INTEGRAÇÃO DA METODOLOGIA DE DESENVOLVIMENTO AJAX COM SERVLETS JAVA

Everton Coimbra de Araújo[1], Juliano Rodrigo Lamb[2], Ricardo Sobjak[3], Jefferson Gustavo Martins[4]

RESUMO: A Internet passou por uma grande mudança para chegar ao estágio de interação com o usuário atual. Este artigo tem o propósito de apresentar a metodologia de desenvolvimento AJAX, tornando aplicativos Web com recursos de interação com o usuário, semelhantes aos oferecidos por aplicativos Desktop, para isso fazendo integração com Servlets Java.

PALAVRAS-CHAVE: desenvolvimento web, conteúdo dinâmico.

1 INTRODUÇÃO

Para aumentar a usabilidade, interatividade e funcionalidade de aplicações *Web*, surgiu um conceito inovador no modo de pensar e desenvolver, *RIA* (*Rich Internet Application*), que une a funcionalidade dos softwares para computadores de mesa aos benefícios oferecidos pela Internet, tornando o uso e a criação de aplicações *Web* mais intuitivas e eficientes.

Os aplicativos de *Rich Internet* são baseados na tecnologia *Rich Client*, que fornece um ambiente dinâmico, com capacidade de hospedagem de aplicativos que são compilados no lado do servidor e recebidos no *browser*, através de requisições HTTP (*HyperText Transfer Protocol*) (COSTA, 2006). RIA são aplicações que buscam prover uma nova classe de *Websites* interativos, com a sofisticação de aplicações *desktop*, mas que não comprometem a facilidade de desenvolvimento, implementação e manuseio dos aplicativos *Web*. A arquitetura típica para uma RIA é mostrada na Figura 1.

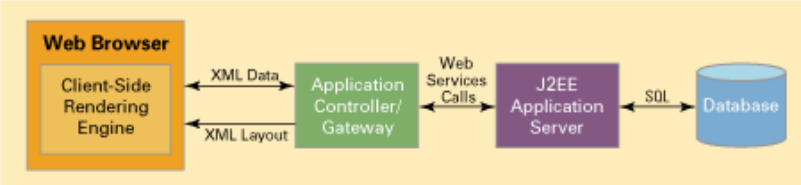

Figura 1. Arquitetura típica RIA (O'ROURKE, 2004).

[1] Mestre em Ciência da Computação, Professor da Universidade Tecnológica Federal do Paraná (UTFPR) – campus de Medianeira, everton@pptinformatica.com.br
[2] Mestre em Engenharia Agrícola com concentração em Engenharia de Sistemas Agroindustriais, Professor da Universidade Tecnológica Federal do Paraná (UTFPR) – campus de Medianeira, juliano@x87.eti.br
[3] Acadêmico do curso de Análise e Desenvolvimento de Sistemas (UTFPR) ricardo_sobjak@hotmail.com
[4] Mestre em Engenharia de Produção e Sistemas, Professor da Universidade Tecnológica Federal do Paraná (UTFPR) – campus de Medianeira, martins@utfpr.edu.br

A adoção crescente da tecnologia *Rich Client* não é uma etapa evolutiva de substituição a HTML. Consiste em oferecer aplicações *Web* com capacidade de navegação mais eficazes e responsivas. A maioria dos aplicativos "*Rich*" são executados no contexto dos *browsers*, dentro das páginas, ou seja, junto com o conteúdo HTML (*Hypertext Markup Language*). Nesse contexto RIA, surge um importante recurso para o desenvolvimento de aplicações *Web*, o AJAX (*Asynchronous Javascript And XML*) (COSTA, 2006).

Desde o início da Internet, os programadores têm encontrado uma série de dificuldades com relação à performance de seus códigos no lado cliente. Em tempo de produção, com uma situação ideal de pouco tráfego e alta velocidade de rede tudo se torna bom, mas quando as páginas são distribuídas na *Web*, os problemas com a demora nos "*reloads*" das páginas, a cada solicitação ao servidor começam a aparecer (AMSTEL, 2006). Com JSP, isso não é diferente. A proposta é oferecer uma aplicação com conteúdo dinâmico, em que o tempo de processamento reduzido, fazendo o uso da linguagem Java através de páginas JSP e Servlets, integrados com a metodologia de desenvolvimento AJAX.

2 MATERIAL E MÉTODOS

O conceito de convergência de diversas tecnologias já existentes para a apresentação de soluções mais eficientes na construção de *Websites*, faz da implementação de códigos com AJAX uma realidade (Figura 2), a qual não se pode ignorar.

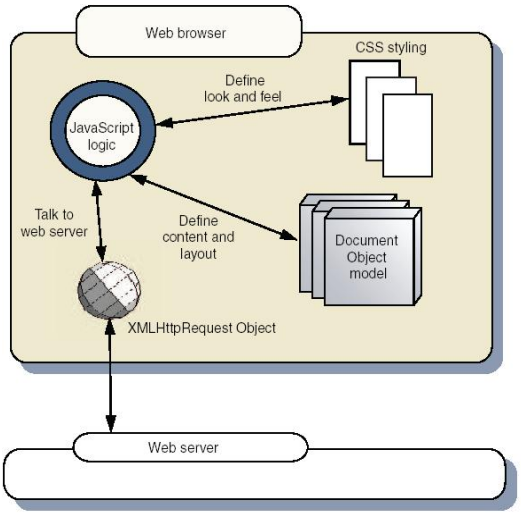

Figura 2. Arquitetura do AJAX englobando seus principais componentes
Fonte: (CRANE, 2006).

