Universidade do Estado do Rio de Janeiro
Centro de Tecnologia e Ciências
Faculdade de Engenharia

Leonardo Chaves Machado

Modelagem e desenvolvimento de sistemas de informações geográficas para *web* com tecnologias de *rich internet applications*

Rio de Janeiro
2009

Leonardo Chaves Machado

Modelagem e desenvolvimento de sistemas de informações geográficas para *web* com tecnologias de *rich internet applications*

Dissertação apresentada como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-graduação em Engenharia da Computação da Universidade do Estado do Rio de Janeiro. área de concentração: Geomática

Orientador: Prof. Dr. Orlando Bernardo Filho
Co-Orientador: Prof. Dr. João Araújo Ribeiro

Rio de Janeiro
2009

CATALOGAÇÃO NA FONTE
UERJ/REDE SIRIUS/CTC/B

M149 Machado, Leonardo Chaves
Modelagem e desenvolvimento de sistemas de informações geograáficas para web com tecnologias de rich Internet applications. / Leonardo Chaves Machado. – 2009.
114f.: il.

Orientador: Prof. Dr. Orlando Bernardo Filho.
Co-orientador: João Araújo Ribeiro.

Dissertação (mestrado) – Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia.
Bibliografia: 114.

1.Sitemas de informação geográficas.
2. Engenharia de software. I. Filho, Orlando Bernardo. II. Ribeiro, João Araújo. III. Universidade do Estado do Rio de Janeiro. IV. Titulo

CDU 528

_____ _____

Assinatura Data

AGRADECIMENTOS

A Petula Guimarães, amiga e colega de hospício, pelo suporte pessoal e observações pertinentes ao texto.

A Marcelo Garcia, meu amigo, pelas boas observações ao texto.

A José Machado, Maria Chaves Machado, Antonio Carlos Chaves Machado e Sandra Maria Chaves Machado, pelo carinho e suporte que me fizeram ser o que sou.

Ao Prof. Dr. Marcelo Sperle, pela generosa orientação acadêmica.

Aos Prof. Dr. Orlando Bernardo Filho e Prof. Dr. João Araújo Ribeiro, por seus papéis de orientadores.

RESUMO

MACHADO, Leonardo Chaves. Modelagem e desenvolvimento de sistemas de informações geográficas para *web* com tecnologias de *rich internet applications. 2009.* 108f. Dissertação (Mestrado em Engenharia da Computação) – Faculdade de Engenharia,Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2009.

Os SIG estão se popularizando cada vez mais e isso tem se dado principalmente através da Internet. Os assim chamados SIG-Web no entanto, quando desenvolvidos com as tecnologias tradicionais de *web*, apresentam as mesmas fraquezas daquelas, a saber: sincronicidade e pobreza na interação com o usuário. As tecnologias usadas para *Rich Internet Applications* (RIA) são uma alternativa que resolvem esses problemas. Na presente dissertação será demonstrada a factibilidade do seu uso para o desenvolvimento de SIG-Web, oferecendo um conjunto de códigos e estratégias para desenvolvimentos futuros,a partir de um conjunto básico de operações a se realizar em um SIG-Web. Adicionalmente será proposta a UWE-R, uma extensão a uma metodologia de engenharia *web* existente, para modelagem de RIA e SIG-Web.

Palavras-chave: Sistema de Informações Geográficas. *Rich Internet Applications* . Engenharia de Software.

ABSTRACT

GIS are getting more popular, mainly due to the Internet. Nonetheless the so called Web-GIS, when developed using traditional web technologies, inherit the same weaknesses from those, i.e.: sinchronicity and poor user interaction. Rich Internet Applications (RIA) technologies are an alternative for those issues. In the present work the feasibility of their usage for Web-GIS will be demonstrated, providing a set of codes and strategies for future developments, taking into account a basic set of Web-GIS operations. Additionally UWE-R, an extension to a web engineering methodology will be proposed for RIA modeling and Web-GIS.

