MODELO SPI (*SINGLE PAGE INTERFACE)* PARA DESENVOLVIMENTO DE APLICAÇÕES WEB

Thiago Nery Teixeira

Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Orientador: Ely Edison Matos

Juiz de Fora, Minas Gerais
Julho de 2009

MODELO SPI (*SINGLE PAGE INTERFACE)* PARA DESENVOLVIMENTO DE APLICAÇÕES WEB

Thiago Nery Teixeira

Monografia submetida ao corpo docente do Departamento de Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora, como parte integrante dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Aprovada pela banca constituída pelos seguintes professores:

____

Ely Edison Matos – orientador
Msc em Modelagem Computacional, UFJF, 2008

____

Prof. Fernanda C. A. Campos
DSc. em Eng. de Sistemas e Computação, COPPE/UFRJ, 1999

____

Prof. Tarcísio de Souza Lima
MSc. em Informática, PUC/RJ, 1988

Juiz de Fora, Minas Gerais
Julho de 2009

Sumário

LISTA DE FIGURAS....iv

LISTA DE TABELAS....vi

LISTA DE REDUÇÕES....vii

RESUMO....viii

1 INTRODUÇÃO....1

1.1 OBJETIVO....2

1.2 ORGANIZAÇÃO DO TRABALHO....3

2 MODELOS DE DESENVOLVIMENTO DE APLICAÇÕES WEB....4

2.1 MODELO DE SUBMISSÃO DE FORMULÁRIOS....4

2.2 MODELO DE RENDERIZAÇÃO PARCIAL....6

2.3 RIA – *RICH INTERNET APPLICATIONS*....7

2.4 *SINGLE PAGE INTERFACE* (SPI)....9

3 *SINGLE PAGE INTERFACE*....11

3.1 AJAX....11

3.1.1 Javascript....15

3.1.2 *Cascading Style Sheets* (CSS)....16

3.1.3 *Document Object Model*....17

3.1.4 XMLHTTPRequest....19

3.2 SPI....20

3.3 PADRÕES DE PROJETO AJAX....21

3.4 FRAMEWORKS AJAX....23

3.4.1 *Google Web Toolkit* (GWT)....23

3.4.2 *Echo2*....24

3.4.3 *Backbase*....25

3.4.4 *Qooxdoo*....25

3.4.5 Dojo....26

4 ESTUDO DE CASO – MIOLO 2.5....28

4.1 ARQUITETURA....29

4.2 CAMADA DE APRESENTAÇÃO....33

4.2.1 Controles....33

4.2.2 Temas....34

4.2.3 *Single Page Interface*....37

4.2.4 Exemplo....40

5 CONSIDERAÇÕES FINAIS.....47
6 - REFERÊNCIAS BIBLIOGRÁFICAS.....49
APÊNDICE 1 – Exemplo de uma *single-page* usando o *framework* Dojo.....51
APÊNDICE 2 – Dados coletados para comparação.....57
APÊNDICE 3 – Código da nova interface do lançamento de notas parciais.....59
ANEXO 1 – Árvore de controles do Miolo 2.5.....72

LISTA DE FIGURAS

Lista de Figuras

Figura 1: Documento texto HTML e sua apresentação em um navegador............................4
Figura 2: Um formulário HTML..5
Figura 3: Modelo de submissão de formulários (ESPOSITO, 2008).....................................6
Figura 4: Modelo de renderização parcial (ESPOSITO, 2008) 7
Figura 5: Elementos SPI em uma página (ESPOSITO, 2008)...9
Figura 6: Modelo de aplicação Ajax (GARRET, 2005)...12
Figura 7: Padrão de interação assíncrona de uma aplicação Ajax (GARRET, 2005) 13
Figura 8: Aplicação web tradicional X Aplicação Ajax (CRANE et al., 2006).........................14
Figura 9: Funcionamento das tecnologias que compões o Ajax (CRANE et al, 2006)...........15
Figura 10: Uma função Javascript...16
Figura 11: Estilo de uma interface de usuário definido via CSS(CRANE et al, 2006)............17
Figura 12: Definição de estilo CSS...17
Figura 13: Estrutura de árvore DOM para uma tabela HTML (W3C, 1998). 18
Figura 14: Exemplo de Javascript para manipulação do DOM..18
Figura 15: Exemplo da notação JSON. 19
Figura 16: Criação de um objeto XMLHTTPRequest (SILVA,2007).....................................20
Figura 17: Exemplo de renderização parcial no Miolo 2..29
Figura 18: Arquitetura em camadas do Miolo...30
Figura 19: Estrutura de diretórios do Miolo..31
Figura 20: Estrutura de diretórios pré-definida de um módulo do Miolo. 32
Figura 21: Exemplo de formulário criado no Miolo...34
Figura 22: Estrutura padrão do tema *blue*..35
Figura 23: Organização lógica da estrutura do tema..35
Figura 24: Estrutura de diretórios do tema *blue*. 36
Figura 25: Código HTML da “*single-page*” base do Miolo..38
Figura 26: Painel de ações do docente..41
Figura 27: Trecho de código do formulário principal com os objetos MDiv 41
Figura 28: Interface de usuário após a escolha da turma...42
Figura 29: Trecho de código do evento Ajax executado...42
Figura 30: Lançamento dos dados da primeira avaliação. 43
Figura 31: Formulário após o lançamento das notas da primeira avaliação.........................43

Figura 32: Formulário após o lançamento de três avaliações..44
Figura 33: *Site Single-Page Interface*...51
Figura 34: Código HTML da “*single page*”...51
Figura 35: Funções Javascript para o menu do site...52
Figura 36: Arquivo HTML com conteúdo para o site...52
Figura 37: Definição do objeto de estado para navegação..53

LISTA DE TABELAS

Tabela 1: Padrões de projeto Ajax..21
Tabela 2: Resultados obtidos para o cenário 1..45
Tabela 3: Resultados obtidos para o cenário 2..45
Tabela 4: Resultados obtidos para o cenário 3..46

LISTA DE REDUÇÕES

AJAX	*Asynchronous Javascript + XML*
API	*Application Programming Interface*
CGI	*Common Gateway Interface*
CSS	*Cascading Style Sheets*
DOM	*Document Object Model*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
JSON	*Javascript Object Notation*
RIA	*RICH INTERNET APPLICATIONS*
SIGA	Sistema Integrado de Gestão Acadêmica
SGBD	Sistema Gerenciador de Banco de Dados
SPI	*Single Page Interface*
URI	*Uniform Resource Identifier*
URL	*Uniform Resource Locator*
W3C	*World Wide Web Consortium*
XHTML	*eXtensible Hypertext Markup Language*
XML	*eXtensible Markup Language*
XSLT	*eXtensible Stylesheet Language Transformations*

RESUMO

Com a evolução das tecnologias web, muitas aplicações *desktop* estão sendo transferidas para esta plataforma. Porém, se deparam com as limitações do ambiente, que, baseado em protocolo do tipo requisição-resposta limita a interação do usuário com a aplicação. Aplicações web clássicas são baseadas em um modelo de interface de múltiplas páginas, no qual cada interação segue um paradigma de seqüência de páginas. O objetivo deste trabalho é apresentar o modelo de desenvolvimento web chamado *Single Page Interface (*SPI*),* ou modelo de página única. Com ele é possível desenvolver aplicações web mais ricas e interativas. É também apresentado um estudo de caso do *framework* Miolo 2.5, que implementa o modelo SPI, com um exemplo de sua utilização

1 INTRODUÇÃO

Desde sua difusão para o público em geral em 1994, a web tem passado por um contínuo processo de evolução. Web é o termo pelo qual se popularizou o serviço internet dedicado ao compartilhamento de documentos multimídia. No início da década de noventa, Tim Berners-Lee, considerado o criador da web, estava trabalhando em um sistema de gerenciamento de informações, no qual o texto poderia conter referências a outros trabalhos, permitindo ao leitor transitar de maneira rápida entre os documentos referenciados. Para a criação e exibição desse tipo de documento, conhecido como hipertexto, Berners-Lee criou uma aplicação a qual deu o nome de *WorldWideWeb* (BERNERS-LEE, 1989).

Quatro princípios nortearam a criação da web (BERNERS-LEE, 1996):

- Independência de especificações;
- O *Uniform Resource Identifier* (URI), uma seqüência de caracteres única, usada para identificar um recurso (como uma página) no ambiente web. A URL(*Uniform Resource Locator*) é uma URI que além de especificar o recurso informa também qual o protocolo será usado para o acesso;
- A linguagem de marcação de hipertexto (HTML - *HyperText Markup Language*), para a exibição e formatação de dados em um documento hipertexto;
- O protocolo de transferência de hipertexto (HTTP - *HyperText Transfer Protocol*), um protocolo de comunicação do tipo requisição-resposta (*request-response*), que não possui informações sobre estado para sistemas de informação distribuídos de hipermídia.

Baseado nesses princípios foi criada uma simples, porém poderosa arquitetura cliente/servidor, na qual os recursos podem ser ligados e acessados facilmente através de navegadores[1]. No início a web era composta apenas por documentos simples de hipertexto, que por sua vez continham referências a outros documentos (hiperlinks). A navegação funciona no momento em que uma requisição é enviada ao servidor, a partir do acesso a uma URL pelo navegador. O servidor se encarrega de localizar e recuperar o documento correspondente e enviá-lo de volta ao navegador, que usa esse documento para recarregar toda a interface. Esses documentos de hipertexto são ditos “estáticos” pois estão armazenados no sistema de arquivos do servidor e não sofrem nenhuma alteração em sua estrutura, da requisição inicial ao seu envio para o navegador. (MESBAH, 2009)

1 Programas utilizados como clientes para acesso a um servidor de documentos hipertexto (servidor web), com o qual se comunicam através do protocolo HTTP. Exemplos de navegadores: Mozilla Firefox, Internet Explorer, Google Chrome.

Com a ampla adoção da web, a necessidade de aplicações mais complexas surgiu e foi sanada com a criação de servidores capazes de usar de linguagens de programação para a criação de páginas de forma dinâmica. Com isso os servidores passaram a atender duas situações: se a requisição aponta para um arquivo armazenado em disco, devolve o conteúdo do arquivo (estático), senão, de acordo com os parâmetros enviados, uma nova página é criada e associada à requisição (dinâmico). As primeiras páginas dinâmicas foram criadas com o auxílio de linguagens de *script* que possibilitam a programação do lado do servidor através de uma interface chamada CGI - *Common Gateway Interface* (ROBINSON, 2004).

Com a crescente popularização, aliado ao seu poder comercial, muitas características que não estavam previstas na concepção original foram incorporadas à web. Cientes desse poder comercial muitas empresas a usaram para apresentar seus produtos e serviços. A web deixou de ser apenas uma fonte de informação, oferecendo serviços como comércio eletrônico e *home-banking*. A partir de 2001, após uma crise financeira que afetou principalmente as empresas “ponto com”[2], muitos concluíram que esse modelo de negócios estava saturado (O'REILLY, 2005). Porém essa crise, longe de indicar o fim do potencial da web, é tida como o ponto de início de um novo modelo de desenvolvimento. A partir de então a web ganhou uma estrutura mais dinâmica, privilegiando a interação com o usuário. Serviços simples, focados na colaboração e participação ganharam destaque (SILVA, 2007).

Surgiu assim um novo conjunto de princípios e práticas para o desenvolvimento web. O termo Web 2.0 foi criado para designar essa nova etapa no contínuo processo de evolução da web, que segundo O'REILLY (2005) possui diversos significados, entre eles: a web como plataforma, dados com importância superior aos documentos, a participação e a colaboração do usuário, a distribuição e a descentralização do conteúdo, aplicativos sendo desenvolvidos para diversos dispositivos, atualizações freqüentes dos serviços e aplicações com design mais rico.

1.1 OBJETIVO

O objetivo deste trabalho é apresentar o modelo de desenvolvimento web chamado *Single Page Interface (*SPI*),* ou modelo de página única. Com ele é possível desenvolver aplicações web mais ricas e interativas. É também apresentado um estudo de caso do *framework* Miolo 2.5, que implementa o modelo SPI, com um exemplo de sua utilização.

---

2 Termo que foi usado para designar as empresas que focaram seu modelo de negócio na internet. Faz menção ao endereço usado nos navegadores para acessar um site. Exemplo: *www.google.com*

1.2 ORGANIZAÇÃO DO TRABALHO

O trabalho está organizado da seguinte forma: no capítulo 2 são mostrados alguns modelos de desenvolvimento utilizados em aplicações web. O capítulo 3 apresenta o modelo SPI, tema desse trabalho, focando nas tecnologias que o compõem. No capítulo 4 é feito um estudo de caso do *framework* Miolo versão 2.5, uma ferramenta que auxilia no desenvolvimento de aplicações web e implementa o modelo SPI. No capítulo 5 são apresentadas as considerações finais e indicações para trabalhos futuros.

2 MODELOS DE DESENVOLVIMENTO DE APLICAÇÕES WEB

As seções a seguir fazem uma rápida revisão dos principais modelos de desenvolvimento utilizados na criação de aplicações web.

2.1 MODELO DE SUBMISSÃO DE FORMULÁRIOS

A linguagem de marcação HTML é utilizada para a criação das páginas web. Foi originalmente desenvolvida por Tim Berners-Lee e se expandiu rapidamente devido à popularização da web, ganhando diversas extensões. Isto motivou a criação de um trabalho conjunto, visando sua especificação e padronização, já que a existência de versões distintas da HTML fere seu propósito original (W3C, 1999a). Basicamente os documentos HTML são arquivos textos que apresentam elementos delimitadores, os comandos de formatação da linguagem, que fornecem informações importantes para o navegador sobre sua apresentação. A Figura 1 apresenta um pequeno documento HTML (os elementos – *tags)*, juntamente com a sua exibição em um navegador.

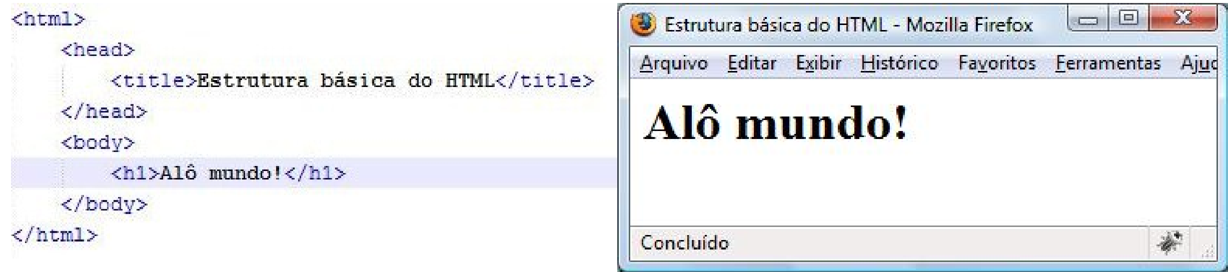

Figura 1: Documento texto HTML e sua apresentação em um navegador.

Os documentos HTML podem apresentar também uma seção chamada "formulários", contendo elementos especiais de marcação, chamados controles e rótulos aplicados a esses controles (W3C, 1999b). Foram os formulários que modificaram a forma original de funcionamento da web, baseada apenas na exibição de conteúdo estático, permitindo ao usuário entrar com informações através do navegador. Essas informações são repassadas para o servidor, processadas e retornadas na forma de uma página HTML criada dinamicamente. (ARAUJO, 1997)

Para o envio de informações através de um formulário HTML existem dois métodos distintos, disponibilizados pelo protocolo HTTP: o *GET* e o *POST* (W3C, 2004). O método

que será utilizado para o envio das informações é definido na criação do formulário HTML através do atributo *method*. O atributo *action* define o *script* do lado do servidor que irá receber e tratar os dados enviados. A Figura 2 mostra a definição de um formulário HTML, com alguns dos controles disponíveis, e sua exibição em um navegador.

GET é o método utilizado para solicitar algum recurso do servidor, como um arquivo ou um *script* CGI. As informações enviadas por um formulário através desse método são vinculadas à URL utilizada para acessar o servidor, o que faz com que as informações fiquem visíveis para o usuário. Supondo que o formulário da Figura 2 utilize o método *GET* para o envio de suas informações teríamos uma URL com o seguinte formato:

```
http://127.0.0.1/adiciona?nome=Nome&sobrenome=Sobrenome&email=email@ufjf.edu.br&sexo=Masculino
```

Dessa forma é feita uma requisição ao endereço acima, e para recuperar as informações enviadas, o *script* definido no atributo *action* utiliza uma variável de ambiente[3] específica para acessar os dados submetidos via método GET. Por exibir as informações na URL seu uso não é recomendado quando o processamento do formulário modifica o estado da aplicação, sendo mais utilizado para a recuperação de dados e solicitações simples ao servidor.

O método POST é recomendado quando o processamento do formulário envolve o armazenamento ou atualização dos dados enviados. Uma requisição que faz uso desse método anexa as informações submetidas ao corpo da mensagem HTTP. Por esse motivo, o método é considerado mais seguro em relação à transferência de dados, já que eles não ficam visíveis aos usuários como no método GET. E são acessados através da entrada de dados padrão do *script* que irá tratar a solicitação.

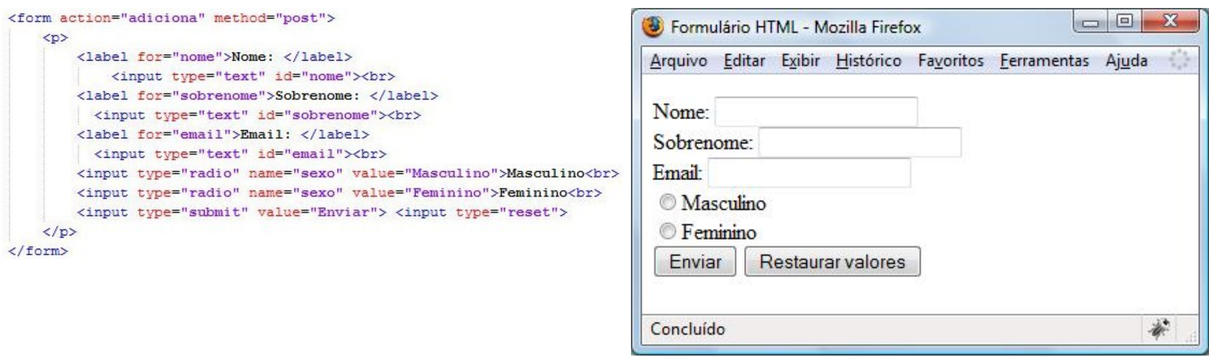

Figura 2: Um formulário HTML

3 Variáveis de ambiente são uma série de informações relevantes sobre o ambiente de execução passados do servidor para o *script* CGI. No caso da recuperação de informações enviadas através do método GET a variável de ambiente que contém os dados é geralmente chamada de QUERY_STRING (ROBINSON, 2004)

No modelo de submissão de formulários ou modelo clássico (*start-stop-start-stop*), o usuário interage com a aplicação através de uma interface basicamente composta por elementos de um formulário HTML (campos de texto, caixas de seleção, botões, etc). Cada ação do usuário resulta em uma requisição HTTP para o servidor onde estão hospedadas as páginas que compõem a aplicação. O servidor se encarrega de realizar o processamento necessário para cada requisição (o que pode envolver a recuperação e transformação de dados) e retornar uma nova página web para o usuário.

Nesse modelo, ilustrado na Figura 3, para cada vez que o usuário submete dados para a aplicação ele deve aguardar até que o servidor retorne uma nova página. Durante esse tempo o servidor lida apenas com uma requisição do usuário, executando cada uma de suas operações seqüencialmente. Caso alguma ação do usuário gere uma nova requisição durante a execução da anterior, as operações que estão sendo executadas são abortadas para que o servidor possa lidar com a nova requisição ou são enfileiradas para processamento posterior. Dessa forma a comunicação entre o cliente e o servidor utilizada por esse modelo é do tipo síncrona.

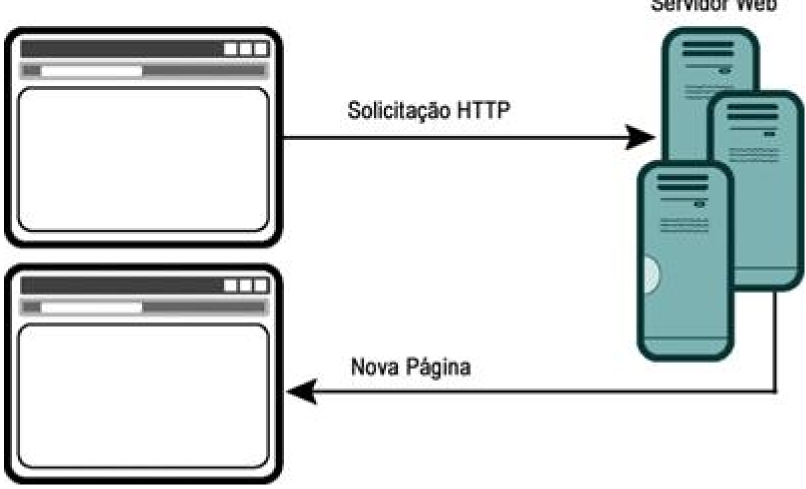

Figura 3: Modelo de submissão de formulários (ESPOSITO, 2008)

2.2 MODELO DE RENDERIZAÇÃO PARCIAL

O modelo de renderização parcial foi criado para evitar o carregamento de toda a página a cada vez que o usuário faz uma requisição. Uma aplicação do lado do cliente intercepta a ação padrão do navegador, que é a de submeter o formulário, e a substitui por uma requisição do tipo XMLHttpRequest, um objeto embutido no navegador que permite a

utilização de requisições assíncronas. Essa nova requisição é enviada ao servidor, que por sua vez retorna apenas o fragmento da página que necessita ser atualizado.

