UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

AJAX NA CONSTRUÇÃO DE UMA APLICAÇÃO WEB PARA MONITORAMENTO DE AMBIENTES EM PLANTAS 2D

MARCUS VINÍCIUS SILVA GOIS

BLUMENAU
2006

2006/1-30

MARCUS VINÍCIUS SILVA GOIS

AJAX NA CONSTRUÇÃO DE UMA APLICAÇÃO WEB PARA MONITORAMENTO DE AMBIENTES EM PLANTAS 2D

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Paulo César Rodacki Gomes, Doutor - Orientador

BLUMENAU
2006

2006/1-30

AJAX NA CONSTRUÇÃO DE UMA APLICAÇÃO WEB PARA MONITORAMENTO DE AMBIENTES EM PLANTAS 2D

Por

MARCUS VINÍCIUS SILVA GOIS

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Paulo César Rodacki Gomes, Doutor – Orientador, FURB
Membro:	Prof. Maurício Capobianco Lopes, Mestre – FURB
Membro:	Prof. Marcel Hugo, Mestre – FURB

Blumenau, 11 de julho de 2006

Dedico este trabalho aos meus pais que sempre me serviram de modelo a ser seguido.

AGRADECIMENTOS

Agradeço acima de tudo aos meus pais Ademir e Ziná, que mesmo estando distantes sempre estiveram presentes em todos os meus momentos.

Aos meus amigos pela compreensão e companheirismo.

Ao meu orientador, Paulo César Rodacki Gomes, por ter acreditado na conclusão deste trabalho.

“A alegria está na luta, na tentativa, no sofrimento envolvido. Não na vitória propriamente dita.”

Mahatma Gandhi

RESUMO

Este trabalho apresenta a especificação e desenvolvimento de uma aplicação *web* para monitoramento gráfico de ambientes físicos para controle de acesso e segurança. Monitorar ambientes de forma gráfica se torna um desafio para aplicativos *web-based* devido a necessidade de utilização de interfaces com muitos recursos de interação. As técnicas apresentadas pelo Ajax são usadas neste trabalho juntamente com a linguagem SVG para possibilitar que as páginas *web*, normalmente estáticas, tenham uma interatividade comparável à de aplicativos *desktop*.

Palavras-chave: Controle de acesso. Ajax. SVG.

ABSTRACT

This work presents the specification and development of a web application for graphical monitor physical environments for access control and security. To monitor environments of graphical form becomes a challenge for web-based applications due to necessity of use interfaces with many interaction resources. The techniques presented by Ajax are used in this work together with language SVG to make possible that the pages web, normally static, have a comparable interactivity to the one of desktop applications.

Key words: Access control. Ajax. SVG.

LISTA DE ILUSTRAÇÕES

Figura 1 - O modelo tradicional de aplicações web comparado com o modelo do Ajax 19
Figura 2 - O padrão de interação síncrono das aplicações web tradicionais e o padrão assíncrono do Ajax 20
Quadro 1 - Exemplo de documento XHTML 22
Quadro 2 - Exemplo de documento CSS 23
Quadro 3 - Função Javascript para obtenção do objeto XMLHttpRequest 27
Quadro 4 - Função para requisitar dados de forma assíncrona ao servidor 28
Quadro 5 - Função que recebe o resultado da requisição de forma assíncrona do servidor 28
Quadro 6 - Exemplo de documento SVG 30
Figura 3 - Imagem criada por documento SVG 30
Quadro 7 - XHTML e SVG integrados 32
Figura 4 - Documento XHTML com SVG exibido em um navegador 32
Figura 5 - Arquitetura de uma aplicação com Servlets 35
Figura 6 - Arquitetura do GWT 39
Figura 7 - Diagrama de casos de uso 43
Figura 8 - Diagrama de classes da estrutura do mapa 48
Figura 9 - Diagrama de classes de identificação de pessoas 49
Figura 10 - Diagrama de classes de informações históricas 50
Figura 11 - Diagrama de classes de interface do aplicativo com os dispositivos 51
Figura 12 – Diagrama de seqüência de recebimento e notificação do evento 53
Figura 13 - Diagrama de seqüência de envio de comandos ao dispositivo 54
Figura 14 - Diagrama de seqüência de consulta das pessoas presentes 55
Figura 15 - Diagrama de seqüência de consulta de eventos ocorridos 56
Figura 16 - Tela principal do sistema 59
Figura 17 - Notificação de evento de acesso negado 60
Figura 18 - Zoom no dispositivo gerador do ultimo evento 61
Figura 19 - Dispositivo do mapa selecionado 61
Figura 20 - Menu sensível ao contexto do dispositivo 62
Figura 21 - Consulta do histórico de eventos 63
Figura 22 - Menu sensível ao contexto do local físico 64
Figura 23 - Consulta de pessoas presentes 64

Quadro 8 - Código para criação do layout da tela principal da aplicação 66

LISTA DE SIGLAS

AJAX – Asynchronous Javascript and XML

CGI – Common Gateway Interface

CSS – Cascading Style Sheets

DOM – Document Object Model

ECMA – European Computer Manufacturers Association

HTML - Hypertext Markup Language

HTTP – Hypertext Transfer Protocol

IIS – Internet Information Services

IDE – Integrated Development Environment

SGML – Standard Generalized Markup Language

SVG – Scalable Vector Graphics

RPC – Remote Procedure Call

VO – Value Object

XHTML – Extensible Hypertext Markup Language

XML – Extensible Markup Language

XSL – Extensible Style Language

WTP – Web Tools Plataform

W3C – World Wide Web Consortium

SUMÁRIO

1 INTRODUÇÃO 13
1.1 OBJETIVOS DO TRABALHO 14
1.2 ESTRUTURA DO TRABALHO 15
2 FUNDAMENTAÇÃO TEÓRICA 16
2.1 CONTROLE DE ACESSO E SEGURANCA 16
2.2 AJAX 17
2.2.1 XHTML 21
2.2.2 *Cascading Style Sheet* 22
2.2.3 Document Object Model 24
2.2.4 Javascript 25
2.2.4.1 O objeto XMLHttpRequest 26
2.3 SCALABLE VECTOR GRAPHICS 29
2.3.1 O SVG e a Web 31
2.4 XML 33
2.5 SERVLETS JAVA 34
2.5.1 O Servlet Container Tomcat 36
2.6 GOOGLE WEB TOOLKIT 37
3 DESENVOLVIMENTO DO TRABALHO 41
3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO 41
3.2 ESPECIFICAÇÃO 42
3.2.1 Casos de uso 43
3.2.1.1 Caso de uso: Gerar eventos 44
3.2.1.2 Caso de uso: Enviar comandos a dispositivos 44
3.2.1.3 Caso de uso: Consultar eventos ocorridos 46
3.2.1.4 Caso de uso: Consultar pessoas presentes 46
3.2.2 Diagramas de classe 46
3.2.2.1 Value objects (VOs) da estrutura do mapa 47
3.2.2.2 VOs de Identificação das pessoas 48
3.2.2.3 VOs de tratamento de informações históricas 49
3.2.2.4 Classes de interface do aplicativo com os dispositivos 50
3.2.3 Diagramas de seqüência 52

3.2.3.1 Recebimento e notificação do evento 52
3.2.3.2 Envio do comando ao dispositivo 53
3.2.3.3 Consultar pessoas presentes 54
3.2.3.4 Consultar eventos ocorridos 55
3.3 IMPLEMENTAÇÃO 56
3.3.1 Técnicas e ferramentas utilizadas 57
3.3.1.1 Plataforma Eclipse 57
3.3.1.1.1 Eclipse Web Tools Plataform (WTP) 57
3.3.1.1.2 SVG Eclipse Plugin 58
3.3.1.2 Navegador Firefox 1.5 58
3.3.2 Operacionalidade da implementação 59
3.4 RESULTADOS E DISCUSSÃO 65
4 CONCLUSÕES 68
4.1 EXTENSÕES 68
REFERÊNCIAS BIBLIOGRÁFICAS 70

1 INTRODUÇÃO

"A mesma simplicidade que permitiu a rápida proliferação da *web* também criou uma separação entre a experiência de usuário que poderia ser provida com interfaces *web* e com aplicações desktop." (GARRETT, 2005, tradução nossa).

Segundo McLellan (2005), um dos inconvenientes clássicos de desenvolver aplicações com interfaces *web* é que, uma vez que a página tenha sido baixada para o cliente, a conexão com o servidor é cortada. Qualquer interação numa interface dinâmica envolve o completo envio da página para o servidor para que este possa reconstruí-la, um processo que tende a tornar as aplicações *web* deselegantes e não responsivas.

Este trabalho propõe utilizar as técnicas do *Asynchronous Javascript and XML* (Ajax) e a linguagem *Scalable Vector Graphics* (SVG) para tentar aproximar a interatividade de uma aplicação *web* da interatividade de uma aplicação desktop.

O termo Ajax é usado para descrever um conjunto de tecnologias que permite prover uma navegação mais natural aos usuários de aplicativos *web*. Teare (2005) diz que, antes do Ajax, os *sites* impunham o paradigma submeter/esperar/exibir aos usuários. O Ajax possui a habilidade de se comunicar com o servidor de forma assíncrona, deixando o usuário livre da espera pela resposta do servidor. Garrett (2005, tradução nossa) levanta ainda a seguinte questão: "Enquanto o servidor está fazendo as suas coisas, o que o usuário está fazendo? Certamente esperando. E, a cada passo da tarefa, o usuário espera mais um pouco".

De acordo com Teare (2005), em uma página desenvolvida utilizando Ajax, quando o usuário clica em um botão, por exemplo, a interface é atualizada imediatamente e uma requisição assíncrona é enviada para o servidor, para executar a atualização de uma tabela de banco de dados. Quando a requisição retorna, somente a parte da página influenciada por esta requisição é atualizada dinamicamente, sem ter que recarregar toda página. Assim, o usuário

pode nem perceber que navegador está se comunicando com o servidor.

Este trabalho apresenta o desenvolvimento de um aplicativo *web* para monitoramento gráfico de ambientes em plantas 2D, que podem ser plantas-baixas, mapas, hierarquias de setores ou qualquer outro tipo de estrutura que se deseje representar. O monitoramento dá-se através de dispositivos instalados nestes ambientes. Estes dispositivos podem ser leitoras de crachá, catracas, sensores de presença, alarmes de incêndio, identificadores de íris, impressões digitais ou padrão vascular, entre outros. Cada um destes dispositivos dispara um tipo específico de evento em resposta a algum acontecimento, como por exemplo, a leitura de um crachá ou a detecção de uma intrusão.

A função do aplicativo monitorador de ambientes é permitir visualizar, em tempo real, informações sobre os eventos gerados, seus dispositivos geradores e a localização destes dispositivos na planta do ambiente monitorado.

O Ajax será utilizado no desenvolvimento deste aplicativo para que se viabilize o monitoramento em tempo real dos ambientes. Com o Ajax será possível que o servidor notifique o cliente sobre o evento gerado imediatamente após o acontecimento deste. Sem o Ajax, o cliente teria que ficar “perguntando” de tempo em tempo para o servidor se aconteceu algum evento. Além disso, as técnicas do Ajax permitem que ao detectar um evento, o navegador atualize somente a parte da página necessária para exibir as informações sobre o evento gerado.

O SVG será utilizado para permitir a utilização de desenhos gráficos vetoriais em páginas *web*. É através da linguagem SVG que será desenhado o ambiente monitorado.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver uma aplicação *web* para monitoramento em

tempo real de ambientes físicos utilizando as técnicas do Ajax e a linguagem SVG.

Os objetivos específicos deste trabalho são:

a) exibir uma visualização gráfica da planta-baixa, mapa, terreno ou outra área qualquer na qual estão instalados dispositivos de controle de acesso;
b) disponibilizar formas de interação típicas de aplicativos *desktop*, tais como *drag-and-drop* e menus sensíveis ao contexto, utilizando-se das técnicas do Ajax;
c) utilizar requisições assíncronas através de *callback* do servidor, para que a aplicação esteja sempre disponível às ações do usuário, mas possa notificá-lo imediatamente após a ocorrência de um evento.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está dividido em quatro capítulos que são referidos a seguir:

No primeiro capitulo, é apresentada, objetivamente, uma introdução ao trabalho, suas motivações, seus objetivos e sua estrutura.

No segundo capítulo, é fornecida uma breve explanação sobre alguns fundamentos que servem de base para este trabalho como controle de acesso e segurança, Ajax e SVG, com foco no seu funcionamento independente da utilização neste trabalho.

No terceiro capítulo, é tratado o desenvolvimento do trabalho, mostrando sua especificação com diagramas de caso de uso, diagramas de classe e de seqüência. É também mostrado um caso de uso da aplicação desenvolvida. O quarto capítulo, apresenta as conclusões do trabalho, suas limitações e possíveis extensões para o mesmo.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo aborda, de forma resumida, conceitos técnicos sobre as principais áreas e tecnologias envolvidas no desenvolvimento deste trabalho como Controle de Acesso e Segurança, Ajax, SVG, *Servlets* Java e o Google *Web Toolkit*. Também serão detalhadas algumas das tecnologias base, utilizadas pelo Ajax e SVG.

2.1 CONTROLE DE ACESSO E SEGURANCA

De acordo com Honey (2000, p. 5), pode-se dizer que o propósito de um sistema de controle de acesso é restringir o acesso de pessoas não autorizadas a um local e facilitar a entrada das pessoas autorizadas. Importantes aspectos deste controle são: determinar quais pessoas poderão ir a determinados locais e em quais momentos, supervisionar tentativas válidas e inválidas de acesso, monitorar os eventos/estados de sensores e equipamentos e a tomada de ações em resposta a eventos ou mudanças de estados em equipamentos.

O controle de acesso se inicia com a identificação da pessoa e do local ao qual ela deseja ter acesso. A partir destas informações, é verificado na lista de permissões desta pessoa, se ela pode acessar o local no horário atual. Em caso positivo, é concedido o acesso ao local requisitado. Existem ainda informações adicionais de controle de acesso como permanência máxima no local, quantidade máxima de acesso por período determinado, sentido ou direção de entrada e saída, procedimentos especiais a serem executados no momento da entrada, saída ou em intervalos de tempo determinados entre outras.

Nos sistemas de controle de acesso, a identificação da pessoa é normalmente (mas não necessariamente) feita por dispositivos eletrônicos. Exemplos destes dispositivos são leitoras de código de barra e de *smartcards*, teclados, leitoras biométricas como identificadoras de

digitais e íris, dispositivos de identificação por voz, padrão vascular e vários outros. A função e o desafio de todos estes dispositivos é identificar de forma simples e precisa as pessoas. A liberação ou bloqueio em sistemas de controle de acesso é geralmente feito por equipamentos dotados de um dispositivo mecânico, controlado por um circuito eletrônico baseado em um microprocessador. Estes equipamentos normalmente são catracas, torniquetes, cancelas, portas eclusas, ou fechaduras eletromagnéticas.

O sistema de controle de acesso deve gerenciar e monitorar o funcionamento dos dispositivos de forma a coordenar suas ações e estados para que reflitam as configurações de permissão de acesso desejadas, prevendo estados de exceção (como falta de energia e mal-funcionamento) e situações de emergência como incêndios.

Um sistema de controle de acesso e segurança, geralmente possui ainda um módulo de monitoramento, que é responsável por fornecer informações sobre os eventos ocorridos nos dispositivos e seus estados. O módulo de monitoramento pode permitir também tomar decisões em resposta aos eventos ocorridos e interagir com os dispositivos através de comandos enviados a eles.

2.2 AJAX

O Ajax, sigla de *Asynchronous Javascript And XML* permite o desenvolvimento de aplicativos *web* com interface “rica”. Crane e Pascarello (2005, p. 5) definem que “rica” refere-se ao modelo de interação do aplicativo com o usuário. Um modelo rico de interação com o usuário é aquele que suporta uma variedade de formas de entrada e responde intuitivamente e em tempo hábil a estas entradas.

Segundo Garrett (2005), o Ajax não é uma tecnologia. Na realidade são várias tecnologias que juntas se combinam de uma maneira extremamente poderosa. O Ajax

incorpora:

a) *Cascading Style Sheets* (CSS) e XHTML para exibição das informações;

b) *Document Object Model* (DOM) para interagir dinamicamente com as informações apresentadas;

c) XMLHttpRequest para trocar dados de forma assíncrona com o servidor;

d) Javascript como linguagem de *script* para suporte às tecnologias acima.

No Ajax, grande parte da codificação do aplicativo *web* fica no cliente, o que faz com que as aplicações se tornem mais responsivas e suportem uma maior variedade de formas de interação com o usuário como *drag-and-drop*, menus sensíveis ao contexto e atualizações dinâmicas de interfaces.

Ao contrário das tradicionais aplicações *web*, onde toda a lógica da aplicação tendia a estar concentrada no servidor, tratada por linguagens como ASP.Net ou Java Server Pages (JSP), com o Ajax, a maior parte do código da aplicação é executada no cliente (*browser*) através de linguagens de *script* como Javascript. A principal razão para isto é poder controlar atualizações precisas apenas nas partes da página onde o usuário está interagindo e não ter que recarregar a página inteira a cada interação do usuário.

