UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

ROBERTO RENATO WELZEL FILHO

CHAMADAS ASSÍNCRONAS COM GWT

TRABALHO DE DIPLOMAÇÃO

TRABALHO DE DIPLOMAÇÃO

MEDIANEIRA
2011

ROBERTO RENATO WELZEL FILHO

CHAMADAS ASSÍNCRONAS COM GWT

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas – da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. MSc. Everton Coimbra de Araújo

MEDIANEIRA

2011

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

Chamadas Assíncronas com GWT

Por

Roberto Renato Welzel Filho

Este Trabalho de Diplomação (TD) foi apresentado às 13:00 h do dia 14 de junho de 2011 como requisito parcial para a obtenção do título de Tecnólogo no Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, *Campus* Medianeira. Os acadêmicos foram argüidos pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Prof. MSc. Everton Coimbra de Araújo
UTFPR – *Campus* Medianeira
(Orientador)

Prof. MSc. Alan Gavioli
UTFPR – *Campus* Medianeira
(Convidado)

Marcela Turim Koschevic
UTFPR – *Campus* Medianeira
(Convidado)

Prof. Juliano Rodrigo Lamb
UTFPR – *Campus* Medianeira
(Responsável pelas atividades de TCC)

DEDICATÓRIA

A Deus, por ter me dado vida, e por ter mandado seu
Filho para que eu pudesse viver para sempre.

“Filho, há mais uma coisa que eu quero dizer:
os livros sempre continuarão a ser escritos;
estudar demais cansa a mente.”
Salomão, no livro de Eclesiastes

RESUMO

O presente trabalho é um estudo sobre o mecanismo de chamada remota de procedimentos do *Google Web Toolkit*. Apresenta uma visão geral de suas operações e também a arquitetura e funcionamento de uma chamada remota. Compara a realização de chamadas assíncronas (AJAX) com uso do jQuery em contraste com o mecanismo de chamada remota (*RPC*) do *GWT*.

Palavras-chave: GWT, Chamada Remota de Procedimentos, AJAX

ABSTRACT

The present work is a study about Google Web Toolkit's remote procedure call mechanism. It presents an overview of how GWT works as well as the architecture and functioning of a remote call. A comparison of asynchronous (AJAX) calls with *jQuery* in contrast with GWTs remote procedure call mechanism is made.

Keywords: GWT, Remote Procedure Call, AJAX

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

AJAX	- *Asynchronous JavaScript and XML*
HTML	- *HyperText Markup Language*
HTTP	- *HyperText Transfer Protocol*
GUI	- *Graphical User Interface*
XML	- *eXtensible Markup Language*
GWT	- *Google Web Toolkit*
RPC	- *Remote Procedure Call*
JVM	- *Java Virtual Machine*
CSS	- *Cascading Style Sheet*
RMI	- *Remote Method Invocation*
DOM	- *Document Object Model*
URL	- *Uniform Resource Locator*
IDE	- *Integrated Development Environment*
JSP	- *Java Server Pages*
ORM	- *Object-relational Mapping*
API	- *Application Programming Interface*

SUMÁRIO

1 INTRODUÇÃO

Com o acesso cada vez mais facilitado à Internet, tornou-se viável o desenvolvimento de sistemas que usam a Web como plataforma (O'REILLY, 2005), sendo que é possível acessar a Internet de praticamente qualquer lugar do mundo.

Esse avanço acelerado fez aumentar o número de usuários da grande rede, e surgiu a necessidade de aplicações Web cada vez mais seguras, dinâmicas e rápidas. Dessa forma a demanda por ferramentas que possibilitassem o desenvolvimento de aplicações dinâmicas no menor tempo possível cresceu muito.

Em função dessa grande demanda, muitos *frameworks* e ferramentas para o desenvolvimento de aplicações Web nasceram. Uma dessas ferramentas é o GWT (*Google Web Toolkit*), que é uma ferramenta de código aberto lançada pela Google.

Este estudo pretende mostrar em termos práticos como funciona o principal método de comunicação entre o cliente e o servidor de uma aplicação desenvolvida com o GWT. Para melhor compreensão do mecanismo de chamadas assíncronas do RPC do GWT, uma comparação será feita entre o funcionamento do mesmo e o AJAX (*Asynchronous JavaScript and XML*) do jQuery.

1.1 OBJETIVO GERAL

Explicar e demonstrar o funcionamento do mecanismo de chamada remota de procedimento utilizado pelo GWT. Apresentar o contraste entre a realização de chamadas assíncronas com o GWT e com o jQuery[1].

1.2 OBJETIVOS ESPECÍFICOS

- Apresentar uma visão do GWT e do seu funcionamento.
- Apresentar a arquitetura e o funcionamento do RPC do GWT.

[1] jQuery é uma biblioteca JavaScript *cross-browser* desenvolvida para simplificar os *scripts client side* que interagem com o HTML. (JQUERY, 2011)

- Demonstrar como os serviços RPC são programados e chamados em uma aplicação Web desenvolvida com o GWT em contraste com o AJAX do jQuery.
- Desenvolver uma aplicação Web para mostrar as vantagens e desvantagens do uso do mecanismo de chamada remota de procedimento (RPC) do GWT.

1.3 JUSTIFICATIVA

A modernização dos navegadores Web e o advento da Web 2.0 fez surgir muitos *frameworks*[2] e ferramentas com a pretensão de agilizar e facilitar o desenvolvimento de aplicações Web.

Foi nesse contexto que a empresa de serviços Web, Google, lançou o *Google Web Toolkit* (GWT), com soluções inovadoras para velhos problemas do mundo do desenvolvimento de aplicações para a Internet (SMEETS et al, 2008, p. 23), como a portabilidade entre navegadores (KEREKI, 2011).

Faz parte do escopo de trabalho de um desenvolvedor buscar o conhecimento de novas tecnologias para que, ao se deparar com uma problemática, ele tenha sempre em mente uma maneira de solucionar o mesmo.

Este trabalho pretende apresentar uma breve visão geral do *Google Web Toolkit* e demonstrar o funcionamento da inovação tecnológica feita pelo mesmo no uso da técnica AJAX. Além disso, possibilitando ao leitor conhecer as principais vantagens do uso da ferramenta na criação de aplicações Web.

1.4 ESTRUTURA DO TRABALHO

O trabalho está dividido em capítulos que abordarão o tema da seguinte forma:

O primeiro capítulo fornece uma breve fundamentação histórica de aplicações JavaScript.

Os capítulos que seguem aprofundam-se no tema, explicando a arquitetura e o funcionamento das chamadas assíncronas, que é a principal inovação do GWT.

[2] Uma abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica.

No terceiro capítulo é apresentado um estudo de caso de uma aplicação desenvolvida com o GWT.

No capítulo finail, compara-se uma aplicação que faz o uso do AJAX (através do jQuery) e uma aplicação que realiza chamadas assíncronas com o RPC do GWT.

2 O *GOOGLE WEB TOOLKIT*

2.1 A INTERNET COMO PLATAFORMA

Quando a Internet surgiu, não se imaginava que ela se tornaria algo tão útil e versátil. A Internet cresceu tanto que acabou se tornando um dos principais meios de comunicação à longa distância. As pessoas começaram a perceber que a Internet não era somente algo para publicação de conteúdos, e sim algo que pode servir como um meio para prestação de serviços. Mudou-se então a forma de ver a Internet, agora como algo pela qual seria possível a disponibilização de serviços que na sua essência são programas e possuem como plataforma a própria Internet (KEREKI, 2011).

Usar a Internet como plataforma se tornou algo interessante (leia-se lucrativo), para o mercado de desenvolvimento de software, tanto que começou-se a desenvolver sistemas que usam a Internet como plataforma. Porém esse uso tem acarretado uma série de necessidades, entre elas a de desenvolver sistemas mais rapidamente, e, conseqüentemente, a necessidade de desenvolvimento de ferramentas que permitem um desenvolvimento rápido sem perda de qualidade.

2.2 VISÃO GERAL

O GWT é um conjunto de ferramentas de código aberto que tem como finalidade fornecer uma maneira de construir e otimizar aplicações complexas que funcionam em um navegador. Ela permite ao desenvolvedor não somente construir um sistema Web com uma interface gráfica atraente, pois é uma ferramenta que trabalha majoritariamente com AJAX, mas também com rapidez, já que ele possui muitos componentes visuais prontos. A ferramenta da Google também facilita o desenvolvimento, sendo que não é necessário conhecimento profundo sobre *Web Standards*[3] (KEREKI, 2011).

Uma das maiores vantagens do uso do GWT é o desempenho de suas aplicações. O uso natural do AJAX para a comunicação do cliente com o servidor faz com que não haja a

[3] Padrões para a construção de páginas definidos pela *World Wide Web Consortium*(W3C) e outras entidades. (SCHMITT et al, 2008)

necessidade de que a página seja recarregada várias vezes até que se uma tarefa seja realizada (DEWSBURY, 2008).