Esse modelo, ilustrado na Figura 4, é útil em atualizações dentro de uma mesma página, por exemplo, quando se faz a paginação de uma listagem de dados. Utilizando esse modelo apenas o fragmento com a listagem é atualizado. Porém a transição de uma página para outra na mesma aplicação ainda necessita de um carregamento total da página. Esse modelo pode ser visto como uma nova maneira de utilização do modelo clássico e por esse motivo está sujeito as mesmas limitações do modelo anterior, principalmente no que tange à comunicação síncrona entre cliente e servidor.

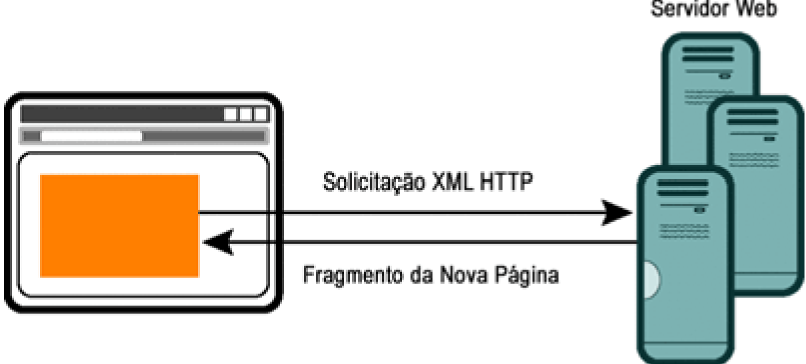

Figura 4: Modelo de renderização parcial (ESPOSITO, 2008)

2.3 RIA – *RICH INTERNET APPLICATIONS*

A partir do momento em que a internet, e principalmente a web, se firmou como uma verdadeira plataforma para o desenvolvimento de aplicações, surgiu o conceito de Aplicações de Internet Rica (RIA – na sigla em inglês). Estas aplicações web têm características e funcionalidades que as aproximam mais dos *softwares* tradicionais (ALLAIRE, 2002).

Em comparação aos outros modelos de desenvolvimento vistos anteriormente, onde o servidor faz a maior parte do processamento, o RIA faz uso intensivo do cliente web. O processamento da interface é transferido para o navegador, deixando o servidor responsável apenas pelos dados da aplicação. Para isso, todas as RIA introduzem uma camada intermediária de código, chamada de *client-engine,* entre o cliente e o servidor. O *client-engine* funciona como uma extensão do navegador, e é responsável pela renderização da interface da aplicação e pela comunicação com o servidor.

Através do *client-engine* é possível oferecer à interface da aplicação características

que não estão presentes no navegador padrão, com o intuito de enriquecer a experiência do usuário, disponibilizando componentes gráficos mais avançados, como controles do tipo arrastar e soltar, redimensionamento de objetos e utilização de recursos multimídia. Dessa forma, a interface se torna mais reativa às ações do usuário, pois, ao contrário de uma aplicação web padrão, não necessita de uma constante interação com o servidor remoto. O uso do *client-engine* torna mais equilibrada a carga de processamento entre o cliente e o servidor, já que o servidor não realiza mais todo o processamento da aplicação, ficando assim disponível para lidar com mais sessões de clientes simultaneamente.

Uma aplicação RIA é capaz de utilizar comunicação assíncrona com o servidor. Os modelos apresentados anteriormente utilizam uma forma de comunicação síncrona, ou seja, cada ação do usuário na interface, como o clique em um botão ou acesso a um *link*, faz uma requisição ao servidor, e o usuário aguarda até que a requisição seja processada. Nesses modelos o usuário só pode executar novas ações depois que ação atual for finalizada. Como uma aplicação RIA faz uso da comunicação assíncrona ela é capaz de fazer mais de uma requisição ao servidor, utilizando o *client-engine* para, baseado em alguma solicitação anterior, antecipar uma futura necessidade de dados e carregá-los no cliente, de modo a acelerar uma resposta posterior.

Da mesma forma que seus maiores benefícios estão associados ao fato de usar intensamente o lado do cliente, as maiores deficiências da utilização do RIA também estão ligadas a esse fato. Para manter a independência de plataforma, as aplicações do lado do cliente muitas vezes são escritas em uma linguagem de programação interpretada (*script*), geralmente Javascript, o que provoca uma sensível redução de desempenho, ou até mesmo o não funcionamento da aplicação, caso o usuário desabilite a execução de *scripts* em seu navegador. O tempo de carregamento da aplicação é outro fator que deve ser considerado, já que o *client-engine* necessita ser transferido do servidor para o cliente para funcionar, e, dependendo do tamanho ou do tipo de solicitação, o carregamento do script pode ser demorado. Um típico RIA requer que a aplicação fique permanentemente conectada à rede.

A tecnologia RIA introduz novas camadas de complexidade na arquitetura de uma aplicação web tradicional, o que requer um design mais rígido, testes mais complexos, dimensionamento da aplicação e suporte. Por esse motivo o processo de desenvolvimento de software torna-se mais longo. Este fato pode ser atenuado com a utilização de um *framework* para padronizar as características principais do design e desenvolvimento RIA.

As alterações introduzidas pela RIA na arquitetura quebram o padrão de uma aplicação web tradicional, que pode ser vista como uma série de páginas, cada uma com solicitações distintas de carregamento, iniciadas por uma requisição. Ao introduzir a

comunicação assíncrona para melhorar a resposta da interface às ações do usuário, aplicações RIA se afastam definitivamente desse modelo, pois o *client-engine* pode prever a necessidade de carregar alguns conteúdos para uso futuro, acabando com a percepção do carregamento de páginas. Usuários acostumados aos padrões de uma aplicação web tradicional podem ter dificuldades para se adaptar ao modelo RIA, pois muitas das funcionalidades disponíveis no modelo tradicional, como o botão "voltar" do navegador, podem ter efeitos diferentes ou indesejáveis em aplicações RIA.

2.4 *SINGLE PAGE INTERFACE* (SPI)

No modelo de SPI, toda interação do navegador com a aplicação web é feita com a utilização de apenas uma página. Nesse modelo a página é composta de elementos visuais que podem ser carregados, atualizados ou substituídos de forma independente. Dessa forma, não é necessário que toda a página seja recarregada a cada ação do usuário. A cada momento são mostrados apenas os elementos visuais e o conteúdo relevante para o estágio em que se encontra a aplicação. (ESPOSITO, 2008)

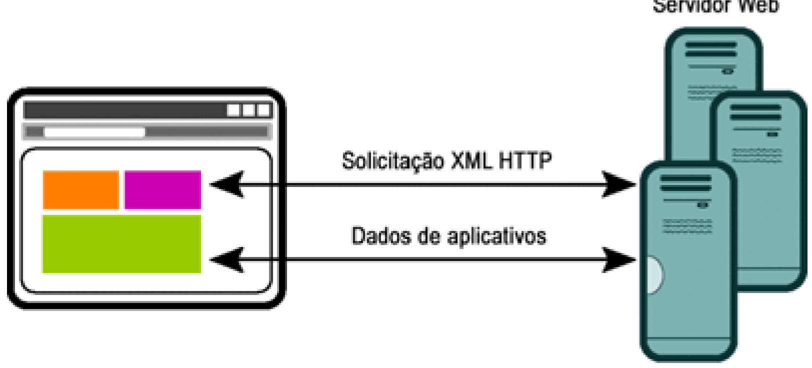

Figura 5: Elementos SPI em uma página (ESPOSITO, 2008)

Geralmente uma aplicação web desenvolvida sob o modelo SPI será uma aplicação RIA, pois um dos objetivos desse modelo é aumentar a interatividade do usuário com a aplicação. Assim, é intensificada a utilização de componentes de interface mais ricos e interativos, capazes de atualizações automáticas e independentes.

Entretanto, nem toda aplicação RIA segue o modelo de página única, podendo oferecer uma interface mais rica para o usuário utilizando outros modelos, como o de seqüência de páginas. Uma outra diferença importante é que algumas aplicações RIA são

fortemente baseadas em *plugins*, ou seja, dependendo da tecnologia utilizada para criar a aplicação é necessário que o usuário instale um programa específico que se acopla ao navegador para poder executá-la. Por outro lado, o SPI se baseia em tecnologias já suportadas pelos navegadores, o que não exige nenhuma instalação adicional.

O próximo capítulo analisa em detalhes esse modelo, com ênfase nas tecnologias utilizadas para a sua implementação.

3 *SINGLE PAGE INTERFACE*

Com a evolução das tecnologias web, muitas aplicações *desktop* estão sendo transferidas para esta plataforma. Entretanto, esse processo resulta em uma grande perda de interatividade do usuário com a aplicação. Aplicações web clássicas são baseadas em um modelo de interface de múltiplas páginas, no qual cada interação segue um paradigma de seqüência de páginas. Enquanto esse modelo se mostra simples e elegante para o intercâmbio de documentos, ele é deficiente para o desenvolvimento de modernas aplicações com interação amigável humano-computador. (MESBAH, 2009)

Em resposta ao limitado grau de interação das aplicações web, surgiu o Ajax. No centro dessa nova abordagem reside o modelo de S*ingle Page Interface,* que amplia a interatividade. Nesse modelo, as mudanças são feitas em componentes individuais da interface de usuário contidas na página web, ao invés de recarregar toda a página.

Este capítulo aborda os principais componentes do Ajax, bem como as características relevantes do modelo SPI.

3.1 AJAX

O termo Ajax (*Asynchronous* Javascript + XML) data de fevereiro de 2005 e sua criação é atribuída a Jesse James Garret, que o utilizou para denominar um conjunto de tecnologias já existentes, mas combinadas de uma nova maneira no desenvolvimento de aplicações web (GARRET, 2005). Assim, o termo Ajax não se refere a uma determinada tecnologia ou linguagem de programação, mas sim a várias tecnologias trabalhando em conjunto, o que proporciona o desenvolvimento de aplicações interativas e dinâmicas, que se aproximam mais do modelo *desktop.*

A utilização do Ajax redefine a interface da aplicação, focando na usabilidade e arquitetura da informação. Isso faz com que apenas as áreas necessárias de uma página sejam atualizadas e elimina a necessidade de carregar completamente a página a cada requisição, diminuindo o tráfego de dados entre o cliente e o servidor (ZAKAS et al., 2006).

Dentre as tecnologias incorporadas pelo Ajax estão XHTML (eXtensible Hypertext Markup Language) e CSS (*Cascading Style Sheets*) para apresentação baseada em padrões web, DOM (*Document Object Model*) para exibição e interação dinâmica, XML (*eXtensible Markup Language*) e XSLT (*eXtensible Stylesheet Language Transformations*) para intercâmbio e manipulação de dados, XMLHttpRequest[4] para recuperação assíncrona

4 Extensão do navegador que permite conexões assíncronas entre a página e o servidor web.

de dados e a linguagem de programação Javascript, com a função de unir as tecnologias apresentadas. (GARRET, 2005)

Uma aplicação web baseada em Ajax adiciona uma camada intermediária entre o cliente e o servidor, conhecida como Ajax-*engine* (Figura 6). Ao invés de carregar uma página no início de uma sessão o navegador carrega a Ajax-*engine*, que passa então a ser responsável tanto pela interface da aplicação quanto pela comunicação com o servidor a partir do comportamento do usuário (GARRET, 2005).

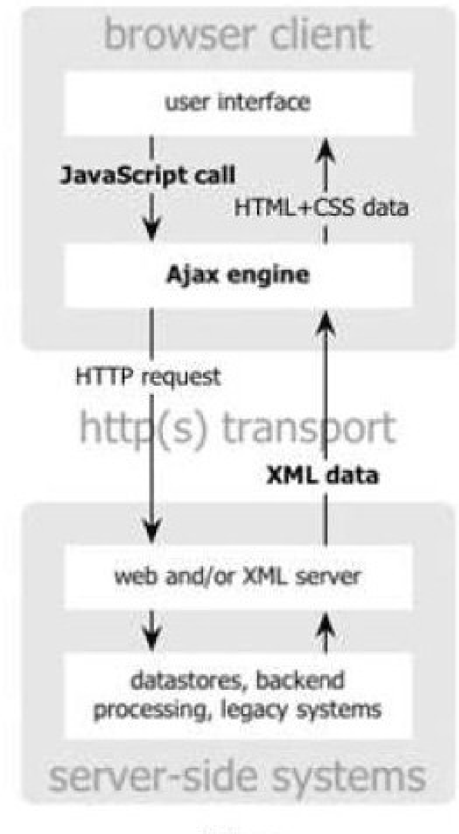

Figura 6: Modelo de aplicação Ajax (GARRET, 2005)

Assim as ações do usuário que normalmente geravam requisições HTTP pelo navegador agora são interceptadas pela Ajax-*engine*, que manipula por si só qualquer requisição realizada que não dependa de processamento do servidor, como uma simples validação, edição de dados em memória, e até mesmo alguma navegação. Nos casos onde é necessária a comunicação com o lado do servidor, a *engine* faz requisições assíncronas

(Figura 7) para prover as respostas necessárias à ação realizada, sem que a interação do usuário com a aplicação seja interrompida.

Ajax web application model (asynchronous)

client

browser UI

user activity

Ajax engine

client-side processing

time

server

input display data transmission server-side processing

Figura 7: Padrão de interação assíncrona de uma aplicação Ajax (GARRET, 2005)

Para uma melhor definição do Ajax é necessário levar em consideração alguns princípios, que explicitam principalmente as diferenças de abordagem em relação ao modelo clássico (CRANE et al, 2006):

1. O navegador hospeda uma aplicação e não conteúdo: em uma aplicação web tradicional toda a informação a respeito do usuário é guardada no servidor. A cada interação do usuário com o *site* um novo documento é enviado e o que estava sendo exibido é descartado e substituído. Uma aplicação Ajax utiliza o navegador para guardar parte da lógica da aplicação, fazendo com que ele receba no início um documento complexo contendo a Ajax-*engine*. Como esse documento é mantido do lado do cliente ele mesmo pode guardar informações importantes sobre o estado da aplicação, evitando o uso de sessões no servidor. A Figura 8 apresenta uma comparação entre o ciclo de vida de uma aplicação web tradicional e uma aplicação Ajax.

2. O servidor fornece dados e não conteúdo: em uma aplicação web tradicional, a cada interação do usuário o servidor envia dados e conteúdo de volta para a aplicação, pois a cada passo é necessário refazer a página com informações sobre disposição dos elementos, menus, barras de navegação, etc (conteúdo) com os dados que foram processados e que serão exibidos para o usuário. Como numa aplicação Ajax o navegador passa a conter parte da lógica da aplicação, inclusive a

parte responsável pelo conteúdo, a comunicação com o servidor torna-se mais eficiente, pois este fica responsável apenas pelo envio de dados (fragmentos de código Javascript, texto, documentos XML). Esse princípio faz com que em uma aplicação Ajax, o tráfego de dados entre o cliente e o servidor seja mais alto no início, momento em que é entregue ao navegador a Ajax-*engine*. Porém à medida que o tempo de interação com a aplicação aumenta, o consumo de largura de banda é relativamente menor quando comparado com uma aplicação web tradicional.

3. A interação do usuário com a aplicação pode ser fluida e contínua: Em uma aplicação web tradicional, cada interação do usuário cria uma interrupção na seqüência de ações, pois é sempre necessário aguardar uma resposta do servidor para prosseguir. Em tarefas transitórias, esse problema não produz muito efeito, porém quando se considera um domínio de negócio diferente, o custo de interrupção da seqüência de trabalho com uma atualização de página se torna proibitivo. Como maior parte da aplicação corre no lado do cliente isto permite que uma aplicação Ajax se comporte como uma aplicação desktop e não seja tão limitada pela rede em termos de respostas do servidor, dando a sensação de continuidade e fluidez nas ações efetuadas. As aplicações Ajax são mais reativas e permitem uma maior riqueza em termos gráficos.

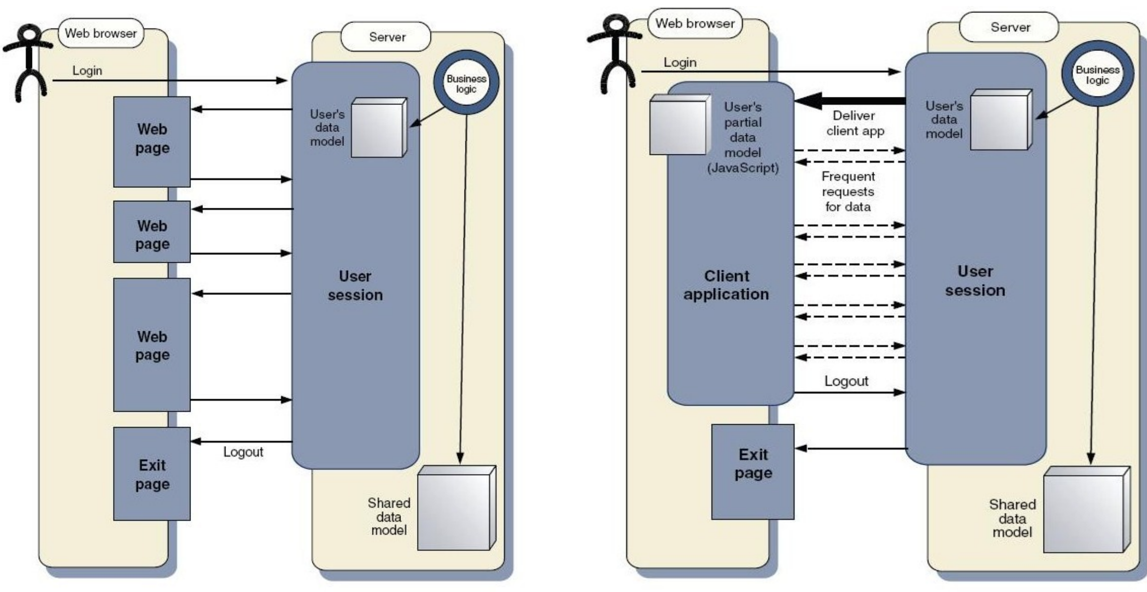

Figura 8: Aplicação web tradicional X Aplicação Ajax (CRANE et al., 2006)

Como Ajax é baseado em padrões web, suportados pela maioria dos navegadores atuais, ele não requer nenhum *plugin*[5] adicional para funcionar. Essa é sua principal diferença e vantagem quando comparado a uma tecnologia RIA. Um outro aspecto que não deve deixar de ser considerado é o enriquecimento da interface do usuário que o Ajax proporciona com o uso de componentes interativos. Alguns exemplos incluem sugestões em campos de texto de acordo com a digitação do usuário, controles do tipo arrastar e soltar e tabelas com ordenação de dados integrada. (MESBAH, 2009)

Como visto anteriormente, é a combinação de tecnologias já bem-estabelecidas e complementares que faz do Ajax único (Figura 9). As seções a seguir contemplam uma visão geral dessas tecnologias.

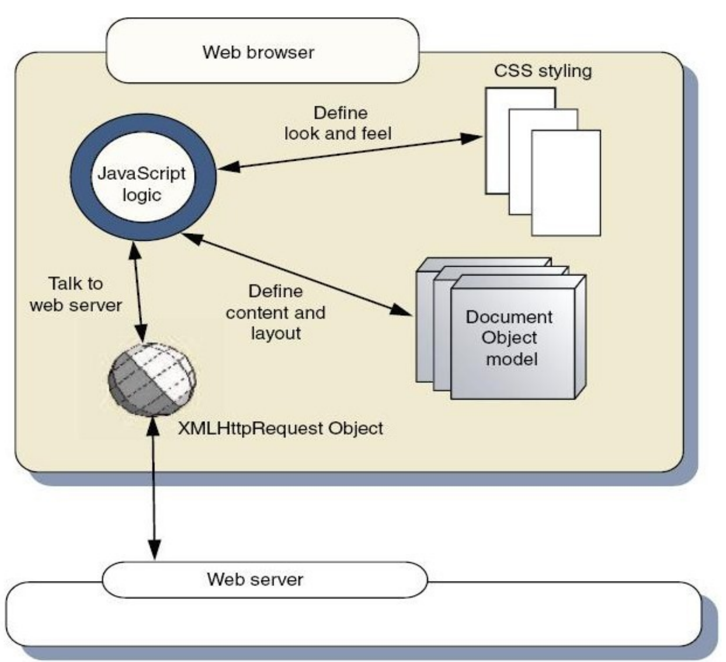

Figura 9: Funcionamento das tecnologias que compões o Ajax (CRANE et al, 2006)

3.1.1 Javascript

Javascript é uma linguagem de programação de uso geral com uma similaridade com a família de linguagens C. É fracamente tipada, ou seja, suas variáveis não são declaradas com um tipo específico, permitindo que para a mesma variável sejam atribuídos valores de diferentes tipos. É interpretada, o que significa que o código fonte da linguagem não é compilado para um código executável, mas sim executado diretamente pelo navegador. Quando um código Javascript é distribuído o código fonte é transmitido direto do servidor web para o cliente.

5 Programa que se acopla ao navegador, estendendo ou criando novas funcionalidades para o mesmo.

Por ser de uso geral a linguagem pode ser usada para a maioria dos algoritmos e tarefas de programação. Conta com suporte nativo a tipos numéricos, cadeia de caracteres, data e hora , expressões regulares, funções matemáticas e geração de números aleatórios. Também é possível definir objetos estruturados, o que auxilia no desenvolvimento de códigos mais complexos.

A Figura 10 apresenta um documento HTML com uma função Javascript, que é definida entre as *tags* <script> e executada quando a página é carregada no navegador.