Keywords: Geographical Information Systems. Rich Internet Applications. Software Engineering

SUMÁRIO

INTRODUÇÃO

Apresentação

Cada vez mais os Sistemas de Informações Geográficas (SIG) estão fazendo parte da vida das pessoas. Haja vista a utilização maciça dos chamados geonavegadores como *Google Earth* (10 milhões de *downloads* desde o lançamento até meados de 2007) e *Microsoft Virtual Earth*. A *web*, de fato, é hoje a maneira mais fácil, rápida e barata de oferecer dados atualizados para usuários não especializados. No entanto o modelo *web* tradicional apresenta algumas características que dificultam a sua utilização como bem denota (DRIVER *et al*., 2005). Tais problemas podem ser resumidos como sendo a sincronicidade e a pobreza de elementos de interface. A sincronicidade implica em uma espera por parte do usuário até que o processamento do lado do servidor termine. A pobreza nos elementos de interface faz com que os usuários comuns se sintam pouco atraídos e tenham dificuldades na utilização desses sistemas.

As tecnologias *Rich Internet Applications* (RIA) oferecem um notável incremento na qualidade da interação humano-computador para as aplicações *web* em geral. Com elas, muitas requisições ao servidor podem ser feitas de modo assíncrono e há uma riqueza nos elementos de interface que dão ao usuário a ilusão de estar utilizando uma aplicação *desktop*. Notadamente para as aplicações SIG-Web o interesse é ainda maior, pois essas contam com uma série de operações do ponto de vista do usuário (*zoom*, deslocamento, colocação de camadas, entre outras) que prescindem de uma comunicação síncrona entre o cliente (navegador) e servidor.

No presente trabalho, será estudada a viabilidade da utilização de tecnologias RIA para o desenvolvimento de SIG-Web. Adicionalmente, será apresentada a *UML For Web Engineering and RIA* (UWE-R), uma extensão a uma metodologia de engenharia *web*, para fazer a modelagem de aplicações RIA e SIG-Web.

Acredita-se que a sinergia potencial entre essas duas tecnologias justifica o presente

estudo. A pouca pesquisa acadêmica relacionada com RIA é outro fator que o motiva e faz com que o presente trabalho tenha seu interesse científico. Como se trata de um estudo focado em operações típicas de um SIG-Web com tecnologias existentes e exemplos de códigos e estratégias, servirá como base para outros pesquisadores desenvolverem SIG-Web futuramente. Do ponto de vista da engenharia de software a contribuição será preencher uma lacuna em modelagem para aplicações RIA e de aspectos de um SIG-Web.

Objetivos

São dois os objetivos desta dissertação de mestrado: prover um conjunto de estratégias e códigos para implementação de SIG-Web, utilizando tecnologias RIA e propor uma metodologia de modelagem para aplicações RIA e SIG-Web, estendendo a UWE.

Hipóteses

São duas as hipóteses a serem discutidas nesta dissertação:

1 – Tecnologias RIA são apropriadas para o desenvolvimento de SIG-Web;

2 – UWE-R é adequada à modelagem de RIA e SIG-Web.

Metodologia

As tecnologias RIA escolhidas para este trabalho serão definidas e justificadas na seção 2. Nessa seção, também definir-se-ão 5 operações típicas de um SIG-Web que serão testadas.

A primeira hipótese foi corroborada ou afastada utilizando as tecnologias referidas para cada uma das 5 operações referidas. Para cada uma delas foram feitos testes de utilização da tecnologia em questão para escrever o código necessário à mesma. Foram testados tanto o formato vetorial quanto o *raster*. Uma nota foi atribuída ao suporte oferecido pela tecnologia à operação, de modo a quantificar o estudo.

Adicionalmente, um conjunto de *sites* com SIG desenvolvidos utilizando RIA foram apresentados e estudados.

Para testar a segunda hipótese a UWE-R foi aplicada a diferentes *sites* contendo SIG-Web desenvolvidos com RIA: (GEABIOS, 2007) feito com Openlaszlo, (GOOGLE MAPS, 2007) feito com GWT e (YAHOO! MAPS, 2007) feito com Flex. Deste modo seu poder expressivo foi avaliado. Para elaborar-se a UWE-R, foram estudadas as metodologias de engenharia *web* existentes.