2.2.1 GWT usa Java

A linguagem de programação usada pelo GWT é Java, ou seja, toda a programação, tanto no lado do cliente como no lado do servidor, é feita em Java (GUPTA, 2008). Isso significa que todos os benefícios da linguagem de programação Java são mantidos, como a independência de sistemas operacionais, e também as características de uma linguagem de programação madura. O Java também tem um grande suporte da comunidade de programação pois existem muitos programadores Java experientes, o que facilitou a difusão da ferramenta bem como a diferença de outros *frameworks* web, que não é grande (KEREKI, 2011).

A necessidade de usar-se o Java em vez de JavaScript para programação vem da complexidade e tamanho crescente de Aplicações de Internet Rica, que são aplicações fornecidas e/ou instaladas através da web, e geralmente funcionam em um navegador (KEREKI, 2011). Porém o fato de usar o GWT não significa que não é mais possível usar o JavaScript, pois a ferramenta ainda permite a interação com código JavaScript já existente sendo que o compilador JavaScript do GWT possibilita a mesclagem de código JavaScript escrito pelo desenvolvedor com o seu próprio código JavaScript.

A programação no lado do servidor suporta todas as funções do Java, pois o servidor executa sobre um container *Servlet*[4] (DEWSBURY, 2007), que por sua vez funciona sobre uma JVM (*Java Virtual Machine*), ou seja, a flexibilidade de acesso a um banco de dados que o Java oferece é mantida (KEREKI, 2011).

2.2.2 GWT e AJAX

[4] Uma extensão genérica do servidor, uma classe Java que pode ser carregada dinâmicamente para expadir a funcionalidade de um servidor (HUNTER & CRAWFORD, 2001).

AJAX (*Asynchronous Javascript and XML*) é uma técnica usada no meio de desenvolvimento Web que possibilita a comunicação assíncrona com o servidor. A comunicação síncrona é o que acontece quando o usuário deve esperar o navegador carregar o conteúdo da página Web que ele deseja acessar, ficando incapaz de interagir com o conteúdo até que ele seja parcialmente ou completamente carregado (SMEETS et al, 2008).

O principal elemento da técnica AJAX é o objeto JavaScript *XMLHttpRequest*. Esse objeto tem como função carregar o conteúdo de um endereço eletrônico (URL – *Uniform Resource Locator*) de maneira assíncrona, possibilitando ao desenvolvedor manipular o resultado através de código JavaScript sem que seja necessário carregá-lo em tela. O desenvolvedor pode então usar esses dados retornados pelo objeto para atualizar a interface gráfica da página Web usando JavaScript, possibilitando ao usuário uma experiência mais rica e rápida, sendo que quando conteúdo do servidor é carregado dessa maneira, o navegador não precisa renderizar esse mesmo conteúdo em tela.

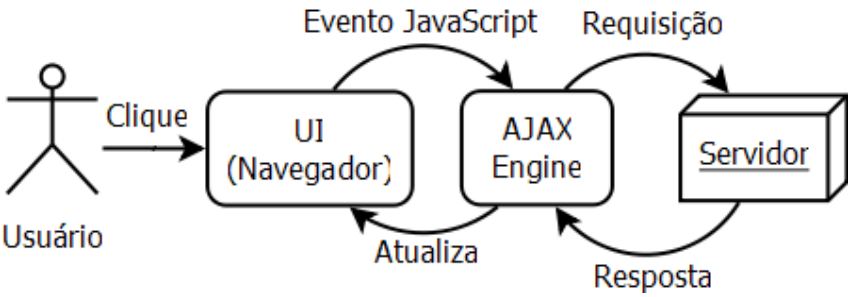

Figura 1 – Ciclo de eventos de uma chamada AJAX

Como a Figura 1 ilustra, a interação do usuário com o navegador dispara as chamadas AJAX que são responsáveis por fazer a comunicação com o servidor através de objetos *XMLHttpRequest*, e não somente o navegador. O elemento "AJAX Engine" representa o código JavaScript que faz o manuseio desses objetos *XMLHttpRequest* e a interação com o DOM[5] (*Document Object Model*). O retorno das chamadas AJAX é então usado por esse código para atualizar a página Web sendo visualizada pelo usuário, sem que ela precise ser renderizada pelo navegador novamente.

[5] DOM é uma interface que não é ligada a um navegador, plataforma, ou linguagem específicos, e permite programas e rotinas, dinâmicamente, acessar e atualizar o conteúdo, estrutura, e estilo de documentos (ZELDMAN, 2006).

O GWT utiliza-se muito da técnica AJAX. Na verdade, é a principal característica de um sistema Web desenvolvido com o GWT, pois a parte gráfica do sistema pode ser toda construída em cima de uma só pagina (KEREKI, 2011), e para que a mudança de estados da aplicação ocorra, a comunicação com o servidor é feita através de chamadas assíncronas, AJAX.

Em nível de programação, essas chamadas assíncronas são feitas através de interfaces de negócio e de implementações dessas interfaces, e é denominado RPC (*Remote Procedure Call*), pois no desenvolvimento da parte visual da aplicação, essas interfaces são instanciadas e seus métodos são chamados remotamente para que a ação seja feita no servidor e retorne um resultado, realizando assim a chamada de um procedimento remotamente.

2.2.3 Construção da GUI

O desenvolvimento da interface gráfica com o GWT é relativamente simples, pois ele já possui componentes prontos para serem usados na construção da mesma, como botões, painéis, listas, *popups* e tabelas, entre outros.

A construção da parte visual de uma aplicação GWT é semelhante ao desenvolvimento de interfaces gráficas *desktop* com o Swing[6] do Java (KEREKI, 2011), pois são conceitualmente muito parecidos entre si, onde ações (como cliques e movimentos do mouse) são gerenciadas como eventos, podendo ser tratados no momento em que acontecem. Essa semelhança ao Swing também facilita o desenvolvimento de ferramentas gráficas que auxiliam na construção de uma GUI (*Graphical User Interface*).

Os *widgets*[7], que são a forma mais atômica dos componentes visuais GWT, são equivalentes aos *Components* do Swing, e alguns também são representantes diretos de tags HTML. Sendo assim, a interface gráfica não é mais codificada com tags, e sim com *widgets*, ou seja, onde existiria um tag *anchor*, existe um *widget Anchor*. Também existem *widgets*

[6] O conjunto de ferramentas padrão para a construção da GUI em Java (HANSON & TACY, 2007).

[7] Widgets são componentes visíveis de uma aplicação GWT que o usuário pode ver em seu navegador (HANSON & TACY, 2007).

mais complexos, como o *Tree*, que representa uma árvore hierárquica, algo que é geralmente visto em aplicações *desktop*. Os eventos desses *widgets* também são programados de forma semelhante ao Swing, pois são componentes orientados a eventos. A maioria dos *widgets* suporta algum tipo de interação com o usuário, e essas interações são tratadas como eventos, sendo codificadas através de *Listeners*, que são interfaces que contém os métodos que são executados quando um evento acontece, como por exemplo um *ClickListener*, que possui o método *onClick*, que é disparado quando o botão é pressionado (clicado).

O *layout* de uma aplicação GWT é baseado em painéis (HANSON & TACY, 2007) dos tipos mais variados, que tornam a aplicação visualmente flexível, bem como o suporte de *popups*, que são tão úteis quanto os painéis. Em vez de usar gerenciadores de *layout* como o Swing faz, o GWT optou pela construção de uma GUI através do uso de tabelas e painéis. Exemplos de *widgets* que são contêiners é o *DecoratorPanel*, que é uma painel com bordas arredondadas, e o *VerticalPanel*, que é um painél genérico que dispõe seus filhos em uma linha vertical, como ilustra a Figura 2.

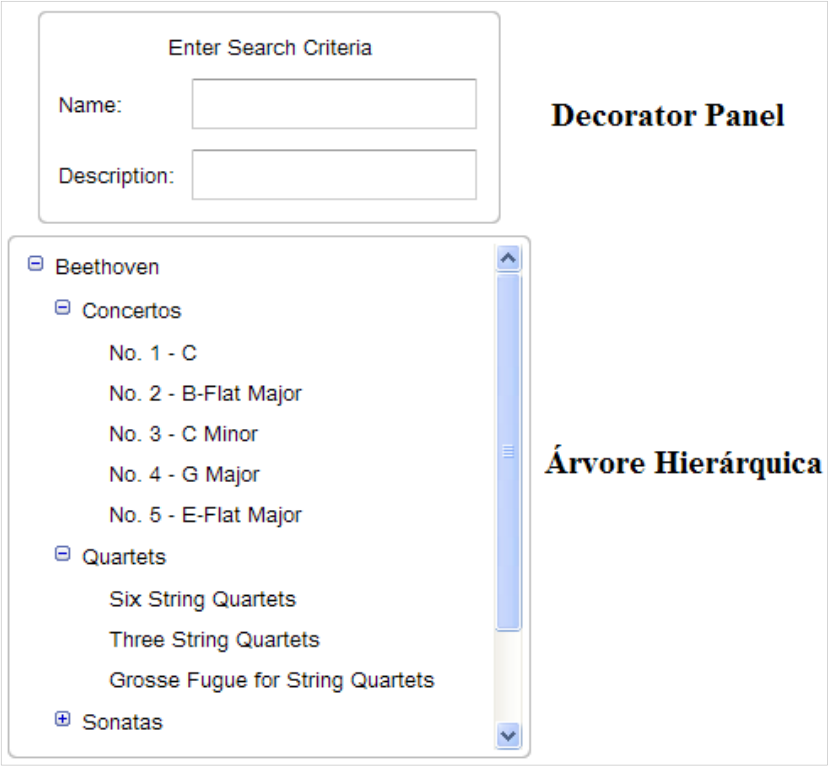

Figura 2 – Exemplos de *Widgets*.