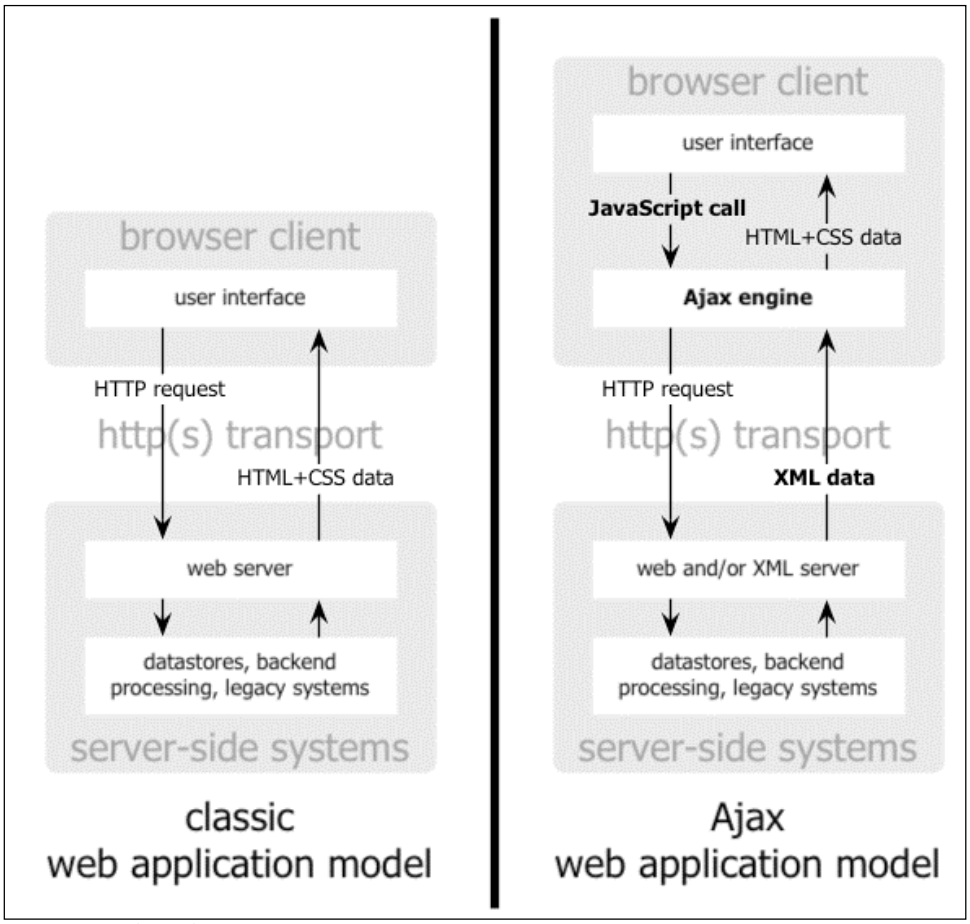
Outra mudança notável nas técnicas do Ajax está no tipo de informação que é enviada pelo servidor. De acordo com Crane e Pascarello (2005, p. 19), no Ajax, o servidor fornece dados e não conteúdo. Em aplicações *web* comuns, é normal que o servidor retorne dados já formatados com *tags Hypertext Markup Language* (HTML) da maneira como devem ser apresentados. Com o Ajax, o servidor fornece apenas os dados, normalmente em formato *Extensible Markup Language* (XML). Toda a camada de apresentação é implementada no navegador através das linguagens de *script*.

No modelo de aplicações *web* clássico, as interações do usuário com a página geram uma requisição *Hypertext Transfer Protocol* (HTTP) do navegador para o servidor, que

recebe, processa e devolve o resultado desta requisição, geralmente em formato HTML. Este modelo síncrono faz com que o usuário e o servidor trabalhem um de cada vez e um aguarde ocioso enquanto o outro trabalha.

Este comportamento era suficiente para os documentos de hipertexto existentes no princípio dos tempos da *web*. Mas segundo Mahemoff (2005), a *web* há muito não é somente utilizada com *sites* que expõem alguma informação. Ela é cada vez mais usada por aplicações completas e complexas, que demandam ricos estilos de interação. E quando as aplicações *desktop* começaram a migrar para a *web*, houve um choque de recursos de usabilidade que não eram suportados pelas páginas.

A Figura 1 mostra o fluxo de interação do navegador *web* com o servidor em uma aplicação *web* clássica em paralelo com uma aplicação Ajax, onde o "Ajax *engine*" é normalmente código Javascript que dita como trafegarão os dados entre navegador e servidor.

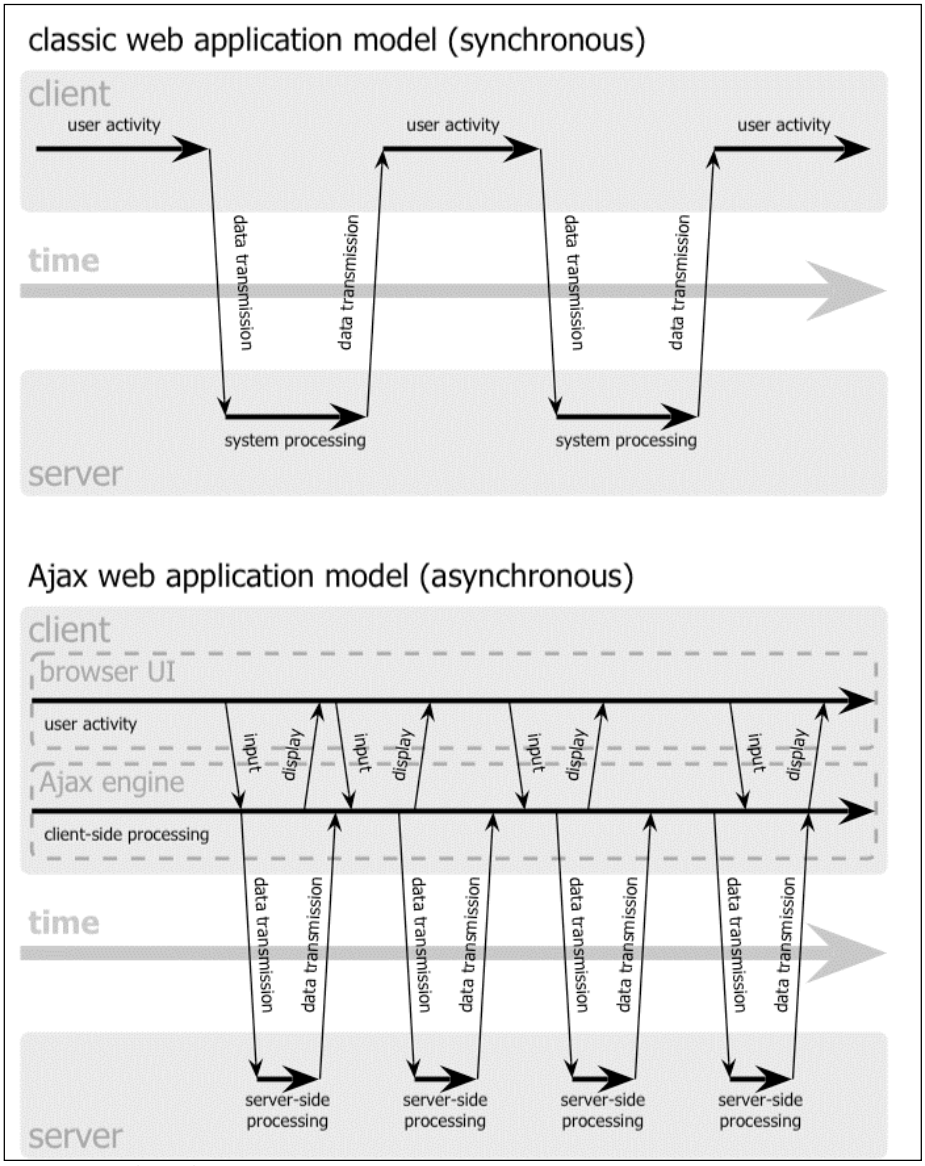


Fonte: Garret (2005).

Figura 1 - O modelo tradicional de aplicações web comparado com o modelo do Ajax

Com o Ajax, as aplicações *web* não precisam mais fazer requisições ao servidor e esperar que ele retorne o que será exibido. Nelas, através do Javascript, o navegador tem o controle do que será enviado ao servidor e o que retornará, e como isto deverá ser exibido na página, isto tudo de forma assíncrona, sem interromper o usuário. Nas aplicações Ajax, o tempo do servidor e do usuário é mais bem aproveitado, permitindo que ambos trabalhem ao mesmo tempo, de forma coordenada, diminuindo os tempos ociosos de ambas as partes.

A Figura 2 mostra a utilização do tempo do usuário e do servidor em uma aplicação *web* clássica comparado com a utilização destes tempos em uma aplicação Ajax.



Fonte: Garret (2005).

Figura 2 - O padrão de interação síncrono das aplicações web tradicionais e o padrão assíncrono do Ajax

Aplicativos que utilizam as técnicas do Ajax tentam tratar a maior parte possível das ações do usuário no próprio cliente, sem que seja necessário requisitar informações para o servidor, e, quando isto é inevitável, é feita uma requisição assíncrona e o servidor retorna apenas de dados (sem formatação) que serão formatados e exibidos pelo cliente.

A utilização do Ajax é recente e ainda não muito difundida pela internet, mas seu uso vem crescendo, principalmente pela influência do Google, que utiliza massivamente o Ajax na construção de aplicativos *web* que desafiam cada vez mais os limites da interatividade das páginas *web*. E sua utilização tende a ser cada vez maior já que suas tecnologias base já são maduras e suportadas por todos os navegadores modernos.

2.2.1 XHTML

A W3C (2002) define o XHTML, ou *Extensible Hypertext Markup Language* como uma reformulação da linguagem de marcação HTML baseada em XML. O XHTML é uma adaptação das *tags* de marcação HTML com regras da XML. O objetivo do XHTML é padronizar a exibição de páginas *web* em diversos dispositivos (televisão, palm, celular, etc.).

O XHTML é baseado no HTML, mas possui sintaxe XML, o que faz com que ele consiga ser interpretado por qualquer dispositivo, independente da plataforma, pois suas marcações possuem sentido semântico para as máquinas. Não existem muitas diferenças entre o HTML e o XHTML. O que faz a maior diferença entre um documento HTML e um XHTML é o bom conhecimento do programador que está construindo os códigos com relação aos padrões de XHTML recomendados pela W3C.

Existe uma tendência de que o XHTML substitua naturalmente o HTML por ser mais simples de ser interpretado que o HTML e seguir padrões de sintaxe mais rígidos. Existem muitos interpretadores para XML e todos eles funcionam com um documento XHTML, já que

este nada mais é do que um documento XML específico. O Quadro 1 exibe um exemplo de documento XHTML.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
<title>Exemplo de página XHTML</title>
</head>
<body>
<h1>Exemplo de XHTML</h1>
<hr />
</body>
</html>
```

Quadro 1 - Exemplo de documento XHTML

Este exemplo descreve uma página XHTML que exibe um título “Exemplo de XHTML”, é fácil notar a semelhança com o HTML já que as *tags* possuem o mesmo nome. A maior diferença está nas declarações iniciais do arquivo XML.

2.2.2 *Cascading Style Sheet*

Cascading Style Sheets ou CSS é uma linguagem de folhas de estilo usada para descrever como um documento deve ser exibido. Segundo a W3schools (2006), o CSS veio para fazer com que o HTML voltasse a ter o seu propósito inicial. As *tags* HTML originalmente deveriam dizer o que são os elementos do documento, como por exemplo, “cabeçalho”, “parágrafo” e “tabela” e não como eles deveriam ser exibidos. Com a popularização da internet, os fabricantes de navegadores começaram a adicionar *tags* proprietárias para formatação dentro do documento HTML.

O crescente número de tags de formatação dentro dos documentos HTML tornou mais trabalhosa a construção de *web sites* e mais difícil a pesquisa de informações relevantes dentro dos documentos HTML.

A criação do CSS retoma a idéia original de que o documento HTML deveria conter apenas informações do que são as partes do documento e o seu conteúdo e a descrição de como este conteúdo deve ser apresentado na tela ou impresso pode ser feita separadamente através dos estilos CSS.

O CSS é utilizando principalmente para descrição do formato de exibição de páginas *web*, mas sua linguagem pode ser usada para formatar qualquer documento XML. É comum ver o CSS sendo usado, por exemplo, para formatação dos documentos SVG. A idéia da formatação através de CSS é que o documento principal contenha somente informação, ou conteúdo e não informações sobre sua apresentação.

O Quadro 2 mostra um exemplo de uma parte de documento CSS que define classes como “mainmenu” e “properties”. Estas classes são associadas a elementos da página XHTML e definem aspectos de apresentação destes elementos como margens, posição, dimensões (largura e altura) e bordas, por exemplo.

```
#mainmenu {
    padding-left: 10px;
}

#properties {
    position: relative;
    display: block;
    float: left;
    width: 100%;
    top: 10px;
    border: 0px;
    left: 0px;
}

#toolBar {
    position: relative;
    display: block;
    height: 30px;
    width: 99.5%;
    vertical-align: middle;
    border-bottom: 1px solid black;
/*  background-color: aqua; */
    padding-left: 2px;
}

#content {
    position: relative;
    width: 99.5%;
    height: 622px;
    padding-left: 2px;
/*  background-color: green; */
    overflow: auto;
}
```

Quadro 2 - Exemplo de documento CSS

Através do CSS é possível ainda controlar aspectos como fontes, cores, alinhamento, fluxo do texto e outras propriedades de formatação. Além destas propriedades estáticas, o CSS também pode definir algumas propriedades dinâmicas como qual estilo deverá ser usado quando o *mouse* estiver sobre o item, ou quando o item receber o foco, por exemplo.

2.2.3 Document Object Model

Document Object Model (DOM) é uma descrição de como documentos XML e HTML podem ser representados em uma estrutura de árvore. O DOM define um conjunto de interfaces de programação orientada a objeto que permitem interpretar estes documentos e realizar operações sobre o seu conteúdo.

Os navegadores *web* implementam várias destas interfaces e permitem sua utilização através do Javascript para manipular elementos de documentos HTML ou XHTML. O DOM foi proposto pela W3C e se tornou uma especificação padrão, independente de linguagem e plataforma, que é implementada por diversas linguagens de programação de diferentes fabricantes.

A representação em forma de árvore requer que o documento inteiro seja lido e armazenado em memória. Por causa disso o DOM é melhor utilizado por aplicações que precisam acessar e manipular os elementos do documento em uma ordem indeterminada e/ou repetidas vezes. Para acessos seqüenciais e para casos onde haverá de somente uma leitura ou gravação o DOM apresenta um custo consideravelmente alto. O modelo de leitura de documentos HTML e XML chamado de *Simple API for XML* (SAX) é vantajoso ao DOM em alguns casos em termos de velocidade e consumo de memória.

2.2.4 Javascript

Segundo Duffy (2003, p.5), o Javascript surgiu com o navegador Netscape 2.0 em 1995 com a intenção de integrar páginas HTML com *applets* Java. Os desenvolvedores rapidamente descobriram seu potencial e passaram a usar o Javascript para adicionar interatividade às páginas *web*.

O Javascript é uma linguagem de programação de *scripts*. Linguagens de programação de *scripts* têm como característica comum não precisar compilar seus códigos fonte para que eles possam ser executados. Em 1996, com o objetivo de diminuir a incompatibilidade entre as implementações, o Javascript tornou-se uma linguagem padronizada pela *European Computer Manufacturers Association* (ECMA).

Por não precisar compilar seus códigos fontes, os *scripts* Javascript somente são interpretados no cliente, ou seja, nos navegadores que irão executá-lo. Isto faz com que o javascript seja uma linguagem independente de plataforma.

Atualmente o Javascript é amplamente suportado pelos navegadores *web* e está presente na maioria das páginas existentes na *web*. A característica que o torna muito atraente para as páginas *web* é a sua capacidade de adicionar lógica e comportamento dinâmico aos *sites*. E o fato de seus *scripts* serem executados do lado cliente, ou seja, dentro do próprio navegador, o que dá bastante agilidade e proporciona maior capacidade de interação com as ações do usuário.

De acordo com Crane e Pascarello (2006, p. 34) o Javascript é o jogador central do Ajax. É ele quem controla todo o fluxo de uma aplicação Ajax. O Javascript é o responsável por enviar as requisições ao servidor, receber as respostas de forma assíncrona e determinar como e quando serão exibidos os resultados.

2.2.4.1 O objeto XMLHttpRequest

O objeto XMLHttpRequest é uma extensão não padronizada do Javascript, suportada pela maioria dos navegadores *web* atuais. Este objeto foi projetado exclusivamente para fazer requisições de forma assíncrona ao servidor. Por não ser padronizado, existem diferentes implementações deste objeto, mas todas compartilham uma interface comum permitindo que a única diferença em sua utilização fique na forma de criação (ou instanciação) do objeto. Existe ainda uma tendência de que todos os navegadores passem a ter suporte nativo a este objeto, o que não acontece com os navegadores atuais. O Internet Explorer até a versão 6, por exemplo exige a utilização de um componente ActiveX para usar este objeto. Na versão 7 beta do navegador da Microsoft, existe o suporte nativo ao XMLHttpRequest, como já acontece com os navegadores Mozilla e Apple Safari.

Crane e Pascarello (2006, p. 56) dizem que antes do objeto XMLHttpRequest já era possível fazer requisições assíncronas ao servidor através da utilização de IFrames ocultos, mas isto é basicamente um truque com um objeto (o IFrame) que foi originalmente criado para exibir conteúdo visível em páginas *web*. A utilização de IFrames também não é mais recomendada pela W3C e este objeto nem mesmo existe no modo *Strict* (um modo de sintaxe mais rígida) do HTML que é o atualmente recomendado para novas páginas.

O objeto XMLHttpRequest por sua vez foi criado especificamente com a intenção de tratar as requisições assíncronas com o servidor e fornece uma série de facilidades para esta função.

O Quadro 3 mostra um exemplo de uma função Javascript para obtenção do objeto XMLHttpRequest. Esta função tenta instanciar o objeto XMLHttpRequest primeiramente como um objeto ActiveX (padrão do Internet Explorer) e caso não consiga, tenta instanciá-lo como um objeto nativo Javascript (padrão para os demais navegadores *web*). Caso ainda

assim não consiga obter uma instância do objeto, é exibido um alerta para o usuário informando que o navegador não suporta o Ajax.

```
function getHTTPRequest() {
    var tempXMLHttp = false;
    if (window.ActiveXObject) {
        var objectNames = [
                        "MSXML2.XMLHTTP",
                        "Microsoft.XMLHTTP"
        ];
        for(var i=0; i<objectNames.length && !tempXMLHttp; i++) {
            try {
                var requestName = objectNames[i];
                tempXMLHttp = new ActiveXObject(requestName);
            } catch(e) {
            }
        }
    }

    if (!tempXMLHttp && window.XMLHttpRequest) {
        try {
            tempXMLHttp = new XMLHttpRequest();
        } catch(e) {
        }
    }

    if ( !tempXMLHttp ) {
        alert("Seu browser não esta preparado para o ajax");
    }
    return tempXMLHttp;
}
```

Quadro 3 - Função Javascript para obtenção do objeto XMLHttpRequest

Depois de criado um objeto XMLHttpRequest o navegador está pronto para fazer requisições assíncronas para o servidor. O Quadro 4 mostra um exemplo de função que faz uma requisição assíncrona ao servidor utilizando o método POST do protocolo HTTP, mas é possível também utilizar o método GET. A maioria dos navegadores do mercado possui suporte a ambos os métodos de requisição.