Nessa metodologia existem alguns elementos a serem considerados. O objeto XMLHttpRequest fornece duas propriedades que concedem acessa à resposta do servidor:

- responseText: fornece a resposta como uma string;
- responseXML: fornece a resposta como um objeto XML.

O uso de responseText é indicado para respostas simples, como exemplo, para exibição em caixa de alerta ou mensagens informativas de sucesso ou falha.

Ao clicar no botão "Enviar (GET)" ou "Enviar (POST)", será chamado a função "processar", que receberá um parâmetro de entrada GET ou POST. Essa função, começa chamando uma instância do objeto XMLHttpRequest, em seguida é criada a string de

consulta e por fim é feita uma verificação a partir da string de entrada. Se o valor passado for GET, então a função “doGet” será chamada, se o valor passado for POST, então será chamada a função “doPost”. A Figura 3 representa a Diagrama de Atividades da Aplicação de Exemplo.

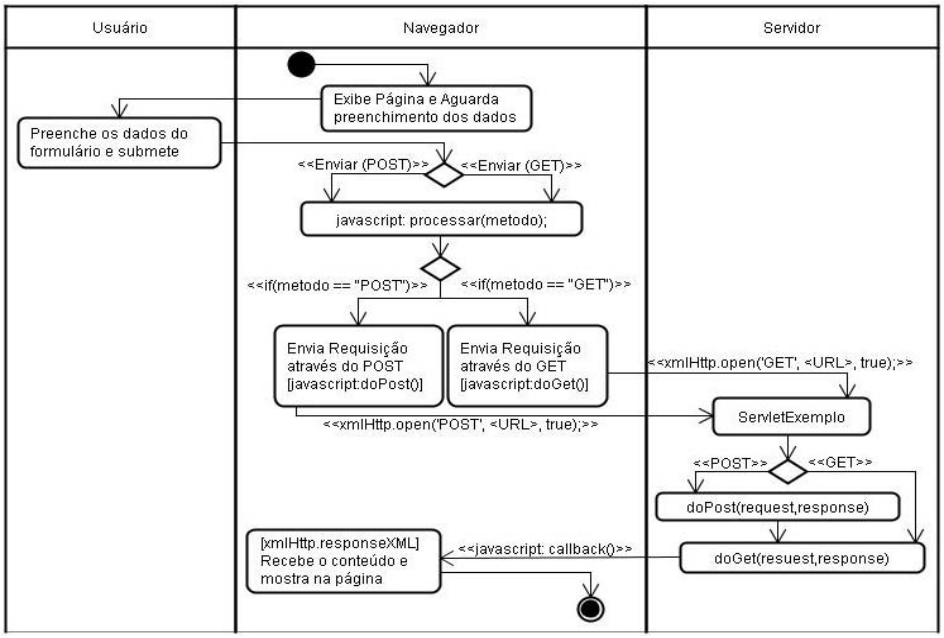

Figura 3. Diagrama de Atividades

A função “*criarQueryString*” é responsável por codificar os parâmetros de entrada como uma string de consulta. Ao clicar no botão “Enviar (GET)” ou “Enviar (POST)”, será chamado a função “processar”, que receberá um parâmetro de entrada GET ou POST. Essa função, começa chamando uma instância do objeto *XMLHttpRequest*, em seguida é criada a string de consulta e por fim é feita uma verificação a partir da string de entrada. Se o valor passado for GET, então a função “*doGet*” será chamada, se o valor passado for POST, então será chamada a função “*doPost*”.

O método GET passa o valor na forma de pares nome/valor como parte da URL da solicitação. A URL do recurso termina com um ponto de interrogação (?) e depois vem os pares nome/valor. Os pares nome/valor estarão na forma nome=valor e serão separados por um E comercial (&). Exemplo de uma URL de solicitação GET: “http://localhost:8080/AppExemplo/ServletExemplo?timeStamp=1159794838187&nome=Ricardo&telefone=(45) 0000-0000&cidade=Medianeira”.

O método POST é quase idêntico ao método GET, pois codifica os parâmetros como pares sendo separados com um E comercial. A principal diferença entre os dois é que o método POST envia a string do parâmetro dentro do corpo da solicitação em vez de acrescentá-la a URL como faz o método GET.

3 RESULTADOS E DISCUSSÕES

A Figura 4 mostra um exemplo de página, ilustrando um formulário de entrada simples com componentes HTML, contendo três caixas de texto, para ser informado nome, telefone e a cidade em que a pessoa mora, também possui dois botões. Os dois botões solicitam uma função de JavaScript que enviam os dados para o servidor, um utilizando o método GET e outro utilizando POST.