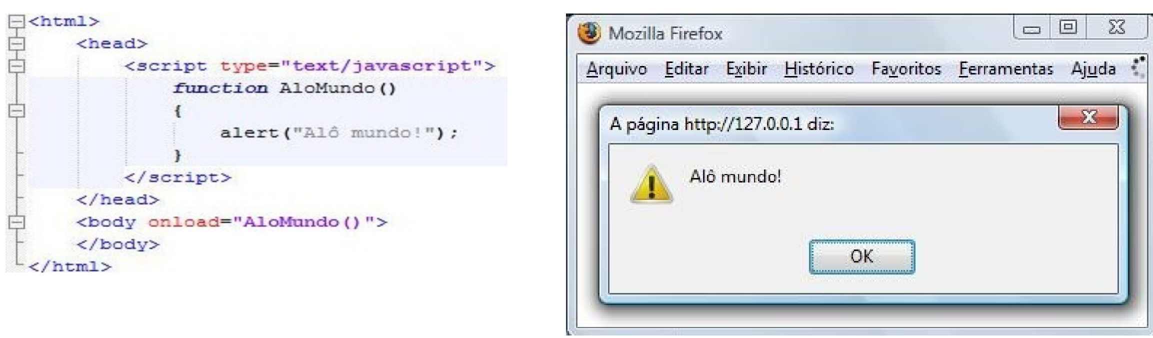

Figura 10: Uma função Javascript

3.1.2 *Cascading Style Sheets* (CSS)

Cascading Style Sheets é um padrão para a apresentação de documentos HTML ou XML. Enquanto a linguagem de marcação HTML define a estrutura de um documento, o CSS define como esses elementos serão exibidos. Sua primeira versão foi definida em 1996 pela W3C[6] e continha a especificação de propriedades de estilo relativas a cores, imagens e fontes. A segunda versão, disponibilizada em 1998, especifica propriedades avançadas em relação ao posicionamento dos elementos na página. Atualmente está em desenvolvimento a terceira versão, que estende a segunda e adiciona algumas funcionalidades, como novas bordas e planos de fundo, texto vertical, interação com usuário e suporte a fala. O uso do CSS proporciona uma separação entre a formatação e o conteúdo de um documento (W3C, 1999) .

Alguns dos benefícios práticos do bom uso de CSS envolvem a redução do tempo de design e desenvolvimento, aceleração do tempo de carga da página, remoção de elementos de apresentação da linguagem HTML e aumento da interoperabilidade aderindo aos padrões web. (REIS, 2007)

Com o Ajax, onde as aplicações web deixam de seguir o modelo de múltiplas

6 *World Wide Web Consurtium:*consórcio de empresas de tecnologia com o intuito de manter um conjunto de padrões para a criação e interpretação de conteúdos web, através de protocolos comuns e fóruns abertos.

páginas, o CSS auxilia provendo um repositório de visuais pré-definidos, que através de programação podem ser aplicados dinamicamente a um determinado elemento. A Figura 11 mostra a utilização de CSS para definir o estilo de uma interface de usuário, as duas telas são geradas a partir do mesmo HTML, e tiveram apenas a definição de estilos alterada. O CSS utilizado na tela a esquerda mantém apenas a informação de posicionamento dos elementos, enquanto a da direita possui informações de cores e imagens. (CRANE et al., 2006). A Figura 12 apresenta a definição de um estilo CSS para um elemento HTML e sua exibição no navegador.

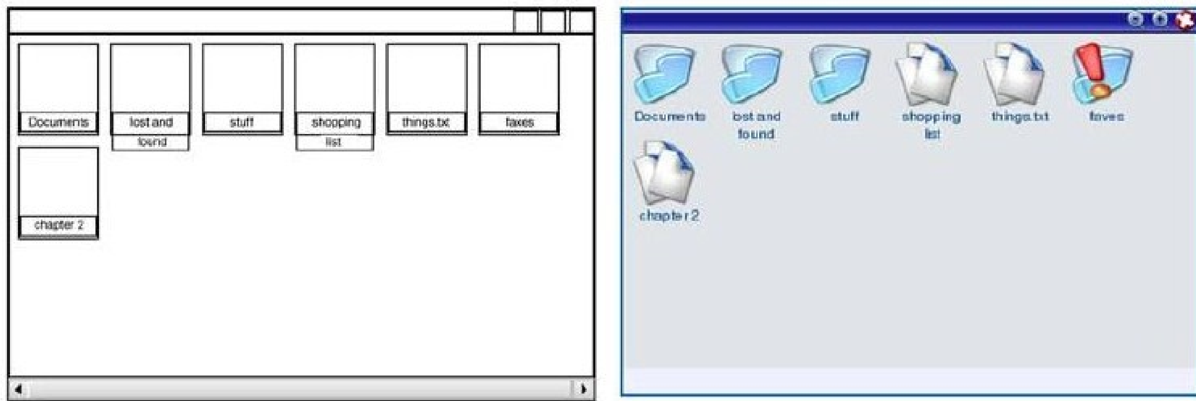

Figura 11: Estilo de uma interface de usuário definido via CSS(CRANE et al, 2006)

```
<html>
    <head>
        <style type="text/css">
            div.item{
            position: relative;
            border: solid #0066aa 2px;
            width: 150px;
            text-align: center;
            font: 18px verdana, arial, helvetica;
        }
        </style>
    </head>
    <body>
        <div class="item">Estilo CSS</div>
    </body>
</html>
```

Mozilla Firefox
Arquivo Editar Exibir Histórico Favoritos
Estilo CSS

Figura 12: Definição de estilo CSS.

3.1.3 *Document Object Model*

O *Documento Object Model* (DOM) é uma padronização para representar documentos HTML ou XML em um modelo de objetos. Baseado no relacionamento entre os elementos que compõem o documento, é criado um modelo em forma de árvore (Figura 13)

que permite, através de programação, o acesso, navegação e manipulação do documento. (W3C, 1998)

No navegador web uma instância de um objeto DOM pode ser vista como a representação em tempo de execução do código HTML. A primeira instância do objeto é criada quando a página é carregada e interpretada pelo navegador. A partir de então é possível utilizar a linguagem Javascript para manipular o documento, o que consiste em inserir, excluir e/ou mover os nós que compõem a estrutura do DOM, bem como alterar suas propriedades. Os resultados dessa manipulação são incorporados de volta à página (MESBAH, 2009). Como permite a mudança em elementos individuais da página, o DOM é amplamente utilizado no Ajax para fazer a maior parte das mudanças na interface de usuário (CRANE et al, 2006).

A Figura 14 apresenta um exemplo da utilização de Javascript para manipulação do DOM. No caso a propriedade *innerHTML* da *tag* <h1>, identificada por *header*, está sendo alterada para um novo valor.

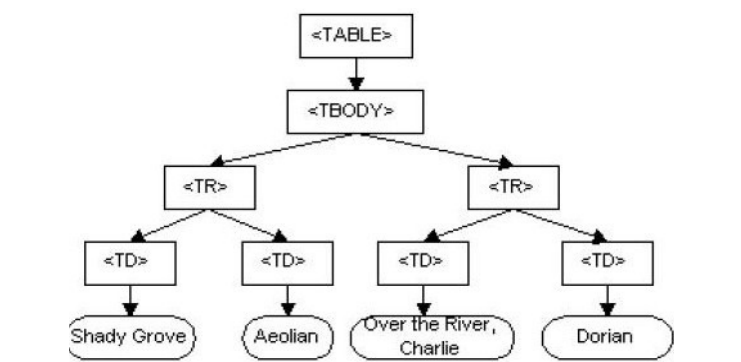

Figura 13: Estrutura de árvore DOM para uma tabela HTML (W3C, 1998).

```
<html>
<body>

<h1 id="header">Cabeçalho Velho</h1>

<script type="text/javascript">
document.getElementById("header").innerHTML="Cabeçalho Novo";
</script>

<p>"Cabeçalho Velho" alterado para "Cabeçalho novo"</p>

</body>
</html>
```

Figura 14: Exemplo de Javascript para manipulação do DOM.

3.1.4 XMLHTTPRequest

XMLHTTPRequest é uma extensão do navegador que permite a conexão assíncrona entre a página e o servidor Web. O objeto do tipo XMLHTTPRequest foi criado explicitamente para a recuperação de dados em segundo plano, e por isso é amplamente usado em Ajax, pois com ele é possível fazer requisições ao servidor, recuperar os dados necessários e alterar os controles da interface de usuário sem que a página seja totalmente recarregada.

Na maior parte das vezes os dados transferidos do servidor para a aplicação estão no formato XML, já que o objeto oferece um bom suporte nativo ao processamento de dados nesse formato. Porém, o XMLHTTPRequest pode receber qualquer informação baseada em texto. Um formato particularmente útil para transmitir dados ao cliente Ajax é o *Javascript Object Notation* (JSON), uma forma compacta de representar um objeto Javascript genérico. A Figura 15 mostra um exemplo da representação da notação JSON.

```
{
    "animal":
    {
        "nome": "Cachorro",
        "classe": "Mamífero"
    }
}
```

Figura 15: Exemplo da notação JSON.

O objeto XMLHTTPRequest tem origem em uma implementação da *Microsoft®* como um componente integrante do navegador Internet *Explorer* 5. Logo os outros navegadores apresentaram suas próprias implementações do objeto, com funcionalidades semelhantes. (MESBA, 2009) Dessa peculiaridade vem o principal problema relacionado a essa tecnologia: a falta de um padrão para o objeto fez com que ele fosse desenvolvido de forma diferente pelos diversos navegadores disponíveis, ficando a cargo do desenvolvedor da aplicação tratar esse problema. A Figura 16 mostra uma possível solução para essa dificuldade apresentando um trecho de código para a criação de um objeto XMLHTTPRequest, que leva em consideração a utilização de navegadores distintos para acesso à aplicação. (SILVA, 2007)

```
01 this.createAJAX = function() {
02     try {
03       this.xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
04     } catch (e1) {
05
06       try {
07         this.xmlhttp = new 08ActiveXObject("Microsoft.XMLHTTP");
09           } catch (e2) {
10           this.xmlhttp = null;
11       }
12       }
13      if (! this.xmlhttp) {
14       if (typeof XMLHttpRequest != "undefined") {
15           this.xmlhttp = new XMLHttpRequest();
16       } else {
17           this.failed = true;
18       }
19      }
20 };
```

Figura 16: Criação de um objeto XMLHTTPRequest (SILVA,2007)

3.2 SPI

No modelo *Single-Page Interface* (SPI), todas as funcionalidades de uma aplicação web, ou pelo menos a maioria delas, ficam contidas em uma única página. Dessa forma, toda interação do usuário com a aplicação se dá em apenas uma página. Essa abordagem, comparada ao modelo clássico de aplicações para a web, é revolucionária. Porém não é nova em relação ao desenvolvimento de aplicações para desktop. (ESPOSITO, 2008) Para aproveitar ao máximo o Ajax é fortemente recomendado seguir o modelo SPI.

Um dos principais objetivos do modelo é enriquecer a experiência do usuário com a aplicação, apresentando uma interface mais rica e interativa. Porém esse modelo traz a tona uma série de questões relacionadas à busca e a manutenção do histórico de navegação. Quanto à busca, as páginas sempre foram rastreadas pelos mecanismos de pesquisa usando um link permanente, fazendo com que cada estado de uma aplicação web fosse associado a uma página com uma URL distinta. No modelo SPI, isso deixa de ser válido, pois, como toda a interação acontece em apenas uma página, não há mais transições de URL marcando estados distintos da aplicação. Esse aspecto também afeta diretamente a questão da manutenção do histórico de navegação, como não há mais a transição entre endereços, o controle oferecido pelos navegadores, compostos pelos botões Voltar e Avançar, deixa de funcionar corretamente.

O modelo SPI oferece as ferramentas necessárias para a criação de aplicações web mais próximas das aplicações desktop, em termos de interatividade e capacidade de resposta, devido às características do Ajax de realizar operações simultâneas (requisições

assíncronas) e manter uma interface de usuário única de atualização automática. Entretanto, quando comparado ao tradicional, o modelo de Single Page Interface apresenta diversas mudanças: não há mais histórico de navegação, favoritos, operações únicas, transição de páginas e links permanentes.

Devido a essa quebra de paradigma introduzido pelo modelo não é possível ignorar os usuários e desenvolvedores acostumados à antiga web e seu paradigma de múltiplas páginas. Assim, quando a navegação (envolvendo o uso dos botões Avançar e Voltar do navegador) é importante, como por exemplo em um web site que exibe conteúdo estático sem alterar a URL da página, alguns recursos devem ser utilizados para simular essa situação no modelo SPI. Uma das propostas de solução para esse problema envolve a alteração manual da propriedade DOM que guarda o endereço da página e a associação de um objeto de estado a cada passo da aplicação. A alteração da propriedade DOM adiciona um item que referencia o estado da aplicação ao histórico do navegador sem que a página seja recarregada. Com o uso de Javascript é possível monitorar o clique em um dos botões de navegação e assim executar funções correspondentes que restauram os estados vinculados a cada item do histórico de navegação (RIECKE et al., 2008). (Ver Apêndice 1 - Exemplo de uma *single-page* usando o *framework* Dojo)

3.3 PADRÕES DE PROJETO AJAX

Para a criação de código no modelo SPI se faz necessário um novo conjunto de padrões de projeto. Padrões de projeto descrevem soluções para problemas recorrentes no desenvolvimento de sistemas de software, definindo o problema, a solução, quando aplicá-las e suas conseqüências. Um padrão de projeto dá um nome para uma solução técnica, tornando-a mais fácil de ser entendida e discutida. A tabela a seguir, lista alguns padrões Ajax, concentradas em técnicas e organizações da interface do usuário (ESPOSITO, 2008)

Tabela 1: Padrões de projeto Ajax

Padrão	Meta
Templates no lado do navegador	O padrão sugere o uso de *templates* HTML que serão elaborados dinamicamente usando dados recuperados dos pontos de extremidade HTTP remotos. Em vez de gerar novamente o *layout* HTML dos dados conforme as solicitações, o padrão sugere que seja configura uma camada de *templates*. Esse padrão representa uma alternativa ao padrão de mensagem HTML.
Proxy entre domínios	O padrão opera uma conexão servidor/servidor com um serviço acessível e exposto publicamente, além de transportar os dados novamente para o cliente. No navegador do cliente, um aplicativo Ajax não tem permissão para se conectar a nenhuma URL fora do domínio da página. No entanto, um *proxy* local no mesmo domínio pode facilmente captar dados de um lugar qualquer e devolvê-los à quem fez a chamada.

Pulsação	Como a maioria dos aplicativos Ajax faz grande parte do trabalho no cliente sem jamais fazer uma submissão direta ao servidor, pode ser necessário informá-lo que um determinado cliente continua ativo. O padrão sugere que o aplicativo do cliente envie periodicamente uma mensagem de "pulsação" para indicar que o aplicativo ainda está carregado e funcionando no navegador.
Mensagem HTML	Os pontos de extremidade HTTP remotos normalmente retornam dados JSON (*JavaScript Object Notation*) no cliente a ser integrado ao DOM existente. Essa tarefa só pode ser realizada por meio de JavaScript. No entanto, caso o código do cliente seja especialmente complexo, ou por motivos de desempenho, é possível retornar HTML (dados e *layout*) do servidor, e não apenas dados sem formatação.
Microlink	O Ajax é essencialmente usado para fazer muitas atividades dentro da mesma página. Para referenciar conteúdo externo é necessário um tipo de hiperlink na página ou microlink. O microlink é uma referência a uma parte de marcação recuperada por meio de chamadas do servidor e, em seguida, inserida na página. Um microlink pode ser um ponto de extremidade HTTP ou um método em um objeto de comando do JavaScript.
JavaScript sob demanda	Ao invés de transferir todo o JavaScript necessário durante a inicialização da página, os arquivos são transferidos conforme a necessidade de utilização. Isso dá às páginas um carregamento mais rápido e não afeta a funcionalidade.
Organização da página	Como grande parte da atividade da aplicação acontece na mesma página, é preciso atualizar o conteúdo e apresentar novas informações à medida que o contexto muda. Esse padrão sugere que se utilize o DOM para adicionar/remover ou mostrar/ocultar elementos para refletir transições de estado.
Atualização periódica	O navegador agenda periodicamente uma solicitação ao servidor para ganhar informações novas e assim atualizar a interface do usuário.
Pop-up	Um pop-up consiste em um conteúdo HTML exibido à frente do conteúdo existente durante um período relativamente curto até que o usuário o descarte. Usado para representar uma caixa de diálogo, quando a aplicação requer uma interação exclusiva do usuário, como uma confirmação de exclusão, por exemplo.
Busca preditiva	O padrão sugere prever as ações mais prováveis do usuário e buscar os dados necessários para essas ações antecipadamente. A implementação desse padrão tem um custo: trata-se, basicamente, de uma estimativa, e, sendo assim, pode estar errada. Embora seja efetivo para melhorar a performance percebida pelo usuário, esse padrão também pode resultar em uma perda de desempenho caso seja implementado insatisfatoriamente ou implementado em um cenário inferior ao ideal por conta de um alto consumo de largura de banda do servidor.
Indicador de progresso	Padrão usado para monitorar o progresso das operações do servidor. Para sua implementação é necessário que a operação sendo executada pelo servidor grave seu próprio progresso em uma área compartilhada, de forma que um serviço de monitoramento executado pelo cliente seja capaz de ler essas informações e passá-las ao usuário no contexto de uma atualização de progresso.
Limitação de envio	Uma das potenciais desvantagens do Ajax é que muitas solicitações podem ser geradas para o servidor em uma única unidade de tempo. O que pode causar um problema de escalabilidade na aplicação. Esse padrão sugere que seja utilizado um temporizador para carregar periodicamente os dados no servidor e um cache local ou uma fila para acumular solicitações que serão enviadas ao servidor.
Tempo limite	No caso de operações pesadas realizadas no cliente como, por exemplo, *streaming* ou atualização periódica, existe o problema de se garantir que cada cliente conectado esteja realmente usando a aplicação. Esse padrão sugere que seja declarado um tempo limite para a operação pesada, e, caso esse tempo seja atingido, a operação pare de executar, retornando apenas se o usuário solicitar que ela continue.
URLs exclusivas	o padrão permite que você atribua URLs distintas a partes diferentes do aplicativo que normalmente refletem estados diferentes. Esse padrão é muito usado para dar suporte ao histórico em aplicativos Ajax.
Espaço de trabalho virtual	O servidor precisa responder a solicitações o mais rápido possível, embora não possa necessariamente devolver todos os dados disponíveis por motivos de largura de banda. Esse padrão sugere que seja criado um espaço de trabalho virtual, que transmita a sensação de que todos os dados estão disponíveis para aplicação, ainda que apenas uma pequena fração deles esteja efetivamente no cliente. É de responsabilidade da aplicação baixar os dados sob demanda e armazená-los localmente em *cache*.

3.4 FRAMEWORKS AJAX

Um *framework* pode ser definido como um conjunto de classes cooperantes usadas para construir um projeto reutilizável para uma determinada classe de software. A utilização de um *framework* para construir uma aplicação dita a arquitetura da mesma e enfatiza a reutilização de projetos. (GAMMA et al., 2009)

Para adicionar algum grau de dinamismo à interface de usuário, os desenvolvedores de aplicações web empregam muito esforço para superar os limites do modelo de sequência de páginas do HTML e as complexidades de programação Javascript do lado do cliente. Para fazer um uso efetivo do Ajax os desenvolvedores precisam ter habilidades avançadas em diversas tecnologias para a web. Em conseqüência, muito esforço também é gasto para testar as aplicações antes que elas entrem em utilização. (MESBAH, 2009)

O uso de um *framework* para o desenvolvimento de aplicações já possui a característica de aumentar a produtividade e a reutilização de código em um projeto, além de proporcionar maior facilidade para a execução de testes. No desenvolvimento de aplicações web sob o paradigma *Single Page Interface*, um *framework* Ajax auxilia pois permite lidar com a complexidade relacionada à integração das tecnologias que compõem esse modelo de forma padronizada, já que se encarrega de boa parte da implementação e gerenciamento necessários para as interações entre o cliente e o servidor. Além de introduzir uma ordem no desenvolvimento e manutenção de aplicações em larga escala.

Devido à importância dada ao Ajax no momento, um vasto número de *frameworks* para essa tecnologia estão sendo desenvolvidos. As seções a seguir descrevem de forma sucinta cinco dessas ferramentas que permitem o desenvolvimento de aplicações web SPI, muitas vezes se encarregando de toda a geração de código Javascript e da interação assíncrona entre o cliente e o servidor, além da integração com bibliotecas de código que permitem a criação de uma interface de usuário rica e a funcionamento independente do navegador utilizado pelo usuário.

3.4.1 *Google Web Toolkit* (GWT)

O Google, empresa líder no serviço de busca na web, antes mesmo da criação do termo já utilizava o conceito de Ajax em várias de suas aplicações, como o GMail[7] e o

7 Serviço de correio eletrônico via web, lançado em 2004, com uma interface de usuário que permite a abertura de diversas mensagens de uma única vez, além da atualização automática da caixa de entrada, mesmo quando o usuário está digitando uma mensagem. Disponível em http://www.gmail.com

Google Maps[8]. Por conta de sua influência, foi uma das empresas que mais contribuiu para dar uma grande visibilidade às aplicações Ajax.(CRANE, et al, 2006). Continuando com sua política, o Google tornou disponível ao público o *framework* utilizado por eles para o desenvolvimento dessas aplicações, que recebeu o nome de GWT.

O GWT permite que a aplicação seja completamente implementada utilizando a linguagem Java[9], tanto do lado do cliente, quanto do servidor. Já que possui um compilador que transforma o código escrito em Java em código Javascript para ser interpretado pelo navegador. Dessa forma mantém toda a funcionalidade da interface do usuário no lado do cliente, diminuindo o número de consultas entre o cliente e o servidor, que só é consultado quando novos dados são necessário para a atualização de um componente.