```
function sendRequest(url, params) {
    var req = getHTTPRequest();
    if (req) {
        req.onreadystatechange = onReadyStateChange;
        req.open("POST", url, true);
        req.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
        req.send(params);
    }
}
```

Quadro 4 - Função para requisitar dados de forma assíncrona ao servidor

Depois de feita a requisição para o servidor, o navegador não espera que esta requisição retorne antes de devolver o controle para o usuário, que pode continuar navegando enquanto o servidor está fazendo os processamentos necessários para devolver o resultado da requisição. Quando o servidor termina o processamento e envia o resultado da requisição para o navegador, uma função (neste caso a função onReadyStateChange) definida pelo desenvolvedor recebe e faz o tratamento do retorno para exibição dos dados recebidos. Tudo isto acontece sem interromper a operação do usuário.

O Quadro 5 mostra um exemplo de função que recebe o resultado da requisição do servidor de forma assíncrona e o encaminha para uma função que cuidará da correta apresentação dos dados para o usuário.

```
var READY_STATE_UNINITIALIZED = 0;
var READY_STATE_LOADING = 1;
var READY_STATE_LOADED = 2;
var READY_STATE_INTERACTIVE = 3;
var READY_STATE_COMPLETE = 4;

function onReadyStateChange() {
    var ready = req.readyState;
    var data = null;
    if ( ready == READY_STATE_COMPLETE ) {
        data = req.responseText;
    } else {
        data = "loading...[" + ready + "]";
    }
    exibir(data);
}
```

Quadro 5 - Função que recebe o resultado da requisição de forma assíncrona do servidor

2.3 SCALABLE VECTOR GRAPHICS

Scalable Vector Graphics (SVG) é uma especificação de linguagem da *World Wide Web Consortium* (W3C) para descrição de gráficos vetoriais bidimensionais através de XML. Segundo a especificação da W3C (2003), a linguagem SVG permite três tipos de objetos gráficos: formas vetoriais (figuras compostas de linhas e curvas), imagens e texto. Os objetos gráficos podem ser manipulados através de transformações geométricas e estilos. O conjunto de recursos inclui ainda transformações aninhadas, trajetos de corte, máscaras alfa, efeitos de filtro e *templates* (modelos) de objetos.

Desenhos SVG podem ser feitos de forma interativa ou dinâmica. Animações podem ser definidas e disparadas de forma declarativa ou via *script* dando grande poder e flexibilidade para esta linguagem.

Aplicações mais sofisticadas com SVG são possíveis pelo uso de linguagens de *script* suplementares como o Javascript e o SVG DOM, que permitem o acesso completo a todos os seus elementos, atributos e propriedades, além de fornecer vários manipuladores de eventos para que possam ser tratadas ações sobre os objetos gráficos. Através do uso de linguagens de *script* integradas aos documentos SVG, é possível construir aplicações completas que reagem a ações do usuário e se adaptam a novas situações utilizando-se apenas dos elementos definidos pela especificação do SVG.

Existe atualmente uma infinidade de formatos de arquivos vetoriais, muitos deles são proprietários ou de domínio específico e não há nenhum dominante. De acordo com Prescod (2003), a revolução que o SVG pode causar na padronização da manipulação de objetos gráficos pode ser comparada à revolução que o protocolo IP causou aos protocolos de rede. O SVG é bem documentado, fácil de trabalhar e livre de licenças. Espera-se que o SVG torne-se um formato padrão para intercâmbio entre formatos, e existem ainda diversas aplicações que

tem adotado o SVG como formato principal para visualização e manipulação de objetos gráficos.

O Quadro 6 mostra um exemplo de documento SVG que define um gradiente, utilizado no preenchimento de um retângulo e três círculos agrupados e preenchidos com cores sólidas. Pode-se observar que a sintaxe da linguagem SVG é baseada em XML, o que faz com que o seu código possa ser lido por qualquer tradutor XML existente no mercado. Isto permite também a sua integração a outros tipos de documentos XML como páginas XHTML, por exemplo.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink"
version="1.1" baseProfile="full" viewBox="0 0 400 500">
<defs>
<linearGradient id="Gradient01" x1="0" x2="0" y1="0" y2="150" gradientUnits="userSpaceOnUse">
    <stop offset="30%" style="stop-color:black"/>
    <stop offset="70%" style="stop-color:yellow"/>
</linearGradient>
</defs>
<rect x="50" y="10" width="350" height="100" style="fill:url(#Gradient01)" />
<g fill-opacity="0.7" stroke="black" stroke-width="0.1cm">
    <circle cx="6cm" cy="2cm" r="100" fill="red" transform="translate(0,50)" />
    <circle cx="6cm" cy="2cm" r="100" fill="blue" transform="translate(70,150)" />
    <circle cx="6cm" cy="2cm" r="100" fill="green" transform="translate(-70,150)" />
</g>
</svg>
```

Quadro 6 - Exemplo de documento SVG

A Figura 3 mostra a exibição gráfica do documento do Quadro 6.

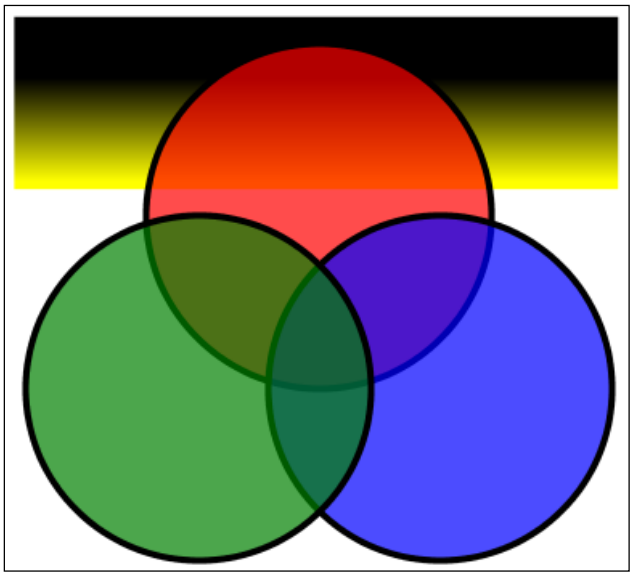


Figura 3 - Imagem criada por documento SVG

2.3.1 O SVG e a Web

O Firefox versão 1.5 foi o primeiro navegador do mercado a oferecer suporte nativo ao SVG (sem a necessidade de *plugins*). A partir desta iniciativa, vários outros navegadores incluíram suporte nativo ao SVG.

A utilização de SVG em conjunto com o HTML ou XHTML cria possibilidades que nunca poderiam ser pensadas utilizando somente HTML, e sua maior utilização certamente causará uma revolução na forma como são as páginas *web* atuais. A linguagem SVG, por ser uma linguagem XML, pode ser combinada com uma página XHTML através de *namespaces* XML permitindo que se alternem ou misturem conteúdos HTML e XML numa mesma página. O resultado disso é o uso do melhor dos dois mundos: o grande poder para manipulação de objetos gráficos do SVG e os recursos de formatação de conteúdo do HTML.

Para exibição de qualquer imagem em páginas HTML, ela tem que ser convertida para formato bitmap (JPEG, GIF, PNG, etc.). Estes formatos tendem a gerar arquivos, grandes e estáticos, sem nenhuma capacidade de interação por parte do usuário.