2.2.4 Compilador JavaScript

O que torna possível ao GWT o uso do Java na construção de uma interface gráfica é o seu compilador JavaScript. A sua função é traduzir o código Java da parte do cliente em JavaScript (COOPER & COLLINS, 2008), para que esse possa ser carregado em uma página HTML e executado pelo cliente.

Esse compilador transforma todos os componentes visuais que foram adicionados no código Java em um ou mais arquivos JavaScript, uma linguagem que o navegador (cliente da aplicação) pode compreender. Estes arquivos fazem exatamente o que foi programado no código Java, porém em JavaScript, para que possa ser carregado por um navegador (KEREKI, 2011). Esse arquivo com extensão .js é então carregado pela página principal da aplicação GWT, mostrando no navegador a parte gráfica do sistema, que foi construída em código Java.

O compilador JavaScript do GWT possui três modos de operação, podendo gerar código JavaScript mais legível ou menos legível. O primeiro modo é o ofuscado, que gera código incompreensível com o objetivo de reduzir ao máximo o tamanho dos arquivos JavaScript produzidos. Pode ser usado também como uma maneira de esconder o código e prevenir que ele seja roubado. O segundo modo é o “gracioso”, que gera código legível mas não compreensível. O terceiro modo é o detalhado, que finalmente gera código legível e compreensível, adicionando o nome qualificade das classes Java envolvidas, sendo útil para encontrar o código Java correspondente (HANSON & TACY, 2007).

Outra característica do compilador JavaScript é que ele lê o código fonte Java, e não as classes binárias geradas pelo compilador Java. Por essa razão todas as classes Java usadas para codificar a aplicação cilente deve estar disponível para o compilador e de acordo com a versão 1.5 do Java. Quando distribuindo uma aplicação para o reuso de código, é necessário fornecer tanto o código fonte quanto o arquivo .jar.

2.2.5 Servlets

Na perspectiva do servidor, a maneira de trocar informações com o cliente é feita através de serviços RPC, que no servidor são nada mais que *Servlets* (KEREKI, 2011) personalizados para serem usados da maneira necessária ao GWT. Porém, na sua essência, e

na visão do servidor, esses serviços remotos possuem todas as características de um *Servlet,* pois, como comentado anteriormente, o servidor é executado em um contêiner *Servlet*.

Portanto, é possível usar todos os recursos de um *Servlet* em um serviço RPC, como o controle de sessões de servidor e o gerenciamento de *cookies*.

2.3 DESENVOLVENDO UMA APLICAÇÃO GWT

O GWT funciona, na sua base, como qualquer outro *framework* para desenvolvimento Web. A diferença é que além de codificar o servidor e editar as páginas de apresentação, é preciso compilar a parte cliente em JavaScript. Na verdade, grande parte da codificação do cliente é feita em Java em vez de HTML (*HyperText Markup Language*) ou *scriptlets*[8].

2.3.1 Client-Side

Como já mencionado, grande parte da codificação da camada de apresentação é feita em Java. Como um todo, o GWT usa uma combinação de Java, JavaScript, HTML e CSS[9] (*Cascading Style Sheet*) para o desenvolvimento da parte do programa visível ao usuário.

O desenvolvimento da camada visual pode ser considerado intuitivo para programadores que já trabalharam com Swing. Uma grande parcela do que é visível ao usuário, e a parte do programa (sistema) realiza a interação com o usuário é codificada em Java. Todo o tratamento de eventos e o fluxo do programa em geral são programados em Java através dos *widgets*, que são os componentes visuais usados para programar a apresentação ao usuário. As instanciações de serviços remotos e as chamadas dos mesmos também são feitas em Java. Todo esse código cliente escrito em Java é então interpretado para JavaScript e otimizado para funcionar nos principais navegadores através o Compilador JavaScript do GWT.

---

[8] Blocos de Código *servlet* usado para construir páginas JSP (*Java Server Pages*). (HUNTER & CRAWFORD, 2001)

[9] São arquivos (folhas de estilo) que consistem de uma ou mais regras que controlam como os elementos selecionados devem ser exibidos (ZELDMAN, 2007).

Como o GWT é um *framework* Web, e utilizado para desenvolver sistemas que serão mostrados em navegadores, não é possível que o HTML e o CSS fiquem de lado. O GWT torna possível a atribuição de identificações (propriedade *id*) e classes CSS aos *widgets* programados em Java (GUPTA, 2008), permitindo assim a personalização do sistema utilizando folhas de estilo (CSS). E quanto a isso, boas práticas e técnicas de *web design* se aplicam perfeitamente. Mas apesar de todas essas funcionalidades, a página em si é apresentada em uma página HTML significando que é possível utilizar *tags* HTML (que é considerado baixo nível quando se usa o GWT) para cobrir quaisquer falhas que tenham sido deixadas pelas outras formas de desenvolvimento.

2.3.2 Server-Side

No lado do servidor as coisas são um pouco diferentes. O que foi desenvolvido no servidor não precisa ser traduzido para JavaScript, pois não é executado no navegador. O código do lado do servidor não possui restrições, ou seja, é lícito usar todos os recursos da linguagem Java (HANSON & TACY, 2007), pois é executado sobre um contêiner *Servlet*, que por sua vez funciona através de uma JVM[10]. Na programação do servidor é possível fazer desde a conexão com um banco de dados para a serialização de objetos, até a chamada de um procedimento remoto de um site de previsão do tempo, por exemplo. No tocante ao desenvolvimento da lógica de negócio, é idêntico a qualquer outra aplicação Web que funciona sobre um contêiner Servlet.

Outra funcionalidade importante que é programada no servidor são as "pontes" entre a lógica de negócio do servidor e a apresentação gráfica no cliente. Esses são os serviços RPC, que nada mais são que a ligação entre o servidor e cliente, através desses serviços, rotinas no servidor são executados e resultados de processamento podem ser retornados para visualização do cliente, como por exemplo a alteração dos dados de um livro, ou serem parte de uma lógica sendo desenvolvida no cliente, como a busca de um valor necessário ao cálculo de um desconto, por exemplo.

No servidor, esses métodos remotos são programados como servlets, e implementam uma interface comum ao servidor e ao cliente que representa o serviço. Para todos os efeitos,

[10] JVM é a máquina virtual que possibilita que aplicações Java sejam executadas em várias plataformas pois faz a interpretação do código Java para linguagem de máquina específica (DEITEL & DEITEL, 2011).

o serviço no servidor é tratado como um servlet, podendo também ser acessado por uma URL única especificada na classe, como por exemplo, *localhost.com/serviço*. Para o cliente, a chamada de um procedimento remoto é simples, pois se necessita apenas instanciar o serviço e chamar o método. A comunicação é feita através de chamadas assíncronas, ou seja, através do uso da técnica AJAX.

2.4 REMOTE PROCEDURE CALL (RPC)

RPC significa "Chamada de procedimento remoto", do inglês *Remote Procedure Call*. O RPC em si não é um conceito novo, pois teve sua primeira descrição em 1976 (IETF, 1976) e se entende por ser uma tecnologia de comunicação entre processos, que geralmente são dois programas, que permite a chamada de um procedimento que se encontra em outro endereço, através de uma rede. Em termos atuais, seria um aplicativo cliente chamando um procedimento em um aplicativo sendo executado em um servidor, ambos separados geograficamente .

Esse conceito foi assimilado na arquitetura do GWT, pois é assim que os serviços remotos que são chamados pelo cliente (no caso, o navegador) funcionam. A aplicação cliente, na verdade. solicita ao servidor a execução de um procedimento específico (havendo passagem de parâmetros ou não) e este, por sua vez, executa o procedimento, podendo retornar um resultado ao cliente ou não (HANSON & TACY, 2007).

