesso.

A.2 Minha Conta

Id do Caso de Uso:	01
Nome do Caso de Uso:	Minha Conta
Criador por:	Ariadne Cruz
Data de Criação:	03/08/2006
Última Atualização Realizada por:	Ariadne Cruz
Data da Última Atualização:	25/08/2006
Atores:	Usuário Comum e Usuário Administrador
Descrição:	Permitir que o usuário possa editar (atualizar, trocar senha e/ou excluir) sua conta.
Pré-condições:	Cliente logado no sistema
Pós-condições:	1. Se o subfluxo "Atualizar Conta" for executado, o sistema irá exibirá uma tela com as opções a serem atualizadas; 2. Se o subfluxo "Excluir Conta" for executado, a conta será desabilitada; 3. Se o subfluxo "Alterar Senha" for executado, o sistema solicitará a inclusão de uma nova senha.
Interface:	

Tab. A.3: Identificação do Caso de Uso Minha Conta.

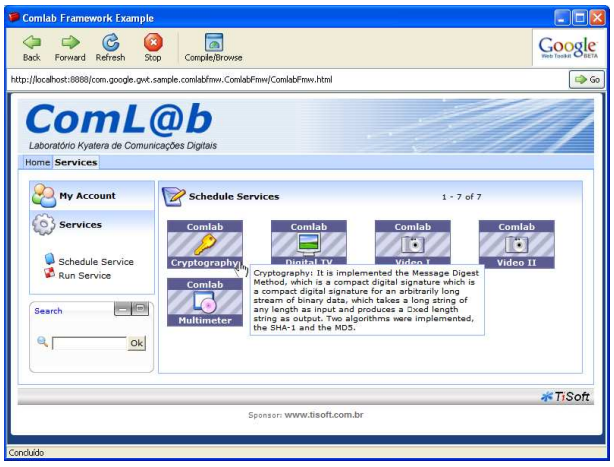
Cenário Principal
1. Este caso de uso inicia-se quando o usuário seleciona a opção “Minha Conta”;
2. De acordo com a opção escolhida pelo usuário, um dos subfluxos será executado: a) Se o usuário desejar atualizar sua conta, o subfluxo “Atualizar Conta” é executado; b) Se o usuário desejar excluir sua conta, o subfluxo "Excluir Conta"é executado; c) Se o usuário desejar alterar a senha, o subfluxo “Alterar Senha” é executado.
Subfluxo Atualizar Conta
1. Este subfluxo inicia-se quando o usuário solicita atualizar a sua conta;
2. O sistema irá exibir uma tela com os dados que podem ser atualizados: a) Nome (*): 15 caracteres b) Sobrenome (*): 20 caracteres; c) Email (*): 25 caracteres; d) Endereço (*): 100 caracteres; e) País (*): Lista (Caso o nome do País não conste na lista, o usuário poderá adicionar um novo nome); f) Estado (*): Lista (Caso o nome do Estado não conste na lista, o usuário poderá adicionar um novo nome); g) Cidade (*): 20 caracteres; h) ZipCode (*): Máscara (Só aceita números); i) Afiliação (*): 20 caracteres; j) Departamento (*): 20 caracteres; l) Telefone (*): 20 caracteres; m) Status (*): Lista (Aluno, Professor, Outro).
3. O sistema exibe uma mensagem informando que as alterações foram efetivas com sucesso.
Subfluxo Excluir Conta
1. Este subfluxo inicia-se quando o usuário solicita excluir sua conta do sistema;
2. O sistema solicita a confirmação da exclusão;
3. O usuário confirma a exclusão;
4. O sistema exibe uma mensagem informando que a exclusão foi efetivada com sucesso. O sistema desabilita a conta do usuário e exclui o usuário de todos as listas de serviços agendados.

Tab. A.4: Caso de Uso Minha Conta

Continua na próxima página

Subfluxo Alternativo (Excluir Conta)
3.1. O usuário não confirma a exclusão e o sistema volta para o painel dos serviços.
Subfluxo Alterar Senha
1. Este subfluxo inicia-se quando o usuário solicita alterar senha
2. O sistema irá exibir uma tela com os atributos a serem atualizados: a) Senha antiga (*): mínimo de 6 caracteres; b) Nova senha (*): mínimo de 6 caracteres; c) Confirmação da Senha (*): mínimo de 6 caracteres.
3. O sistema exibe uma tela de que as alterações de senha foram efetivadas com sucesso, caso contrário o sistema reinicia o passo 2.