O Quadro 7 mostra um documento XHTML integrado com um documento SVG. A Figura 4 mostra como fica este documento quando exibido por um navegador *web*. No documento inicialmente são definidos dois *namespaces* para separação dos conteúdos XHTML e SVG dentro do mesmo arquivo XML. A partir da definição dos *namespaces*, o navegador ou outro programa que esteja interpretando o documento, consegue distinguir a qual linguagem se refere cada *tag*.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:svg="http://www.w3.org/2000/svg">
<head>
<title>XHTML e SVG</title>
</head>
<body>
<h1>XHTML integrado com SVG</h1>
<hr />
  <svg:svg viewBox="0 0 400 500">
  <svg:defs>
    <svg:linearGradient id="Gradient01" x1="0" x2="0" y1="0" y2="150"
    gradientUnits="userSpaceOnUse">
      <svg:stop offset="30%" style="stop-color:black"/>
      <svg:stop offset="70%" style="stop-color:yellow"/>
    </svg:linearGradient>
  </svg:defs>
  <svg:rect x="50" y="10" width="350" height="100" style="fill:url(#Gradient01)" />
  <svg:g fill-opacity="0.7" stroke="black" stroke-width="0.1cm">
  <svg:circle cx="6cm" cy="2cm" r="100" fill="red" transform="translate(0,50)" />
  <svg:circle cx="6cm" cy="2cm" r="100" fill="blue" transform="translate(70,150)" />
  <svg:circle cx="6cm" cy="2cm" r="100" fill="green" transform="translate(-70,150)" />
</svg:g>
</svg:svg>
</body>
</html>
```

Quadro 7 - XHTML e SVG integrados

Pode-se observar que o documento XHTML é um documento XML simples, que define dois *namespaces* um para *tags* XHTML e outro para *tags* SVG e através dos prefixos definidos para os *namespaces* o interpretador consegue identificar a qual conjunto de *tags* determinada *tag* pertence.

Figura 4 - Documento XHTML com SVG exibido em um navegador

Com o SVG as páginas poderiam ser cheias de formas arredondadas, gradientes, transparências e outras formas de imagem sem o custo que elas têm em formato *bitmap*. E tudo isto com possibilidade de zoom, rotação, translação e outras transformações sem perda de qualidade e sem recarregar a imagem.

2.4 XML

O XML é uma linguagem de marcações de propósito geral utilizada para definição de linguagens de marcação de propósito específico. Ou seja, o XML é uma maneira de se descrever dados e pode também conter dados.

Segundo Lowe e Wilde (2002, documento eletrônico) o XML foi originalmente criado pela W3C para ser uma variante simplificada da *Standard Generalized Markup Language* (SGML), mas logo a W3C percebeu que ele seria ideal para resolver algumas das limitações do HTML. O XML poderia substituir o conjunto de *tags* predefinidas do HTML por um conjunto de *tags* definidas pelo usuário, somada com informações sobre como estas *tags* deveriam ser exibidas. A linguagem XML deveria seguir os seguintes princípios:

a) separação do conteúdo e da formatação;

b) legibilidade tanto por humanos quanto por máquinas;

c) possibilidade de criação de tags sem limitação;

d) criação de arquivos para validação de estrutura;

e) concentrar-se na estrutura da informação e não na sua aparência.

Segundo Walsh (1998), a especificação do XML é ainda complementada por outras especificações relacionadas como, por exemplo:

a) XML *Pointer Language* (XPointer): descreve como endereçar recursos;

b) XML *Linking Language* (XLink): descreve como dois ou mais recursos se

relacionam;

c) *Extensible Style Language* (XSL): define uma linguagem para criação de folhas de estilo sobre documentos XML.

Como o XML é uma linguagem de marcação de propósito geral, ela também prevê a possibilidade de validação de documentos. Documentos XML podem ser validados em duas categorias: bem formados e válidos. Documentos bem formados são documentos que seguem aos padrões básicos do XML, como estrutura de elementos e atributos, pois um documento que não é bem formado, não é considerado um documento XML. Já um documento válido, é um documento que segue um conjunto mais restrito de *tags*, que pode ser definido de acordo com a finalidade do documento através de seu *namespace*.

A validade de documentos XML específicos pode ser determinada através de linguagens como *Document Type Definition* (DTD) ou XML *Schema*. Estas linguagens possuem uma sintaxe especial para definir parâmetros como nomes de elementos permitidos, seqüência válida de elementos, atributos requeridos, valores de atributos do tipo correto, entre outros. A partir desta definição da estrutura do documento, ele pode ser verificado. E considerado válido ou não para aquela subespecificação da linguagem.

O XML evoluiu e tornou-se rapidamente um padrão para todo tipo de intercâmbio de informações, e também é usado para armazenamento de informações, linguagens de *scripts* e uma série de outras aplicações. Linguagens de programação modernas como Java e .Net tem seu ambiente e linguagens fortemente baseados em arquivos XML.

2.5 SERVLETS JAVA

A Sun Microsystems (2005) define os *servlets* Java como a tecnologia Java para estender e desenvolver servidores *web*. Os *servlets* disponibilizam uma tecnologia

independente de plataforma baseada em componentes para desenvolvimento de aplicações *web*, sem as limitações de desempenho dos programas CGI.

De acordo com Kurniawan (2002), os *servlets* são a base do desenvolvimento de aplicações *web* utilizando a linguagem Java e uma das suas mais importantes tecnologias. Em uma aplicação *web* Java, os *servlets* são responsáveis por receber requisições HTTP de páginas *web*, fazer os processamentos no servidor e devolver os resultados da requisição. A especificação mais atual dos *servlets*, versão 2.4 foi divulgada pela Sun em 2005 e serve como tecnologia base para diversas outras especificações como JSP e Java *Server Faces* (JSF).

A Figura 5 mostra a arquitetura de uma aplicação utilizando Java Servlets, onde o cliente é o navegador instalado na máquina do usuário e o *web container* e banco de dados estão em um ou mais servidores da *web*.

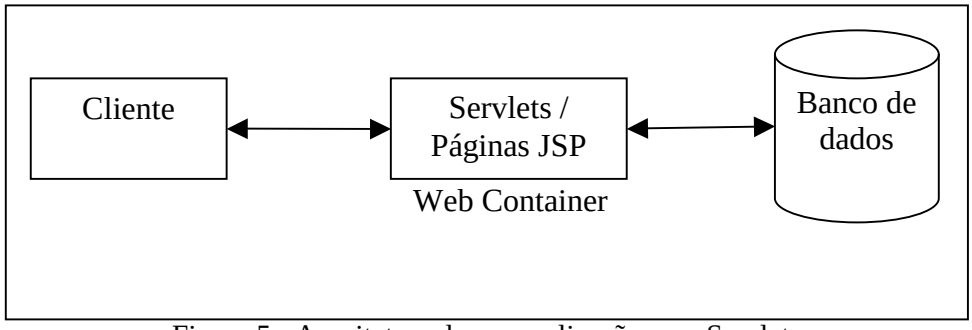


Figura 5 - Arquitetura de uma aplicação com Servlets

Kurniawan (2002) afirma que existem diversas tecnologias para criação de aplicativos *web* como ColdFusion, PHP e ASP.Net, mas os *servlets* oferecem alguns benefícios que não são necessariamente encontrados em outras tecnologias como:

a) desempenho: os *servlets* têm um desempenho superior ao do CGI porque eles não necessitam criar um novo processo para cada requisição do cliente. Todas as requisições podem ser tratadas por um único processo que é iniciado e mantido pelo *web container*. Quando o *servlet* finaliza o processamento da requisição, ele permanece na memória aguardando por novas requisições do cliente;

b) portabilidade: como as outras tecnologias Java, os *servlets* são portáveis. Eles

podem ser movidos para outros sistemas operacionais sem necessidade de nova compilação ou alterações nos fontes;

c) rápido ciclo de desenvolvimento: como uma tecnologia Java, os *servlets* compartilham a rica biblioteca Java, que possui implementações de diversas classes para ajudar a aumentar a produtividade do processo de desenvolvimento;

d) aplicações robustas: por serem aplicações Java, os *servlets* compartilham das vantagens de se executar em uma máquina virtual como por exemplo o *garbage collector*, livrando o desenvolvedor de preocupações com alocação, liberação ou vazamentos de memória.

Um *servlet* é um programa Java comum, executado em um computador que possui um aplicativo chamado de *Web Container*, onde os *servlets* são disponibilizados. O *web container* também é responsável por fazer o papel do servidor *web* em si. Os *servlets* geralmente são responsáveis pelas camadas de negócios e persistência de uma aplicação em três camadas. A partir dos *servlets* são feitas consultas em bancos de dados, arquivos ou outros computadores de uma rede.

A partir do momento em que o *servlet* recebe uma requisição do navegador, seu funcionamento é exatamente igual ao de qualquer aplicativo em uma máquina local. São permitidos acessos ao disco, ao banco de dados, comunicação com outras máquinas de uma rede ou qualquer outra coisa que se poderia fazer em uma aplicação normal. Além disso, tem-se a possibilidade de enviar o retorno das operações executadas de volta para o navegador que fez a requisição em um formato pré-estabelecido, como o XML por exemplo.

2.5.1 O Servlet Container Tomcat

O Tomcat é um *web container* desenvolvido pela Apache Software Foundation que

implementa as especificações de *servlets* e JSP da Sun Microsystems. O Tomcat fornece um ambiente para execução de código Java em cooperação com um servidor *web*.

De acordo com Kurniawan (2002), o Tomcat é o mais popular e o único reconhecido como oficial entre os vários *web containers* existentes atualmente. Originalmente desenvolvido pela Sun Microsystems, seu código fonte foi entregue à fundação Apache em 1999, onde se tornou *open-source* e foi incluído como parte do projeto Jakarta.

O Tomcat é totalmente escrito em Java, o que o torna independente de plataforma. Sua versão mais recente, a 5.5 implementa as especificações de *servlets* versão 2.4 e JSP versão 2.0 da Sun Microsystems.

A fundação Apache (2001) classifica o Tomcat como um servidor *web* completo, já que ele pode funcionar isoladamente, sem a utilização de outros servidores e atender sozinho tanto às requisições HTTP de páginas HTML estáticas como dinâmicas. O Tomcat pode também ser configurado para funcionar como um *servlet container* para outros servidores, como o Internet Information Services (IIS) ou o Apache, por exemplo.

O *web container* é responsável, por receber as requisições HTTP e encaminhá-las ao *servlet* correto e fazer com que o retorno da requisição chegue até o cliente através do protocolo HTTP. Além desta função, a especificação de *servlets* e JSP funciona como um contrato que especifica o que o servidor deve fazer. Suas funções incluem administração de segurança, concorrência, gerenciamento de tempo de vida de *servlets*, transações, distribuição e outros serviços.

2.6 GOOGLE WEB TOOLKIT

O Google *Web Toolkit* (GWT) é um *framework* para desenvolvimento de aplicações Ajax utilizando a linguagem Java. O GWT funciona como um compilador de código Java,

onde a saída, no lugar de *byte-code*, é Javascript. O GWT permite que os recursos como orientação a objetos e modularização, disponíveis na linguagem Java sejam utilizados na criação de aplicativos *web* e automatiza a chamada de métodos remotos encapsulando a camadas de *Remote Procedure Call* (RPC) sobre o *container* Tomcat.