Esses serviços são chamadas remotas realizadas de forma assíncrona e formam o principal método de comunicação entre o servidor e o cliente em uma aplicação desenvolvida com o GWT. A implementação do RPC no GWT é, a nível de navegador, um aperfeiçoamento da técnica AJAX, com relação à maneira que a comunicação entre o cliente e o servidor é feita, visto que no servidor essas chamadas são tratadas por servlets. Portanto, a chamada de um procedimento do servidor pelo cliente é mais que apenas uma chamada assíncrona, pois é possível usar os recursos de um servlet a partir de uma chamada do cliente, como, por exemplo, armazenar um objeto na sessão do cliente que fez a chamada do procedimento.

2.4.1 Arquitetura

2.4.1.1 Server-side

Existe grande semelhança entre o modelo do RPC do GWT e o Java RMI (*Remote Method Invocation*) (GUPTA, 2008), sendo que os dois modelos possuem o mesmo objetivo: chamar procedimentos remotamente. Portanto, é necessário ter uma interface comum, que tanto o servidor, como o cliente conhecem. Tendo essa interface, é preciso de uma implementação para essa interface, essa implementação é a programação que fica no servidor e é executada pelo servidor quando o cliente invoca métodos daquela interface. No diagrama (Figura 3) é possível perceber a relação que existe entre as classes de um serviço RPC. Essas classes, porém, ficam no servidor, e formariam, na verdade, a espinha dorsal de uma aplicação desenvolvida com o GWT, pois a lógica de negócio que necessita de uma quantidade maior de processamento deve ficar no servidor para não sobrecarregar o cliente.

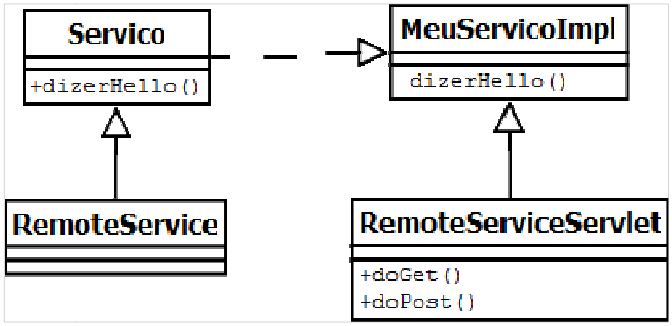

Figura 3 – O lado do servidor de um serviço RPC.

Um serviço RPC , no tocante ao servidor, é constituído de três classes:

- A interface que dita os métodos que o servidor irá implementar e que o cliente poderá chamar remotamente. No diagrama (Figura 3) é representada pela interface *Servico*. Ela deve ser herdada da classe marcadora *RemoteService*.
- A interface *RemoteService* que é uma simples interface marcadora que todos os serviços RPC devem herdar, para que o GWT possa identificá-los.
- A classe *RemoteServiceServlet,* que na verdade é um *HttpServlet* e portanto possui todos os métodos de um Servlet como o *doGet* e o *doPost*. Toda implementação de um serviço deve ser subclasse de *RemoteServiceServlet*, sendo assim, todo serviço *RPC* é também um Servlet.

- A implementação da interface do serviço, que na Figura 1 é representada pela classe *MeuServicoImpl*. Essa classe contém o código que é executado quando um dos métodos da interface do serviço é chamado.

São essas as classes que constituem a parte de um serviço RPC que executa no servidor. Em seguida veremos as classes que constituem um serviço RPC no cliente.

2.4.1.2 Client-side

A parte menos complexa é a que reside no cliente quando ele carrega a aplicação no seu navegador. Uma aplicação cliente GWT consiste em um ou mais arquivos JavaScript que são carregados pelo navegador a partir de uma página estática qualquer (KEREKI, 2011). O navegador, que é na verdade o cliente da aplicação GWT, executa esses arquivos que, através da técnica AJAX, fazem uso dos serviços RPC que são disponibilizados pelo servidor.

Porém, para que esses arquivos existam, é preciso codificar essas interfaces gráficas e também definir o seu funcionamento e a lógica que será usada na comunicação com o servidor. A único código que é comum ao servidor e ao cliente é a interface de negócio, porém para o cliente é necessário mais do que somente essa interface, é necessário também uma interface especial chamada de "contraparte (versão) assíncrona", que nada mais é que uma adaptação da interface de negócio usada para fazer chamadas assíncronas dos procedimentos da interface que o servidor implementa.

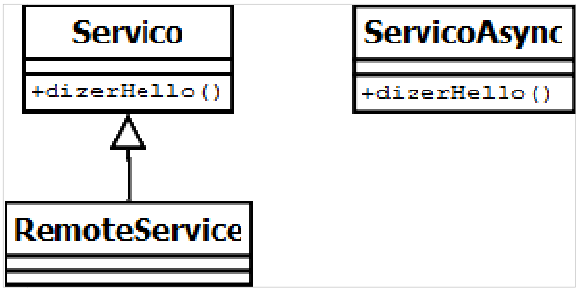

Figura 4 – O lado do cliente de um serviço *RPC*.

No tocante ao cliente, são três as classes envolvidas (representadas pela Figura 4):

- A interface de negócio, identificada na Figura 4 pelo nome *Servico*, que é a classe compartilhada entre o servidor e o cliente e que contém os métodos que podem ser chamados pelo cliente. Ela é herdada da classe marcadora *RemoteService*.

- A interface *ServicoAsync*, que é a versão assíncrona da interface *Servico*. Note que não existe relação de herança explícita entre as interfaces *Servico* e *ServicoAsync*, contudo todos métodos da interface *Servico* existem na interface *ServicoAsync* na sua forma assíncrona: com um parâmetro a mais que representa o retorno (resultado) do servidor.

2.4.2 Funcionamento

O funcionamento de uma chamada a um serviço RPC é, na verdade, de fácil compreensão, pois, de maneira geral, ela funciona como uma chamada AJAX. O que difere de uma chamada AJAX comum é que o GWT possui uma camada de serialização. Essa camada transforma os dados transmitidos em objetos antes que esses dados sejam manipulados pelo código criado pelo desenvolvedor. A codificação é feita acima dessa camada, ou seja, do ponto de vista do desenvolvimento, a troca de dados se torna uma troca de objetos entre cliente e servidor.

2.4.2.1 Client-side

O manuseio de dados do lado do cliente é feito pelo código JavaScript gerado pelo compilador do GWT que é o código Java compilado para JavaScript.

Para que o cliente e o servidor consigam se comunicar, antes que os dados sejam enviados ao servidor através de uma chamada *POST*, eles são convertidos em uma cadeia de caracteres (ilustrada pela Figura 5) que contém todas as informações necessárias para que a requisição seja processada pelo serviço disponível no servidor.

```
7|0|7|http://localhost:8080/myapp/myapp/|9F10D5EBA83843ED3D2564FA27A70B7B|org.filho.client.
GreetingService|greetServer|org.filho.shared.Bean/1532308294|java.lang.Integer/3438268394|G
WT User|1|2|3|4|1|5|5|6|1|7|
```

Figura 5 – Exemplo de conteúdo de uma requisição enviada ao servidor pelo cliente GWT.

Após a serialização dos dados necessários, temos uma cadeia de caracteres parecida com a mostrada na Figura 5. Ela contém os argumentos do método que está sendo invocado, bem como o nome do próprio método, as classes envolvidas, entre outras informações, delimitadas pelo caractere *pipe* (|). Ao receber essas informações, o servidor desserializa essa cadeia de caracteres, convertendo-a em objetos compreensíveis ao código, executa o serviço e retorna o resultado (caso houver) já serializado ao cliente. O cliente então usa esses dados para continuar o fluxo normal da aplicação.

A programação dessas chamadas remotas é feita através das interfaces assíncronas. Essas interfaces são "cópias" das interfaces de negócio porém trazem uma diferença: todos os métodos possuem um argumento a mais, o argumento de retorno da chamada assíncrona, conforme é ilustrado na Figura 6. Esse argumento adicional substitui o retorno do método, ou seja, o tipo do retorno agora é informado na generalização da interface *AsyncCallback*. Ao realizar a programação da parte do cliente somente as interfaces assíncronas são usadas, pois para cada chamada de um serviço é preciso definir a ação a ser tomada com o retorno do serviço.

Interface de Negócio

```
@RemoteServiceRelativePath("greet")
public interface GreetingService extends RemoteService {
  String greetServer(String name) throws IllegalArgumentException;
}
```

Versão Assíncrona

```
public interface GreetingServiceAsync {
  void greetServer(String input, AsyncCallback<String> callback)
      throws IllegalArgumentException;
}
```

Figura 6 – Comparação entre uma interface de negócio e sua versão assíncrona.

A interface *AsyncCallback* é usada para manipular o retorno da chamada ao servidor (DEWSBURY, 2008). Nela é informado o tipo do retorno (deve ser o mesmo tipo contido na interface de negócio) e também possui os métodos *onSuccess*, que é executado quando a chamada é concluída com êxito, e *onFailure*, que fornece um *Throwable* e é executado

quando ocorre um erro na chamada. O método *onSuccess* fornece um objeto-resultado que pode ser usado pelo cliente (caso o servidor retorne algo) para concluir ou dar continuidade ao fluxo da operação.