Para o desenvolvimento dos códigos aqui apresentados foi utilizado um computador Intel Pentium Dual *Core* de 2.8GHz e 2GB de RAM, rodando o sistema operacional GNU/Linux kernel 2.6.22 (distribuição Fedora *Core* 6). Para a modelagem em UML foi utilizada a versão 5.0.2 da ferramenta Jude Community.

Organização

Esta dissertação está dividida em 5 capítulos. Inicialmente é feita, na seção 1, a apresentação dos conceitos fundamentais necessários para o entendimento deste trabalho. São discutidos panoramicamente os conceitos de SIG e SIG-Web, RIA e Engenharia *Web*. Nele são apresentadas as várias metodologias de modelagem *web* e sistemas multimídia, como preâmbulo necessário para as discussões seguintes. São resumidas 14 metodologias existentes.

Na segunda seção, apresenta-se um estudo comparativo de 3 tecnologias RIA aplicadas a 5 operações básicas de um SIG-Web, tanto no formato *raster* quanto no vetorial. Ele também trata de demonstrar o uso efetivo de algumas tecnologias RIA em *sites* que são SIG-Web. O objetivo deste é testar a primeira hipótese do trabalho.

A terceira seção é sobre UWE-R, a extensão a uma das metodologias apresentadas anteriormente para possibilitar a modelagem de sistemas com tecnologias RIA e SIG-Web.

O objeto da quarta seção é a aplicação da UWE-R a 3 diferentes SIG-Web existentes a

fim de testar sua capacidade expressiva.

Por fim são apresentadas as conclusões e sugestões de trabalhos futuros.

Contribuições

A UWE-R proposta neste trabalho é uma das poucas metodologias propostas até agora para modelagem de sistemas *web* com tecnologias RIA. Tal inovação obteve publicações recentes, inclusive em revista internacional (MACHADO; BERNARDO FILHO; RIBEIRO, 2008) e (MACHADO; BERNARDO FILHO; RIBEIRO, 2009). Além disso, este trabalho chama a atenção da comunidade de desenvolvimento em SIG para a relevância de RIA. Comprovam esse fato as publicações obtidas em congressos (MACHADO; BERNARDO FILHO; RIBEIRO, 2007) e (MACHADO; BERNARDO FILHO; RIBEIRO, 2008a).

1 – FUNDAMENTOS TEÓRICOS

1.1- Introdução

Este trabalho se apóia nos seguintes fundamentos: SIG e SIG-Web, RIA e Engenharia *Web*. Para cada um desses fundamentos, haverá uma seção a seguir. Em cada uma delas explicam-se os conceitos fundamentais e explicitam-se as ligações entre os mesmos.

1.2 - SIG e SIG-Web

Sistema de Informação Geográfica, como definido em (BURROUGH; MCDONNELL, 1998) "é um conjunto poderoso de ferramentas para coletar, armazenar, obter, transformar e apresentar dados espaciais do mundo real para um conjunto particular de propósitos". Como tal, podemos notar sua complexidade. Para fins deste estudo, serão focadas as funcionalidades que são visíveis ao usuário do SIG (apresentação e algumas transformações). Ainda citando os mesmos autores, pode-se modularizar um SIG nos seguintes componentes: interface com o usuário, apresentação e relatórios, entrada de dados, base de dados geográfica e transformações, uu seja, nesta dissertação o foco será nos 2 primeiros componentes e em alguns aspectos do último. Aqui também sempre assume-se uma arquitetura cliente-servidor, por se tratar de mundo *web*.

De acordo com (COUCLELIS, 1992), objetos que são manipulados em um SIG "podem ser contados, movidos, empilhados, rotacionados, coloridos, rotulados, cortados, separados, fatiados, combinados, visualizados de diferentes ângulos, sombreados, aumentados, diminuídos (...), como uma variedade de objetos sólidos do cotidiano que não têm nenhuma relação específica com Geografia". Ora, todas essas operações de um SIG não tem uma razão intrínseca para serem realizadas no lado do servidor. Eis a motivação para tratá-las com RIA.