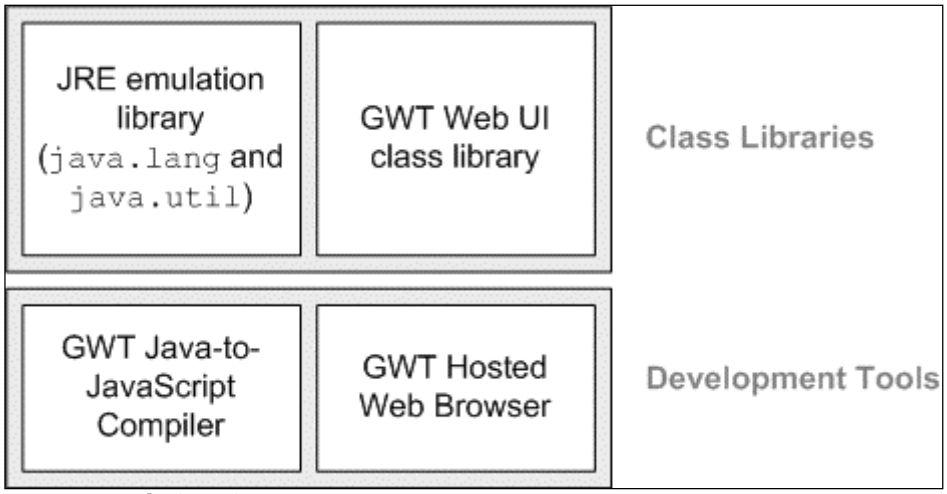
Segundo o Google (2006), escrever aplicações *web* dinâmicas é um processo tedioso e bastante suscetível a erros, onde o desenvolvedor gasta 90% do tempo contornando incompatibilidades entre navegadores *web* e plataformas.

O GWT permite que se utilize qualquer ambiente integrado de desenvolvimento para desenvolvimento Java e ainda permite a utilização de utilitários comuns na linguagem Java como o *framework* de testes JUnit, para criação de testes unitários. A utilização do GWT pode ser ainda combinada com código HTML e Javascript escrito manualmente, permitindo integração e intercâmbio de informações entre o código gerado pelo GWT e o código escrito manualmente.

A utilização do GWT oferece as seguintes vantagens sobre o uso do Javascript de forma manual:

a) checagem estática de tipos da linguagem Java, o que aumenta a produtividade e reduz a quantidade de erros no código;
b) utilização de recursos como complemento de código e refatoramento, disponíveis nas principais IDEs para desenvolvimento Java;
c) geração automática de código específico para vários tipos de navegador *web*;
d) a maioria dos erros típicos do Javascript como incompatibilidade de tipos ou chamadas inválidas de função podem ser apontados em tempo de compilação e não no momento da execução como acontece normalmente com o Javascript.

A Figura 6 mostra a arquitetura do GWT com seus quatro componentes: o compilador Java-JavaScript, um navegador *web*, e duas bibliotecas de classes Java.



Fonte: Google (2006).

Figura 6 - Arquitetura do GWT

Os quatro componentes do GWT são:

a) GWT *Java-to-Javascript Compiler*: é o compilador que traduz o código escrito na linguagem Java para código Javascript. Este compilador é utilizado normalmente no momento da distribuição da aplicação;

b) GWT *Hosted Web Browser*: é um navegador *web* específico que permite executar as aplicações GWT diretamente do IDE de desenvolvimento no modo chamado de "hospedado". Neste modo, o código Java não é compilado para Javascript. Ele é executado diretamente pelo navegador do GWT e pode ainda ser depurado;

c) JRE *emulation library*: são implementações em Javascript das classes disponíveis na biblioteca padrão Java, incluindo a maioria das classes do pacote java.lang e várias classes do pacote java.util;

d) GWT *Web UI class library*: é um conjunto de classes que permite a criação de interfaces com o usuário em navegadores *web*. É através destas classes, que são criadas as páginas *web* propriamente ditas.

As aplicações típicas GWT normalmente requerem que o navegador faça o *download* de aproximadamente 100K de código Javascript, para que a aplicação possa iniciar. Após iniciada, a aplicação possui desempenho similar a de aplicações escritas manualmente com Javascript.

Entre os principais recursos disponíveis no GWT estão:

a) componentes de interface dinâmicos e reusáveis: é possível criar componentes compostos de outros componentes, controlar o *layout* das páginas através de painéis e compartilhar componentes através de arquivos JAR;
b) RPC simplificado: para comunicar a aplicação *web* com o servidor, o desenvolvedor precisa apenas definir as classes Java serializáveis que serão utilizadas e o GWT cuida do processo de serialização e desserialização destas classes permitindo inclusive hierarquias polimórficas de classes e tratamento de exceções;
c) gerenciamento do histórico do navegador *web*: um dos problemas típicos de aplicações Ajax é o funcionamento dos botões de navegação “Página anterior” e “Próxima página” do navegador *web*. O GWT possui um mecanismo para gerenciamento do histórico mantido pelo navegador *web* que registra todas as transições, mesmo dinâmicas de páginas e direciona corretamente os botões de navegação;
d) depuração real: no ambiente de produção, o código é compilado em Javascript e executado em um navegador independente, o que dificulta a depuração, mas durante a fase de desenvolvimento do aplicativo, ele é executado na máquina virtual Java e como todo código Java, pode ser depurado inclusive utilizando os depuradores integrados das principais IDEs do mercado;
e) compatibilidade entre navegadores *web*: o código Javascript gerado pelo GWT, é suportado automaticamente pelos navegadores Internet Explorer, Firefox, Mozilla, Safari e Opera sem que seja necessária nenhuma codificação específica;
f) interoperabilidade: é possível combinar Javascript escrito manualmente com o gerado pelo GWT utilizando o recurso Javascript *Native Interface* (JSNI).

3 DESENVOLVIMENTO DO TRABALHO

O software desenvolvido por este trabalho é um aplicativo *web-based* para monitoramento de ambientes em plantas 2D voltado para a área de segurança patrimonial que tem por finalidade permitir uma visualização gráfica do ambiente monitorado com seus dispositivos sensores instalados e fornecer informações a respeito dos eventos gerados por estes dispositivos.

Este capítulo aborda o desenvolvimento do trabalho, mostrando a especificação com diagramas de casos de uso, diagrama de classes e diagrama de seqüência.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Abaixo são detalhados os requisitos funcionais (RF) e os requisitos não-funcionais (RNF) da aplicação:

a) o sistema deverá exibir uma planta baixa ou mapa gráfico do ambiente monitorado permitindo identificar salas, setores ou divisões destes ambientes (RF);
b) o sistema deverá permitir aproximar ou distanciar (*zoom*) a visualização da planta exibida na página (RF);
c) o sistema deverá permitir navegar para cima, para baixo e para os lados na planta exibida na página, através de *drag-and-drop* caso a planta não possa ser exibida por completo na janela do navegador (RF);
d) o sistema deverá exibir e identificar os dispositivos instalados na planta (RF);
e) o sistema deverá destacar, na visualização da planta, o dispositivo que dispara um evento, permitindo diferenciá-lo dos demais (RF);
f) o sistema deverá armazenar e exibir, informações históricas de todos os eventos

ocorridos (RF);

g) o sistema deverá ser *web-based*, ou seja, funcionar em um navegador *web* (RNF);

h) o sistema deverá exibir os eventos ocorridos com um atraso máximo de cinco segundos para um ambiente onde haja banda livre de rede igual ou superior a 10 Mbitse uma base com até 1.000 dispositivos instalados (RNF);

i) o sistema deverá exibir a relação das pessoas presentes em um local físico determinado, em um tempo máximo de três segundos para um ambiente onde haja banda livre de rede igual ou superior a 10 Mbits e uma base com até 5.000 pessoas cadastradas (RNF);

j) o sistema deverá possuir uma interface fácil de ser usada, permitindo executar ações sobre os dispositivos ou locais físicos específicos da planta utilizando o botão direito do mouse e menus sensíveis ao contexto (RNF).

3.2 ESPECIFICAÇÃO

Para representação da especificação deste trabalho foi escolhido o padrão *Unified Modeling Language* (UML) por atender completamente a todas as necessidades de representação das soluções dos problemas que este trabalho se propõe a tratar. Foram também utilizadas técnicas da modelagem ágil (AMBLER, 2003) que é uma metodologia prática para modelagem de sistemas usando somente os recursos essenciais da UML para solução do problema proposto pelo trabalho.

Para criação dos diagramas deste neste trabalho foi utilizada a ferramenta *Enterprise Architect* (SPARX SYSTEMS, 2006) devido à sua facilidade de utilização e suporte a todos os diagramas da especificação da UML 2.0.

A seguir serão exibidos os diagramas de caso de uso do problema, diagramas de

classes e diagramas de seqüência. Também serão abordadas as ferramentas utilizadas na especificação.

3.2.1 Casos de uso

Os casos de uso representam as interações entre o sistema e agentes externos ao sistema. Neste trabalho foram definidos os seguintes casos de uso:

a) gerar eventos;

b) enviar comandos a dispositivos;

c) consultar eventos ocorridos;

d) consultar pessoas presentes.

A seguir serão detalhados os casos de uso do sistema. A Figura 7 demonstra o diagrama de casos de uso do sistema.

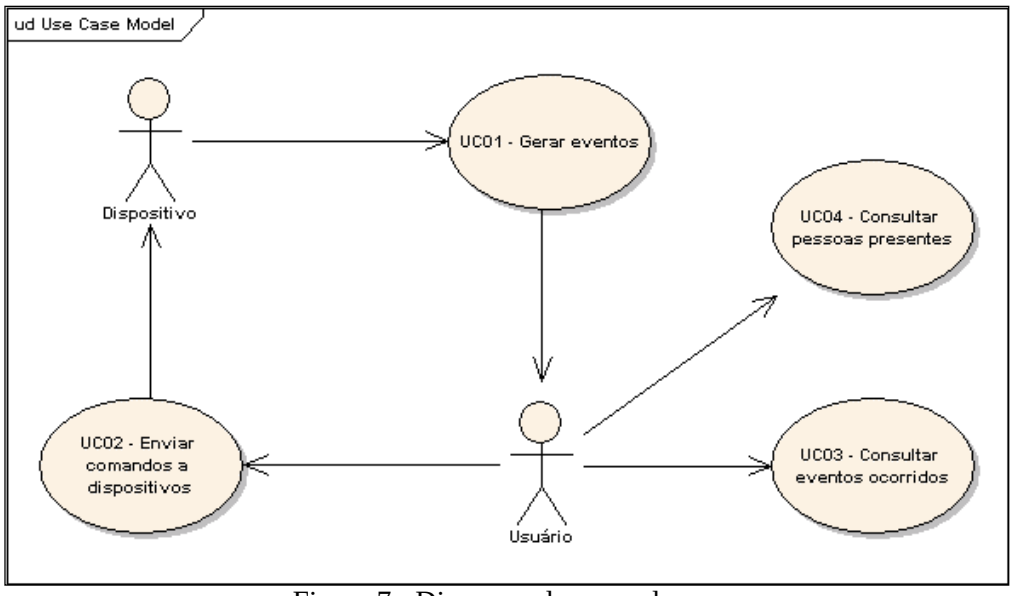


Figura 7 - Diagrama de casos de uso

3.2.1.1 Caso de uso: Gerar eventos

Um dispositivo fisicamente instalado na planta monitorada notifica o sistema da ocorrência de algum evento. O sistema recebe este evento e destaca imediatamente o seu dispositivo gerador na árvore e sua representação no mapa. A ocorrência do evento é gravada em um histórico. O sistema executa uma ação específica de acordo com o tipo do evento ocorrido. Os tipos de evento que serão tratados neste trabalho são:

a) evento de alarme: o dispositivo gerador do evento deve receber o foco na árvore e sua representação gráfica deve ficar piscando até que o evento seja tratado;
b) evento de acesso concedido: é exibida a identificação da pessoa que obteve o acesso concedido. O dispositivo que concedeu o acesso recebe o foco na árvore e sua representação gráfica é selecionada com a cor azul;
c) evento de acesso bloqueado: é exibida a identificação da pessoa que obteve o acesso bloqueado. O dispositivo que bloqueou o acesso recebe o foco na árvore e sua representação gráfica é selecionada com a cor vermelha.

Os eventos disparados por cada dispositivo estão cadastrados no sistema e associados ao dispositivo gerador do evento. Cada evento deve possuir os seguintes dados:

a) código do evento;
b) descrição do evento;
c) possíveis dispositivos geradores do evento.

3.2.1.2 Caso de uso: Enviar comandos a dispositivos

Um dispositivo fisicamente instalado na planta está cadastrado no sistema e pode estar associado a uma série de comandos específicos. Estes comandos fazem com que o dispositivo

execute ações físicas na planta como liberação ou bloqueio de portas, desligamento de alarmes, abertura ou fechamento de cancelas ou exibição de mensagens para uma pessoa identificada. A seguir segue uma lista de comandos suportados pelos dispositivos reconhecidos por este trabalho:

a) catraca:
   - bloqueio: trava o giro da catraca, impedindo passagens,
   - liberação: libera o giro da catraca, liberando passagens,
   - exibir mensagem: exibe uma mensagem de texto em um visor digital da catraca;

b) cancela:
   - fechamento: abaixa a cancela, bloqueando passagens,
   - abertura: levanta a cancela, liberando passagens;

c) sensor de presença (Alarme):
   - ligar: liga a detecção de presença do sensor,