Quando o usuário realiza uma ação que requer algum processamento do lado do servidor, os argumentos do serviço a ser chamado são serializados, enviados ao servidor, que por sua vez desserializa os argumentos, faz o processamento e retorna a resposta para o cliente. A Figura 7 ilustra como é feita a invocação de um serviço remoto pelo cliente.

```
greetingService.greetServer(nome, newAsyncCallback<String>() {
  public void onFailure(Throwable erro) {
    // Quando a chamada falha esse trecho de código é executado
    ...
  }

  public void onSuccess(String resultado) {
    // Quando a chamada é concluída com êxito esse trecho é executado
    ...
  }
});
```

Figura 7 – Invocação de um serviço remoto pelo cliente.

2.4.2.2 Server-Side

O funcionamento de uma aplicação GWT é mais simples do lado do servidor do que do lado do cliente. Isso se deve ao fato de que o GWT foi desenvolvido, principalmente, para resolver problemas que ocorrem no cliente (navegador) (SMEETS et al, 2008) e não no servidor. O funcionamento do lado do servidor, portanto, não difere muito do ordinário.

Os serviços RPC são fornecidos através de servlets, portanto, ao incluí-los em uma aplicação, é necessário que essa aplicação esteja sendo executada em um contêiner servlet. Nesses serviços é possível usar todos os recursos de um servlet, como controle de sessão e *cookies*. O desenvolvimento desses serviços é realizado acima da camada de serialização do *GWT*, pois a desserialização dos argumentos de uma chamada vinda do cliente é feita antes da manipulação dos dados pelo código do servlet.

O servidor recebe a requisição (os dados enviados por essa solicitação são parecidos com o exemplo da Figura 3), desserializa os dados, através desses dados identifica o servlet e o método sendo chamado e chama esse método passando os argumentos (que já foram desserializados e se encontram em forma de objetos). O método executa e retorna o resultado,

o *Toolkit*, por sua vez, serializa esse resultado e envia ao cliente na forma exemplificada na Figura 8.

```
//OK[1,["Hello, org.filho.shared.Bean@3fb2d7df!<br><br>I am running jetty/6.1.26.<br><br>It looks like you are using:<br>Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWebKit/534.16 (KHTML, like Gecko) Chrome/10.0.648.151 Safari/534.16"],0,7]
```

Figura 8– Exemplo de retorno de uma requisição enviada ao cliente pelo servidor.

2.5 *SERIALIZATION POLICY* (POLÍTICA DE SERIALIZAÇÃO)

Para que uma chamada remota seja possível, o GWT precisa transferir objetos não só entre dois locais fisicamente separados, mas também entre duas aplicações sendo executadas em duas linguagens diferentes, a saber, o cliente que funciona com JavaScript, e o servidor que trabalha com Java.

A política de serialização (*Serialization Policy*) foi a maneira que o GWT encontrou para dar suporte à classes usadas no código Java do servidor no JavaScript. Essa política é gerada em tempo de compilação, e determina quais classes o código JavaScript deve dar suporte (KEREKI, 2011). É possível especificar no XML de configuração uma lista negra de classes, caso haja a necessidade de evitar que o código JavaScript dê suporte aquela classe específica, diminuindo assim o tamanho do arquivo .js gerado.

3 ESTUDO DE CASO

Em seguida é desenvolvida uma aplicação de cadastro de favoritos utilizando a versão 2.2.0 do *Google Web Toolkit* em conjunto com o banco de dados db4o[11]. Essa aplicação tem como objetivo demonstrar em funcionamento não a ferramenta como um todo, mas apenas uma parte dela, incorporando o uso do RPC para a realização de chamadas assíncronas.

3.1 O PROJETO

A idéia do projeto é permitir que o usuário possa se cadastrar e, uma vez que logado no sistema, cadastrar seus favoritos para que ele possa acessá-los mais tarde. Isso inclui o cadastro de usuários com *login* e senha e a autenticação desses dados (a Figura 9 ilustra a tela em que essa autenticação é feita) pelo servidor para o acesso aos favoritos. É possível também o cadastro dos favoritos que são compostos pelo nome e endereço eletrônico, e a remoção de favoritos já cadastrados.

O Quadro 1 ilustra os principais requisitos que o sistema implementa usando como ferramenta de desenvolvimento o GWT.

ID	FUNCIONALIDADE	DESCRIÇÃO
R1	Cadastrar Usuário	O sistema deve permitir que o usuário se cadastre.
R2	Autenticar Usuário	O sistema deve autenticar o usuário no sistema mediante a informação de login e senha.
R3	Trocar de Usuário	O sistema deve permitir que o usuário faça *logoff*, permitindo a troca de usuários
R4	Criar Favorito	O sistema deve permitir que o usuário cadastre links informando o nome e o endereço.
R5	Editar Favorito	O sistema deve permitir a edição de um favorito pelo usuário.
R6	Remover Favorito	O sistema deve permitir que o usuário remova um link previamente cadastrado.
R7	Listar Favorito	O sistema deve permitir ao usuário visualizar todos os seus links em forma de lista.

Quadro 1 – Lista de requisitos da aplicação de exemplo.

[11] Db4o é um banco de dados orientado a objeto que facilita a persistência de dados pois não é necessário fazer um mapeamento ORM (*Object-Relational Mapping*).

Percebe-se que a lista de requisitos não é muito longa, porém o desenvolvimento desse pequeno exemplo gera uma grande quantidade de código, e consequentemente um esforço razoável de codificação. Essa é uma desvantagem do GWT, pois mesmo que a aplicação seja simples, o mínimo de código necessário para o seu funcionamento é volumoso.

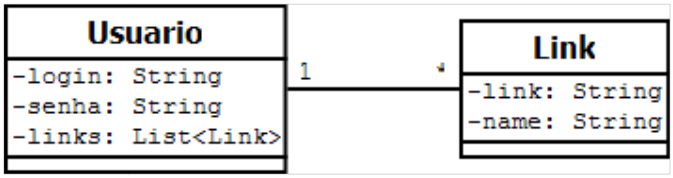

Figura 9 – Relação entre os beans da aplicação.

No servidor o banco de dados usado para armazenar os dados foi o db4o, que é um banco de dados orientado a objetos. A relação entre as classes utilizadas para o armazenamento em banco é representada pela Figura 9.

O bean *Usuario* é responsável por armazenar todos os dados do usuário, que nessa aplicação são o login e a senha. Encontra-se também no bean *Usuario* a lista de *Link*, que são os favoritos salvos por aquele usuário. O bean *Link*, por sua vez, representa um Favorito, e possui os atributos link, que é o endereço eletrônico do Favorito, e nome, que é o nome dado ao Favorito e é mostrado na lista de Favoritos do usuário.

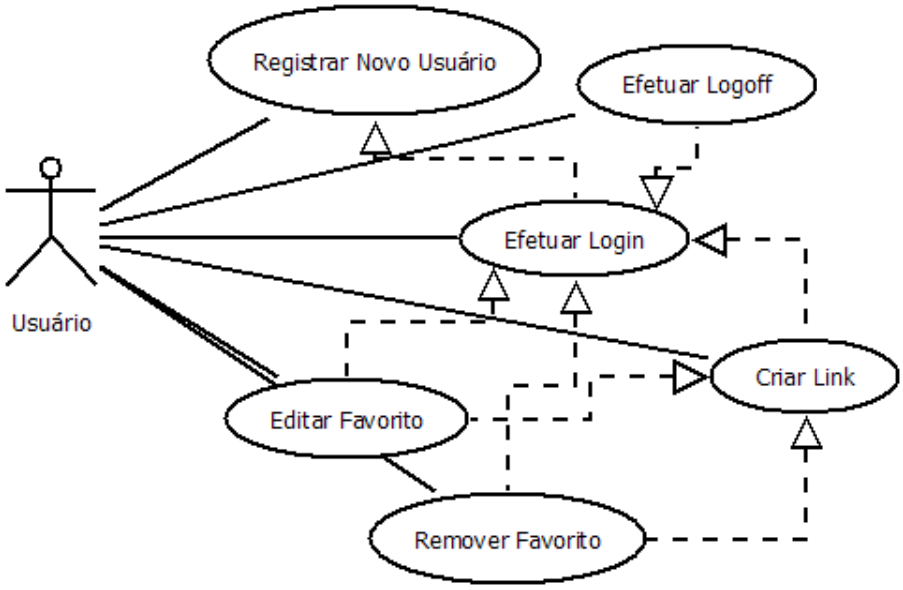

Figura 10 – Caso de uso geral da aplicação em questão.