Uma rica biblioteca de componentes de interface de usuário acompanha o GWT, além de ferramentas para comunicação com o servidor, internacionalização e gerenciamento do histórico do navegador. Devido às características da linguagem Java, o GWT oferece aos programadores a possibilidade do uso de ferramentas sofisticadas para o desenvolvimento, como checagem de erro, checagem de tipo, depuração e testes do código.(CRUZ, 2007)

3.4.2 Echo2

O Echo2 é um *framework* Ajax que permite aos desenvolvedores a criação de aplicações web orientadas a objeto e fortemente baseada em componentes de interface com o usuário. Provê APIs[10] para o gerenciamento do estado de uma aplicação e de sua interface. Aplicações Echo podem ser inteiramente criadas do lado do servidor usando a linguagem de programação Java. (MESBAH, 2009)

Três módulos distintos fazem parte do Echo2: framework de aplicação, mecanismo de renderização e *container* da aplicação. O framework de aplicação é responsável por representar e gerenciar o estado da aplicação e de sua interface. O mecanismo de renderização é responsável pela exibição dos componentes de interface do usuário e pela

8 O Google Maps mistura um visualizador de mapas com um serviço de buscas. O mapa em si possui recursos em Ajax, o que possibilita uma navegação sem o recarregamento da página, além da exibição de informações relativas à área visualizada com apenas um clique. Disponível em http://www.google.com/maps

9 Java é uma linguagem de programação orientada a objeto desenvolvida na década de 90 por uma equipe de programadores chefiada por James Gosling, na empresa Sun Microsystems. Ao contrário das linguagens convencionais, que são compiladas para código nativo, os programas Java podem ser executados em qualquer sistema operacional, desde que este possua a *Java Virtual Machine* (JVM) instalada. A JVM é o programa que converte o código Java em comandos que o sistema operacional possa executar.

10 *Application Programming Interface* ou Interface de Programação de Aplicativos é um conjunto de rotinas e padrões estabelecidos por um software para a utilização de suas funcionalidades. Permite usar os serviços de um software sem conhecer detalhes de sua implementação.

comunicação cliente/servidor. O *container* da aplicação atua como uma extensão do mecanismo de renderização responsável por manter a sincronização do estado entre o cliente e o servidor quando ocorre alguma ação do usuário na interface.

3.4.3 *Backbase*

O *Backbase* é um framework comercial fornecido pela empresa de mesmo nome localizada em Amsterdam. O elemento chave do framework é chamado de *BackBase Cliente Run-time* (BCR), que é o Ajax *engine*, escrito em Javascript e executado no navegador. Suas principais funcionalidades são: criar a interface de página única (SPI) e gerenciar seus controles, interpretar código Javascript e manter a sincronização do estado da aplicação entre o cliente e o servidor. (MESBAH, 2009)

O *Backbase* disponibiliza uma linguagem de marcação chamada de *Backbase Tag Library* (BTL) que oferece uma biblioteca de controles de interface com o usuário, um mecanismo para anexar ações a esses controles e facilidades para conexões assíncronas como o servidor. Do lado do servidor, o *Backbase* permite a implementação em linguagem Java ou .NET.[11]

Por ser uma solução proprietária, este *framework* só podes ser usado gratuitamente para fins não comerciais.

3.4.4 Qooxdoo

Qooxdoo é um *framework* Ajax que inclui suporte para o desenvolvimento Javascript profissional e comunicação cliente/servidor de alto nível. Mesmo sendo um framework totalmente baseado em Javascript, oferece um rico conjunto de controles de interface com o usuário bem próximos a elementos nativos de uma aplicação desktop. Inteiramente baseado em classes, tenta elevar as características de orientação a objetos presente no Javascript. Para a utilização do Qooxdoo, não é necessário ao desenvolvedor conhecimento de CSS e HTML, porém deve-se ter boas noções de programação Javascript.

Mesmo utilizando a filosofia de linguagens como o Delphi[12], ainda não possui um editor visual maduro como os existentes para Java. Um outro ponto negativo é que sua plataforma, código necessário para a execução no cliente, é relativamente grande, o que faz

11 Iniciativa da empresa *Microsoft®* que visa uma plataforma única para o desenvolvimento e execução de sistemas e aplicações. O código gerado para .NET pode ser executado em qualquer dispositivo que possua o framework da plataforma.

12 Linguagem de programação fortemente baseada em elementos visuais, permite a definição da interface e até mesmo parte da estrutura da aplicação com o auxílio de ferramentas visuais.

com que o tráfego entre o cliente e o servidor seja muito alto no início da aplicação (CRUZ, 2007).

3.4.5 Dojo

O Dojo é um conjunto de ferramentas para auxiliar na construção de aplicações baseadas no navegador. Ele é construído usando Javascript do lado do cliente para aumentar a capacidade dos navegadores modernos. Sua proposta é a de tornar a interface de usuário de uma aplicação baseada no navegador (e, conseqüentemente, uma aplicação web) indistinta daquelas incluídas em aplicações locais. O próprio navegador, hoje presente nos mais diversos dispositivos, passa a ser a plataforma de interface com o usuário. Para isso, o Dojo corrige alguns problemas relativos a essa abordagem, como a incompatibilidade existente entre os diversos navegadores disponíveis no mercado. Isso significa que um código escrito em Dojo irá funcionar em qualquer navegador suportado pelo *framework*. (RIECKE et al., 2008)

Como se apresenta como um conjunto de ferramentas o Dojo é composto de uma coleção de *scripts* estáticos escritos em Javascript. Para sua utilização não é necessário plugin para o navegador ou componentes do lado do servidor. O código fonte do Dojo é livre e está disponível em sua distribuição, que inclui as seguintes características (DOJO, 2008):

- Funções e bibliotecas para abstrair a manipulação do DOM (Documento Object Model);
- Funções e bibliotecas para corrigir erros graves dos navegadores, como vazamento de memória;
- Um extenso conjunto de controles de interface de usuário;
- Bibliotecas independentes para a implementação de características avançadas para o navegador. Essas bibliotecas podem ser carregadas sob demanda, de acordo com a necessidade de utilização.

Dojo é dividido em três projetos. O primeiro, também chamado Dojo, é a fundação sobre a qual todo o resto é construído, inclui os recursos que lidam com a normalização dos navegadores, extensões para o núcleo da biblioteca Javascript e para a manipulação do DOM, funções de localidade e internacionalização e diversas outras funções. O segundo, chamado Dijit, é o sistema de controles de interface com o usuário construídos com o Dojo, esses controles recebem o nome de *widget* ou componente Dijit e são construídos a partir de HTML e Javascript. E por fim o Dojox, bibliotecas criadas com o intuito de ampliar ou oferecer novas funcionalidades ao Dojo (DOJO, 2008).

Em relação ao Ajax, o Dojo oferece uma série de facilidades para permitir a comunicação do cliente com o servidor sem o recarregamento da página. Estas facilidades estão encapsuladas em API's construídas no topo do XMLHTTPRequest. É importante ressaltar que o Dojo é um *framework* que funciona do lado do cliente, o que faz com que mais código seja executado desse lado da aplicação. Mas ainda é necessário a utilização de uma linguagem intermediária, para, por exemplo, fazer a conexão com um banco de dados e passar os resultados de volta num formato compreensível para o Dojo. Qualquer linguagem que se comunique com um servidor HTTP e tenha suporte à banco de dados pode ser usada (RIECKE et al., 2008).

Por sua robustez e capacidade de integração com projetos já existentes (RUSSEL, 2007), o Dojo foi escolhido para a implementação do Ajax na versão 2.5 do *framework* Miolo, que será o foco do próximo capítulo.

4 ESTUDO DE CASO – MIOLO 2.5

Segundo MATOS(2009), o Miolo é um framework para a criação de sistemas de informação acessíveis via web, baseado na linguagem PHP5, scripts Javascript e conceitos de Programação Orientada a Objetos.

O Miolo é um framework brasileiro, que teve seu desenvolvimento iniciado em 2001 na universidade UNIVATES, localizada em Lajeado – RS, e a partir de 2004 passou para a responsabilidade da Solis, uma cooperativa de software livre (SOLIS, 2007). Em 2003 a UFJF adotou o Miolo para o desenvolvimento de seus sistemas corporativos, com isso, diversas adaptações e melhorias foram adicionadas ao framework o que deu origem, em 2005, à sua segunda versão, interna à UFJF. A partir de 2006 teve início um intercâmbio entre a UFJF e a Solis com o intuito de unificar as versões do Miolo, fato que se concretizou em 2008 com o lançamento da versão 2.5 unificada.

O Miolo provê uma infra-estrutura para o desenvolvimento de sistemas de informação acessíveis via web, o que permite ao desenvolvedor manter o foco de trabalho nas regras de negócio e não nos detalhes de implementação. No Miolo um sistema é construído a partir do desenvolvimento de módulos, que integrados, formam uma aplicação. Dessa forma cada instalação do framework está associada a uma única aplicação. Por exemplo, o Sistema Integrado de Gestão Acadêmica da UFJF, aplicação desenvolvida com o Miolo, é composta por vários módulos (ensino, recursos humanos, biblioteca, administração) que integrados formam o sistema de informação da instituição.

Como é voltado especificamente para o desenvolvimento de aplicações web, o framework, desde sua primeira versão, buscou oferecer recursos para vencer as limitações específicas desse ambiente, que é baseado num protocolo de comunicação do tipo *request-response* e que não guarda informações sobre o estado e a conexão. Esses recursos incluem um rico conjunto de componentes de interface com o usuário e o controle de sessões, mantendo no servidor as informações importantes sobre a interação do usuário com a aplicação. Porém, ainda estava preso ao modelo de submissão de formulários, pois a cada envio de informações pelo usuário os dados eram processados pelo servidor e a página novamente recarregada com todo o conteúdo.

Sua segunda versão já apresentava algumas características do Ajax, principalmente no que diz respeito à atualização de apenas determinadas áreas da página. Por exemplo em um formulário que apresenta os dados pessoais do usuário, é possível atualizar apenas a caixa de seleção do município de nascimento de acordo com uma alteração da caixa de seleção do estado de nascimento. Escolhendo o estado de Minas

Gerais a caixa de seleção de municípios é carregada com uma lista das cidades mineiras (Figura 17). Isso é feito sem que o formulário seja inteiramente submetido, a alteração é detectada do lado do cliente e uma requisição do tipo XMLHttpRequest é enviada ao servidor, que retorna apenas o fragmento que necessita de atualização. Porém, quando os dados são submetidos, a informação é processada pelo servidor e a página novamente recarregada. Assim a versão 2 do MIOLO se enquadra no modelo de renderização parcial.

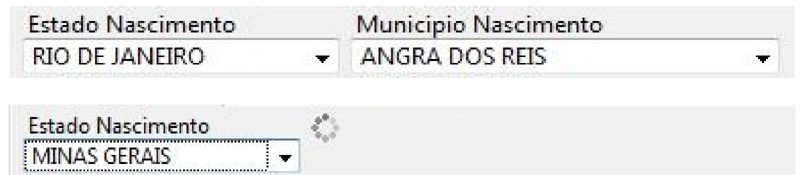

Figura 17: Exemplo de renderização parcial no Miolo 2.

A versão 2.5 adota de vez o novo modelo de desenvolvimento web, implementando o conceito SPI e mantendo todas as interações entre o cliente e o servidor baseadas em Ajax. Como não poderia deixar de ser também há uma grande mudança na interface de usuário, com o intuito de oferecer uma experiência mais rica e fluida, e controles mais avançados, aproximando ainda mais as aplicações desenvolvidas com o Miolo a uma aplicação desktop.

Algumas das principais funções implementadas pelo framework são:

- Arquitetura em camadas
- Possibilita o padrão MVC (Model-View-Controller)
- Rico conjunto de componentes UI , usando PHP5 e Javascript (DOJO)
- Modelo de programação event-driven , com forte uso de Ajax
- Gerenciamento de sessão e estado
- Aplicações independentes do navegador
- Segurança (autenticação, permissões, logs)
- Abstração de acesso a Banco de Dados
- Camada de persistência de objetos: MOQL (Miolo Object Query Language)
- Customização da UI através de temas e templates
- Geração de arquivos PDF (EzPDF, JasperReports)

4.1 ARQUITETURA

O Miolo adota uma arquitetura em camadas composta pela camada de

apresentação, camada de negócios, camada de integração e camada de recursos (Figura 18).

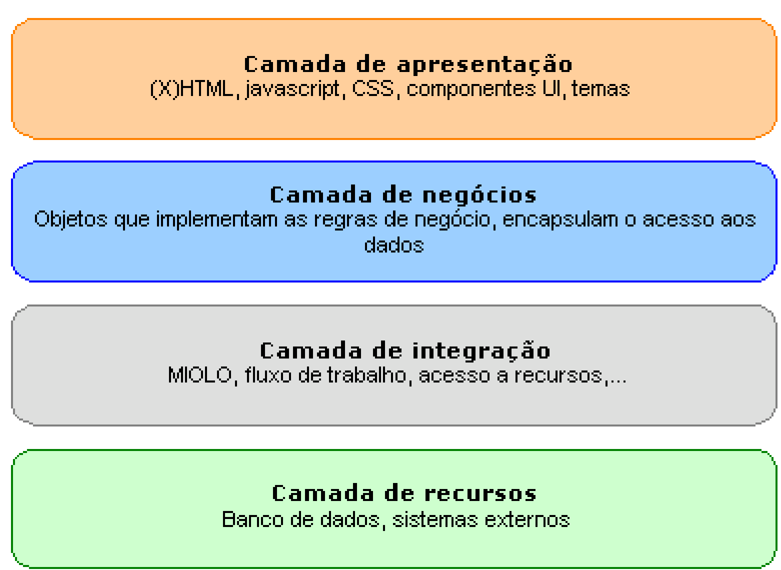

Figura 18: Arquitetura em camadas do Miolo

A camada de apresentação é responsável pela geração de arquivos, renderização dos controles HTML e criação dos scripts Javascript enviados ao cliente. Também fazem parte dessa camada as classes criadas pelos desenvolvedores da aplicação para a definição da interface do usuário (formulários, listagens e relatórios). Na camada de negócios ficam as classes criadas pelo desenvolvedor para representar as regras de negócio ou domínio da aplicação, são usadas pela camada de apresentação e acessam o banco de dados pela camada de recursos. A camada de integração disponibiliza a classe MIOLO, que representa o próprio framework, expondo métodos para facilitar o acesso e intercâmbio entre as demais camadas. Através dessa camada também é possível, tanto ao framework quanto à aplicação desenvolvida, acessar recursos e funcionalidades da linguagem ou do sistema operacional. Por último, a camada de recursos contém as classes responsáveis por abstrair o acesso às bases de dados, tornando as classes da camada de negócios independentes do SGBD utilizado

No Miolo, uma instalação do framework está associada a uma única aplicação, todos os seus arquivos ficam sob um mesmo diretório. A Figura 19 apresenta uma visão geral da estrutura de diretórios do Miolo, tendo como base o diretório C:\Miolo:

```
-----<diretório-base>(C:\Miolo)
          |
          +--- bin
          +--- classes
          |         +--- compatibility
          |         +--- contrib
          |         +--- database
          |         +--- doc
          |         +--- etc
          |         +--- extensions
          |         +--- ezpdf
          |         +--- flow
          |         +--- interfaces
          |         +--- model
          |         +--- persistence
          |         +--- pslib
          |         +--- security
          |         +--- services
          |         +--- ui
          |         +----+--- controls
          |         +----+--- fonts
          |         +----+--- painter
          |         +----+--- report
          |         +--- utils
          |         +--- miolo.class.php
          |         +--- support.inc.php
          +--- docs
          +--- etc
          |         +--- miolo.conf
          |         +--- passwd.conf
          |         +--- mkrono.conf
          +--- html
          |         +--- downloads
          |         +--- images
          |         +--- reports
          |         +--- scripts
          |         +--- themes
          |         +---+--- blue
          |         +---+--- mystica
          |         +---+--- system
          |         +---+--- ...
          |         +--- index.html
          |         +--- index.php
          +--- locale
          +--- modules
          |         +--- admin
          |         +--- modulo1
          |         +--- modulo2
          |         +--- ....
          +--- var
                    +--- db
                    +--- log
                    +--- report
                    +--- trace
```

Figura 19: Estrutura de diretórios do Miolo

No diretório *classes* ficam armazenadas as classes que formam o núcleo do Miolo. Nelas estão implementados os métodos para prover os serviços das diversas camadas que compõem o framework. O diretório *docs* armazena a documentação do Miolo. Em *etc* estão os arquivos de configuração para o funcionamento do sistema, eles guardam informações particulares sobre a instalação, como caminho dos arquivos, banco de dados utilizados, URL do servidor e o módulo responsável pela administração da aplicação. O diretório *html* é o

único que deve estar visível para o servidor web, ele que contém o ponto de entrada para a aplicação e é responsável também por armazenas os *scripts* que serão executados do lado do cliente e as definições do tema utilizado pela aplicação. Em *locale* está o sistema usado para internacionalização. *Modules* contém os módulos que formam a aplicação, cada um deles possui uma estrutura de diretórios pré-definidas, responsáveis por armazenas as classes do domínio de negócio da aplicação e também os elementos de interface com o usuário, como formulários, listas e relatórios (Figura 20). O diretório var armazena os arquivos gerados pelo framework, como relatórios, registros de utilização do sistema e processos de depuração.

```
-----<diretório-base>(C:\Miolo)
       |
       +--- modules
       |       +--- modulo1
       |       |       +--- classes
       |       |       +--- forms
       |       |       +--- menus
       |       |       +--- sql
       |       |       +--- handlers
       |       |       +--- grids
       |       |       +--- reports
       |       |       +--- etc
       |       |       +--- html
       |       |       |      +--- images
       |       |       |      +--- files
       |       |       +--- ui
       |       |              +--- controls
```

Figura 20: Estrutura de diretórios pré-definida de um módulo do Miolo.

Dentre os arquivos que compõem a instalação do framework, podemos destacar como principais os seguintes:

<miolo>/html/index.html: arquivo acessado inicialmente pelo servidor web, responsável por iniciar o processo de criação do ambiente para o sistema. Cria um *frameset*[13] com um *frame* chamado *content*, onde efetivamente é exibido o conteúdo das páginas do sistema para o usuário. O *frame content* chama o arquivo index.php. O objetivo do uso de *frames* é evitar que as URL's usadas pelo framework sejam exibidas no navegador.

<miolo>/html/index.php: é o manipulador principal do Miolo. Utilizado por todos os módulos e sempre presente nos links usados pelo framework. Seu papel principal é instanciar um objeto MIOLO, que é a principal classe do framework, e tratar as solicitações feitas pelo usuário.

<miolo>/etc/miolo.conf: arquivo que mantém as configurações do ambiente

<miolo>/classes/support.inc.php: Define as funções globais do framework.

13 Elemento HTML que permite dividir a página de forma que cada parte gerada exiba um documento independente. Essas partes de um *frameset* são chamados *frames.*

<miolo>/html/scripts/m_*.js: Arquivos com as funções javascript utilizadas pelo framework e seus componentes.

<miolo>/classes/miolo.class.php: Principal classe do MIOLO. Define os métodos principais do framework e permite o acesso aos serviços implementados pelas demais classes.

4.2 CAMADA DE APRESENTAÇÃO

Na versão 2.5, a camada de apresentação recebeu as maiores alterações, pois as principais mudanças foram no tratamento da interface com o usuário e no uso de temas, visando a implementação do modelo SPI. Entre os objetivos que guiaram a criação da nova versão estão a unificação de bibliotecas Javascript com o uso do *framework* Dojo (seção 3.4.5), manter as interações entre cliente e o servidor baseadas em Ajax, com flexibilidade para a atualização de diversos elementos simultaneamente, e o uso de *templates* para a construção dos temas. (MATOS, 2009).

As seções a seguir descrevem os elementos da camada de apresentação e seu funcionamento na versão 2.5 do Miolo.

4.2.1 Controles

Para a construção da interface com o usuário o Miolo provê uma série de controles visuais (Figura 21), que são classes programadas em PHP5, organizadas de forma hierárquica (ver ANEXO 1 – Árvore de controles do Miolo) e que encapsulam controles HTML ou construídos com Javascript[14]. Essas classes contêm a lógica de programação envolvida no funcionamento do controle e, se necessário, o código Javascript associado. Nelas também estão indicadas as classes CSS usadas para a exibição correta do controle nos navegadores, processo chamado de renderização.

Para a renderização os controles possuem um método responsável pela geração do código HTML correspondente chamado *generateInner*. Esse método do controle funciona em conjunto com a classe do framework MHTMLPainter (<miolo>/classes/ui/painter), que possui os métodos para a geração final do código HTML. Cada método dessa classe recebe um objeto do tipo controle e gera o código HTML referente a ele.

É importante ressaltar que no Miolo existe o conceito de controles atômicos, que são aqueles renderizados diretamente através de uma tag HTML. Os métodos da classe

14 Os controles do Miolo estão integrados com os *widgets Dijit* do Dojo.

MHTMLPainter retornam o código referente à renderização de um controle atômico. Como regra geral, um controle não-atômico é formado pela composição de controles atômicos, e sua renderização é feita com a execução do método correspondente para cada um de seus componentes.