   - desligar: desliga a detecção de presença do sensor,
   - disparar alarme: dispara uma sirene conectada ao sensor de presença,
   - parar alarme: desliga a sirene conectada ao sensor de presença, até que seja detectada uma nova presença.

Para enviar um comando para um dispositivo, o usuário deve clicar com o botão direito do mouse sobre a representação gráfica do dispositivo no mapa, o sistema deverá exibir um menu com opções primárias, entre elas a opção “Comando”. Ao selecionar este item do menu, o sistema deverá exibir um sub-menu com a lista de comandos suportados pelo dispositivo selecionado, para que o usuário possa escolher um.

3.2.1.3 Caso de uso: Consultar eventos ocorridos

O usuário deve selecionar na visualização gráfica um dispositivo ou local físico e clicar com o botão direito do mouse. O sistema exibirá então um menu de contexto, um dos itens do menu deve ser “Histórico de Eventos”. Ao selecionar esta opção, o sistema deverá exibir uma janela solicitando o período que se deseja consultar. Informando o período e clicando em “Pesquisar”, o sistema deverá exibir uma lista de todos os eventos gerados pelo dispositivo ou local físico selecionado no período informado. Caso tenha sido selecionado um local físico, deve incluir os eventos dos locais físicos filhos. Ao clicar sobre um evento exibido, o sistema deverá mostrar informações adicionais sobre o evento, caso existam, como dados da pessoa que o gerou.

3.2.1.4 Caso de uso: Consultar pessoas presentes

O usuário deve selecionar, na visualização gráfica, um local físico da planta e clicar com o botão direito do mouse. O sistema exibirá então um menu, um dos itens do menu deve ser “Pessoas presentes”. Ao selecionar esta opção, o sistema deverá exibir uma janela com a relação das pessoas presentes no local físico no momento. Deve haver a opção de consultar as pessoas presentes nos locais físicos filhos do local selecionado.

3.2.2 Diagramas de classe

Em modelos orientados a objetos, o diagrama de classes é a representação estática das classes de um sistema. As classes, por sua vez, servem para representar o que será manipulado pelo sistema. São nos diagramas de classes que estão representados os relacionamentos como

associações, composições, agregações e especializações entre as classes, com suas multiplicidades, papéis e regras. Este trabalho, por seguir os padrões da UML, utiliza este tipo de diagrama para representar suas estruturas de dados.

3.2.2.1 Value objects (VOs) da estrutura do mapa

Os *value objects*, também conhecidos como *Detail Object*, *Data Transfer Object* e *Replicate Object* são objetos que contém apenas dados, e que normalmente, em aplicações multicamadas são utilizados para transitar informações entre as camadas na aplicação. A utilização deste tipo de objetos segue um padrão de projeto também chamado de *Value Object* (JVALUE, 1998) e é bastante utilizado em cenários onde um cliente necessita requisitar dados de um servidor remoto. Este trabalho utiliza VOs para representar objetos básicos que refletem informações da camada de persistência (ou camada de dados).

O diagrama exibido na Figura 8 mostra as classes envolvidas na exibição estática do mapa de ambientes.

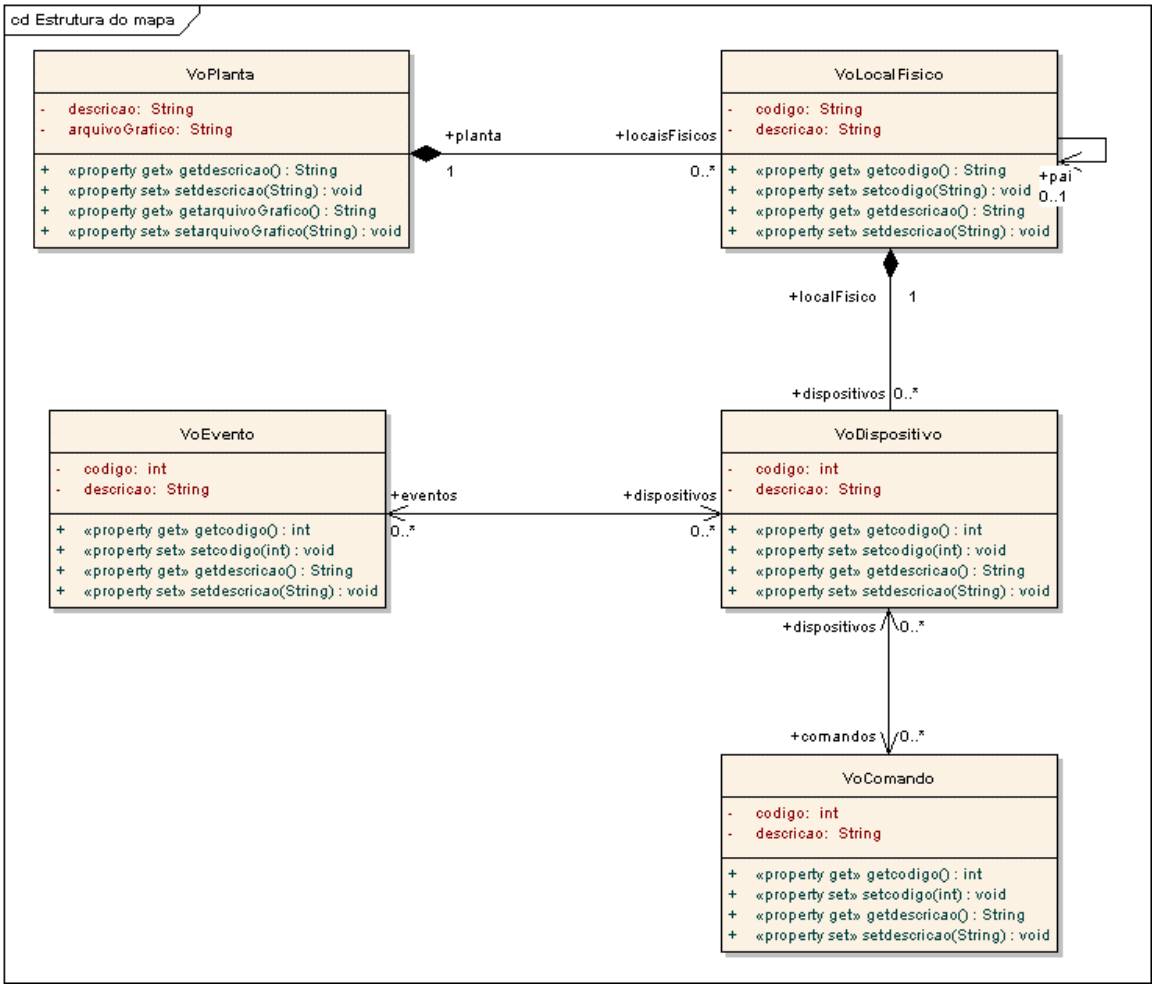


Figura 8 - Diagrama de classes da estrutura do mapa

Este diagrama mostra que a raiz do monitoramento de acesso é a classe VoPlanta que contém um *link* para o arquivo de visualização gráfica do mapa de ambientes, e a partir desta classe, são localizados seus locais físicos, dispositivos, eventos e comandos do dispositivos. Como o aplicativo aqui descrito é apenas um monitorador de eventos, a alimentação dos dados destas classes está fora do escopo deste trabalho.

3.2.2.2 VOs de Identificação das pessoas

Os VOs de identificação de pessoas são o elo entre o que é lido por um dispositivo e uma pessoa. Em um aplicativo de segurança, uma das funções dos dispositivos de controle de acesso é a identificação da pessoa que está requerendo o acesso. Esta identificação pode ser

feita de várias maneiras.

As classes derivadas de IdentificadorPessoa implementam formas de se chegar a uma pessoa a partir de dados lidos por dispositivos de reconhecimento. Depois de encontrar a pessoa, o sistema consegue verificar seus direitos de acesso por dispositivo e faixa horária.

Este trabalho propõe tratar apenas identificação por crachá, mas o diagrama exibido na Figura 9 mostra a estrutura básica para adição de outros tipos de reconhecimento.

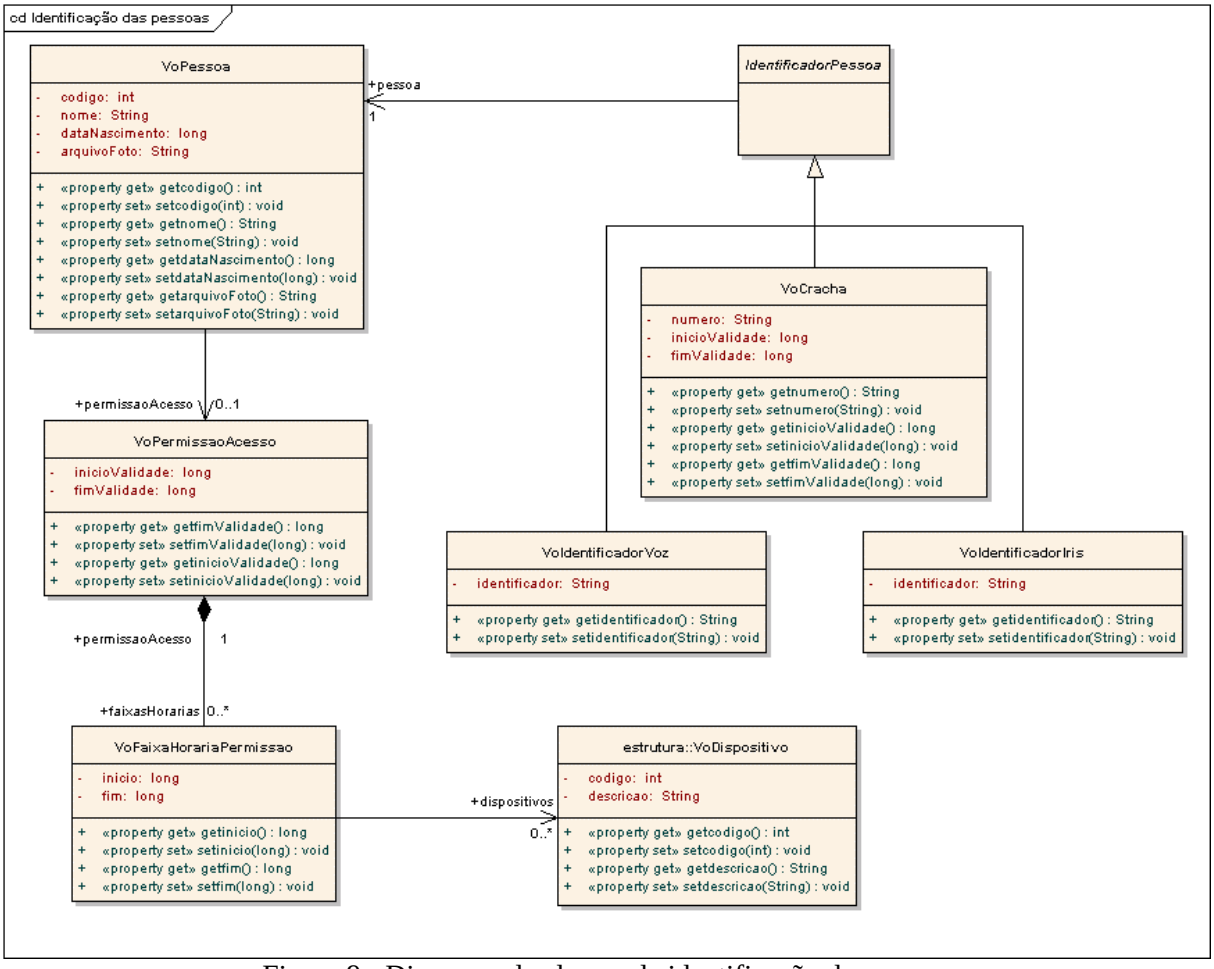


Figura 9 - Diagrama de classes de identificação de pessoas

3.2.2.3 VOs de tratamento de informações históricas

Os VOs de informações históricas são utilizados para armazenar informações sobre as ações do usuário, sobre os dispositivos do sistema e os eventos gerados pelos dispositivos

instalados no ambiente, além do tratamento dado a estes eventos. A Figura 10 exibe o diagrama de classes de VOs de informações históricas do sistema.

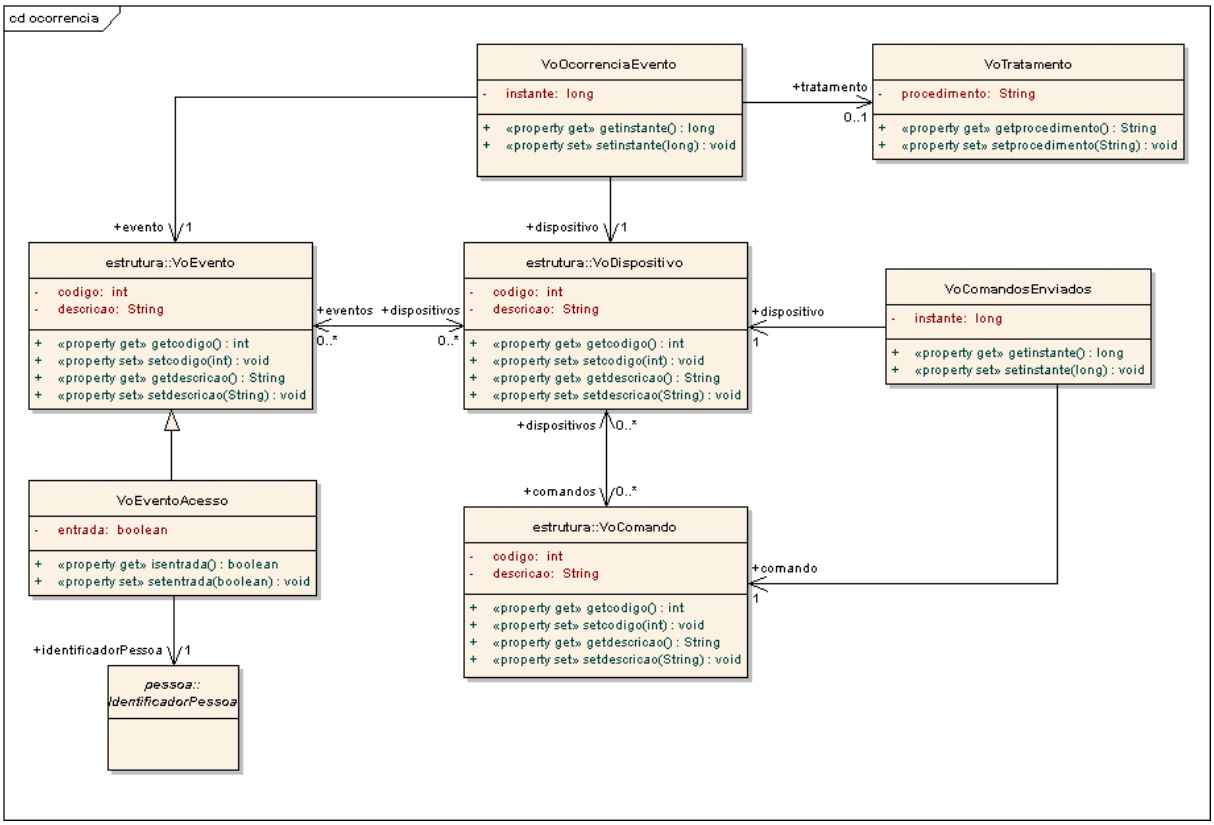


Figura 10 - Diagrama de classes de informações históricas

A classe VoOcorrenciaEvento armazena um registro do momento do acontecimento dos eventos e seus respectivos dispositivos geradores e tratamento dado a estes eventos. A classe VoComandosEnviados armazena registros dos comandos enviados aos dispositivos instalados na planta, juntamente com o momento do seu envio.

3.2.2.4 Classes de interface do aplicativo com os dispositivos

As classes de interface do aplicativo com dispositivos são utilizadas para receber eventos e enviar comandos aos dispositivos.

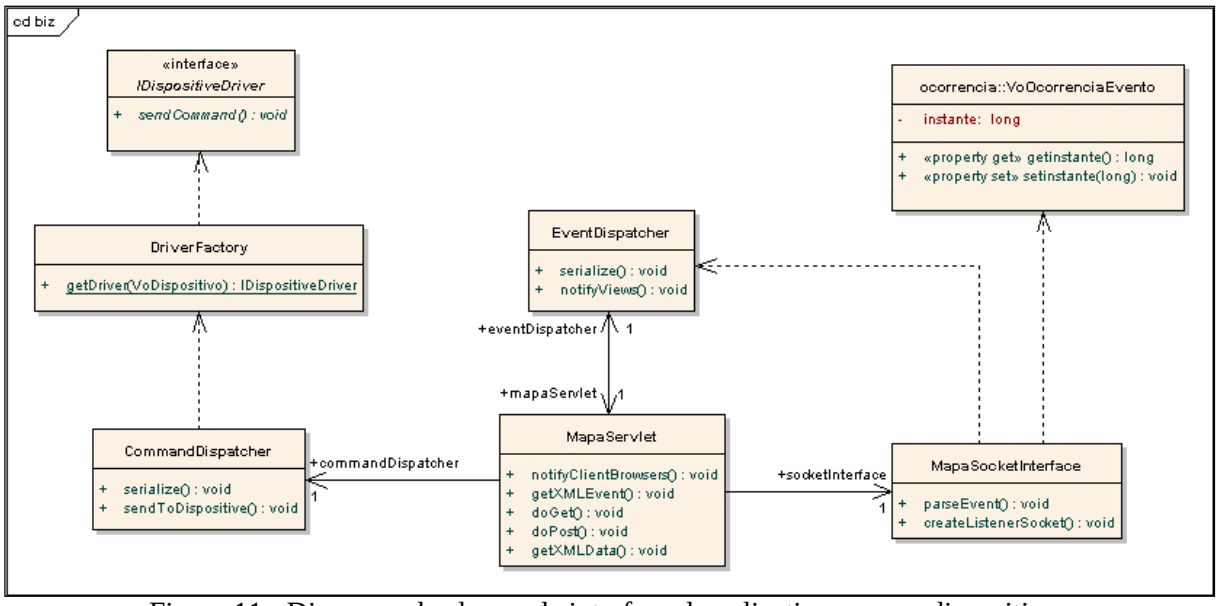


Figura 11 - Diagrama de classes de interface do aplicativo com os dispositivos

A classe MapaServlet é um *servlet* Java. Ela é instanciada pelo *servlet container* e mantém ativo um *socket* implementado pela classe MapaSocketInterface. Esta classe recebe, via socket, as notificações de ocorrência de evento geradas pelos dispositivos em formato texto (este trabalho utiliza notação XML para esta notificação), interpreta estas notificações, transforma-as em um objeto do tipo VoOcorrenciaEvento e o envia para a classe EventDispatcher.

A classe EventDispatcher é responsável por persistir o evento e enviá-lo para o navegador *web* através da classe MapaServlet.

O diagrama da Figura 11 mostra também a interface para envio de comandos aos dispositivos, onde a classe MapaServlet recebe do navegador *web* um envio de comando e o repassa para a classe CommandDispatcher que é responsável por persistir este comando e envia-lo para o *driver* específico do dispositivo através da classe DriverFactory. Esta classe reconhece os *drivers* presentes através de arquivos descritores armazenados na pasta da aplicação. Cada arquivo descritor precisa ter o endereço RMI do *driver* e a lista de dispositivos suportados por ele.

Após encontrar o *driver* certo para o dispositivo que receberá o comando, a classe

DriverFactory instancia o *driver* do dispositivo e envia o comando. Todos os *drivers* devem implementar a interface IDispositiveDriver para que sejam suportados.

3.2.3 Diagramas de seqüência

Diagrama de seqüência é um tipo de diagrama usado na UML, representando a seqüência de processos ou mais especificamente, as mensagens trocadas entre os objetos.

Os diagramas de seqüência descrevem a maneira como grupos de objetos colaboram em algum comportamento ao longo do tempo. Ele registra o comportamento de um único caso de uso, exibindo os objetos e as mensagens passadas entre esses objetos no caso de uso. O projeto de um caso de uso pode ter uma grande quantidade de métodos em muitas classes diferentes, tornando difícil determinar a seqüência global do comportamento. Nestes casos, o diagrama de seqüência se torna muito útil para definir os fluxos de comunicação entre os objetos.

A seguir serão apresentados alguns diagramas de seqüência utilizados na realização dos principais casos de uso do sistema.

3.2.3.1 Recebimento e notificação do evento

O diagrama de seqüência de “recebimento e notificação do evento” faz parte do caso de uso Gerar Evento e trata de como o sistema receberá o evento do dispositivo e como este evento chegará até o navegador *web*.

A Figura 12 mostra o fluxo de mensagens desde a geração do evento pelo dispositivo, seu recebimento pelas classes de interface da aplicação até a exibição deste evento no navegador *web*.

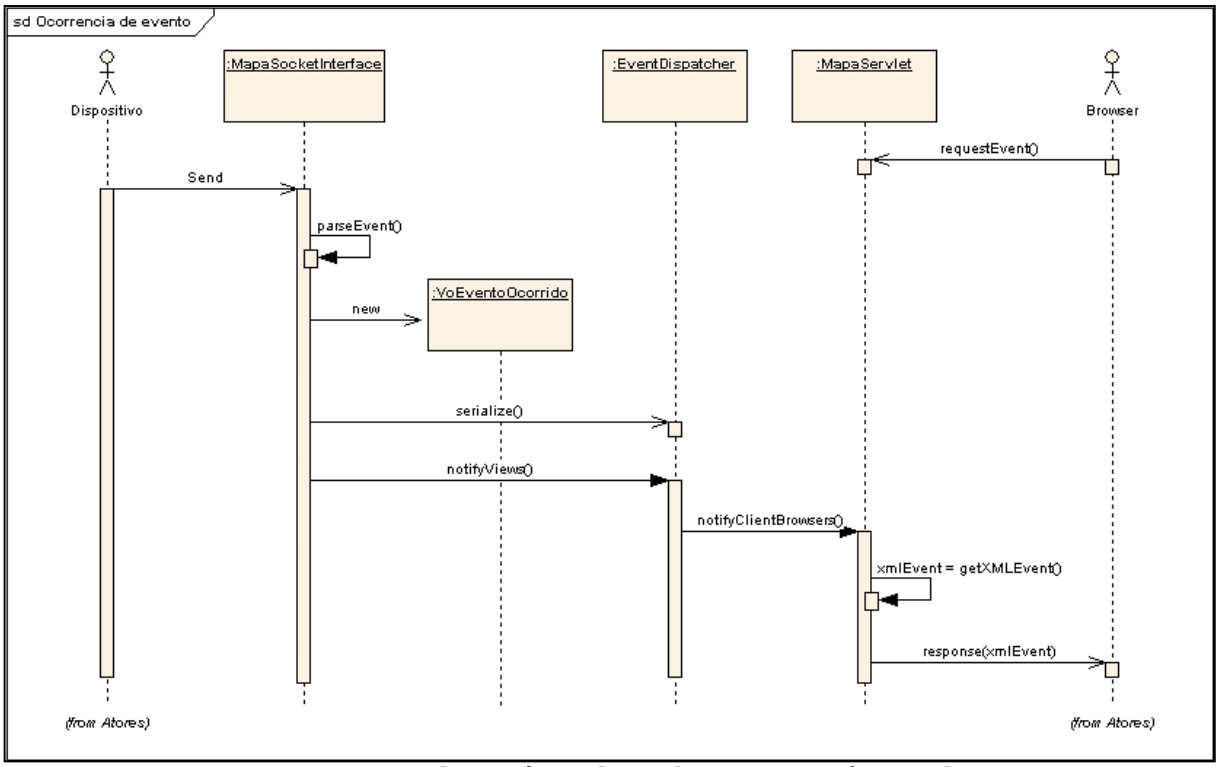


Figura 12 – Diagrama de seqüência de recebimento e notificação do evento

3.2.3.2 Envio do comando ao dispositivo

O diagrama de seqüência de “envio de comandos ao dispositivo” faz parte do caso de uso Enviar comandos a dispositivos e cuida do fluxo de dados, desde a ação do usuário no navegador até o momento em que o comando chega até o dispositivo, passando por um *driver* específico implementado para cada dispositivo.

A Figura 13 mostra o fluxo de mensagens desde a seleção, pelo usuário, do dispositivo e do comando a ser executado no navegador *web*, passando pelas classes de controle do sistema até chegar à interface do *driver* responsável pelo dispositivo instalado na planta.

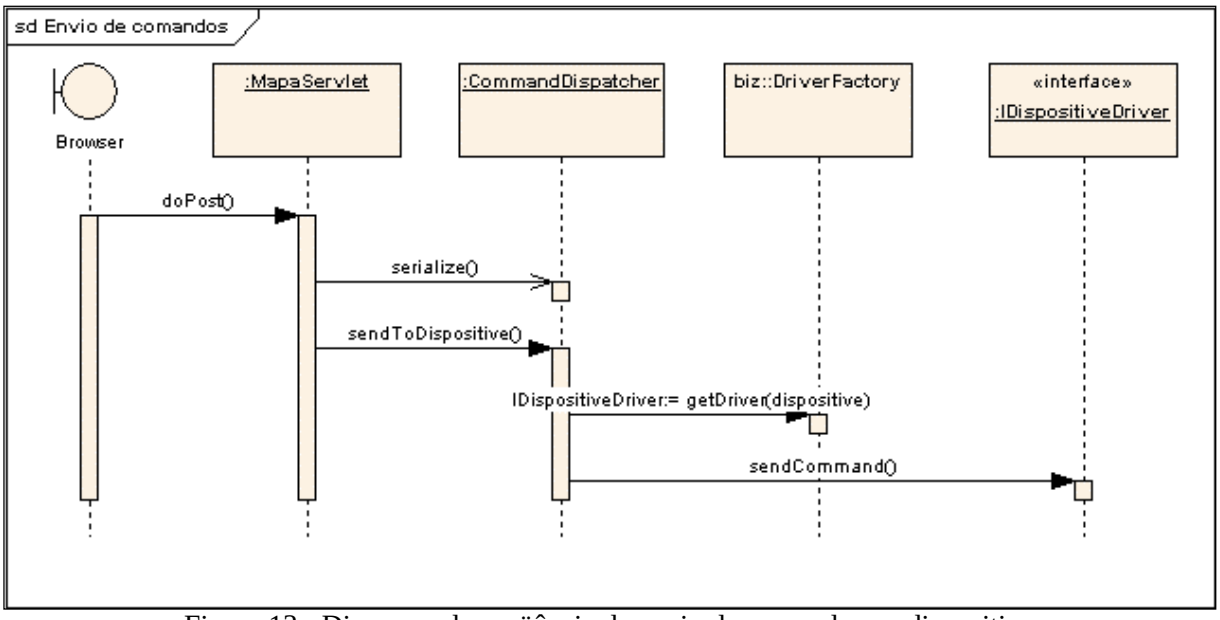


Figura 13 - Diagrama de seqüência de envio de comandos ao dispositivo

3.2.3.3 Consultar pessoas presentes

O diagrama de seqüência de “consulta de pessoas presentes” faz parte do caso de uso Consultar pessoas presentes e tem a finalidade de exibir um relatório das pessoas presentes em um local físico em um determinado instante. O diagrama de seqüência ilustra os objetos envolvidos nesta consulta e a ordem das mensagens trocadas por eles.

Este diagrama de seqüência ilustra o processo de consulta da base histórica de eventos de acesso para obtenção dos eventos de entrada que não possuem um equivalente de saída.

A Figura 14 mostra o fluxo de mensagens desde a seleção, pelo usuário, do local físico a ser consultado no navegador *web*, passando pelas classes de controle do sistema até chegar às classes de armazenamento de informações históricas de eventos do sistema.

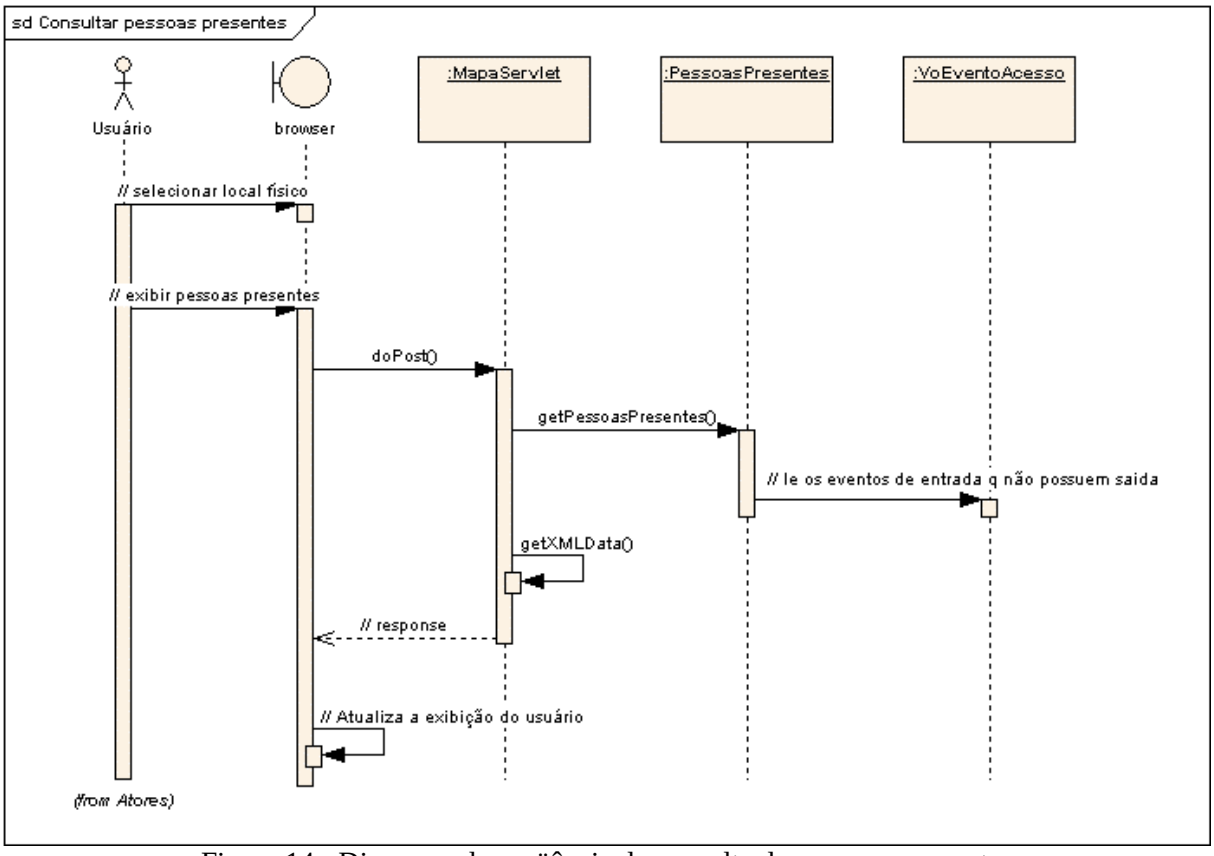


Figura 14 - Diagrama de seqüência de consulta das pessoas presentes

É interessante observar que a consulta de pessoas presentes pode ser considerada uma especialização da consulta de eventos ocorridos. Todavia este caso de uso merece destaque por possuir requisitos não funcionais de desempenho específicos mais rígidos que a consulta de eventos ocorridos, tais como, exibir o resultado da consulta em até um segundo para um cenário específico.