Através da Figura 10 é perceptível que todas as ações, salvo o registro de um novo usuário, necessitam que o usuário esteja logado no sistema. É uma maneira de separar os

dados por usuário e também de garantir a segurança dos dados, impedindo que eles sejam acessados por outro usuário. O usuário pode também efetuar *login*, e após logado, pode fazer o *logoff*. Para a autenticação do usuário, a aplicação faz o uso da API (*Application Programming Interface*) Servlet do serviço RPC, conforme ilustra a Figura 11, que possibilita o armazenamento de dados na sessão do usuário. O nome de *login* do usuário é então armazenado na sessão até que essa sessão seja invalidada. Caso o usuário volte a acessar a aplicação sem realizar o *logoff*, o servidor verifica que já existe um usuário naquela sessão e carrega os seus dados automaticamente, sem que o usuário precise realizar um novo login.

```
public class LoginServiceImpl extends RemoteServiceServlet implements
LoginService {
  public Boolean login(String login, String password) {
    // Verificar se o usuário existe mesmo
    Usuario res = Persistence.getUsuario(login);
    Boolean result = res != null && res.getSenha().equals(password);
    if (result) {
      getThreadLocalRequest().getSession().setAttribute("user", login);
      System.out.println("Usuário \"" + login + "\" fez login.");
    }
    return result;
  }
  ...
```

Figura 11 – Método responsável pela autenticação do usuário.

O diagrama de sequências (Figura 12) apresentado retrata a sequência de eventos do cadastro de um Favorito. O processo inicia quando o usuário seleciona a opção de cadastro de Favorito, informando os dados e clicando o botão "OK". Em seguida o *Cliente*, que é o navegador, realiza a chamada de um serviço RPC do servidor. O *Servidor*, por sua vez, faz as verificações necessárias, grava o Favorito e retorna o Favorito recém salvo para o *Cliente*, que então atualiza a interface gráfica para mostrar esse Favorito em tela.

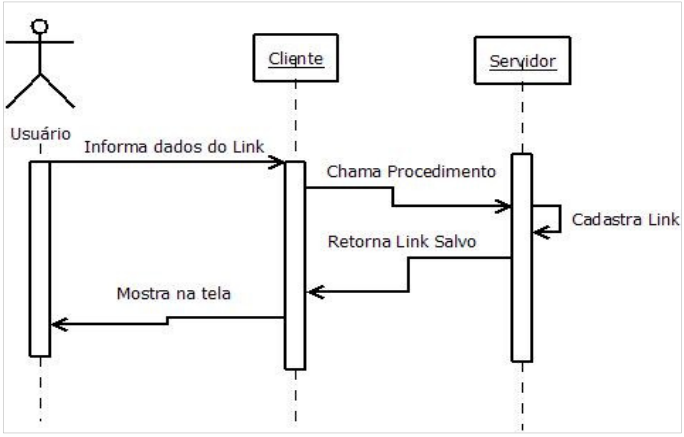

Figura 12 – Diagrama de sequência do cadastro de um Favorito.

Processo semelhante acontece quando um Favorito é removido, porém em vez do usuário informar os dados do favorito ele clica no botão marcado por um “X”. O servidor exclui o favorito a aplicação cliente remove o objeto da tela.

Essas duas operações foram codificadas usando os serviços RPC do GWT. Em cada uma das operações, quando o botão que realiza a ação é acionado, uma chamada assíncrona é feita ao servidor, passando como parâmetros o nome do usuário autenticado e os dados do link a ser cadastrado/removido. O servidor então retorna uma mensagem de sucesso caso a operação tenha sido bem-sucedida, caso contrário retorna uma mensagem de erro.

É importante ressaltar que não foi necessário recarregar a página para cadastrar um novo Favorito, pois quem faz a interação com o servidor é a AJAX *Engine* (ver Figura 1), que nesse caso é o cliente GWT. Essa *engine* é a responsável por fazer a comunicação com o servidor e atualizar a interface gráfica, evitando assim que seja necessário acessar um novo endereço eletrônico para realizar o cadastro. Essa é uma das principais características de aplicações AJAX, a atualização da interface gráfica através do código JavaScript, que realiza a manipulação do DOM.

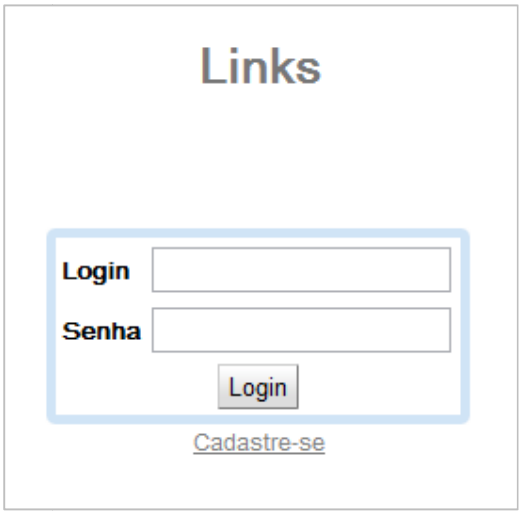

Figura 13 – Tela de login da aplicação de Favoritos.

A tela de login representada pela Figura 13 foi criada usando os *widgets* do GWT. Ela é composta por um *DecoratorPanel* e dentro desse painel foi adicionada uma tabela com o botão e os campos de texto onde o *login* e senha são informados.

Ao acessar a página principal, *index.html*, o arquivo .js principal (chamado de *bootstrap*) é carregado, e esse código JavaScript então determina o arquivo .js que será

carregado em seguida de acordo com o navegador que está sendo usado, bem como a versão do mesmo.

No código, o carregamento da aplicação é feito dentro de um *Entry Point*, ou Ponto de Entrada. Esse Ponto de Entrada é uma classe que implementa a interface *EntryPoint* e que possui somente o método *onModuleLoad*, que é chamado quando a aplicação inicia (HANSON & TACY, 2007). No exemplo em questão, esse método realiza a montagem do painel de login e o exibe para o usuário, conforme ilustra a Figura 14.

```
public class LinkEntry implements EntryPoint {
      RegisterPanel regPanel;
      LoginView loginView;
      Anchor reg;

      public void onModuleLoad() {
            // Init o login view
                  loginView = new LoginView() {

       ...
```

Figura 14 – Carga do painel de *login* no *EntryPoint* da aplicação.

Uma vez que o usuário foi autenticado pela aplicação, a tela principal da aplicação (ilustrada pela Figura 15) é aberta, e seus Favoritos são carregados. A essa tela deu-se o nome de *HomeView*, por ser a visualização principal da aplicação. Nela é possível adicionar, remover e editar Favoritos através dos botões de ação mostrados no lado de cada Favoritos. Nessa tela também é mostrado o nome do usuário e o botão para a realização do logoff do usuário, onde o servidor invalida a sessão do mesmo.

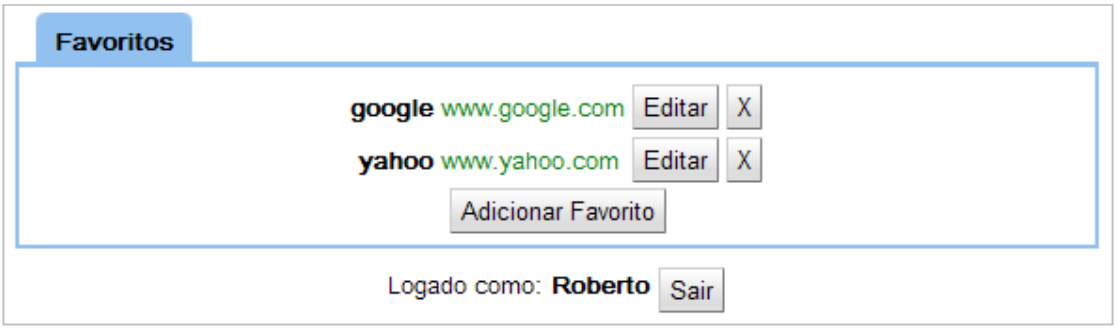

Figura 15 – A tela principal da aplicação, chamada de *HomeView*.

Os botões localizados ao lado da descrição do Favorito são os botões usados para manipular o Favorito. O botão Editar abre uma tela com os dados do Favorito para edição do endereço eletrônico, enquanto o botão "X" remove o Favorito da lista.

Para fazer o cadastro de um Favorito o usuário deve acionar o botão “Adicionar Favorito” que abre a tela de cadastro de Favorito, conforme a ilustração da Figura 16.

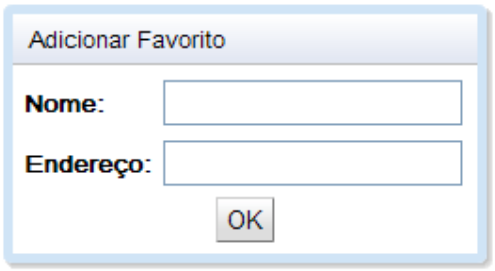

Figura 16 – Tela onde os dados do Favorito são informados.