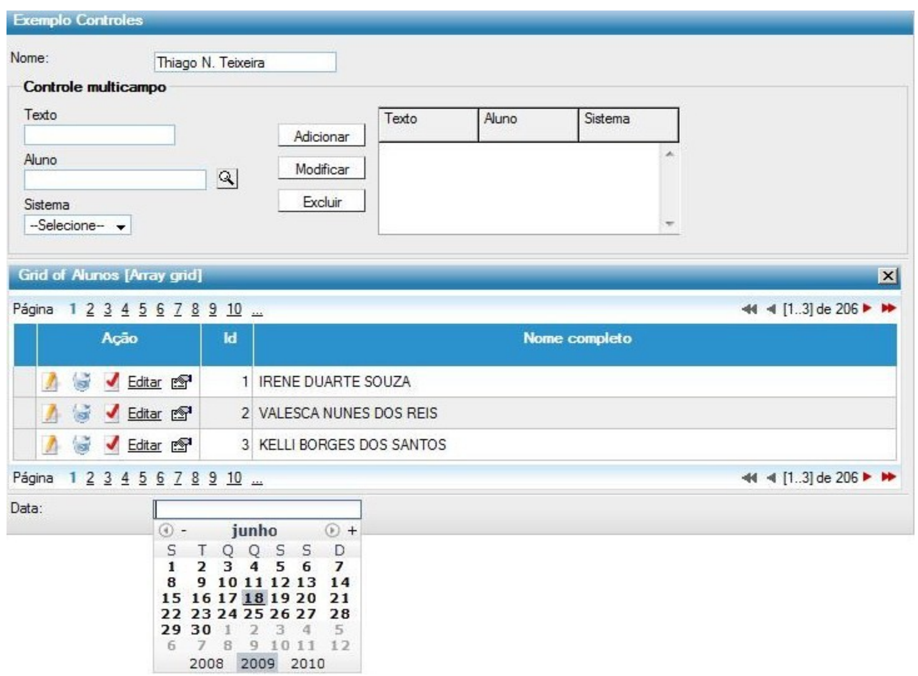

Figura 21: Exemplo de formulário criado no Miolo

4.2.2 Temas

No ambiente do Miolo, tema é a definição do layout da página HTML que será enviada ao navegador. O tema pode ser considerado um receptáculo para os controles que serão renderizados, e é formado por vários elementos (título, barra de navegação, menus, área de conteúdo, barra de status). Cada um desses elementos é representando por um controle da classe *MThemeElement.* A definição e a manipulação do tema é feita por algumas classes internas ao framework Miolo, por esse motivo a responsabilidade do tema não se restringe apenas a definir como a página será dividida, mas também como os controles HTML serão renderizados (MATOS, 2009). A Figura 22 destaca as área que compõem a estrutura do tema "*blue",* distribuído junto com o Miolo.

Figura 22: Estrutura padrão do tema *blue*

Cada uma dessas áreas é um elemento do tema (class *MThemeElement)*, e faz parte da estrutura de sua organização lógica, que é formada pela classe que representa o próprio tema (classe *Theme*) e seus diversos elementos. Por sua vez cada elemento do tema pode ser composto por um ou mais controles do Miolo (Figura 23). Cada ThemeElement é renderizado como um controle HTML Div[15], com um atributo identificador.

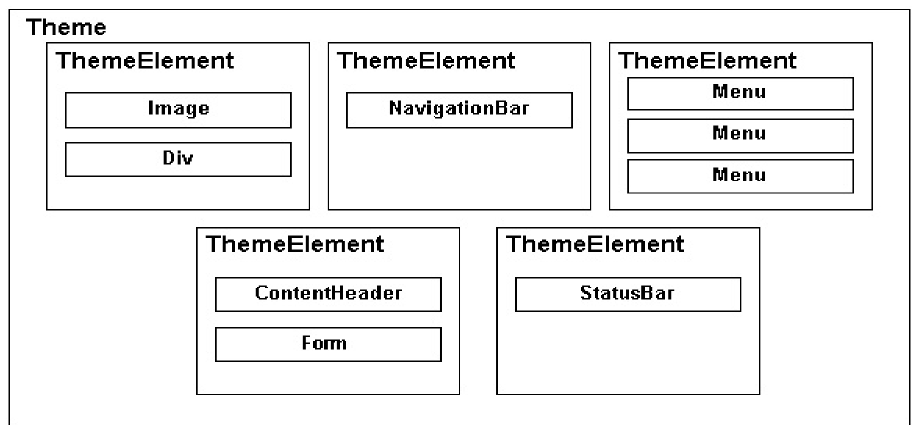

Figura 23: Organização lógica da estrutura do tema

Na estrutura de arquivos do Miolo os temas estão localizados no diretório <miolo>/html/themes. Cada tema existente na instalação do framework, e também aqueles que forem criados, deve estar presente em um subdiretório dos temas (Figura 24). Por exemplo, o tema *blue* está localizado em <miolo>/html/themes/blue. A definição do tema que será utilizado deve ser feita no arquivo de configuração (miolo.conf).

15 Uma *tag* div no HTML é usada onde uma parte, ou partes, do documento que não podem ser descritas por outros elementos HTML. No caso a tag div está sendo usada para definir áreas na página que receberão o código HTML de cada controle.

```
-----<diretório-base>(C:\Miolo)
       |
       +--- html
       |       +--- themes
       |       |       +--- blue
       |       |       |      +--- images
       |       |       |      +--- templates
       |       |       |      |      +--- base.php
       |       |       |      |      +--- content.php
       |       |       |      |      +--- default.php
       |       |       |      |      +--- menu.php
       |       |       |      |      +--- navbar.php
       |       |       |      |      +--- window.php
       |       |       |      +--- blue.css
       |       |       |      +--- dojo.css
       |       |       |      +--- miolo.css
       |       |       |      +--- theme.class.php
```

Figura 24: Estrutura de diretórios do tema *blue*.

A renderização do tema em uma página HTML é feita através de um objeto da classe MPage, que é uma representação no framework de um documento HTML. Esse objeto atua como um intermediário entre o tema e o navegador, pois sua responsabilidade é fazer chamadas aos métodos do tema que retornam o código HTML dos elementos, e repassá-los ao navegador, no formato de uma página completa ou apenas das partes que que necessitam de atualização. No caso, o tema é representado pela classe Theme<tema>, definida no arquivo theme.class.php (Figura 24).

Na versão 2.5 do Miolo o conteúdo HTML de cada elemento que compõe o tema é definido através de *templates*, que são ferramentas usadas para separar o conteúdo da apresentação no desenvolvimento web, permitindo representar o modelo padrão de uma página e indicar as áreas do documento que serão atualizadas. Os *templates* se localizam no diretório <miolo>/html/themes/<nome do tema>/templates. Em um tema devem estar definidos os seguintes *templates:*

- base.php: usado para renderização da página base (ver 4.2.3 Single Page Interface);
- default.php: usado para a renderização do conteúdo completo, incluindo todos os elementos do tema;
- window.php: usado para a renderização das janelas, na maior parte das vezes apenas o elemento “*content”.*

Na classe Theme<tema> devem ser definidos métodos para a geração dos layouts específicos (*base, default, window* etc.) e para cada um dos elementos do tema (*navbar, menu, content* etc). Cada um dos métodos utiliza os *templates* definidos.

O uso de *templates* integrados com a classe de controle dos temas permite uma flexibilização, de forma que a geração de uma área específica do tema possa ser

customizada ou mesmo omitida.

4.2.3 *Single Page Interface*

As aplicações criadas com o Miolo seguem o modelo SPI, de forma que uma única página é criada, e, de acordo com a interação do usuário com a aplicação, apenas determinadas áreas que a compõem são atualizadas.

No acesso à aplicação é criada a página principal, que tem seu conteúdo HTML definido pelo *template* base.php do tema que está sendo utilizado. Ela irá servir de base para a renderização dos conteúdos obtidos via Ajax. Pela definição dos temas do Miolo a página é dividida em "elementos do tema" (MThemeElements representados por elementos <div>). A cada chamada Ajax, o conteúdo HTML obtido como resposta, deve ser renderizado no elemento do tema correspondente.

Em uma aplicação web, os formulários (representado pela seção <form> do HTML) são responsáveis por agrupar os elementos de interação com o usuário. Uma página HTML pode conter mais de uma dessas seções. No caso da *single-page* do Miolo (Figura 25) cada formulário é renderizado dentro de um elemento do tema (<div>), que por sua vez contém todos os controles cujos conteúdos serão enviados juntos em uma submissão do formulário. O formulário principal é chamado "__mainForm" e é nele que são renderizados os controles enviados pelo servidor assim como os diversos elementos do tema (*top, menus, navigation, content* etc)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<title>Miolo Framework</title>

<link rel="stylesheet" type="text/css" href="http://127.0.0.1/themes/blue/dojo.css">
<link rel="stylesheet" type="text/css" href="http://127.0.0.1/themes/blue/miolo.css">

<meta http-equiv="Content-Type" content="0">
<meta name="Generator" content="MIOLO Version Miolo 2.5; http://www.miolo.org.br">
<script type="text/javascript"> djConfig={usePlainJson:true,  parseOnLoad:true}</script>
<script type="text/javascript" src="http://127.0.0.1/scripts/dojoroot/dojo/dojo.js"></script>
<script type="text/javascript" src="http://127.0.0.1/scripts/prototype/prototype.js"></script>
<script type="text/javascript" src="http://127.0.0.1/scripts/m_miolo.js"></script>
<script type="text/javascript" src="http://127.0.0.1/scripts/m_page.js"></script>
<script type="text/javascript" src="http://127.0.0.1/scripts/m_ajax.js"></script>
<script type="text/javascript" src="http://127.0.0.1/scripts/m_encoding.js"></script>
<script type="text/javascript" src="http://127.0.0.1/scripts/m_form.js"></script>
<script type="text/javascript" src="http://127.0.0.1/scripts/m_compatibility.js"></script>
<script type="text/javascript" src="http://127.0.0.1/scripts/m_md5.js"></script>
<script type="text/javascript" src="
http://127.0.0.1/scripts/jscookmenu/jscookmenu.js"></script>
<script type="text/javascript" src="
http://127.0.0.1//scripts/jscookmenu/jscookmenu_office.js"></script>
```

```
<script type="text/javascript">
dojo.require("dojo.parser");
dojo.require("miolo.Dialog");
dojo.require("dijit.form.DateTextBox");
dojo.require("dojox.layout.ContentPane");

//-->
</script>

<script type="text/javascript">
function init()
{
	miolo.doHandler("<?php echo $action ?>",'__mainForm');
}
dojo.addOnLoad(init);
//-->
</script>

</head>
<body class="mThemeBody">
<!-- begin of page -->
<div id="page49bff27a56937">

        <div id="stdout" class="mStdOut" dojoType="dojox.layout.ContentPane"></div>

        <!-- begin of form __mainForm -->
        <div id="__mainForm__scripts" dojoType="dojox.layout.ContentPane" layoutAlign="client"
executeScripts="true" cleanContent="true">
        </div>
        <div id="__mainForm" dojoType="dojox.layout.ContentPane" layoutAlign="client"
executeScripts="true" cleanContent="true">
</div>
<!-- end of form __mainForm -->
</div>
<!-- end of page -->
</body>
</html>
```

Figura 25: Código HTML da “single-page” base do Miolo

Nesta página, temos:

- Folha de Estilos por tema: para controles do Dojo (dojo.css) e do Miolo (miolo.css)
- Scripts: scripts básicos do MIOLO, usados por vários forms, mais scripts básicos do *framework* DOJO
- O corpo (*body*) da página HTML é constituído de um único div, que representa a página (renderizada pelo objeto MPage). Em uma mesma página é possível agregar vários formulários (objetos MForm ou MWindow).

O código HTML gerado inicialmente será incluído no elemento principal da aplicação, <div id=”__mainForm”>. A partir daí, todas as chamadas seguintes resultantes da execução da aplicação devem ser feitas via Ajax, de modo que apenas os elementos contidos no elemento principal sejam atualizados e os outros elementos da página, como cabeçalho, menu, barra de navegação permaneçam no estado em que foram carregados originalmente. Porém, nada impede que sejam feitas chamadas usando GET, o que provocaria uma nova renderização da página principal e a repetição do processo.

Para as chamadas Ajax, o Miolo provê uma infra-estrutura do lado do cliente representada principalmente pelo arquivo m_miolo.js, que contém os métodos Javascript para a interação entre o navegador e o servidor. Como visto acima, a primeira página é gerada com o acesso à aplicação (chamada GET iniciada quando é chamado o endereço diretamente no navegador), a partir daí o formulário principal é criado e os seguintes métodos são utilizados:

- *doPostBack(eventTarget, eventArgument, formSubmit)* : simula a submissão via método POST da página.
- *doLinkButton(url, eventTarget, eventArgument, formSubmit)* : simula a submissão da página (POST) o handler indicado por url.
- *doAjax(eventTarget, eventArgument, formSubmit)* : faz uma chamada Ajax.
- *doHandler(url, formSubmit)* : simula uma chamada GET para URL indicada.
- *doLink(url, formSubmit)* : simula uma chamada GET, alterando o atributo *action* do formulário para URL indicada.
- *doRedirect(url, element)* : simula um redirecionamento para outro endereço HTTP

Do lado do servidor, duas classes principais, a MPage e a MAjax, oferecem suporte ao uso de Ajax. A classe MPage é usada para estruturar a renderização do código HTML a ser enviado ao navegador e seus métodos associados ao uso de Ajax são:

- *generateForm* : renderiza o conteúdo de __mainForm
- *generateBase*: gera a página principal, ou quando é feita uma chamada via GET (sem Ajax)
- *generateAjax* : responde a uma chamada AJAX (com preenchimento de um ou vários elementos)

A classe MAjax representa a biblioteca Ajax no lado do servidor. Seus principais métodos são:

- *setResponseControls($controls,$elements='')* : $*controls* e $*elements* definem, respectivamente, o controle (ou vetor de controles) que serão exibidos e o identificador (ou vetor de identificadores) dos elementos que receberão o código HTML gerado através dos controles.
- *setResponse($controls,$elements='',$scripts=array())*: define ao mesmo tempo os controles, os elementos e os *scripts* que serão enviados como retorno de uma chamada Ajax.

4.2.4 Exemplo

A aplicação escolhida para servir de exemplo do uso do Miolo 2.5 foi o lançamento de notas parciais do SIGA (Sistema Integrado de Gestão Acadêmica). O SIGA é o sistema de informação corporativo da UFJF que abrange tanto a área acadêmica quanto a área administrativa da instituição. Tem como base de desenvolvimento o framework Miolo. (MATOS E RIBEIRO, 2007)

O motivo principal da escolha é que a aplicação atual está fortemente baseada no modelo de transição de páginas, com requisições do tipo GET e POST combinadas para levar a diversos tipos de operação, como o cadastro de avaliações e o lançamento de notas em si.

No modelo original, para o lançamento da primeira avaliação de uma turma e de suas respectivas notas foram levantadas sete tarefas distintas. São elas:

1. Definição do tipo de cálculo;
2. Clique no botão para adicionar avaliação;
3. Enviar os dados da avaliação;
4. Clique no botão OK para o aviso de avaliação inserida com sucesso;
5. Clique no link para lançar as notas da avaliação;
6. Clique no botão para gravar os dados das notas lançadas;
7. Clique no botão Voltar para notas parciais e assim visualizar os dados lançados.

A interface da aplicação foi refeita para utilizar os recursos da versão 2.5 do framework, buscando a diminuição dos passos para completar a tarefa e o aumento da taxa de resposta da aplicação. A diminuição dos passos da tarefa tem um impacto direto no tempo gasto para completar o objetivo da aplicação. O aumento da taxa de resposta permite ao usuário ter um fluxo de trabalho contínuo, sem a necessidade de esperar por informações do programa (WHITE, 2006).

Foi criada uma área de desenvolvimento com o Miolo 2.5 instalado e com acesso a um banco de dados que reproduz a estrutura utilizada no SIGA, povoado com dados fictícios para a simulação do lançamento de notas. Isso envolveu a criação de um curso, grade curricular, professor, turma e alunos. Com esses dados foi feita a matrícula dos alunos na turma.

Com o ambiente preparado, o primeiro passo para o desenvolvimento da nova interface foi criar um ponto de entrada para a aplicação, no painel de ações do módulo Ensino, disponível aos docentes da UFJF, chamado Minhas Turmas (Figura 26).

Figura 26: Painel de ações do docente

O acesso ao *link* Minhas turmas exibe um formulário contendo uma caixa de seleção com as turmas ministradas pelo docente no período. Esse formulário foi dividido em alguns elementos de tema (<div>) (Figura 27) que serão preenchidos por um evento associado à mudança na caixa de seleção (Figura 28) . O evento disparado pela escolha da turma é uma chamada Ajax e utiliza o método *setResponseControls* para passar os conteúdos HTML gerados pelos controles para os elementos div do formulário (Figura 29).

```
public function createFields()
{
    $ano = date('Y');
    $turmas = $this->docente->listTurmasAno($ano);

    $fields = array(
            new MSelection('selTurmas','','Turma',$turmas->result),
            new MSeparator(),
            new MDiv('divPeriodo'),
            new MDiv('divSituacaoTurma'),
            new MDiv('divTipoCalculo'),
            new MDiv('divInfo'),
            array(
                new MDiv('divFrmTvc'),
                new MDiv('divListTvc')
            ),
            new MDiv('divMatriculados')
        );

    $this->setFields($fields);

    $this->divFrmTvc->addStyle('width','300px');
    $this->divListTvc->addStyle('width','300px');
    $this->divFrmTvc->addAttribute('dojoType','dojox.layout.ContentPane');

    $this->setShowPostButton(FALSE);

    $this->selTurmas->addEvent('change',':doTvcMatriculas');
}
```

Figura 27: Trecho de código do formulário principal com os objetos MDiv

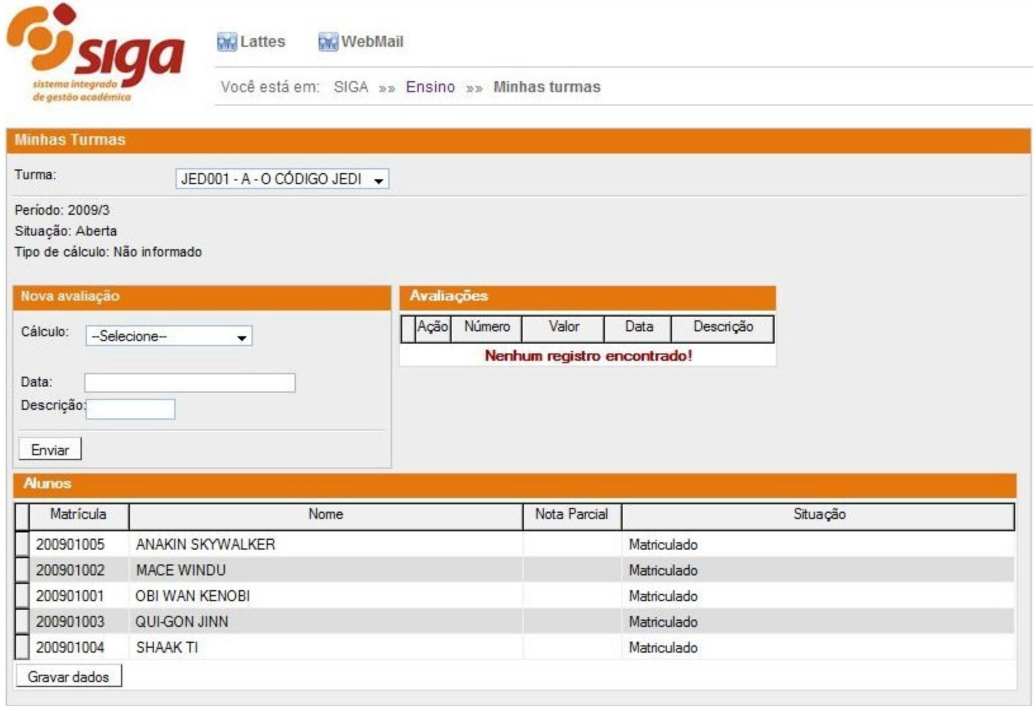

Figura 28: Interface de usuário após a escolha da turma.