A.3 Catálogo de Serviços

Id do Caso de Uso:	02
Nome do Caso de Uso:	Catálogo de Serviços
Criador por:	Ariadne Cruz
Data de Criação:	03/08/2006
Última Atualização Realizada por:	Ariadne Cruz
Data da Última Atualização:	25/08/2006
Atores:	Usuário Comum e Usuário Administrador
Descrição:	Permitir que o usuário possa agendar os serviços, visualizar os serviços existentes no sistema e executar os serviços.
Pré-condições:	Cliente logado no sistema
Pós-condições:	1. Se o subfluxo "Agendar Serviço" for executado, uma lista contendo todos os serviços que não sejam livres será exibida. 2.. Se o subfluxo "Executar Serviço" for executado, uma lista contendo todos os serviços livres e todos os serviços que foram agendados por aquele usuário será exibida.
Interface	

Tab. A.5: Identificação do Caso de Uso Catálogo de Serviço.

Cenário Principal

Tab. A.6: Caso de Uso Catálogo de Serviço

Continua na próxima página

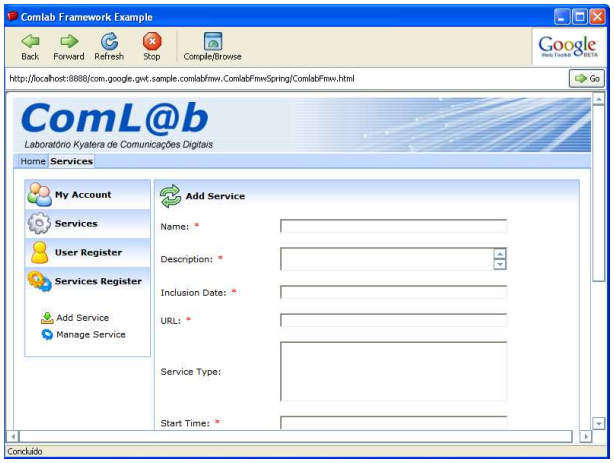
1. Este caso de uso inicia-se quando o usuário seleciona a opção “Serviços”;
2. De acordo com a opção escolhida pelo usuário, um dos subfluxos será executado: a) Se o usuário desejar consultar os serviços disponíveis para o agendamento e/ou agendá-los, o subfluxo “Agendar Serviços” deverá ser executado. b) Se o usuário desejar executar os serviços previamente agendados ou os serviços livres, o subfluxo “Executar Serviço” deverá ser executado.

Subfluxo Agendar Serviços
1. Este subfluxo inicia-se quando o usuário deseja consultar e/ou agendar os serviços disponíveis para agendamento;
2. O sistema exibe uma lista com todos os serviços disponíveis para o agendamento, como a opção de incluir o serviço e/ou visualizar a sua descrição.
3. O usuário seleciona o serviço que deseja incluir ou consultar: a) Se o usuário desejar apenas visualizar a descrição, basta passar o mouse sobre o ícone do serviço, e o sistema exibirá a descrição. b) Se o usuário escolher a opção incluir. O sistema solicitará o preenchimento de alguns dados: b.1) Data de Utilização(*): Calendário b.2) Hora de Início(*): Uma lista com os horários disponíveis irá aparecer; b.3) Limite(*): HH:mm.

Subfluxo Alternativo 1 (Agendar Serviços)
1. Após o passo 3.b, o sistema verificará se há disponibilidade para a data e o horário; E se o tempo estimado do experimento não excedeu o tempo limite cadastrado no sistema para cada Serviço.
2. Se existir, o sistema informará que o serviço foi agendado com sucesso e o serviço é incluído na lista de serviços agendados do usuário. Caso contrário, será pedido que o usuário preencha novamente os dados para o agendamento do serviço, e o sistema reinicia o passo 3.b

Catálogo de Serviços - Subfluxo Executar Serviços.
1. O subfluxo inicia-se quando o usuário solicita consultar os serviços livres e/ou os serviços que ele agendou;
2. O sistema apresenta uma lista com todos os serviços agendados e os serviços livre;
3. Se o usuário desejar executar um desses serviços, o sistema exibirá a janela com o serviço e o usuário poderá interagir com a aplicação.