Após o cadastro de um Favorito, é possível removê-lo clicando no botão marcado por um “X” que fica ao lado dos dados do Favorito, o qual remove o Favorito imediatamente, sem confirmaçãp. Caso o usuário deseje editar um Favorito ele deve usar o botão “Editar”, que fica ao lado do botão de remoção. Essa ação abre um pequeno painel com os dados do Favorito e permite o usuário alterar somente o endereço usado dele, pois o nome é a chave primária, e não pode ser alterada.

Para realizar a troca de usuários deve ser usado o botão Sair, que então fará o logoff do usuário. Para isso o servidor invalida a sessão do usuário para que quando ele retornar à página principal da aplicação a tela de login (Figura 14) seja mostrada.

A *HomeView* foi construída usando um *DecoratedTabPanel*, que é um *Widget* que representa um painel com abas. Dentro desse painel foi colocada a tabela de Favoritos, que é carregada de forma assíncrona através de um serviço RPC que retorna os Favoritos do usuário da sessão. A nível de código, a montagem do painel é muito semelhante à montagem de telas feitas com o Swing, onde no construtor da classe todos os componentes daquela tela são instanciados e organizados dentro da estrutura de painéis.

3.2 REALIZANDO UMA CHAMADA ASSÍNCRONA

Para fazer uma chamada assíncrona através do GWT é preciso primeiro ter um serviço criado. Esse serviço consiste de uma interface de negócio (Figura 17) e a sua versão assíncrona (Figura 18), e também a implementação da interface de negócio.

```
@RemoteServiceRelativePath("login")
```

```
public interface LoginService extends RemoteService {
  Boolean login(String login, String password);
}
```

Figura 17 – Exemplo de interface de negócio de um serviço RPC.

```
public interface LoginServiceAsync {
  void login(String login, String password, AsyncCallback<Boolean>
callback);
}
```

Figura 18 – Exemplo de interface assíncrona.

Na aplicação é necessário instanciar uma representação da interface assíncrona para que possamos chamar os seus métodos, conforme a ilustração da Figura 19.

```
LoginServiceAsync loginService = GWT.create(LoginService.class);
```

Figura 19 – Criação de uma representação da classe *LoginService* no cliente.

Após a criação do objeto que será usado para fazer as chamadas assíncronas já podemos utilizar os métodos que usaremos para realizar a chamada. Para a execução do método, além dos argumentos necessários para a execução do serviço, devemos informar um argumento do tipo *AsyncCallback*, que é uma interface que contém os métodos *onError* e *onSuccess*. A realização dessa chamada é ilustrada pela Figura 20, representada pelo código usado na aplicação em questão para efetuar a autenticação de um usuário no sistema.

No método *onSuccess*, que é executado quando a chamada é concluída com êxito, pode-se fazer a manipulação do retorno da chamada, como por exemplo, realizar as rotinas necessárias quando um login é feito. No método *onError* é feito o tratamento do erro caso a chamada não tiver êxito, e para isso é fornecido um objeto Throwable, que contém informações mais detalhadas sobre o erro. No exemplo em questão esse método é executado somente quando ocorre algum erro no processamento do servidor.

```
String login = "filho", senha = "filho";
```

```
loginService.login(login, senha, new AsyncCallback<Boolean>() {
  public void onFailure(Throwable erro) {
    // Quando a chamada falha esse trecho de código é executado
    ...
  }

  public void onSuccess(Boolean resultado) {
    // Quando a chamada é concluída com êxito, esse trecho é executado
    if (resultado) {
      ...
    }
  }
});
```

Figura 20 – Manipulação do retorno da chamada assíncrona.

4 COMPARAÇÃO (RPC GWT/AJAX JQUERY)

4.1 JQUERY

jQuery é uma biblioteca JavaScript código aberto que visa facilitar o desenvolvimento de páginas Web dinâmicas (JQUERY, 2010) simplificando a interação entre o código JavaScript e o documento HTML (DOM). Ele permite a manipulação (busca e alteração) do CSS de elementos do DOM através de funções que fazem a seleção de elementos que se encaixam em critérios fornecidos através de um *String* de consulta, como ilustra a Figura 21. Também tem suporte para a criação de animações, manipulação de eventos, e também o uso de AJAX.

```
$(".big") // Retorna os elementos da página que pertencem à classe CSS
"big"
$(".big[text]") // Retorna os elementos da classe "big" que contém um
atributo chamado "text"
$("div.big") // Retorna todos os elementos "div" que pertencem à classe
"big"
```

Figura 21 – Exemplos do uso do seletor CSS do jQuery.

Outra maneira do jQuery facilitar o desenvolvimento é lidar com algumas diferenças de padronização entre os navegadores (CASTLEDINE & SHARKIE, 2010). Um grande probema do JavaScript é que cada navegador, e as vezes até versões diferentes de um mesmo navegador, trabalha com JavaScript de maneira diferente, o que dificulta a padronização de uma página Web pois é necessário criar trechos de código a mais para contemplar essas diferenças e fazê-lo funcionar em todos os principais navegadores. O jQuery tenta amenizar essas diferenças em sua implementação incluindo *hacks* e *fixes* para que o código escrito com ele funcione da mesma maneira em versões e marcas de navegadores diferentes.

4.1.1 AJAX com jQuery

O uso da técnica AJAX nao foge desse paradigma, pois ela é feita de forma diferente dependendo da versão e do fornecedor do navegador. A biblioteca em questão procura tratar essa diferença unificando o uso do AJAX em um só conjunto de funções.

O jQuery fornece uma maneira facilitada de trabalhar com AJAX pois simplifica o uso de funções que fazem chamadas assíncronas (Figura 22), porém essas chamadas ainda precisam ser codificadas e inseridas manualmente direto na página HTML através de um arquivo .js.

```
$.ajax("style.css").success(function(data) {
      // O elemento com id "conteudo" recebe o retorno da requisição AJAX
      $("#conteudo").val(data);
});
```

Figura 22 – Exemplo de uso de AJAX usando a biblioteca jQuery.

4.2 EM CONTRASTE

A comparação entre o RPC do GWT e o AJAX com jQuery foi feita usando um exemplo simples que tem como objetivo a comparação entre as duas metodologias de comunicação com o servidor. Foi implementado o processo que a Figura 23 representa tanto com o RPC do GWT quanto com o AJAX do jQuery, e então foi realizada uma comparação entre os dois.

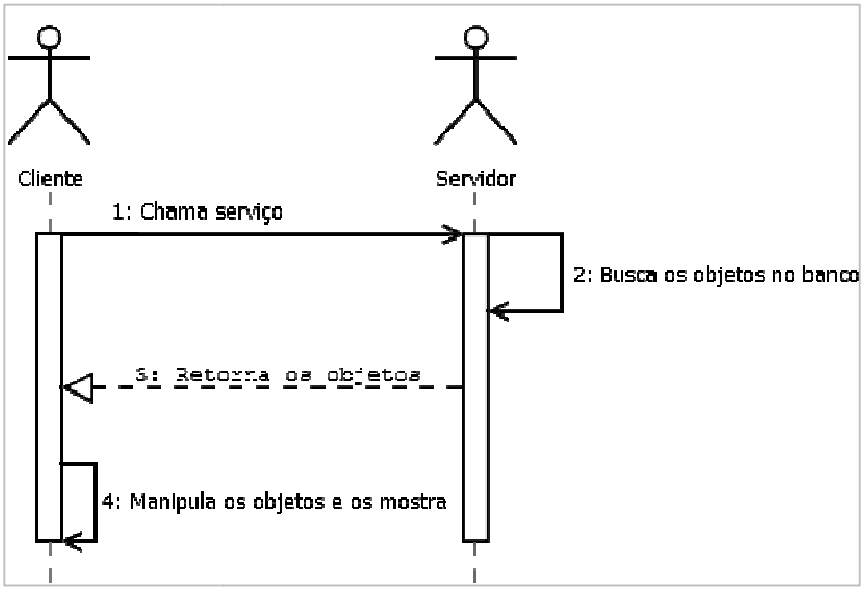

Figura 23 – Processo comum entre as duas formas de desenvolvimento.

O processo da Figura 23 foi executado de duas formas. A primeira delas usando o RPC do GWT para fazer a chamada ao servidor e mostrar os resultados na página Web e para isso será preciso criar uma pequena aplicação. A segunda forma será feita utilizando o AJAX

do jQuery para fazer a chamada assíncrona. Essa chamada irá acessar um serviço no servidor que retornará um objeto JavaScript com os resultados para facilitar a renderização dos mesmos, simulando o que RPC do GWT faz.

4.2.1 Complexidade

A primeira observação feita na abordagem do GWT é que a complexidade da aplicação é razoável, se comparado à simplicidade do processo do exemplo. Foi necessário configurar um serviço para o servidor e também realizar a chamada desse serviço no cliente, tratando exceções e criando rotinas.

Utilizando a técnica AJAX com o jQuery, a complexidade da aplicação diminui, pois ela possui somente um pequeno comando que faz a chamada do serviço e incorpora os resultados à página.