```
$controls = array(
    $form,
    $listTvc,
    $matriculados,
    new MLabel("Período: $turma->ano/$turma->semestre"),
    new MLabel("Situação: $turma->situacao"),
    new MLabel("Tipo de cálculo: $msgTipoCalculo"),
);
$updates = array(
    'divFrmTvc',
    'divListTvc',
    'divMatriculados',
    'divPeriodo',
    'divSituacaoTurma',
    'divTipoCalculo',
);
$this->manager->ajax->setResponseControls($controls,$updates);
```

Figura 29: Trecho de código do evento Ajax executado.

Na nova interface, a escolha do tipo de cálculo a ser usado para o fechamento da turma é feita em conjunto com o lançamento dos dados referentes a primeira avaliação. No caso, a data e uma descrição para a avaliação. Caso o tipo de cálculo seja Média Ponderada ou Soma um campo para a informação do valor é exibido.

Após o preenchimento dos dados o clique no botão Enviar da caixa Nova avaliação dispara um novo evento Ajax, que salva os dados da avaliação no banco de dados e é responsável por atualizar o elemento Avaliações com a listagem das avaliações lançadas para a turma e também o elemento Alunos, onde é exibido uma nova coluna com os campos para receber a nota relativa à avaliação lançada. O elemento de informação é atualizado com a mensagem de que a avaliação foi cadastrada (Figura 30).

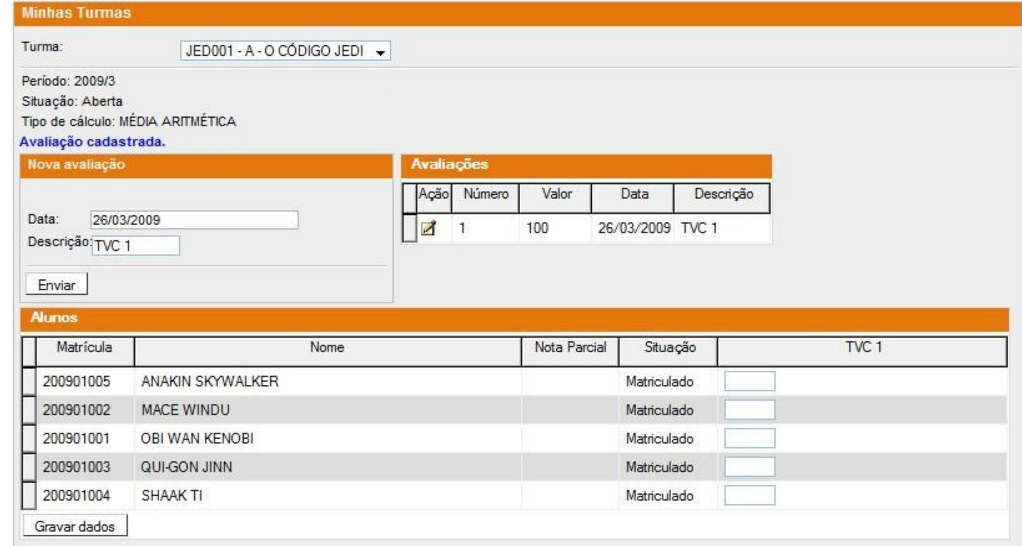

Figura 30: Lançamento dos dados da primeira avaliação.

Após esse passo a interface já está pronta para receber as notas dos alunos, que é feita lançando os respectivos valores nos campos de texto da coluna do TVC 1. Com os dados lançados o clique no botão Gravar Dados dispara o evento Ajax responsável por gravar as notas lançadas no banco de dados e por atualizar a listagem dos alunos, que passa a exibir a nota parcial calculada até o momento (Figura 31).

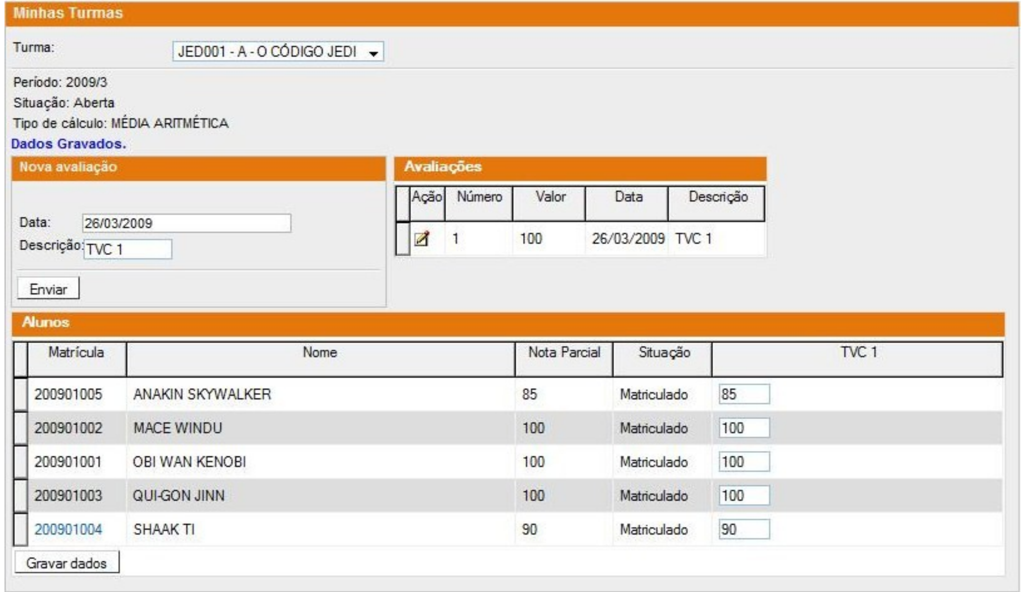

Figura 31: Formulário após o lançamento das notas da primeira avaliação.

Para o lançamento das demais notas das turmas o processo se repete, lançando os dados da nova avaliação e em seguida as notas dos alunos. Para cada avaliação lançada

os elementos Avaliações e Alunos vão sendo atualizados, mostrando os dados das avaliações e as colunas para o lançamento respectivamente. É possível também fazer a edição de uma avaliação clicando no ícone de editar na listagem das turmas, feito isso apenas o elemento Nova Avaliação é atualizado tornando-se Editar Avaliação (Figura 32).

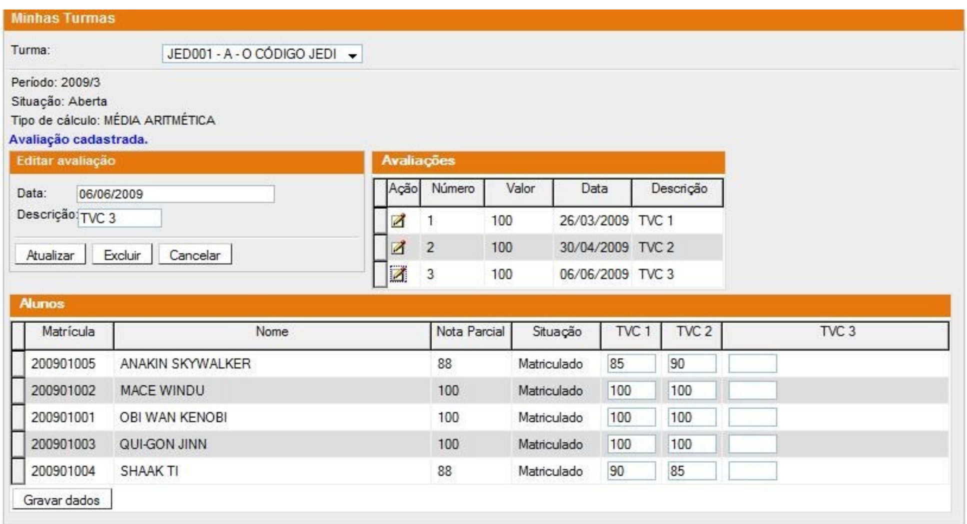

Figura 32: Formulário após o lançamento de três avaliações.

Com a redefinição da interface, o lançamento de notas para uma avaliação da turma passou de sete para duas tarefas: enviar os dados da avaliação e gravar os dados das notas dos alunos. Fato que foi possível graças à implementação SPI da versão 2.5 do Miolo, com as interações baseadas em Ajax e as atualizações de vários elementos simultaneamente.

Levando em consideração as variáveis tempo e quantidade de *bytes* transferidos foi elaborado um comparativo da aplicação de lançamento de notas levando em consideração três cenários distintos:

1. Lançamento de notas tradicional na versão 2 do Miolo comparado com o mesmo na versão 2.5 do *framework*;
2. Lançamento de notas tradicional na versão 2.5 do Miolo comparado com a interface otimizada;
3. Lançamento de notas tradicional na versão 2 do Miolo comparado com a interface otimizada.

Em cada um dos cenários foi considerada o lançamento de notas para a primeira avaliação de uma turma. O que envolve a definição do tipo de cálculo a ser utilizado, informação dos dados da avaliação e o lançamento de notas de cada aluno. Para fazer a comparação foi tomado o tempo global para o término da tarefa, obtido como uma média de

cinco operações repetidas para cada cenário (Ver APÊNDICE 2 – Dados coletados para comparação). Os dados lançados foram sempre os mesmos: tipo de dado média aritmética, data 01/06/2009 e descrição TVC 1 com nota 100 para todos os alunos. O tempo considerado foi apenas o da interação entre o navegador e o cliente, sendo ignorada o tempo que o usuário leva para informar os dados. Para os *bytes* transferidos foi utilizado a quantidade de tráfego recebido pelo navegador até a conclusão da operação.

Para o cenário 1, onde foi comparado as versões 2 e 2.5 executando o lançamento de notas tradicional, há um aumento de desempenho considerável no que diz respeito aos *bytes* transferidos. Isso porque na versão 2 toda vez que os dados são enviados a página é refeita, o que implica em dizer que na maior parte das vezes que há uma atualização todo o conteúdo HTML necessário para a renderização da página é transferido novamente para o navegador. Como a versão 2.5 utiliza o modelo SPI a transferência de dados cai bastante, porque com a página única apenas os elementos que sofrem atualizações são renderizados. Também há uma melhoria relativa no tempo de uso da aplicação. A tabela 2 apresenta os dados obtidos para esse cenário:

Tabela 2: Resultados obtidos para o cenário 1

	Miolo 2	Miolo 2.5	Aumento de desempenho	Aumento desempenho(%)
Bytes transferidos	385638,4	84172,8	301465,6	78,17
Tempo (segundos)	11,9	9,438	2,464	20,7

No cenário 2, a versão 2.5 é utilizada em ambos os casos, e a comparação é feita entre o lançamento de notas tradicional e sua versão otimizado. Nesse caso há um ganho considerável no tempo, pois o número de tarefas necessárias para completar a operação foi diminuída de sete passos da versão tradicional para dois na versão otimizada. E como as atualizações ocorrem em apenas alguns elementos da página e não há tanta transição entre os formulários como na aplicação tradicional, a versão otimizada gera uma grande diminuição no número de bytes transferidos. A tabela 3 apresenta os dados obtidos para esse cenário:

Tabela 3: Resultados obtidos para o cenário 2

	Tradicional	Otimizada	Aumento de desempenho	Aumento de desempenho (%)
Bytes transferidos	84172,8	19046,4	65126,4	77,37
Tempo (segundos)	9,44	2,938	6,500	68,87

O cenário 3 compara o lançamento de notas da versão 2 do Miolo com a interface

otimizada para a mesma operação na versão 2.5. O aumento de desempenho no que diz respeito aos bytes transferidos é novamente explicado pelo modelo SPI da versão 2.5 e as otimizações feitas para a utilização efetiva do Ajax. O aumento no ganho de tempo se justifica pela redução do número de tarefas necessárias para completar a operação. A tabela 4 apresenta os dados obtidos para esse cenário:

Tabela 4: Resultados obtidos para o cenário 3

	Miolo 2	Otimizada	Aumento de desempenho	Aumento de desempenho (%)
Bytes transferidos	385638,4	19046,4	366592	95,06
Tempo (segundos)	11,9	2,938	8,964	75,31

5 CONSIDERAÇÕES FINAIS

A popularização da web a tornou alvo para o desenvolvimento de aplicações, com muitos desenvolvedores migrando seus sistemas para essa plataforma. Porém esbarraram com as severas limitações inerentes desse ambiente, que é construído em cima de um protocolo do tipo *request-response.* Mesmo assim, principalmente pelas vantagens oferecidas por essa plataforma, como a simplificação da atualização dos sistemas e a possibilidade de acesso ao sistema de qualquer computador que tenha uma conexão com a *internet*, soluções foram buscadas para vencer essa barreira inicial.

A maior parte dessas soluções tomaram como ponto de partida as aplicações *desktop*, por sua já bem estabelecida base de usuários e documentação. Assim surgiram soluções que tentam aproximar as aplicações web de aplicações desktop, oferecendo formas de interação do usuário com a aplicação mais ricas e fluidas, rompendo de vez com o modelo de espera e recarregamento da interface a cada interação do usuário. Desses novos modelos destacam-se os aplicações que se baseiam em um cliente rico (RIA) e as interfaces de página única (*single page interface*), fortemente baseada em AJAX.

O modelo SPI no desenvolvimento web cria uma única página para a aplicação onde os elementos que compõem a página são atualizados de acordo com a interação do usuário com a aplicação. O primeiro impacto desse modelo é na transferência de dados entre o cliente e o servidor. Como as páginas não são recarregadas a cada submissão das informações para o servidor o fluxo de dados diminui consideravelmente.

Como exemplo da implementação do modelo SPI, foi apresentado no trabalho o *framework* Miolo 2.5, que permite a criação de aplicações web baseadas nesse modelo. O *framework* é a base de desenvolvimento do SIGA, sistema de informação corporativo da UFJF. Do SIGA foi escolhida o lançamento de notas parciais para ter sua interface refeita de acordo com as funcionalidades oferecidas pela nova versão do framework. A nova interface buscou diminuir o número de passos necessários para a conclusão da tarefa, e foi dividida em elementos que vão sendo atualizados de acordo com a interação do usuário com a aplicação.

Foram feitas algumas medições com o intuito de comparar o desempenho da tarefa de lançamento de notas parciais na versão 2 do Miolo, baseada no modelo de submissão de formulário, com a versão 2.5. Os resultados obtidos confirmam a diminuição do fluxo de transferência de dados prevista pelo modelo. No caso da comparação entre o modelo tradicional de lançamento de notas na versão 2 com a interface otimizada na versão 2.5 o aumento de desempenho para essa variável foi de 95%.

Outra variável utilizada foi o tempo de interação entre o cliente e o servidor. Na mesma situação anterior, o aumento de performance para essa variável foi de 75%.

Duas sugestões podem ser apontadas como trabalhos futuros: o levantamento de interfaces do SIGA que podem ser otimizadas, trazendo os benefícios do modelo SPI para o sistema da UFJF, e uma comparação entre os *frameworks* Ajax apresentados nesse trabalho e sua efetiva aplicação no desenvolvimento de aplicações usando o modelo SPI.

6 - REFERÊNCIAS BIBLIOGRÁFICAS

ALLAIRE, J. **Macromedia Flash MX—A next-generation rich client.** 2002. Disponível em: <http://download.macromedia.com/pub/flash/whitepapers/richclient.pdf>. Último acesso em: 28 jun. 2009.

ARAÚJO, J. G. R. **O Desenvolvimento de Aplicações WEB.** 1997. Disponível em: <http://www.rnp.br/newsgen/9710/n5-3.html>. Último acesso em: 28 jun. 2009.

BERNERS-LEE, T. **Information Management: A Proposal**. 1989. Disponível em: <http://www.w3.org/History/1989/proposal.html>. Último acesso em: 28 jun. 2009.

BERNERS-LEE, T. **The World Wide Web: Past, Present and Future**. 1996. Disponível em: <http://www.w3.org/People/Berners-Lee/1996/ppf.html>. Último acesso em: 28 jun. 2009.

CRANE, D et al. **Ajax in Action**. 1ª. ed. Greenwich: Maning Publications Co, 2006. 650p.

CRUZ, A. **Projeto e Implementação de um Framework paraWebLabs Baseado em Ajax e Padrões de Projeto**. Campinas – SP, 2007. 116p. Dissertação (Mestrado em Engenharia Elétrica) – Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas

DOJO. **The Official Dojo Documentation.** 2008. Disponível em: <http://docs.dojocampus.org>. Último acesso em: 28 jun. 2009.

ESPOSITO, D. **Single-page Interface and AJAX Patterns.** 2008. Disponível em: <http://msdn.microsoft.com/en-us/magazine/cc507641.aspx>. Último acesso em: 28 jun. 2009.

GAMMA, E et al. **Design Patterns: Elements of Reusable Object-Oriented Software.** Addison-Wesley, 1998. 395p.

GARRET, J. J. **Ajax: A New Approach to Web Applications.** 2005. Disponível em: <http://adaptivepath.com/ideas/essays/archives/000385.php>. Último acesso em: 28 jun. 2009.

MATOS, E. E. **Framework MIOLO 2.5**. 2009. Disponível em: <svn://svnext.ufjf.br/miolo/trunk>. Último acesso em: 28 jun. 2009.

MATOS, E. E. RIBEIRO, C. A. **Sistema Integrado de Gestão Acadêmica: a experiência da UFJF**. 2007. Disponível em: <http://www.viadigital.org.br/index.php?option=com_docman&task=doc_download&gid=27&Itemid=99999999>. Último acesso em: 28 jun. 2009.

MESBAH, A. **Analysis and Testing of Ajax-based Single-page Web Applications.** Delft-Holanda, 2009. 208p. Tese (Doutorado em Engenharia de *Software*) - *Delft University of Technology.*

O'REILLY, T. **What Is Web 2.0.** 2005. Disponível em: <http://oreilly.com/web2/archive/what-is-web-20.html>. Último acesso em: 28 jun. 2009.

REIS, T. R. **Desenvolvimento Web com o Uso de Padrões: Tecnologias e Tendências**. Juiz de Fora – MG. 2007. 73 p. Monografia final de curso (Bacharelado em Ciência da Computação). Universidade Federal de Juiz de Fora.

RIECKE, C. et al. **Mastering Dojo Javascript and Ajax Tools for Great Web Experience**. 1ª. ed. Pragmatic Bookshelf, 2008. 545p.

ROBINSON, D. **The Common Gateway Interface (CGI) Version 1.1.** 2004. Disponível em: http://www.ietf.org/rfc/rfc3875.txt. Último acesso em: 28 jun. 2009.

RUSSESL, M. **Dojo Fundamentals: Creating Object-Oriented Widgets**. 1ª. ed. O'Reilly Media, 2007. 55p.

SILVA, C. K. P. **Web 2.0: A migração para a Web social.** Juiz de Fora – MG. 2007. 68 p. Monografia final de curso (Bacharelado em Ciência da Computação). Universidade Federal de Juiz de Fora.

SOLIS. **SOLIS – Cooperativa de Soluções Livres**. 2007. Disponível em: <http://www.solis.coop.br>. Último acesso em: 28 jun. 2009.

W3C. **What is the Document Object Model?** 1998. Disponível em: <http://www.w3.org/TR/1998/WD-DOM-19980720/introduction.html>. Último acesso em: 28 jun. 2009.

W3C. **HTML 4.01 Specification.** 1999a. Disponível em: <http://www.w3.org/TR/REC-html40>. Último acesso em: 28 jun. 2009.

W3C. **Forms.** 1999b. Disponível em: <http://www.w3.org/TR/html401/interact/forms.html>. Último acesso em: 28 jun. 2009.

W3C. **URIs, Addressability, and the use of HTTP GET and POST.** 2004. Disponível em: <http://www.w3.org/TR/html401/interact/forms.html>. Último acesso em: 28 jun. 2009.

WHITE, A. **Measuring the Benefits of Ajax.** 2006. Disponível em: <http://www.developer.com/java/other/article.php/10936_3554271_1>. Último acesso em: 28 jun. 2009.

APÊNDICE 1 – Exemplo de uma *single-page* usando o *framework* Dojo

Nessa seção será vista a construção de uma *single-page* usando os recursos do *framework* Dojo e uma das soluções oferecidas para simular a navegação no modelo SPI. Consiste de um *site* simples, composto por quatro páginas contendo a mesma estrutura onde apenas o conteúdo varia.

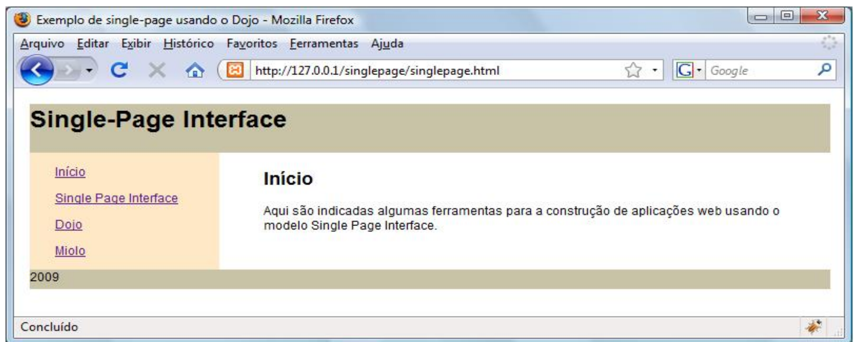

Figura 33: *Site Single-Page Interface.*

Para sua criação, é utilizado um único arquivo HTML. Esse arquivo carrega de forma dinâmica apenas a parte de conteúdo da página, através de chamadas ao objeto XMLHTTPRequest quando um item do menu é clicado.

```
<html>
<head>
  <title>Exemplo de single-page usando o Dojo
  </title>

  <style type="text/css">
    @import "singlepage.css";
  </style>

  <script
    type="text/javascript"
    src="/dojoroot/dojo/dojo.js"
    djConfig="isDebug: true"></script>

  <script
    type="text/javascript"
    src="singlepage.js"></script>
  </head>
  <body>
<div id="pagina">
    <div id="topo"><H1>Single-Page Interface</H1></div>
<div id="principal">
        <div id="conteudo">
        </div>
</div>
<div id="navegacao">
    <div id="menu">
    <p><a href="#inicio">Início</a></p>
    <p><a href="#single">Single Page Interface</a></p>
    <p><a href="#dojo">Dojo</a></p>
    <p><a href="#miolo">Miolo</a></p>
</div>
</div>
<div id="rodape">2009</div>
</div>
<body>
```

Figura 34: Código HTML da “*single page*”.

Os menus são implementados por um conjunto de linhas dentro de um elemento *div* que, quando clicados, chamam um manipulador que executa a função dojo.xhrGet. Essa função faz parte da API do Dojo relativa à utilização do objeto XMLHTTPRequest e é responsável por recuperar o conteúdo e passá-lo para o div#conteudo.

A Figura 35 apresenta as funções criadas no arquivo *singlepage.js* para o controle dos menus. A função menuPrincipal é chamada quando um item do menu é clicado, a partir daí ela executa a função carregaConteudo, que recebe o identificador do conteúdo e faz uma chamada ao servidor usando o método *xhrGet* da API do Dojo.

Para que o servidor faça a leitura do arquivo que contém o conteúdo a ser exibido, é passado como parâmetro para o *xhrGet* o identificador do item do menu mais a extensão ".htm". O arquivo é lido e o elemento div#conteudo é atualizado, recebendo o código HTML na sua propriedade *innerHTML*. Como são quatro os identificadores nos menus, sendo eles inicio, single, dojo e miolo, terão que existir quatro arquivos com o conteúdo dessas seções, sendo eles inicio.htm, single.htm, dojo.htm e miolo.htm. A Figura 36 apresenta o conteúdo do arquivo miolo.htm.

```
function carregaConteudo(identificador){
  dojo.xhrGet({
    url:  identificador + ".htm",
    handleAs: "text",
    handle: function(response){
      dojo.byId("conteudo").innerHTML = response;
    }
  });
}
function menuPrincipal(e){
  dojo.stopEvent(e);
  var identificador= e.target.getAttribute("href").split("#")[1]
  carregaConteudo(identificador);
  dojo.back.addToHistory(new Estado(identificador));
}
```

Figura 35: Funções Javascript para o menu do site.