A.4 Gerenciar Serviços

Id do Caso de Uso:	03
Nome do Caso de Uso:	Gerenciar Serviços
Criador por:	Ariadne Cruz
Data de Criação:	03/08/2006
Última Atualização Realizada por:	Ariadne Cruz
Data da Última Atualização:	25/08/2006
Atores:	Usuário Administrador
Descrição:	Permitir que o administrador possa incluir, atualizar, excluir ou consultar um serviço.
Pré-condições:	Usuário administrado logado no sistema.
Pós-condições:	1. Se o subfluxo "Incluir Serviço" for executado, os dados do novo serviço serão armazenados no sistema, juntamente com a classe de implementação do serviço; 2.Se o subfluxo "Editar Serviço" for executado, uma lista com todos os serviços será exibida com as opções de consultar, atualizar e excluir.
Interface	

Tab. A.7: Identificação do Caso de Uso - Gerenciar Serviços.

Gerenciar Serviços - Cenário Principal.

Tab. A.8: Caso de Uso Gerenciar Serviços

Continua na próxima página

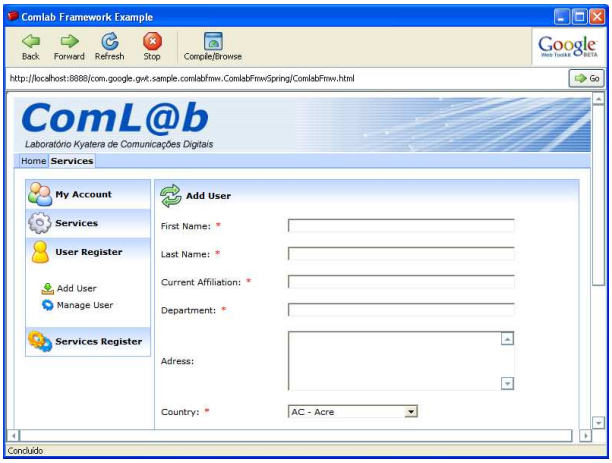
1. Este caso de uso inicia-se quando o administrador seleciona a opção “Gerenciar Serviços”;
2. De acordo com a opção escolhida pelo administrador, um dos subfluxos será executado: a) Se o administrador desejar incluir um novo serviço, o subfluxo “Incluir Serviço” é executado; b) Se o administrador desejar consultar, atualizar ou excluir um dos serviços, o subfluxo “Editar Serviço” é executado.
Subfluxo Incluir Serviço
1. Este subfluxo inicia-se quando o administrador solicita incluir um novo serviço ao sistema;
2. O sistema solicita ao administrador o preenchimento de alguns atributos: a) Nome do Serviço (*) - 15 caracteres; b) Descrição do Serviço (*) - 255 caracteres; c) Data de Inclusão (*); d) Link para o Serviço (*) - 100 caracteres; e) Hora de Inicio; f) Hora de Fim; g) Hora Limite; h) Interface; i) Bean;
3. O administrador deverá ainda fazer o upload da classe de implementa o serviço e a sua interface;
4. Após a confirmação, o sistema solicitará o login e a senha do administrador para confirmar a inclusão do novo serviço.
Gerenciar Serviços - Subfluxo Manter Serviço.
1. Este subfluxo inicia-se quando o administrador solicita consultar, atualizar ou excluir um dos serviços existentes no sistema;
2. O sistema irá exibir uma lista com todos os serviços existentes no sistema, com as opções de consultar, atualizar ou excluir: a) Se o administrador escolher a opção consultar: O sistema apresenta uma tela com a descrição do serviço e os usuários que teriam agendado o serviço; b) Se o administrador escolher a opção atualizar: O sistema exibirá uma tela com os atributos que podem ser alterados; b.1) Nome do Serviço; b.2) Descrição do Serviço; b.3) URL do Serviço; b.4) Interface;