3.2.3.4 Consultar eventos ocorridos

O diagrama de seqüência de “consulta de eventos ocorridos” faz parte do caso de uso Consultar eventos ocorridos e tem a finalidade de exibir um relatório com informações históricas sobre os eventos ocorridos em um local físico em um determinado período. O diagrama de seqüência ilustra os objetos envolvidos nesta consulta e a ordem das mensagens trocadas por eles.

Este diagrama de seqüência ilustra o processo de consulta da base histórica de eventos para obtenção dos eventos ocorridos dentro de condições de filtro específicas.

A Figura 15 mostra o fluxo de mensagens, desde a seleção, pelo usuário, do local físico e dos filtros da consulta no navegador *web*, passando pelas classes de controle do sistema, até chegar às classes de armazenamento de informações históricas de eventos do sistema.

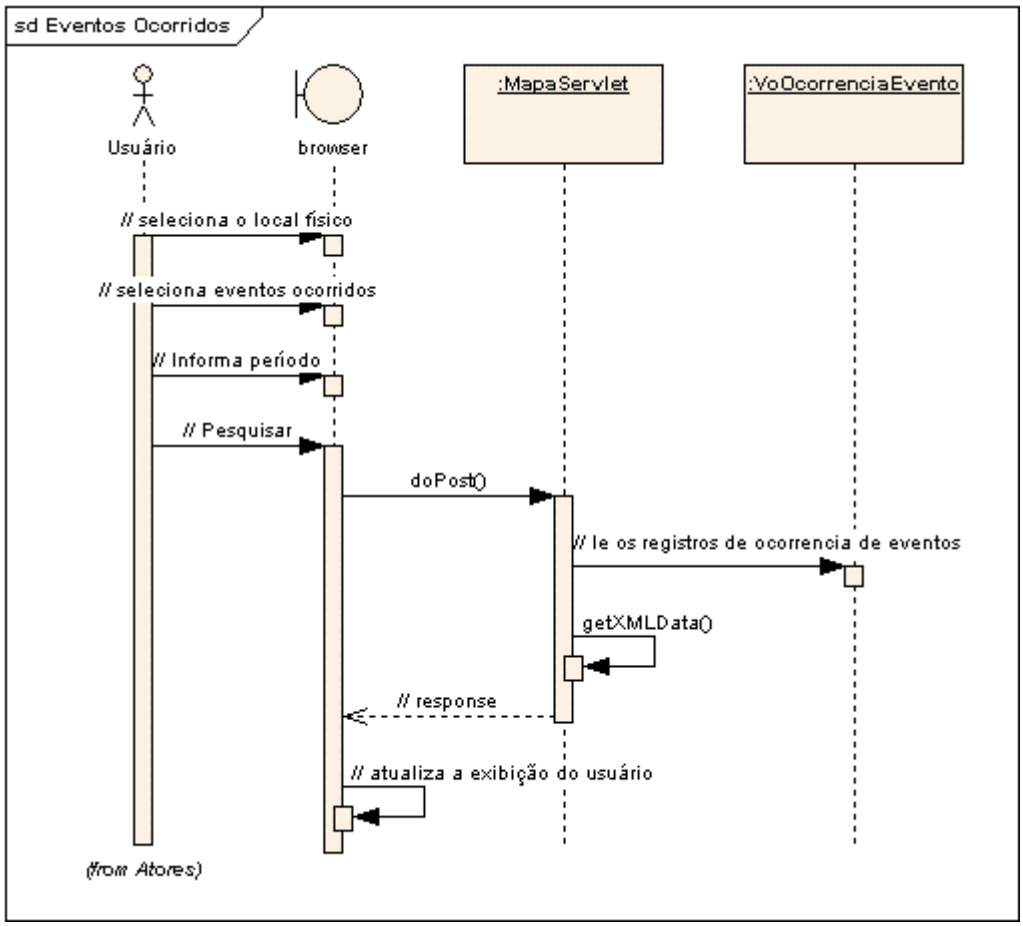


Figura 15 - Diagrama de seqüência de consulta de eventos ocorridos

3.3 IMPLEMENTAÇÃO

As sessões seguintes descrevem as ferramentas e técnicas utilizadas para o desenvolvimento do trabalho e a operacionalidade do aplicativo.

3.3.1 Técnicas e ferramentas utilizadas

Para a fase de desenvolvimento do aplicativo deste trabalho foi utilizada basicamente a plataforma Eclipse (ECLIPSE, 2006) com o auxílio de alguns *plugins* específicos. A seguir será detalhada cada ferramenta utilizada.

3.3.1.1 Plataforma Eclipse

O Eclipse é um IDE extensível, multi-plataforma totalmente implementado em linguagem Java para desenvolvimento de aplicações em diversas linguagens de programação. A plataforma Eclipse disponibiliza um *framework* baseado em *plugins* que permite a fácil extensão do ambiente de desenvolvimento para os mais diversos propósitos.

A plataforma Eclipse em si já é formada por um conjunto de *plugins* que adicionam a ela algumas funcionalidades importantes para os desenvolvedores como acesso a repositórios CVS integrado, editor de *scripts* ANT para criação de *scripts* de compilação do projeto personalizados e pacote de testes unitários baseada em JUnit entre outros. Este trabalho utiliza ainda alguns *plugins* adicionais para o Eclipse que serão descritos a seguir.

3.3.1.1.1 Eclipse Web Tools Plataform (WTP)

O Eclipse *Web Tools Plataform* (ECLIPSE WEB TOOLS PLATAFORM, 2006) é um projeto de código aberto que estende a plataforma Eclipse com ferramentas para desenvolvimento de aplicações J2EE para *web*. O projeto WTP inclui editores para HTML, Javascript, CSS, JSP, XML e outros formatos comumente utilizados no desenvolvimento de aplicativos *web*. O WTP possui também assistentes para criação de projetos J2EE, publicação

dos projetos em servidores de aplicação ou *web containers* e geração automática de descritores XML.

Este trabalho utilizou o *plugin* WTP para edição de páginas XHTML, criação dos *scripts* Javascript e de documentos XML. O *plugin* WTP também foi utilizado pela sua integração com o *container* Tomcat.

3.3.1.1.2 SVG Eclipse Plugin

O SVG Eclipse Plugin (SVG ECLIPSE PLUGIN, 2006) é baseado em um projeto da Fundação Apache chamado Batik. O Batik é um conjunto de ferramentas criadas em Java para manipulação de imagens no formato SVG. Este projeto fornece ferramentas para interpretação, geração, visualização e conversão de imagens SVG para formatos como JPEG, PNG ou Tiff.

O *plugin* adiciona à IDE do Eclipse ferramentas e assistentes que permitem criar, editar e visualizar os arquivos SVG dentro do próprio ambiente de desenvolvimento do Eclipse. Existem ainda recursos para impressão das imagens e ajuda on-line. As imagens SVG utilizadas por este trabalho foram editadas e testadas utilizando o SVG Eclipse plugin.

3.3.1.2 Navegador Firefox 1.5

O Firefox 1.5 é um navegador *web* livre, de código fonte aberto e multi-plataforma desenvolvido pela Mozilla Corporation (MOZILLA CORPORATION, 2006) juntamente com centenas de voluntários. O navegador Firefox 1.5 foi escolhido para este trabalho por atender de forma mais completa às especificações de padrões definidos pela W3C e por possuir suporte nativo à especificação da linguagem SVG além de ser um navegador livre, ou seja,

gratuito e disponível para os principais sistemas operacionais existentes no mercado.

3.3.2 Operacionalidade da implementação

A operacionalidade da implementação será demonstrada através da simulação de um caso de utilização do sistema.

Quando a página principal do sistema é aberta é exibido do lado esquerdo da tela árvore com a hierarquia dos locais físicos cadastrados para a planta atualmente selecionada (a planta é selecionada através do *combobox* “Planta”, acima da árvore). Do lado direito da tela é exibida a visualização gráfica da planta selecionada com seus locais físicos e dispositivos instalados. Neste momento o usuário já está monitorando os eventos ocorridos na planta.

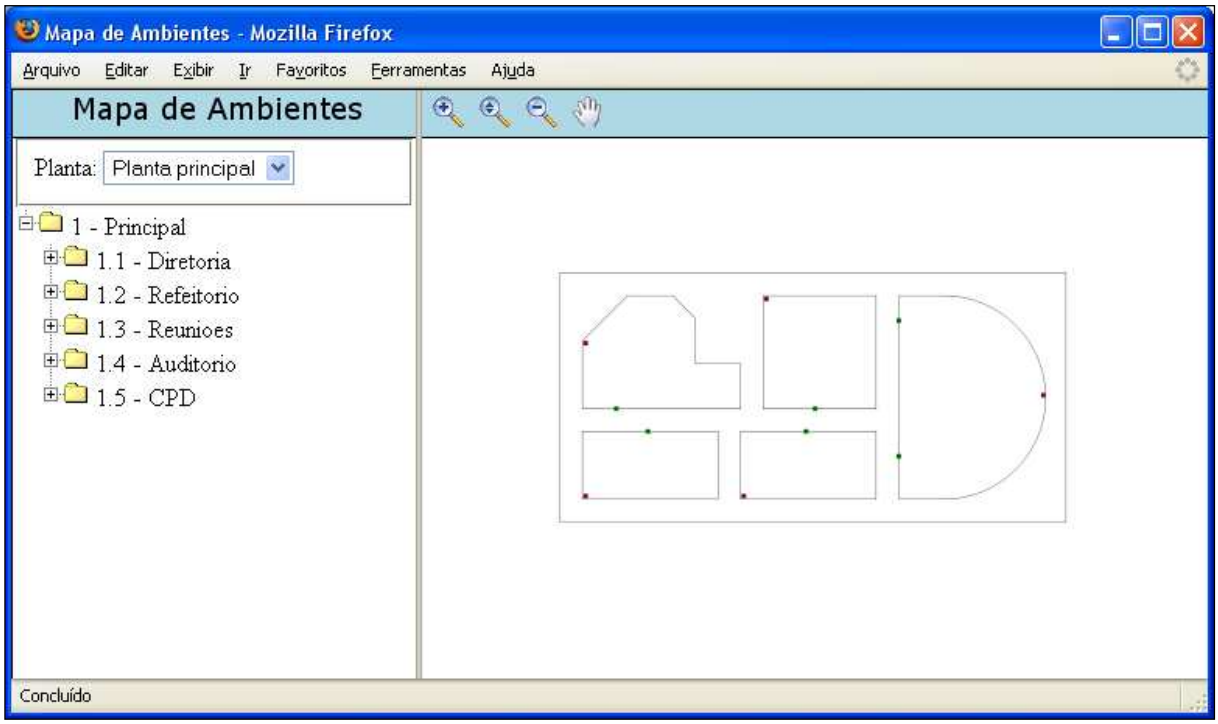


Figura 16 - Tela principal do sistema

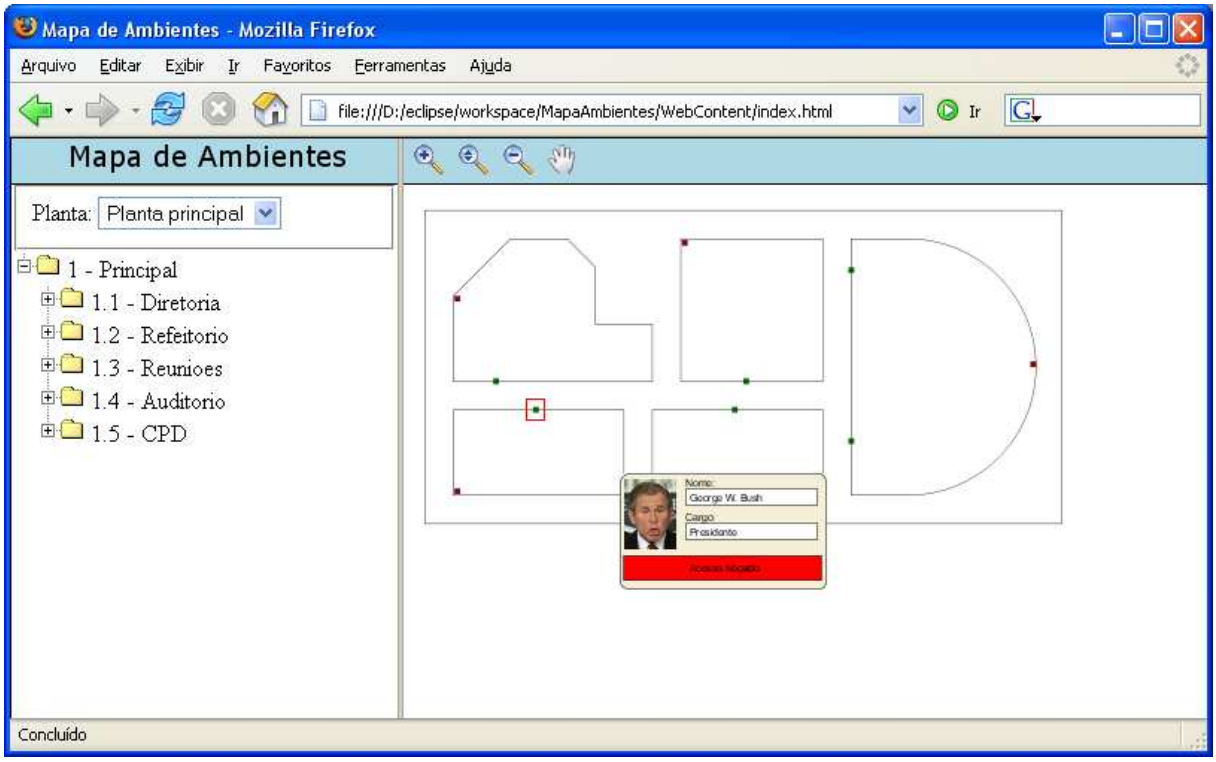


Figura 17 - Notificação de evento de acesso negado

A Figura 17 mostra um exemplo do comportamento do sistema quando ocorre uma tentativa de acesso negado a um local físico. Neste caso, um funcionário tentou acessar um local físico para o qual não tem direitos de acesso. Deve-se observar que a notificação da ocorrência de acesso inválido somente acontece caso o local físico tenha sido configurado para exibir notificações da ocorrência deste tipo de evento.

Após a notificação de negação de acesso, caso se deseje verificar os eventos ocorridos neste dispositivo no dia de hoje, o usuário executa uma ação de zoom sobre o local da notificação para facilitar a seleção do dispositivo.

Para se executar a função de zoom, deve-se selecionar na barra de ferramentas o ícone de uma lupa com um símbolo “+” e clicar com o botão esquerdo do mouse sobre o local do mapa que se deseja ampliar.

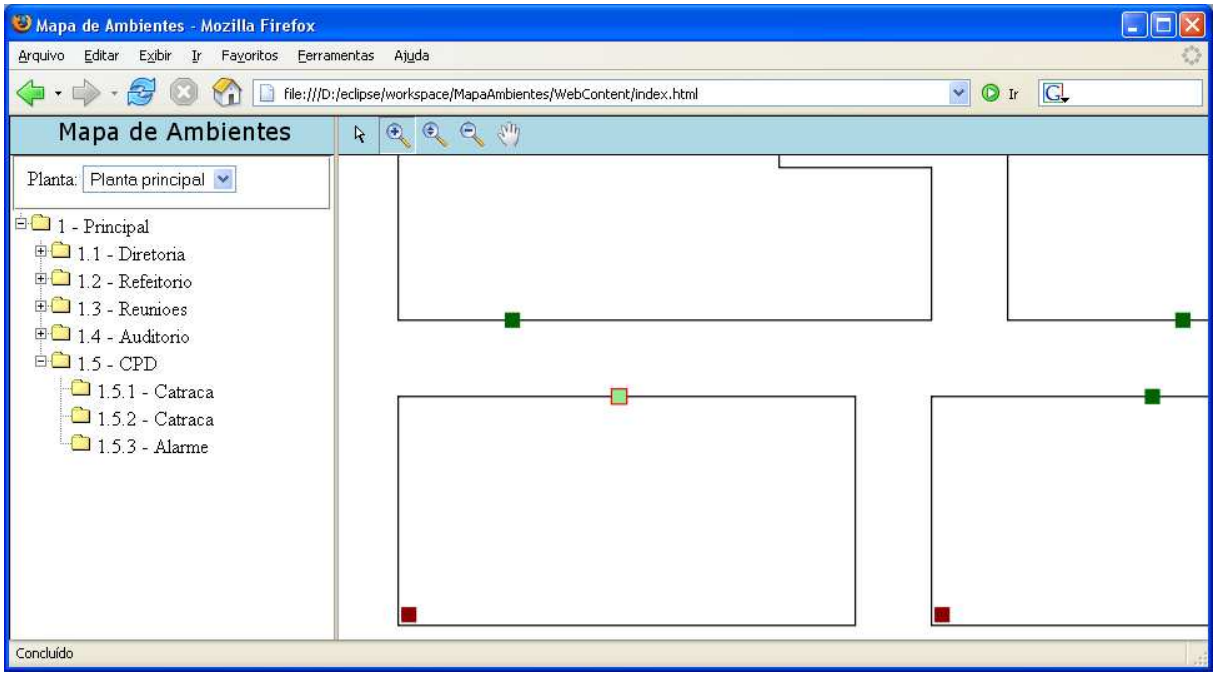


Figura 18 - Zoom no dispositivo gerador do ultimo evento

Depois de feito o zoom sobre o dispositivo desejado, o usuário seleciona o dispositivo, clicando sobre ele com botão esquerdo do mouse.

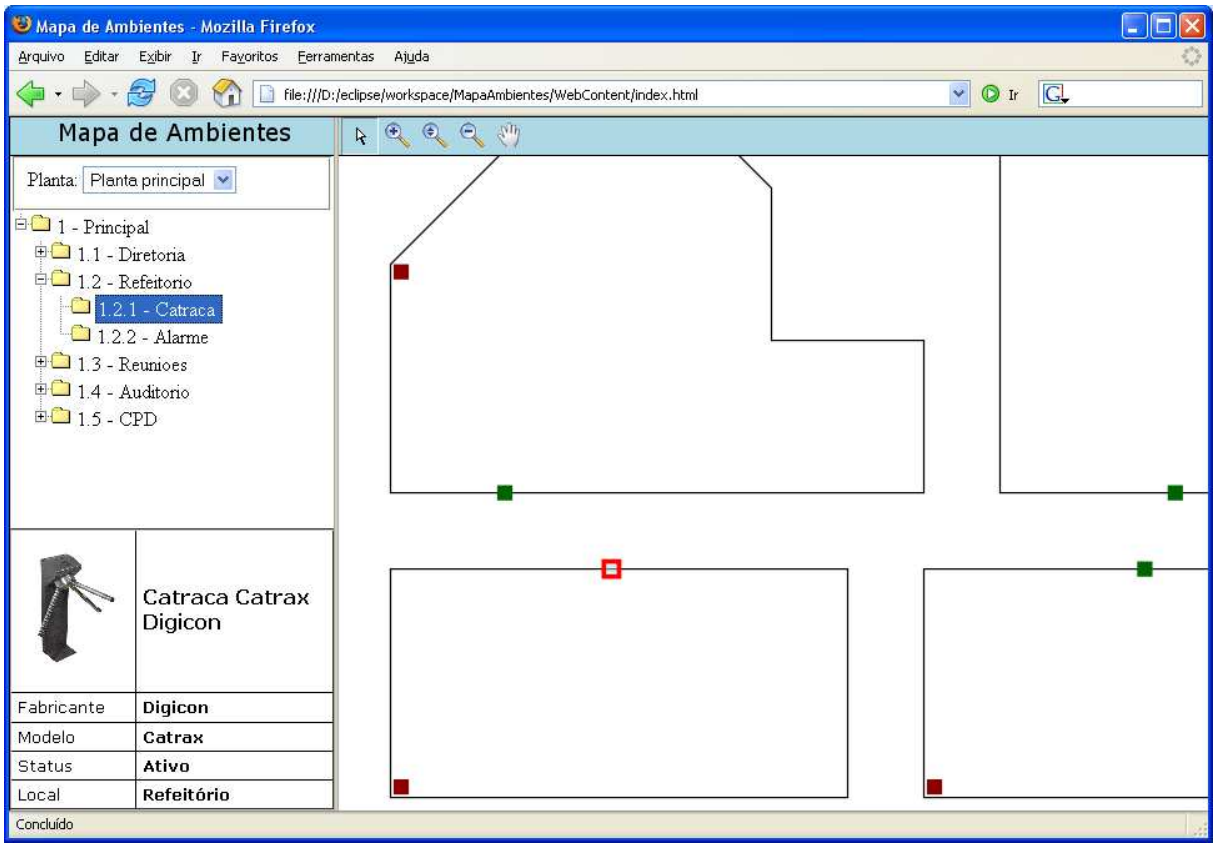


Figura 19 - Dispositivo do mapa selecionado

Pode-se observar na Figura 19 que quando um dispositivo da planta é selecionado,

seus dados são exibidos em uma tabela de propriedades logo abaixo da arvore de locais físicos e dispositivos. A seleção de um dispositivo também faz com que seu correspondente na árvore de locais físicos e dispositivos receba o foco e fique destacado.

Depois de selecionado o dispositivo, clicando com o botão direito do mouse sobre ele, deverá ser exibido um menu sensível ao contexto com as opções disponíveis para o dispositivo selecionado, como mostra a Figura 20.

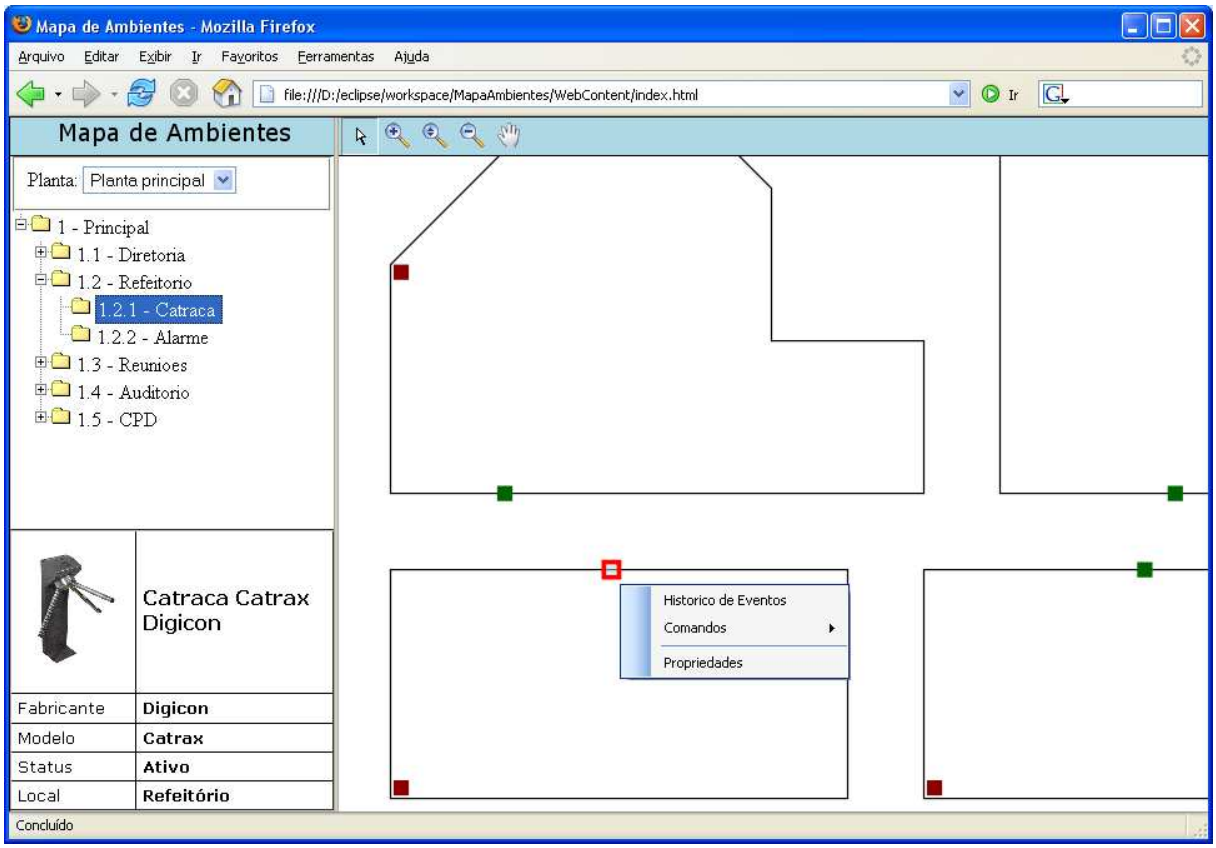


Figura 20 - Menu sensível ao contexto do dispositivo

Selecionando-se a opção “Histórico de Eventos” no menu do dispositivo, será exibido o formulário de consulta de histórico de eventos.

No formulário de consulta de históricos de eventos (Figura 21), informa-se o período desejado e clica-se em pesquisar. A lista dos eventos ocorridos será exibida em uma grid. O botão fechar fecha o formulário e retorna ao mapa de ambientes.

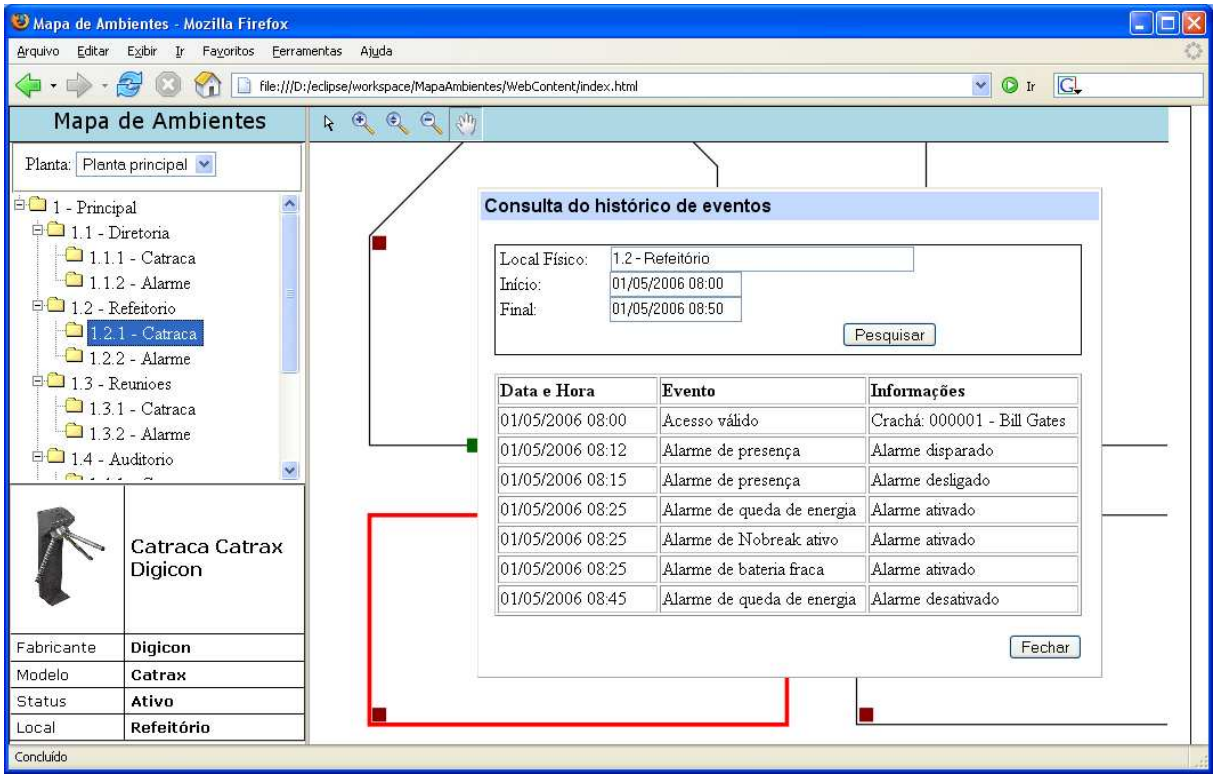


Figura 21 - Consulta do histórico de eventos

O usuário do sistema tem ainda a opção de consultar a relação de pessoas presentes em um local físico no momento atual. Para isto, seleciona-se o local físico desejado e clica-se sobre ele com o botão direito do mouse, um menu sensível ao contexto é exibido com as opções do local físico. A Figura 22 mostra a tela da aplicação com um local físico selecionado e o menu sensível ao contexto.

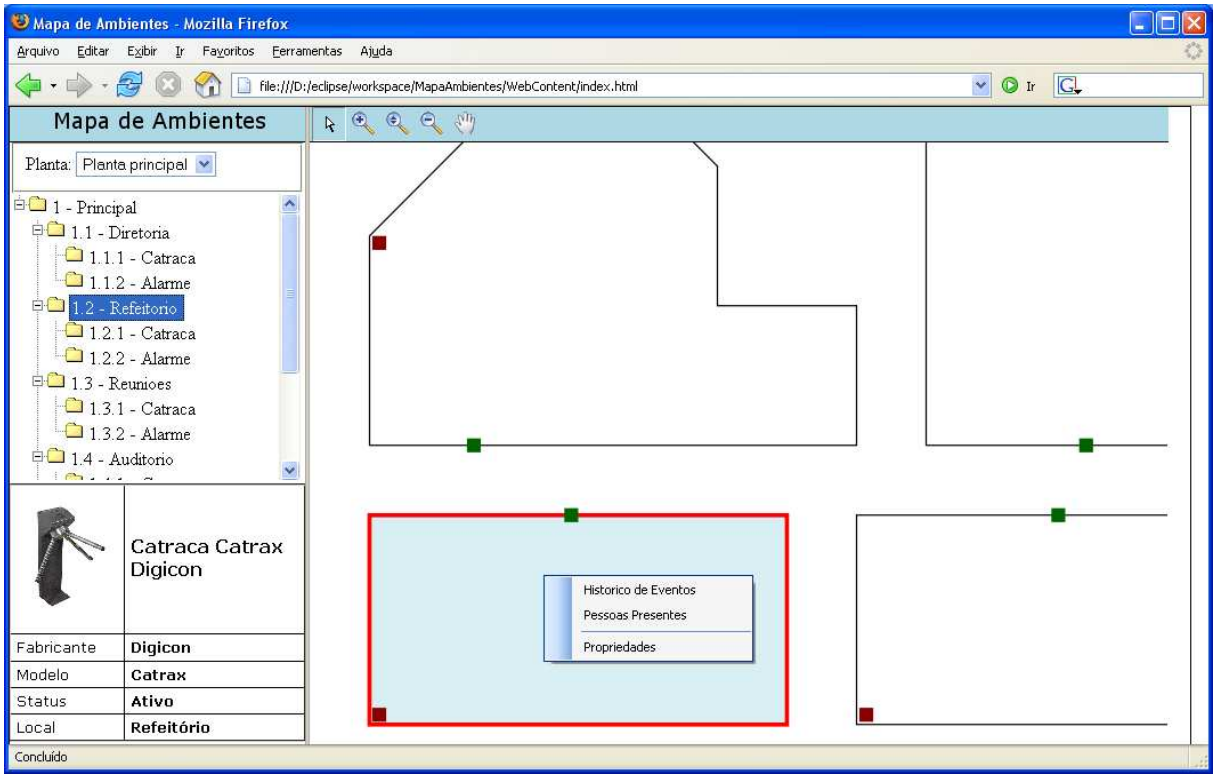


Figura 22 - Menu sensível ao contexto do local físico

Selecionando-se a opção “Pessoas Presentes” no menu do dispositivo, será exibido o formulário de consulta de pessoas presentes (Figura 23).

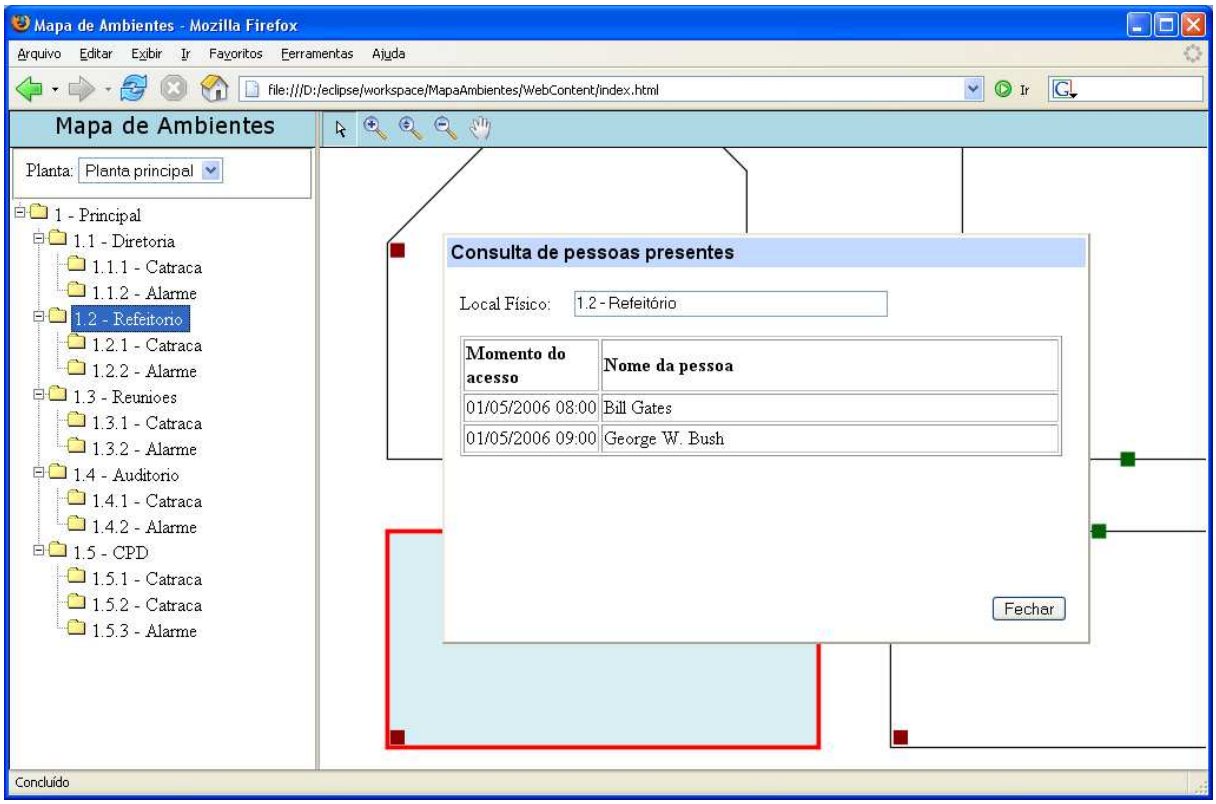


Figura 23 - Consulta de pessoas presentes

O formulário de pessoas presentes exibe, de forma dinâmica os nomes das pessoas que estão presentes em um determinado local físico do mapa.

3.4 RESULTADOS E DISCUSSÃO

O monitoramento de ambientes é apenas uma parte de um sistema de segurança. Ele não precisaria necessariamente ser feito por um aplicativo gráfico. Ele poderia também ser feito através de informações textuais. Todavia, o monitoramento gráfico torna mais clara a localização dos eventos ocorridos, bem como permite um melhor entendimento do ambiente monitorado facilitando a referência aos dispositivos e locais físicos, além de ter um forte apelo comercial, principalmente quando este monitoramento acontece através de um navegador *web*.

O aplicativo desenvolvido por este trabalho possui diversas limitações de funcionalidade e é fortemente dependente de outros aplicativos que alimentam sua a base de dados. Normalmente um aplicativo de monitoramento de ambientes também permite configurações de permissões, cadastros de pessoas e dispositivos e, em alguns casos, permite o desenho do ambiente monitorado ou importação de plantas criadas por aplicativos específicos da área de engenharia.

As técnicas do Ajax se mostraram muito eficientes na manipulação das requisições assíncronas e na padronização do formato de intercâmbio de dados. A utilização massiva do Javascript, porém, demonstrou que ainda é necessária a criação de ferramentas que facilitem sua organização e manipulação. Durante o desenvolvimento deste trabalho foi percebido que à medida que a quantidade de código Javascript aumentava, como é normal em uma aplicação Ajax, a dificuldade de organização, manutenção e correção de *bugs* do sistema crescia consideravelmente.

Problemas como falta de escopos bem definidos, linguagem procedural e falta de consistência de tipos foram constantes na fase de implementação. A utilização do *framework* GWT ajudou bastante na função de abandonar a codificação Javascript direta e passar a codificar em Java, onde se consegue reduzir drasticamente a quantidade de erros. Ferramentas como o refatoramento e assistentes de código do Eclipse, com o GWT puderam ser usados para gerar código Javascript.

O Quadro 8 mostra o código utilizado para criação do layout da tele principal da aplicação. Pode se notar que o código que será convertido pelo GWT para Javascript é similar ao código utilizado pelo Java para criação de diálogos utilizando Swing.