```
<h2>Miolo 2.5</h2>
<p>Framework para o desenvolvimento de aplicações web com o modelo
Single-Page Interface. Utiliza o toolkit Dojo e a linguagem PHP. </p>
<p>Disponível em: <a href="http://www.miolo.org.br">http://www.miolo.org.br</a></p>
```

Figura 36: Arquivo HTML com conteúdo para o *site*.

Para resolver o problema da navegação quando se usa uma única página, o Dojo inclui um módulo chamado *dojo.back*. A idéia é alterar manualmente a propriedade *window.location* do DOM (*Document Object Model),* o que tem o efeito de mudar a URL na barra de endereço e incluir um item ao histórico do navegador, sem recarregar a página.

Para que o *dojo.back* funcione corretamente é necessário criar uma classe que

armazene o estado da navegação e passá-lo para o *dojo.back* a cada mudança. Na Figura 35 o método *dojo.back.addToHistory* faz o armazenamento dos estados de acordo com a navegação nos menus.

O objeto de estado precisa incluir o atributo *changeUrl,* que guarda o identificador associado com um estado em particular. E também os métodos que serão chamados quando o *dojo.back* detecta uma navegação do tipo Avançar ou Voltar (o *dojo.back* monitora o uso dos botões do navegador). Quando o botão Voltar é clicado o método *back* é chamado, da mesma forma, para o botão Avançar é chamado o método *forward.*

A Figura 37 apresenta a definição do objeto de estado que deve ser passado aos métodos do *dojo.back.*

```
var Estado = function(identificador){
  this.changeUrl = identificador;
}
dojo.extend(Estado, {
  back: function() {
    carregaConteudo(this.changeUrl);
  },
  forward: function(){
    carregaConteudo(this.changeUrl);
  }
});
```

Figura 37: Definição do objeto de estado para navegação.

Dessa forma, cada vez que um botão de Avançar ou Voltar é clicado, a função correspondente (*back* ou *forward)* é chamada no contexto do objeto de estado associado com o item do histórico de navegação. Esse objeto guarda o identificador que é usado para carregar o respectivo conteúdo.

O Dojo pode ser encontrado em http://dojotoolkit.org, e apesar de poder ser executado diretamente no navegador é recomendável sua instalação em um servidor web. O exemplo apresentado parte do princípio de que o Dojo está instalado na pasta /dojoroot/dojo/, por sua vez localizada na pasta raiz dos documentos HTML do servidor web. Os arquivos criados para o exemplo também estão nessa pasta, são eles:

- *singlepage.htm*: definição da página principal;
- *singlepage.css:* estilos CSS utilizados;
- *singlepage.js:* funções Javascript para a navegação;
- *inicio.htm:* conteúdo da seção inicial;
- *single.htm:* conteúdo da seção sobre o modelo SPI
- *dojo.htm:* conteúdo da seção sobre o Dojo

- *miolo.htm:* conteúdo da seção sobre o *framework* Miolo 2.5

singlepage.html

```
<html>
<head>
  <title>Exemplo de single-page usando o Dojo
  </title>

  <style type="text/css">
    @import "singlepage.css";
  </style>

  <script
    type="text/javascript"
    src="/dojoroot/dojo/dojo.js"
    djConfig="isDebug: true"></script>

  <script
    type="text/javascript"
    src="singlepage.js"></script>
  </head>
  <body>
<div id="pagina">
    <div id="topo"><H1>Single-Page Interface</H1></div>
<div id="principal">
        <div id="conteudo">
        </div>
</div>
<div id="navegacao">
    <div id="menu">
    <p><a href="#inicio">Início</a></p>
    <p><a href="#single">Single Page Interface</a></p>
    <p><a href="#dojo">Dojo</a></p>
    <p><a href="#miolo">Miolo</a></p>
</div>
</div>
<div id="rodape">2009</div>
</div>
<body>
```

singlepage.css

```
@import "../../dojo/resources/dojo.css";
body{
       margin:10px 15px;
       font: 12px Arial, Helvetica, sans-serif;
       text-align:center;
       }
#pagina {
       width:760px;
       text-align:left;
       margin:0 auto;
       }
#topo {
       height:50px;
    background-color: #C8C2A7;
       }

#principal {
       width:578px;
       float:right;
       }
#navegacao {
       width:180px;
       background-color: #FEE8C6;
       float:left;
       }
#principal * {
       padding: 0 20px;
       }
#navegacao * {
```

```
        padding: 0 8px;
        }
#rodape {
        clear:both;
        height:20px;
    background-color: #C8C2A7;
        }

 div.menu {
  background-color: #F0F0F0;
  padding: .25em;
  border: 2px solid black;
  border-left: 0;
  border-right: 0;
}
```

singlepage.js

```
dojo.require("dojo.back");
(function() {
  function getIdentificador(){
    var parts= window.location.href.split("#");
    if (parts.length==2) {
      return parts[1];
    } else {
      return "inicio";
    }
  }
  var Estado = function(identificador){
    this.changeUrl = identificador;
  }
  dojo.extend(Estado, {
    back: function() {
      carregaConteudo(this.changeUrl);
    },
    forward: function(){
      carregaConteudo(this.changeUrl);
    }
  });
  function carregaConteudo(identificador){
    dojo.xhrGet({
      url:  identificador + ".htm",
      handleAs: "text",
      handle: function(response){
        dojo.byId("conteudo").innerHTML = response;
      }
    });
  }
  function menuPrincipal(e){
    dojo.stopEvent(e);
    var identificador= e.target.getAttribute("href").split("#")[1]
    carregaConteudo(identificador);
    dojo.back.addToHistory(new Estado(identificador));
  }
  dojo.addOnLoad(function(){
    dojo.connect(dojo.byId("menu"), "click", menuPrincipal);
    var inicio= getIdentificador();
    carregaConteudo(inicio);
    dojo.back.setInitialState(new Estado(inicio));
  })
})();
```

inicio.htm

```
<h2>Início</h2>
<p>Aqui são indicadas algumas ferramentas para a construção de aplicações web usando o modelo
Single Page Interface.</p>
```

single.htm

```
<h2>Single Page Interface</h2>
<p>Modelo de desenvolvimento web focado em aplicações de página única, comunicação assíncrona
com o servidor e ricos componentes de interface com o usuário</p>
```

dojo.htm

```
<h2>Dojo</h2>
<p>Toolkit Javascript para ampliar as capacidades do navegador, oferece um extenso conjunto
de controles de interface de usuário para a criação de aplicações web</p>
<p>Disponível em: <a href="http://dojotoolkit.org">http://dojotoolkit.org</a></p>
```

miolo.htm

```
<h2>Miolo 2.5</h2>
<p>Framework para o desenvolvimento de aplicações web com o modelo Single-Page Interface.
Utiliza o toolkit Dojo e a linguagem PHP. </p>
<p>Disponível em: <a href="http://www.miolo.org.br">http://www.miolo.org.br</a></p>
```

APÊNDICE 2 – Dados coletados para comparação

A seção 4.2.4 apresenta um exemplo de aplicação desenvolvida na versão 2.5 do Miolo. Para efeito de comparação, foram coletados o tempo gasto e os bytes transferidos no lançamento de notas do SIGA. Essa seção apresenta os dados coletados para a geração da média do tempo total da operação.

Tarefas que tiveram o tempo medido para a operação de lançamento de notas tradicional:

1.Definição do tipo de cálculo;

2.Clique no botão para adicionar avaliação;

3.Enviar os dados da avaliação;

4.Clique no botão OK para o aviso de avaliação inserida com sucesso;

5.Clique no link para lançar as notas da avaliação;

6.Clique no botão para gravar os dados das notas lançadas;

7.Clique no botão Voltar para notas parciais e assim visualizar os dados lançados.

Tempo (em segundos) coletado para o lançamento de notas tradicional na versão 2 do Miolo:

		Tarefas							Total
		1	2	3	4	5	6	7	
MEDIDAS	1	1,95	1,61	1,81	1,70	1,68	1,89	1,69	12,31
	2	1,94	1,47	1,66	1,65	1,62	1,99	1,62	11,94
	3	1,92	1,47	1,54	1,61	1,59	1,80	1,62	11,55
	4	1,91	1,49	1,59	1,78	1,57	1,94	1,63	11,91
	5	1,94	1,45	1,56	1,68	1,71	1,84	1,63	11,80

Média: 11,9 segundos

Transferência: 376,6 KB

Tempo (em segundos) coletado para o lançamento de notas tradicional na versão 2.5 do Miolo:

		Tarefas							Total
		1	2	3	4	5	6	7	
MEDIDAS	1	1,45	1,20	1,38	1,35	1,29	1,62	1,31	9,60
	2	1,46	1,22	1,40	1,36	1,29	1,54	1,37	9,64
	3	1,42	1,23	1,32	1,31	1,32	1,47	1,37	9,44
	4	1,43	1,19	1,33	1,29	1,25	1,43	1,29	9,21
	5	1,39	1,23	1,31	1,31	1,28	1,45	1,33	9,30

Média: 9,44 segundos

Transferência: 82,2 KB

Tempo (em segundos) coletado para o lançamento de notas com interface otimizada na versão 2.5 do Miolo:

		Tarefas		Total
		Adicionar Avaliação	Gravar dados	
MEDIDAS	1	1,59	1,43	3,02
	2	1,49	1,43	2,92
	3	1,53	1,36	2,89
	4	1,45	1,41	2,86
	5	1,50	1,50	3

Média: 2,94 segundos

Transferência: 18,6KB

APÊNDICE 3 – Código da nova interface do lançamento de notas parciais

A seção 4.2.4 apresenta o lançamento de notas do SIGA refeito para aproveitar os recursos da versão 2.5 do *framework* MIOLO. Para isso foram criados novos elementos de interface com o usuário, que, no MIOLO, são representados por formulários e listagens (*grids):*

- *frmMinhasTurmas.class*: formulário principal. Responsável por fazer a divisão inicial dos elementos que serão atualizados de acordo com a interação entre o usuário e a aplicação. Ele também contém os métodos Ajax que são disparados de acordo com determinados eventos, como inclusão de uma nova avaliação e gravação das notas dos alunos.
- *frmTvc.class*: formulário para a inclusão de uma nova avaliação.
- *frmTvcEdit.class*: formulário para a edição de avaliações já existentes.
- *gridTvc.class*: lista as avaliações lançadas para uma turma. A partir dele é possível editar os dados de uma avaliação.
- *gridNotasMatriculados.class*: lista os alunos matriculados em uma turma. A cada avaliação lançada é exibido uma nova coluna para o lançamento da nota de um aluno.
- *gridHistNotasMatriculados.class*: lista as notas parciais e a nota final dos alunos. Exibido após o fechamento da turma.

frmMinhasTurmas.class

```
<?
class frmMinhasTurmas extends MForm
{
        public $docente;
    public $ui;

        function __construct($docente)
        {
                $this->docente = $docente;
                parent::__construct('Minhas Turmas');
                $this->ui = $this->manager->getUi();
                $this->eventHandler();
        }

        public function createFields()
        {
                $ano = date('Y');
                $turmas = $this->docente->listTurmasAno($ano);

                $fields = array(
                        new MSelection('selTurmas','','Turma',$turmas->result),
                 new MSeparator(),
                 new MDiv('divPeriodo'),
                 new MDiv('divSituacaoTurma'),
```

```
                new MDiv('divTipoCalculo'),
                new MDiv('divInfo'),
                       array(
                       new MDiv('divFrmTvc'),
                       new MDiv('divListTvc')
                       ),
                       new MDiv('divMatriculados')
               );

              $this->setFields($fields);

       $this->divFrmTvc->addStyle('width','300px');
       $this->divListTvc->addStyle('width','300px');
       $this->divFrmTvc->addAttribute('dojoType','dojox.layout.ContentPane');

              $this->setShowPostButton(FALSE);

              $this->selTurmas->addEvent('change',':doTvcMatriculas');
       }

       public function doDetalhesTurma($args)
       {
              $idTurma = $args->selTurmas;
              if ( $idTurma )
              {
               $turma = $this->manager->getBusiness('ensino','turma');
               $turma->getByIdTurma($idTurma);
               $form = $this->ui->getForm('ensino','frmTurma',$turma,'minhasturmas');
               $this->manager->ajax->setResponseControls($form,'detalheTurma');
              }
       }

    public function doTvcMatriculas($args)
    {
              $idTurma = $args->selTurmas;

        if ( $idTurma )
        {
            $turma = $this->manager->getBusiness('ensino','turma');
            $turma->getByIdTurma($idTurma);

            if ( $turma->situacao == 'Fechada' )
            {
                $form = new MDiv('divBlank');
                $matriculados = $this->ui->getGrid('ensino','gridHistNotasMatriculados',
$turma);
            }
            else
            {
                $form = $this->ui->getForm('ensino','frmTvc',$turma,'minhasturmas');
                $matriculados = $this->getMatriculados($turma);
            }
            $listTvc = $this->ui->getGrid('ensino','gridTvc',$turma);

            if ( $turma->tipocalculo )
            {
                $tabelaGeral = $this->manager->getBusiness('common','tabelageral');
                $tabelaGeral->getByItem('GA_TIPOCALCULO',$turma->tipocalculo
);
                $msgTipoCalculo = $tabelaGeral->item2;
            }
            else
            {
                $msgTipoCalculo = "Não informado";
            }
            $controls = array(
                $form,
                $listTvc,
                $matriculados,
                new MLabel("Período: $turma->ano/$turma->semestre"),
                new MLabel("Situação: $turma->situacao"),
                new MLabel("Tipo de cálculo: $msgTipoCalculo"),
            );
```

```
        $updates = array(
            'divFrmTvc',
            'divListTvc',
            'divMatriculados',
            'divPeriodo',
            'divSituacaoTurma',
            'divTipoCalculo',
        );

        $this->manager->ajax->setResponseControls($controls,$updates);
    }
}

public function doAdicionaTvc($args)
{
          $idTurma = $args->selTurmas;
    $data->descricao = $args->descricao;
    $data->data = $args->data;
    $data->peso = $args->peso;
    $data->idturma = $idTurma;

    $tvc = $this->manager->getBusiness('ensino','tvc');
    $tvc->setData($data);
          $tvcs = $tvc->GetTvcsTurma();
    if ( $tvcs->eof() )
    {
        $tvc->numtvc = 1;
    }
    else
    {
        $tvc->numtvc = $tvcs->rowCount+1;
    }
    $turma = $this->manager->getBusiness('ensino','turma');
    $turma->getByIdTurma($idTurma);

    if ( $args->tipocalculo )
    {
        $turma->tipocalculo = $args->tipocalculo;
        $turma->save();
    }
    //testar se peso do tvc esta dentro do permitido para o tipo de calculo

    if ( $turma->tipocalculo == 2 )
    {
        $tvc->peso = 100;
    }
    else
    {
        if ( $tvc->peso > 40 )
        {
            $msg = "Peso da avaliação maior que o permitido";
            $errTvc = TRUE;
        }
        else
        {
           $somapeso = $turma->GetTotalPesoTurma();
            $somapeso = $somapeso[0][0];
            if ( ($somapeso + $tvc->peso) > 100 )
            {
                $valor = 100 - ($somapeso);
                $msg = "Peso da avaliação não pode ser superior a $valor";
                $errTvc = TRUE;
            }
        }
    }
    if ( ! $errTvc )
    {
        try
        {
            $info = new MLabel('Avaliação cadastrada.','blue',TRUE);
            $tvc->Insert();
        }
```

```
        catch ( Exception $e )
        {
            $info = new MLabel('Erro:'.$e->getMessage(),'red',TRUE);
        }

        $listTvc = $this->ui->getGrid('ensino','gridTvc',$turma);
        $matriculados = $this->getMatriculados($turma);

        $controls = array(
            $info,
            $listTvc,
            $matriculados
        );

        $updates = array(
            'divInfo',
            'divListTvc',
            'divMatriculados'
        );

        if ( $args->tipocalculo )
        {
            //atualizar o proprio formulario de tvc's
            $form = $this->ui->getForm('ensino','frmTvc',$turma,'minhasturmas');
            //atualizar o label de tipo de calculo

            $tabelaGeral = $this->manager->getBusiness('common','tabelageral');
            $tabelaGeral->getByItem('GA_TIPOCALCULO',$turma->tipocalculo);

            $controls[] = $form;
            $controls[] = new MLabel("Tipo de cálculo: $tabelaGeral->item2");
            $updates[] = 'divFrmTvc';
            $updates[] = 'divTipoCalculo';
        }
    }
    else
    {
        $controls = array(
            new MLabel($msg,'red',TRUE)
        );
        $updates = array(
            'divInfo'
        );
    }
    $this->manager->ajax->setResponseControls($controls,$updates);
}

public function doExibePeso($args)
{
    $tipocalculo = $args->tipocalculo;
    //para soma e media ponderada exibir o campo peso
    if ( $tipocalculo == 1 || $tipocalculo == 3 )
    {
        $controls = array(
            $peso = new MTextField('peso','','Peso',4),
            $labelPeso = new MLabel('Peso:')
        );
    }
    else
    {
        $controls = array(
            $peso = new MDiv('divPeso'),
            $labelPeso = new MDiv('divLabelPeso')
        );
    }
    $updates = array(
        'divPeso',
        'divLabelPeso'
    );
    $this->manager->ajax->setResponseControls($controls,$updates);
}

public function doGravaNotas($args)
```

```
    {
        $notas = $args->idtvc;
        $turma = $this->manager->getBusiness('ensino','turma');
        $turma->getByIdTurma($args->selTurmas);

        foreach ( $notas as $idTvc => $matriculas )
        {
            foreach ( $matriculas as $matricula => $nota )
            {
                if ( $nota )
                {
                    $objNota = $this->manager->getBusiness('ensino','nota');
                    $objNota->update2($nota,$idTvc,$turma,$matricula);
                }
            }
        }
        $matriculados = $this->getMatriculados($turma);
        $controls = array(
            new MLabel('Dados Gravados.'.$args->setTurmas,'BLUE',TRUE),
            $matriculados
        );
        $updates = array(
            'divInfo',
            'divMatriculados'
        );
        $this->manager->ajax->setResponseControls($controls,$updates);
    }

    public function doEditTvc($args)
    {
        $idtvc = $args;
        $tvc = $this->manager->getBusiness('ensino','tvc');
        $tvc->getById($idtvc);
        $form = $this->ui->getForm('ensino','frmTvcEdit',$tvc,'minhasturmas');
        $this->manager->ajax->setResponseControls($form,'divFrmTvc');
    }

    public function doUpdateTvc($args)
    {
        $idtvc = $args->idtvcUpd;
        $idturma = $args->selTurmas;
        $tvc = $this->manager->getBusiness('ensino','tvc');
        $tvc->getById($idtvc);

        if ( $args->peso )
        {
            $tvc->peso = $args->pesoUpd;
        }
        $tvc->data = $args->dataUpd;
        $tvc->descricao = $args->descricaoUpd;

        try
        {
            $info = new MLabel('Avaliação atualizada.','blue',TRUE);
            $tvc->save();
        }
        catch ( Exception $e )
        {
            $info = new MLabel('Erro:'.$e->getMessage(),'red',TRUE);
        }

        $turma = $this->manager->getBusiness('ensino','turma');
        $turma->getByIdTurma($args->selTurmas);
        $form = $this->ui->getForm('ensino','frmTvc',$turma,'minhasturmas');
        $listTvc = $this->ui->getGrid('ensino','gridTvc',$turma);

        $controls = array(
            $info,
            $form,
            $listTvc
        );

        $updates = array(
```

```
        'divInfo',
        'divFrmTvc',
        'divListTvc'
    );
    $this->manager->ajax->setResponseControls($controls,$updates);
}

public function doDeleteTvc($args)
{
    $idtvc = $args->idtvcUpd;
    $idturma = $args->selTurmas;
    $tvc = $this->manager->getBusiness('ensino','tvc');
    $tvc->getById($idtvc);
    try
    {
        $info = new MLabel('Avaliação excluída.','blue',TRUE);
        $tvc->delete();
    }
    catch ( Exception $e )
    {
        $info = new MLabel('Erro:'.$e->getMessage(),'red',TRUE);
    }
    $turma = $this->manager->getBusiness('ensino','turma');
    $turma->getByIdTurma($args->selTurmas);
    $form = $this->ui->getForm('ensino','frmTvc',$turma,'minhasturmas');
    $listTvc = $this->ui->getGrid('ensino','gridTvc',$turma);
    $matriculados = $this->getMatriculados($turma);

    $controls = array(
        $info,
        $form,
        $listTvc,
        $matriculados
    );

    $updates = array(
        'divInfo',
        'divFrmTvc',
        'divListTvc',
        'divMatriculados'
    );
    $this->manager->ajax->setResponseControls($controls,$updates);
}

public function doCancelUpdateTvc($args)
{
    $idturma = $args->selTurmas;
    $turma = $this->manager->getBusiness('ensino','turma');
    $turma->getByIdTurma($args->selTurmas);
    $form = $this->ui->getForm('ensino','frmTvc',$turma,'minhasturmas');
    $info = new MLabel('Atualização da avaliação cancelada.','blue',TRUE);
    $controls = array(
        $info,
        $form
    );

    $updates = array(
        'divInfo',
        'divFrmTvc'
    );
    $this->manager->ajax->setResponseControls($controls,$updates);
}

public function getMatriculados($turma)
{
    $gridMatriculados = $this->ui->getGrid('ensino','gridNotasMatriculados',$turma);
    return $gridMatriculados;
}

public function doFechaTurma($args)
{
    //testar se é possível fechar a turma
    $idturma = $args->selTurmas;
```

```
        $objMatricula = $this->manager->getBusiness('ensino','matricula');
        $objNota = $this->manager->getBusiness('ensino','nota');
        $objTurma = $this->manager->GetBusiness('ensino','turma');
       $objTurma->GetByIdTurma($idturma);