Tab. A.8: Caso de Uso Gerenciar Serviços

Continua na próxima página

c) Se o usuário escolher a opção excluir: O sistema informa se há usuários que agendaram o serviço e solicita confirmação para a exclusão;
3. Após a confirmação, o sistema solicitará o login e a senha do administrador para confirmar a inclusão ou exclusão do novo serviço.
4. O sistema exibe uma tela que a operação foi efetuada com sucesso.
Gerenciar Serviços - Subfluxo Alternativo 1 (Manter Serviço).
c.1) O administrador não confirma a exclusão e o sistema reinicia o passo 2.

A.5 Gerenciar Usuários

Id do Caso de Uso:	04
Nome do Caso de Uso:	Gerenciar Usuários
Criador por:	Ariadne Cruz
Data de Criação:	03/08/2006
Última Atualização Realizada por:	Ariadne Cruz
Data da Última Atualização:	25/08/2006
Atores:	Usuário Administrador
Descrição:	Permitir que o administrador possa incluir, atualizar, excluir ou consultar um usuário.
Pré-condições:	Usuário administrado logado no sistema.
Pós-condições:	1. Se o subfluxo "Incluir Usuário" for executado, os dados do novo usuário serão armazenados no sistema; 2.Se o subfluxo "Editar Usuário" for executado, uma lista com todos os usuários será exibida com as opções de consultar, atualizar e excluir.
Interface	

Tab. A.9: Identificação do Caso de Uso - Gerenciar Usuários.

Gerenciar Usuários - Cenário Principal.
1. Este caso de uso inicia-se quando o administrador seleciona a opção "Gerenciar Usuários";

Tab. A.10: Caso de Uso Gerenciar Usuários

Continua na próxima página

2. De acordo com a opção escolhida pelo administrador, um dos subfluxos será executado: a) Se o administrador desejar incluir um novo usuário, o subfluxo "Incluir Usuário" é executado; b) Se o administrador desejar consultar, atualizar ou excluir um dos usuários, o subfluxo "Editar Usuário" é executado.
Subfluxo Incluir Usuário
1. Este subfluxo inicia-se quando o administrador solicita incluir um novo usuário ao sistema;
2. O sistema solicita ao administrador o preenchimento de alguns atributos: a) Login (*) - 15 caracteres; b) Senha (*) - mais do que 6 caracteres; c) Confirmação da Senha (*) - mais do que 6 caracteres ; d) Nome (*) - 15 caracteres; e) Sobrenome (*) - 20 caracteres; f) Endereço de email (*) - 25 caracteres; g) Tipo (*) - 1 caracteres;
Gerenciar Serviços - Subfluxo Manter Usuário.
1. Este subfluxo inicia-se quando o administrador solicita consultar, atualizar ou excluir um dos usuários existentes no sistema;
2. O sistema irá exibir uma lista com todos os usuários existentes no sistema, com as opções de consultar, atualizar ou excluir: a) Se o administrador escolher a opção consultar: O sistema apresenta uma tela com a descrição do usuários; b) Se o administrador escolher a opção atualizar: O sistema exibirá uma tela com os atributos que podem ser alterados; b.1) Nome; b.2) Sobrenome; b.3) Endereço de email; b.4) Cidade; b.5) Estado; b.6) País; b.7) Afiliação; b.8) Endereço; b.9) Status; b.10) Tipo; c) Se o usuário escolher a opção excluir: O sistema solicita confirmação para a exclusão;

Tab. A.10: Caso de Uso Gerenciar Usuários

Continua na próxima página

3. Após a confirmação, o sistema solicitará o login e a senha do administrador para confirmar a inclusão ou exclusão do novo usuário.
4. O sistema exibe uma tela que a operação foi efetuada com sucesso.

Gerenciar Usuários - Subfluxo Alternativo 1 (Manter Usuário).

Gerenciar Usuários - Serviços/Usuários.

1. Este subfluxo inicia-se quando o administrador solicita visualizar a lista de todos os serviços agendados por usuário;
2. O sistema irá exibir apresentar uma lista com todos os serviços e que usuários estão agendados para cada serviço.