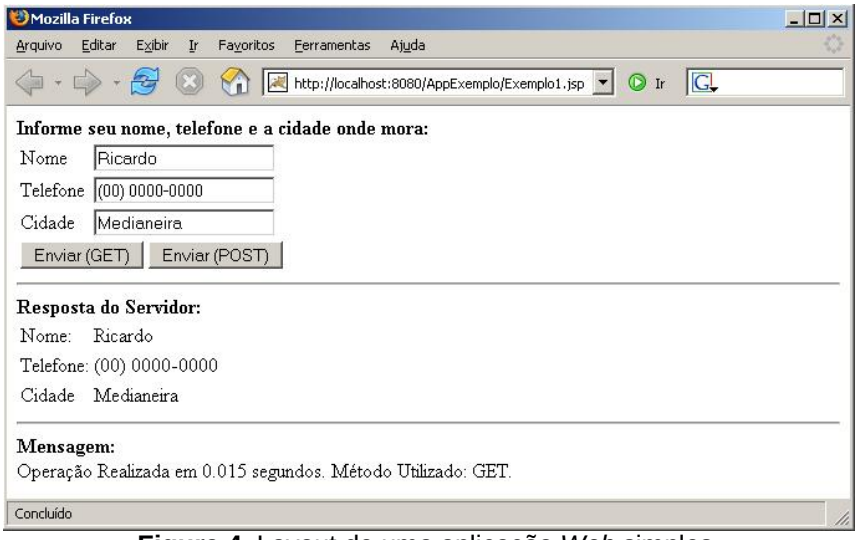

Figura 4. Layout de uma aplicação *Web* simples.

Ao verificar as técnicas para o uso do método GET e POST, é possível concluir que são muito parecidas, diferenciando-se na maneira de como os parâmetros são enviados para o servidor. Em ambas funções JavaScript, "*doGet*" e "*doPost*", é configurada a propriedade "*onreadystatechange*" do objeto *XMLHttpRequest* para usar a função "*callback*". Essa propriedade, contém um ponteiro que aponta para uma função de JavaScript, que é acionada a cada mudança de estado através do manipulador de eventos.

O método *open*() do objeto *XMLHttpRequest*, é implementado de maneira diferente nas funções "*doGet*" e "*doPost*. Na função "*doGet*", o primeiro parâmetro indica que essa é uma requisição do tipo GET e o segundo parâmetro especifica a URL da extremidade de destino, juntamente com os dados a serem enviados. Na função "*doPost*", o primeiro parâmetro é especificado como uma requisição POST e o segundo parâmetro indica a URL apenas do caminho de quem estará processando as informações. O método "*send*()", envia a solicitação para o servidor. Utilizando o método POST, é passado uma string, contendo os dados "nome/valor", como parâmetro no método "*send*". Ao usar GET, o método "*send*" terá como parâmetro apenas a palavra *null*.

Ao utilizar o método POST, é necessário garantir que o servidor tenha conhecimento que os parâmetros possam ser encontrados dentro do corpo da solicitação, o método "*setRequestHeader*" é chamado e configurado o valor de *Content-Type* com "*application/x-www-form-urlencoded*".

Na URL mostrada no exemplo, existe um parâmetro chamado "*timestamp*" que representa a hora atual. Isso é feito porque alguns navegadores armazenam em *cache* os resultados de várias solicitações *XMLHttpRequest* feitas para a mesma URL. Acrescentar o carimbo de hora atual garante a exclusividade da URL, impedindo que os navegadores armazenem os resultados em *cache*.

No método "*doGet*", os três campos de entrada são recuperados a partir do objeto de solicitação com o uso do método "*getParameter*". Logo após é configurado o tipo de conteúdo da resposta com "*text/xml*". É feita uma chamada ao método "*criarXML*", que retorna um conteúdo XML, que em seguida é gravada no fluxo de saída da resposta e, para concluir, o fluxo é fechado.

4 CONCLUSÃO

Essa metodologia de desenvolvimento vem ganhando espaço no meio dos desenvolvedores de aplicações *Web*, por apresentar menor uso de banda, resposta mais

rápida, além de oferecer maior interatividade e usabilidade ao usuário, tornando a página mais natural, menos fria, impessoal e melhor apresentável. O desenvolvedor J2EE, não necessita se especializar em uma nova tecnologia, viabilizando o uso da poderosa linguagem Java e a integração dos recursos AJAX.

REFERÊNCIAS

AMSTEL, F. V. **Webdesigner 2.0**. Web Insider. Disponível em <http://webinsider.uol.com.br/vernoticia.php/id/2770>. Acesso em 28 de setembro de 2006.

COSTA, H. F. D. D. **Internet Rica (RIA)**. Disponível em <http://www.henry.eti.br/pagina.php?IdPagina=258>. Acesso em 29 de setembro de 2006.

CRANE, D.; PASCARELLO, E.; JAMES, D. **Ajax In Action.** Manning Publications Co. Greenwich, 2006.

O'ROURKE, C. **A Look at Rich Internet Applications**. Oracle Magazine. Edição Julho/2004.