```
public DockPanel createStaticLayout() {
    DockPanel panel = new DockPanel();

    panel.setSize("100%", "100%");
    panel.setHorizontalAlignment( DockPanel.ALIGN_LEFT );

    // Painel da tree e propriedades
    DockPanel vpLeft = new DockPanel();
    vpLeft.setSize("250px", "100%");
    vpLeft.setStyleName("leftDiv");

    vpLeft.add( getTitulo(), DockPanel.NORTH );
    vpLeft.add( getCombo(), DockPanel.NORTH );
    vpLeft.add( getInfo(), DockPanel.SOUTH );
    Widget tree = getTree();
    vpLeft.add( tree, DockPanel.CENTER );
    vpLeft.setCellHeight(tree, "100%");

    // Painel do mapa
    mapaPanel.setSize("100%", "100%");
    DockPanel dpRight = new DockPanel();
    dpRight.setSize("100%", "100%");
    dpRight.setStyleName("rightDiv");
    dpRight.add( getBotoes(), DockPanel.NORTH );
    dpRight.add( mapaPanel, DockPanel.CENTER );
    dpRight.setCellHeight(mapaPanel, "100%");

    panel.add(vpLeft, DockPanel.WEST);
    panel.add(dpRight, DockPanel.CENTER);
    panel.setCellWidth(dpRight, "100%");

    return panel;
}
```

Quadro 8 - Código para criação do layout da tela principal da aplicação

Por ser um framework muito recente, o GWT ainda não possui muitos componentes e não suporta algumas formas de interação tais como, botão direito do mouse e botões do tipo *push-button*. Além disso, ele não é compatível com o conteúdo SVG. Para adicionar, no protótipo implementado, as funcionalidades ausentes no GWT, teve-se que usar codificação Javascript manual integrada com o GWT, mas ainda assim suscetível a todos os problemas do Javascript.

A aplicação implementada por este trabalho poderia ter sido construída utilizando o Macromedia Flash (MACROMEDIA, 2006), que possui recursos similares aos da dupla Ajax e SVG, mas este trabalho optou por mostrar as tecnologias alternativas ao Flash. Além disso, páginas implementadas em Flash costumam ter desvantagens como maior dificuldade de localização de conteúdo por ferramentas de busca, custo do ambiente de desenvolvimento e necessidade de instalação de *plugins* específicos.

Por fim, todos os requisitos funcionais do aplicativo foram atingidos. O caso de uso de envio de comandos para dispositivos foi projetado, mas não foi implementado por este trabalho. Os requisitos não funcionais foram atingidos sem maiores dificuldades devido à característica assíncrona das requisições do Ajax que fazem que os primeiros resultados das consultas já sejam exibidos mesmo antes da consulta terminar. Assim não é preciso transferir do servidor para o navegador, grandes blocos de dados de uma só vez e ainda permite que servidor e navegador executem suas tarefas simultaneamente.

4 CONCLUSÕES

Este trabalho propôs e apresentou a implementação de uma aplicação para o monitoramento gráfico de ambientes. O aplicativo desenvolvido por este trabalho pode vir a substituir módulos de monitoramento desktop existentes, com a vantagem de não haver mais problemas de distribuição de versões ou instalações, além de benefícios como administração centralizada de rotinas como atualizações e backup.

A utilização das técnicas do Ajax e do SVG para criação de aplicativos mostrou-se muito promissor para criação de uma nova geração de aplicativos *web-based* sem que haja necessidade de se abrir mão de recursos dos aplicativos desktop.

As técnicas do Ajax e do SVG, por serem ainda pouco utilizadas, demonstraram-se ainda pouco padronizadas e documentadas. Notaram-se principalmente a ausência ou escassez de documentos do tipo “boas práticas” de programação. Foi observado que estas tecnologias ainda têm um longo caminho a seguir até tornarem-se padrão de mercado, mas mesmo durante o período de desenvolvimento deste trabalho foi notável a sua evolução, com novas ferramentas, ambientes de desenvolvimento e novas formas de utilização surgindo a cada semana.

A seção extensões sugere possíveis implementações que complementariam as funcionalidades da aplicação *web* de monitoramento de ambientes em plantas 2D.

4.1 EXTENSÕES

Este trabalho contempla apenas o módulo de monitoramento de ambientes do controle de acesso e segurança. Isto faz com que este sistema seja dependente de outros módulos para manutenção dos dados do sistema e cadastros. Como sugestão para extensão deste trabalho

poderiam ser implementadas algumas outras funcionalidades de um sistema de controle de acesso e segurança como:

a) criação de filtros de monitoramento de eventos como por exemplo notificar somente determinados eventos em dispositivos específicos;
b) implementação do módulo de envio de comandos para dispositivos;
c) controles de notificação de ocorrência de eventos, como emails, SMS ou alarmes encadeados.

Poderia também ser sugerida a implementação de uma ferramenta *web* para a criação e edição do mapa de ambientes diretamente no navegador *web*, possibilitando criação (ou desenho) da planta monitorada com seus locais físicos, adição de dispositivos nos locais físicos e configuração de suas propriedades.

REFERÊNCIAS BIBLIOGRÁFICAS

AMBLER, Scott W. **Modelagem ágil**: práticas eficazes para a programação extrema e o processo unificado. Porto Alegre: Bookman, 2003. 352p.

APACHE SOFTWARE FOUNDATION. **Apache tomcat**. [S.l.], 2001. Disponível em: <http://tomcat.apache.org/tomcat-3.3-doc/tomcat-ug.html>. Acesso em: 23 abr. 2006.

CRANE, David; PASCARELLO, Eric. **Ajax in action**. Greenwich: Manning, 2005.

DUFFY, Scott. **How to do everything with javascript**. Berkeley: McGraw-Hill/Osborne, 2003.

ECLIPSE. **Welcome**. [S.l.], 2006. Disponível em: <http://www.eclipse.org/>. Acesso em: 23 abr. 2006.

ECLIPSE WEB TOOLS PLATFORM. E**clipse wtp project**. [S.l.], 2006. Disponível em: <http://www.eclipse.org/webtools/>. Acesso em: 23 abr. 2006.

LOWE, David; WILDE, Erik. **XPath, XLink, XPointer, and XML**: a practical guide to web hyperlinking and transclusion. [S.l.], 2002.

GARRETT, Jesse James. **Ajax:** a new approach to web applications. San Francisco, 2005. Disponível em: <http://www.adaptivepath.com/publications/essays/archives/000385.php>. Acesso em: 23 abr. 2006.

GOOGLE. **Google web toolkit**. [S.l.], 2006. Disponível em: <http://code.google.com/webtoolkit/>. Acesso em: 27 maio 2006.

JVALUE. **Why value objects?**. [S.l.], 1998. Disponível em: <http://www.jvalue.org/explanation.html>. Acesso em: 29 abr. 2006.

KURNIAWAN, Budi. **Java for the web with servlets, JSP, and EJB:** a developer's guide to J2EE solutions. [S.l.]: New Riders Publishing, 2002.

MACROMEDIA. **Flash Professional.** [S.l.], 2006. Disponível em: <http://www.adobe.com/products/flash/flashpro>. Acesso em: 10 jul. 2006.

MAHEMOFF, Michael. **Ajax patterns**. [S.l], 2005. Disponível em: <http://ajaxpatterns.org/#Background>. Acesso em: 23 abr. 2006.

MCLELLAN, Drew. **Very dynamic web interfaces**. [S.l.], 2005. Disponível em: <http://www.xml.com/pub/a/2005/02/09/xml-http-request.html>. Acesso em: 23 abr. 2006.

MOZILLA CORPORATION. **Firefox 1.5**. [S.l.], 2006. Disponível em: <http://www.mozilla.com/firefox/>. Acesso em: 29 abr. 2006.

PRESCOD, Paul. **SVG**: a sure bet. [S.l.], 2003. Disponível em: <http://xml.com/pub/a/2003/07/16/svg-prescod.html>. Acesso em: 08 abr. 2006.

SPARX SYSTEMS. **Enterprise Architect**. [S.l.], 2006. Disponível em: <http://www.sparxsystems.com.au/products/ea.html>. Acesso em 09 jul. 2006.

SUN MICROSYSTEMS. **Java servlet technology**. [S.l.], 2005. Disponível em: <http://java.sun.com/products/servlet/>. Acesso em: 15 maio 2006.

SVG ECLIPSE PLUGIN. **Summary**. [S.l.], 2006. Disponível em: <http://sourceforge.net/projects/svgplugin>. Acesso em: 29 abr. 2006.

TEARE, David. **An introduction to Ajax**. [S.l], 2005. Disponível em: <http://dev2dev.bea.com/pub/a/2005/08/ajax_introduction.html>. Acesso em: 23 abr. 2006.

WALSH, Norman. **A technical introduction to XML**. [S.l.], 1998. Disponível em: <http://www.xml.com/pub/a/98/10/guide0.html>. Acesso em: 23 abr. 2006.

W3C. **Scalable vector graphics (SVG) 1.1 specification**. [S.l], 2003. Disponível em: <http://www.w3.org/TR/SVG11/>. Acesso em: 23 abr. 2006.

W3C. **The extensible hypertext markup language**. [S.l], 2002. Disponível em: <http://www.w3.org/TR/xhtml1/#xhtml/>. Acesso em: 23 abr. 2006.

W3SCHOOLS. **Introduction to CSS**. [S.l.], 2006. Disponível em: <http://www.w3schools.com/css/css_intro.asp>. Acesso em: 29 abr. 2006.