        //o metodo abaixo verifica se ainda existem alunos sem notas lançadas na turma.
        $verificaNotas = $objNota->verificaFechamentoTurma($idturma);
        if ( ! $verificaNotas->eof() )
        {
            $info = new MLabel('Ainda existem alunos sem notas','red',TRUE);
            $controls = array(
                $info
            );
            $updates = array(
                'divInfo'
            );
        }
        else
        {
            try
            {
                //nota calculada pelo sistema
                $origemNota = '2';
                $repNota = $objNota->listFrequenteSemNota($idturma);
                if ( $repNota->result )
                {
                    foreach ($repNota->result as $rs)
                    {
                        $matricula = $rs[0];
                        $objMatricula->GetByMatriculaTurma($matricula, $idturma);
                   $objMatricula->UpdateNotaConceito(0,60,$objTurma->disciplina,$origemNota);
                    }
                }
                $notas = $objNota->GeraNotaFinalTurma($idturma);
                if($notas)
                {
                    foreach($notas as $notasFinais)
                    {
                        $notasFinais[1]= ereg_replace(",",".",$notasFinais[1]);
                        $objMatricula->GetByMatriculaTurma($notasFinais[0], $idturma);
                        //O método abaixo atualiza a tabela ga_matricula. Os parâmetros sao:
nota e mínimo para aprovação, disciplina e origem da notafinal - 0 DARA, 1 Professor, 2
Sistema
                        $objMatricula->UpdateNotaConceito($notasFinais[1],60,$objTurma-
>disciplina,$origemNota);
                    }
                }
            //atualizar situacao da turma para fechada e fazer o procedimento de
transferencia das notas parciais
                $objTurma->beginTransaction();
                $objTurma->fechar();
                $objTurma->transfereTurmaHistNotas();
                $objTurma->getByIdTurma($idturma); //para atualizar a situacao
                $objTurma->endTransaction();

                $gridHistNotas = $this->ui->getGrid('ensino','gridHistNotasMatriculados',
$objTurma);
                $listTvc = $this->ui->getGrid('ensino','gridTvc',$objTurma);

                $controls = array(
                    new MLabel('Turma fechada com sucesso.','blue',TRUE),
                    new MLabel("Situação: $objTurma->situacao"),
                    new MDiv('divBlank'), //para tirar o formulario de inclusao de notas
                    $gridHistNotas,
                    $listTvc
                );

                $updates = array(
                    'divInfo',
                    'divSituacaoTurma',
                    'divFrmTvc',
                    'divMatriculados',
                    'divListTvc'
```

```
                );
            }
            catch ( Exception $e )
            {
                $info = new MLabel('Erro: '.$e->getMessage(),'red',TRUE);
                $controls = array(
                    $info
                );
                $updates = array(
                    'divInfo'
                );
            }
        }
        $this->manager->ajax->setResponseControls($controls,$updates);
    }
}
?>
```

frmTvc.class

```
<?
class frmTvc extends MForm
{
    public $turma;

    function __construct($turma)
    {
        $this->turma = $turma;
        parent::__construct('Nova avaliação');
        $this->eventHandler();
    }

    function createFields()
    {
        $tabelaGeral = $this->manager->getBusiness('common','tabelageral');
        $query = $tabelaGeral->listByTabela('GA_TIPOCALCULO');

        $fields = array(
            new MSelection('tipocalculo','','Cálculo',$query->result),
            new MTextField('peso','','Peso',4),
            array(
                new MDiv('divLabelPeso'),
                new MDiv('divPeso'),
            ),
            new MCalendarField('data','','Data'),
            new MTextField('descricao','','Descrição')
        );

        $this->setFields($fields);

        if ( $this->turma->tipocalculo )
        {
            $this->setFieldAttr('tipocalculo','visible',FALSE);
            $this->setFieldAttr('divPeso','visible',FALSE);
            if ( $this->turma->tipocalculo == 2 )
            {
                $this->setFieldAttr('peso','visible',FALSE);
            }
        }
        else
        {
            $this->setFieldAttr('peso','visible',FALSE);
            $this->tipocalculo->addEvent('change',':doExibePeso');
        }

        $buttons = array(
            new MButton('btnNovaAvaliacao','Enviar'),
        );

        $this->setButtons($buttons);

        $this->btnNovaAvaliacao->addEvent('click',":doAdicionaTvc");
```

```
    }
}
?>
```

frmTvcEdit.class

```
<?
class frmTvcEdit extends MForm
{
       public $tvc;

       function __construct($tvc)
       {
        $this->tvc = $tvc;
        parent::__construct('Editar avaliação');
       }

    function createFields()
    {
        $fields = array(
            new MTextField('pesoUpd',$this->tvc->peso,'Peso',4),
            new MCalendarField('dataUpd',$this->tvc->data,'Data'),
            new MTextField('descricaoUpd',$this->tvc->descricao,'Descrição'),
            new MHiddenField('idtvcUpd',$this->tvc->idtvc)
        );

        $this->setFields($fields);

        $this->tvc->retrieveAssociation('turma');

        if ( $this->tvc->turma->tipocalculo == 2 )
        {
            $this->setFieldAttr('pesoUpd','visible',FALSE);
        }

        $buttons = array(
            new MButton('btnUpdate','Atualizar'),
            new MButton('btnDelete','Excluir'),
            new MButton('btnCancel','Cancelar'),
        );

        $this->setButtons($buttons);
        $this->setButtonAttr('btnDelete','visible',FALSE);

        $this->btnUpdate->addEvent('click',":doUpdateTvc");
        $this->btnCancel->addEvent('click',":doCancelUpdateTvc");

       if (is_null($this->tvc->VerificaLancamentoNotas($this->tvc->idtvc)))
       {
            $this->setButtonAttr('btnDelete','visible',TRUE);
            $this->btnDelete->addEvent('click',":doDeleteTvc");
       }
    }
}
?>
```

gridTvc.class

```
<?
class gridTvc extends MObjectGrid
{
    function __construct($turma)
    {
        $MIOLO = Miolo::getInstance();

        if ( $turma->tipocalculo == 3 )
        {
            $label = "Peso (%)";
        }
        else
        {
```

```
            $label = "Valor";
        }

        if ( $turma->situacao == 'Fechada' )
        {
            $exibirBotaoEditar = FALSE;
        }
        else
        {
            $exibirBotaoEditar = TRUE;
        }

        //imagem para o botão
        $ui = $MIOLO->getUi();
        $img = $ui->getImage( '', 'button_edit.png' );

        $columns = array(
            new MObjectGridColumn('idtvc','Id','left',false,0,FALSE),
            new MObjectGridControl(new MImageButton('btnEdit','Editar','',
$img),'idturma','Ação','left',false,'5%',$exibirBotaoEditar),
            new MObjectGridColumn('numtvc','Número','left',false,'20%',TRUE),
            new MObjectGridColumn('peso',$label,'left',false,'25%',TRUE),
            new MObjectGridColumn('data','Data','left',false,'20%',TRUE),
            new MObjectGridColumn('descricao','Descrição','left',false,'30%',TRUE),

        );

        $turma->retrieveAssociation('tvc');

        $href = $MIOLO->getCurrentUrl();
        parent::__construct($turma->tvc,$columns,$href,0);
        $this->setRowMethod('gridTvc','editar');
        $this->setTitle('Avaliações');
        $this->setClose(NULL);
    }

    public function editar($i, $row, $actions, $columns)
    {
        $idtvc = $row[0];
        $idturma = $row[1];
        $columns[1]->control[$i]->setHref(":doEditTvc;$idtvc");
    }
}
?>
```

gridNotasMatriculados.class

```
<?
class gridNotasMatriculados extends MGrid
{
    function __construct($turma)
    {
        $MIOLO = Miolo::getInstance();
        $turma->retrieveAssociation('turmanota');

        $turma->retrieveAssociation('tvc');

        $quantTvc = count($turma->tvc);

        $control = new MTextField($tvc->idtvc,'',$tvc->num);
        $columns[] = new MGridColumn('Idturma','left',true,'10%',FALSE);
        $columns[] = new MGridColumn('Codigos','left',true,'10%',FALSE);
        $columns[] = new MGridColumn('tipocalculo','left',true,'10%',FALSE);
        $columns[] = new MGridColumn('Matrícula','left',true,'10%',TRUE);
        $columns[] = new MGridColumn('Nome','left',true,'40%',TRUE);
        $columns[] = new MGridColumn('Nota Parcial','left',true,'10%',TRUE);
        $columns[] = new MGridColumn('Situação','left',true,'10%',TRUE);
        if ( $quantTvc > 0 )
        {
            $largura = str_replace('.',',',$rest/$quantTvc);
            foreach ( $turma->tvc as $tvc )
            {
```

```
                $control = new MTextField("idtvc"."[".$tvc->idtvc."]"."[%3%]",'','',4);
                $columns[] = new MGridControl($control,'TVC '.$tvc->numtvc,'left',false,
$largura.'%',TRUE);
              //armazenar os codigos dos tvcs adicionados no grid, escolhido fazer dessa
forma pra evitar sucessivos acesso ao banco para recuperar dados do tvc no rowMethod
                $tvcs[] = $tvc->idtvc;
            }
        }

        $query = $turma->listMatriculadosNotas();

        if ( $tvcs )
        {
            $strTvc = implode('_',$tvcs);
        }
        while ( ! $query->eof() )
        {
            $matricula = $query->fields('matricula');
            $nome = $query->fields('nome');
            $nota = $query->fields('nota');
            $situacao = $query->fields('situacao');

            $data[] = array($turma->idturma,$strTvc,$turma->tipocalculo,$matricula,$nome,
$nota,$situacao);
            $query->moveNext();
        }

        $href = $MIOLO->getCurrentUrl();
        parent::__construct($data,$columns,$href,0);
        $this->setTitle('Alunos');
        $this->setClose(NULL);

        $button = new MButton('btnGravaDados','Gravar dados');
        $button->addEvent('click',":doGravaNotas");

        $this->setControls($button);

       $somapeso = $turma->GetTotalPesoTurma();
        if ( count($tvcs) >= 3 && $somapeso[0][0] >= 100 )
        {
            $objNota = $MIOLO->getBusiness('ensino','nota');

            $verificaNotas = $objNota->verificaFechamentoTurma($turma->idturma);
            if ( $verificaNotas->eof() )
            {
                $button = new MButton('btnFechaTurma','Fechar Turma');
                $button->addEvent('click',":doFechaTurma");
                $this->setControls($button);
            }
        }

        if ( $turma->possuiNotasParciaisLancadas() )
        {
            //so executa o row method caso existam notas lançadas para a turma.
            $this->setRowMethod('gridNotasMatriculados','notas');
        }
    }

    public function notas($i, $row, $actions, $columns)
    {
        $MIOLO = Miolo::getInstance();
        $idturma = $row[0];
        $codigos = $row[1]; //codigos dos tvcs
        $tipocalculo = $row[2];
        $matricula = $row[3];

        //$nome = $row[4]; //nao usa
        //notaparcial = $row[5]; //nao usa
        //situacao = $row[6];

        //a partir de row[7] as colunas do grid aumentam de acordo com o numero de tvcs da
turma, uma nova coluna para cada tvc, o primeiro seria a coluna 7, o segundo a 8 e assim por
diante
```

```
        if ( $codigos )
        {
            $k = 7; //representa o índice inicial da coluna de tvc's
            $idTvcs = explode('_',$codigos);
            foreach ( $idTvcs as $idTvc )
            {
                $nota = $MIOLO->getBusiness('ensino','nota');
                $nota->getById($matricula,$idturma,$idTvc);
                $columns[$k]->control[$i]->setValue($nota->nota);
                //armazenando as notas para calculo da parcial;
                if ( $nota->nota && $nota != 'Erro' )
                {
                    $notas[$idTvc] = $nota->nota;
                }
                $k++;
            }
            if ( $notas )
            {
                $notaParcial = $nota->getNotaParcial($notas,$tipocalculo);
                $columns[5]->control[$i]->setValue($notaParcial);
            }
            unset($notas);
        }
    }
}
?>
```

gridHistNotasMatriculados.class

```
<?
class gridHistNotasMatriculados extends MGrid
{
    function __construct($turma)
    {
        $MIOLO = Miolo::getInstance();
        $rest = 30;

        $turma->retrieveAssociation('tvc');

        $quantTvc = count($turma->tvc);

        $control = new MTextField($tvc->idtvc,'',$tvc->num);
        $columns[] = new MGridColumn('Idturma','left',true,'10%',FALSE);
        $columns[] = new MGridColumn('Codigos','left',true,'10%',FALSE);
        $columns[] = new MGridColumn('Matrícula','left',true,'10%',TRUE);
        $columns[] = new MGridColumn('Nome','left',true,'40%',TRUE);
        $columns[] = new MGridColumn('Nota Final','left',true,'10%',TRUE);
        $columns[] = new MGridColumn('Situação','left',true,'10%',TRUE);
        if ( $quantTvc > 0 )
        {
            $largura = str_replace('.',',',$rest/$quantTvc);
            foreach ( $turma->tvc as $tvc )
            {
                $columns[] = new MGridColumn('Avaliação '.$tvc->numtvc,'left',false,
$largura.'%',TRUE);

                //armazenar os codigos dos tvcs adicionados no grid, escolhido fazer dessa
forma pra evitar sucessivos acesso ao banco para recuperar dados do tvc no rowMethod
                $tvcs[] = $tvc->idtvc;
            }
        }

        $query = $turma->listMatriculadosNotas();

        if ( $tvcs )
        {
            $strTvc = implode('_',$tvcs);
        }
        while ( ! $query->eof() )
        {
            $matricula = $query->fields('matricula');
            $nome = $query->fields('nome');
```

```
            $nota = $query->fields('nota');
            $situacao = $query->fields('situacao');

            $data[] = array($turma->idturma,$strTvc,$matricula,$nome,$nota,$situacao);
            $query->moveNext();
        }

        $href = $MIOLO->getCurrentUrl();
        parent::__construct($data,$columns,$href,0);
        $this->setTitle('Alunos');
        $this->setClose(NULL);
        $this->setRowMethod('gridHistNotasMatriculados','notas');
    }

    public function notas($i, $row, $actions, $columns)
    {
        $MIOLO = Miolo::getInstance();
        $idturma = $row[0];
        $codigos = $row[1]; //codigos dos tvcs
        $matricula = $row[2];

        //$nome = $row[3]; //nao usa
        //notaparcial = $row[4]; //nao usa
        //situacao = $row[5];

        //a partir de row[6] as colunas do grid aumentam de acordo com o numero de tvcs da
turma, uma nova coluna para cada tvc, o primeiro seria a coluna 6, o segundo a 7 e assim por
diante
        if ( $codigos )
        {
            $k = 6; //representa o índice inicial da coluna de tvc's
            $idTvcs = explode('_',$codigos);
            foreach ( $idTvcs as $idTvc )
            {
                $nota = $MIOLO->getBusiness('ensino','histnota');
                $nota->getByMatriculaTurmaTvc($matricula,$idturma,$idTvc);
                $columns[$k]->control[$i]->setValue($nota->nota);
                $k++;
            }
        }
    }
}
?>
```

ANEXO 1 – Árvore de controles do Miolo 2.5

Apresentação da árvore de controles do Miolo 2.5, organizada de forma hierárquica.

```
+--MStyle
+--MAttributes
+--MdragDropControl
+--+--MDraggable
+--+--MDroppable
+--MComponent
+--+--MControl
+--+--+--MPage
+--+--+--MDiv
+--+--+--+--MHr
+--+--+--+--MCSSBox
+--+--+--+--MBoxTitle
+--+--+--+--MModuleHeader
+--+--+--+--MEditor
+--+--+--+--MSeparator
+--+--+--+--MSpacer
+--+--+--+--MContainerControl
+--+--+--+--+--MBox
+--+--+--+--+--MAccordion
+--+--+--+--+--MContainer
+--+--+--+--+--+--MHContainer
+--+--+--+--+--+--MVContainer
+--+--+--+--+--+--MBasePanel
+--+--+--+--+--+--+--MPanel
+--+--+--+--+--+--+--+--MActionPanel
+--+--+--+--+--+--MBaseGroup
+--+--+--+--+--+--+--MRadioButtonGroup
+--+--+--+--+--+--+--MCheckBoxGroup
+--+--+--+--+--+--+--MLinkButtonGroup
+--+--+--+--+--+--+--MToolBar
+--+--+--+--+--MTabContainer
+--+--+--+--+--MBaseGrid
+--+--+--+--+--+--MGrid
+--+--+--+--+--+--+--MDataGrid
+--+--+--+--+--+--+--+--MDataGrid2
+--+--+--+--+--+--+--MLookupGrid
+--+--+--+--+--+--+--+--MLookupObjectGrid
+--+--+--+--+--+--+--+--MLookupQueryGrid
+--+--+--+--+--+--+--MObjectGrid
+--+--+--+--+--+--+--MPDFReport
+--+--+--+--+--+--+--MGridAjax
+--+--+--+--+--MBaseForm
+--+--+--+--+--+--MForm
+--+--+--+--+--+--+--MFormAjax
+--+--+--+--+--+--+--MCompoundForm
+--+--+--+--+--+--+--MCSSForm
+--+--+--+--+--+--+--MCSSPForm
+--+--+--+--+--+--+--MIndexedForm
+--+--+--+--+--+--+--MTabbedForm
+--+--+--+--+--MTheme
+--+--+--+--+--MThemeElement
+--+--+--+--+--MThemeBox
+--+--+--MTabbedFormPage
+--+--+--MTableCell
+--+--+--MTableRow
+--+--+--MTable
+--+--+--+--MSimpleTable
+--+--+--+--+--MTableRaw
+--+--+--+--+--+--MTableXML
+--+--+--MContent
+--+--+--+--MFileContent
+--+--+--MContentHeader
+--+--+--MAreaContainer
+--+--+--MIFrame
```

```
+--+--+--MImageFormLabel
+--+--+--MGridColumn
+--+--+--+--MGridHyperlink
+--+--+--+--+--MDataGridHyperlink
+--+--+--+--+--MObjectGridHyperlink
+--+--+--+--MGridControl
+--+--+--+--+--MDataGridControl
+--+--+--+--+--MObjectGridControl
+--+--+--+--+--MPDFReportControl
+--+--+--+--MDataGridColumn
+--+--+--+--MObjectGridColumn
+--+--+--+--MPDFReportColumn
+--+--+--MGridAction
+--+--+--+--MGridActionIcon
+--+--+--+--MGridActionText
+--+--+--+--MGridActionSelect
+--+--+--+--MDataGridAction
+--+--+--+--MObjectGridAction
+--+--+--MGridFilter
+--+--+--+--MGridFilterText
+--+--+--+--MGridFilterSelection
+--+--+--+--MGridFilterControl
+--+--+--MGridNavigator
+--+--+--MJavascriptCode
+--+--+--MFormControl
+--+--+--+--MInputControl
+--+--+--+--+--MActionControl
+--+--+--+--+--+--MButton
+--+--+--+--+--+--+--MButtonAjax
+--+--+--+--+--+--+--MInputButton
+--+--+--+--+--+--+--+--MButtonFind
+--+--+--+--+--+--+--MButtonImage
+--+--+--+--+--+--+--MButtonWindow
+--+--+--+--+--+--+--MDropDownButton
+--+--+--+--+--+--MLink
+--+--+--+--+--+--+--MActionHyperLink
+--+--+--+--+--+--+--MLinkBack
+--+--+--+--+--+--+--MOpenWindow
+--+--+--+--+--+--+--MGridHeaderLink
+--+--+--+--+--+--+--MLinkButton
+--+--+--+--+--+--+--+--MButtonClose
+--+--+--+--+--+--+--+--MButtonHelp
+--+--+--+--+--+--+--+--MButtonMinimize
+--+--+--+--+--+--+--MImageLink
+--+--+--+--+--+--+--+--MImageLinkLabel
+--+--+--+--+--+--+--+--MImageButton
+--+--+--+--+--+--+--+--+--MImageButtonLabel
+--+--+--+--+--MTextField
+--+--+--+--+--+--MCalendarField
+--+--+--+--+--+--MCurrencyField
+--+--+--+--+--+--MPasswordField
+--+--+--+--+--+--MFileField
+--+--+--+--+--+--MHiddenField
+--+--+--+--+--+--MMultiLineField
+--+--+--+--+--+--+--MmultiTextField3
+--+--+--+--+--+--+--+--MmultiTextField4
+--+--+--+--+--+--MmultiTextField2
+--+--+--+--+--+--MLookupField
+--+--+--+--+--+--+--MLookupTextField
+--+--+--+--+--+--+--MLookupFieldValue
+--+--+--+--+--MChoiceControl
+--+--+--+--+--+--MCheckBox
+--+--+--+--+--+--MRadioButton
+--+--+--+--+--MListControl
+--+--+--+--+--+--MSelection
+--+--+--+--+--+--+--MComboBox
+--+--+--+--+--+--MMultiSelection
+--+--+--+--+--+--MMultiSelectionField
+--+--+--+--+--+--MOrderedList
+--+--+--+--+--+--MUnOrderedList
+--+--+--+--+--MOption
+--+--+--+--+--MInputGridColumn
+--+--+--+--+--MInputGrid
+--+--+--+--+--MPrompt
```

```
+--+--+--+--+--MIndexedControl
+--+--+--+--MOutputControl
+--+--+--+--+--MSpan
+--+--+--+--+--+--MHint
+--+--+--+--+--MBaseLabel
+--+--+--+--+--+--MPageComment
+--+--+--+--+--+--MSeparator
+--+--+--+--+--+--MLabel
+--+--+--+--+--+--+--MRawText
+--+--+--+--+--+--MFieldLabel
+--+--+--+--+--+--MTextHeader
+--+--+--+--+--+--MText
+--+--+--+--+--+--+--MTextLabel
+--+--+--+--+--+--+--MGDText
+--+--+--+--MImage
+--+--+--+--+--MImageLabel
+--+--+--+--MDragDrop
+--+--+--+--MHtmlForm
+--+--+--+--MTextTable
+--+--+--+--MtoolBarButton
+--+--+--+--MTreeMenu
+--+--+--MValidator
+--+--+--MWindow
+--+--+--MStatusBar
+--+--+--MOptionGroup
+--+--+--MOptionListItem
+--+--+--MOptionList
+--+--+--+--MDhtmlMenu2
+--+--+--+--MMenu
+--+--+--+--+--MNavigationBar
```