Após a observação da complexidade de cada abordagem é possível concluir que a complexidade da aplicação GWT é ligeiramente maior que a complexidade da aplicação jQuery. Acontece dessa maneira pois o GWT é uma ferramenta de desenvolvimento de aplicações Web. Ele fornece ferramentas para o desenvolvimento tanto o lado do cliente quanto o lado do servidor (criação de serviços) e realiza uma integração transparente entre os dois.

O jQuery, entretanto, facilita o trabalho somente no lado do cliente. Se comparado ao GWT, a biblioteca é uma ferramenta simples, porém como o pequeno exemplo usado também é simples, ela se ajusta de maneira satisfatória às necessidades.

Outra observação feita foi quanto a quantidade de código. O desenvolvimento do exemplo com o GWT necessitou de um volume maior de código do que a sua versão feita com o jQuery. Obteve-se o mesmo desempenho com o jQuery que com o RPC do GWT. Enquanto que com o AJAX do jQuery o problema foi solucionado com um simples bloco de código JavaScript, com o GWT foi necessário a criar uma classe completa para a chamada do serviço remoto e manipulação do retorno.

4.2.2 Usabilidade

Para o uso do GWT não é necessário que o desenvolvedor saiba usar ou tenha experiência com JavaScript, porém a falta desse conhecimento pode levar a dificuldades no desenvolvimento. Outro ponto relevante é que o GWT possui o apoio de várias IDEs (*Integrated Development Environments*) e também *plugins* que facilitam o desenvolvimento de aplicações Web, sendo que o Java possui grande aceitação da comunidade de desenvolvimento (KEREKI, 2011).

Quanto à usabilidade do jQuery, é desnecessário dizer que o conhecimento de JavaScript é necessário, tanto para o desenvolvimento de aplicações como o uso do AJAX. Porém para que o desenvolvedor consiga realizar um uso eficiente da biblioteca é recomendável que ele tenha conhecimento na área de *Web Standards*, bem como de manipulação do DOM.

4.2.3 Manutenibilidade

A manutenibilidade diz respeito a facilidade de modificação e à procura e tratamento de erros. Ao comparar os dois exemplos verificamos qual a facilidade de alterar a aplicação, como por exemplo, adicionar a funcionalidade de alteração dos dados, e também como ambos facilitam ou não o tratamento de erros.

A manutenção do código do GWT é relativamente simples, pois a descrição dos erros é bem legível, e não existem muitos segredos envolvendo os recursos usados. O tratamento de exceções ao codificar uma chamada remota é feito em um método chamado *onFailure*. Nesse método são realizadas todas as verificações pertinentes à exceção, sendo que esse método tem como argumento um objeto *Throwable*. Dessa forma o tratamento da exceção é centralizado e facilita a identificação do erro, bem como a interação da aplicação com o usuário, caso um erro venha a ocorrer.

Quanto ao jQuery, ele não oferece feramentas para a realização do *debugging* do código, como achar e tratar os erros fica a critério do desenvolvedor. Contudo a alteração de uma chamada assíncrona, caso seja necessário, é simples, pois não existem interfaces de negócio e nem cópias assíncronas, e também não é preciso compilar o projeto para colocá-lo em produção.

4.3 CONSIDERAÇÕES FINAIS

Colocando as duas ferramentas em contraste, nota-se que a principal diferença entre o *Remote Procedure Call* do *Google Web Toolkit* e a funcionalidade AJAX da biblioteca JavaScript jQuery é o nível de abstração.

O RPC do GWT é uma parte – um módulo – dentro de um conjunto completo de ferramentas para o desenvolvimento de aplicações Web. Esse conjunto de ferramentas possibilita o intercâmbio de objetos entre servidor e cliente de modo transparente, sem que o desenvolvedor precise, manualmente, interpretar os dados vindos do servidor para que eles se tornem usáveis no cliente. Ou seja, o GWT abstrai a comunicação do cliente com o servidor, onde o desenvolvedor não trabalha mais com chamadas assíncronas e manipulação do DOM, mas sim com a troca de objetos entre o cliente e o servidor.

O AJAX do jQuery, por sua vez, trabalha um nível abaixo, com chamadas assíncronas (através do objeto JavaScript *XMLHttpRequest*) de endereços eletrônicos (URL). O nível de abstração é muito menor do que a do RPC do GWT, pois os dados retornados por essas chamadas são dados brutos, sem nenhuma interpretação, e que , antes que se tornem usáveis, necessitam dessa interpretação. Com o jQuery essa tarefa deve ser realizada pelo desenvolvedor, que deve usar bibliotecas de terceiros (o que aumentaria o tamanho da aplicação) ou codificar a sua própria implementação, que demandaria um conhecimento e tempo extra.

De forma resumida, ambas as ferramentas (o RPC do GWT e o AJAX com jQuery) usam as mesmas técnicas para comunicar-se com o servidor. A diferença é que o RPC do GWT fornece os resultados interpretados e prontos para serem usados, abstraindo o uso de chamadas assíncronas, enquanto o AJAX feito com o jQuery retorna os dados brutos, sem nenhuma forma de interpretação, cabendo ao desenvolvedor a interpretação dos dados retornados pela chamada assíncrona.

5 CONSIDERAÇÕES FINAIS

5.1 CONCLUSÃO

Diante das informações apresentadas é possível concluir que o GWT é uma ferramenta inovadora, que fornece vantagens ao desenvolvedor. Uma facilidade de grande importância é a eliminação da necessidade de adaptar o código para funcionar em vários navegadores.

Quanto à utilização de chamadas assíncronas, o GWT possibilitou a integração da programação orientada a objeto no cliente quanto no servidor, algo que até então era feito de maneira separada. O desenvolvimento da comunicação entre o cliente e o servidor pode ser feito de maneira transparente deixando que a ferramenta tome conta da integração entre os dois.

Entretanto pode-se identificar algumas desvantagens. Entre elas a dificuldade da manutenibilidade do código, onde para realizar uma tarefa simples se torna inviável o uso do GWT, por ser uma ferramenta robusta que necessita de mais recursos para funcionar.

5.2 TRABALHOS FUTUROS

Verifica-se que o GWT, apesar de ser uma ferramenta já difundida, está em constante desenvolvimento. Quando esse trabalho foi iniciado, a ferramenta encontrava-se na versão 1.5, ao escrever essa conclusão, um ano e meio depois, ela se encontra na versão 2.3, agora com suporte à HTML5 e integração com serviços da Google. A Web evolui muito rápido, creio que não seja possível prever o seu futuro. Portanto se faz necessário que a busca do desenvolvedor pelo conhecimento que o auxiliará em seu trabalho (em específico da tecnologia em questão, o GWT) , seja continuada e aprofundada, pois agora existem novos recursos a serem analisados e relevados.

REFERÊNCIAS BIBLIOGRÁFICAS

CASTLEDINE, Earle; SHARKIE, Craig. **jQuery: Novice to Ninja.** Victoria: Sitepoint Pty. Ltd., 2010.

COOPER, Robert T.; COLLINS, Charlie E.. **GWT In Practice.** Connecticut: Manning Publications Co., 2008.

DEITEL, Paul; DEITEL, Harvey. **Java for Programmers Second Edition.** Massachusetts: Prentice Hall, 2011.

DEWSBURY, Ryan. **Google Web Toolkit Applications.** Boston: Prentice Hall, 2008.

GOOGLE. **What is Deferred Binding?**. Disponível em: <http://code.google.com/intl/pt-BR/webtoolkit/doc/1.6/FAQ_Client.html#What_is_Deferred_Binding?>. Acesso em 21 mar. 2011.

GUPTA, Vipul. **Accelerated GWT: Building Enterprise Google Web Toolkit Applications.** California: Apress, 2008.

HANSON, Robert; TACY, Adam. **GWT In Action: Easy Ajax with the Google Web Toolkit.** Connecticut: Manning Publications Co., 2007.

HUNTER, Jason; CRAWFORD, William. **Java Servlet Programming, 2nd Edition.** California: O'Reilly, 2001.

INTERNET ENGINEERING TASK FORCE. Disponível em <http://tools.ietf.org/html/rfc707>. Acesso em: 14 mar. 2011.

JQUERY. Disponível em <http://jquery.com>. Acesso em 24 mar. 2011.

KEREKI, Federico. **Essential GWT Building for the Web with Google Web Toolkit 2.** Massachusetts: Addison-Wesley, 2011.

O'REILLY, Tim. **What Is Web 2.0**. Disponível em: <http://oreilly.com/pub/a/web2/archive/what-is-web-20.html> Acesso em 6 jun. 11.

SMEETS, Bram; BONESS, Uri; BANKRAS, Roald. **Beginning Google Web Toolkit: From Novice to Professional.** California: Apress, 2008.

SCHMITT, Christopher et al. **Adapting to Web Standards: CSS and Ajax for Big Sites**. California: New Riders, 2008.

ZELDMAN, Jeffrey. **Designing With Web Standards, Second Edition.** California: New Riders, 2007.