1.3 – *Rich Internet Applications*

Há algumas definições de RIA (KLEIN; CARLSON; MCEWANN, 2007) e (ADOBE, 2007). Aqui propõe-se a seguinte: são aplicações que se beneficiam da ubiqüidade da *web* e melhoram seu modelo de interação tradicional, adicionando elementos de interface mais ricos juntamente com um mecanismo flexível e assíncrono de comunicação com o servidor.

São inúmeras as tecnologias relacionadas com RIA. A maior parte delas envolve, de algum modo, AJAX. AJAX é a sigla para *Asynchronous Javascript with Xml*. Com AJAX tem-se um conjunto de tecnologias (XML, DOM, XHTML, *Javascript* e *XMLHttpRequest*) que permitem a interação assíncrona com o servidor (ZAKAS; MCPEAK; FAWCETT, 2007). No entanto, RIA não se limita a AJAX. Tecnologias como Flex da Adobe e Java da Sun Microsystems também oferecem mecanismos assíncronos de comunicação com o servidor e a riqueza da interface gráfica.

A cada dia, novas ferramentas, *frameworks* e *Application Programming Interfaces* (APIs) RIA surgem. Não é uma tarefa trivial manter-se atualizado com todas as possibilidades. São selecionados, portanto, alguns critérios para a escolha das tecnologias RIA deste trabalho. A justificativa de cada um deles segue:

- Requisitos do navegador: esse critério visa a restringir ao mínimo possível a necessidade de *software* adicional instalado no navegador;
- Suporte ao desenvolvimento: a existência de comunidades ou mesmo de suporte formal pago, garante o apoio aos desenvolvedores para os problemas de implementação de um SIG-Web;
- Maturidade: apesar de RIA ser uma idéia bastante recente, alguns ciclos de *releases* em uma determinada tecnologia oferecem uma confiança maior na sua utilização;
- Presença de APIs de terceiros e extensões: RIA é recente e sua utilização para SIG-Web ainda mais. Portanto sem a satisfação deste critério, uma tecnologia RIA sozinha

dificilmente ofereceria as ferramentas necessárias para esse tipo de *software;*

- Suporte em *Integrated Development Environment* (IDEs): IDEs aumentam a produtividade e são um quesito importante na escolha de uma tecnologia de desenvolvimento;
- Licenciamento *free software*: pela mesma razão que são necessárias extensões e APIs de terceiros, também podem ser necessárias mudanças na própria tecnologia. Mudanças essas que apenas o acesso ao código fonte faria possível;
- Existência de *framework*: linguagens como *Javascript* são fracamente tipadas, de difícil construção de testes unitários e *debug*. Além do que, muitas vezes exigem o uso de muitas bibliotecas externas nem sempre compatíveis. *Frameworks* eliminam ou diminuem a necessidade de escrever em *Javascript*, oferecendo um arcabouço uniforme e confiável de desenvolvimento.

1.4 – Engenharia *Web*

A *Unified Modelling Language* (UML) tornou-se a linguagem padrão para modelar aplicações orientadas a objetos. Tanto no mundo acadêmico quanto no mundo comercial, seus diagramas e definições são utilizadas em modelos de aplicações em geral. No entanto quando se trata de modelar aplicações *web* novos desafios se impõem, pois a navegação, interação com o usuário, tipos de conteúdos não encontram suficiente suporte no núcleo da UML. Tanto é assim que (PRECIADO; LINAJE; SANCHÉZ, 2005) cita 15 diferentes metodologias e linguagens para modelar sistemas *web*. Algumas baseiam-se em UML e outras não. RIA são uma evolução das aplicações *web* tradicionais. De acordo com os mesmos autores nenhuma dessas metodologias e linguagens são adequadas o suficiente para modelar RIA. RIA possuem uma série de características, como continuidade visual da interface, possibilidade de interação assíncrona com o servidor e incorporação de conteúdo multimídia que não são contempladas satisfatoriamente por nenhuma dessas metodologias.

1.4.1 - Metodologias de modelagem W*eb* existentes

Nesta seção, são apresentadas as metodologias de modelagem existentes para *web* e sistemas hipermídia. O objetivo é oferecer uma visão panorâmica do que já existe nessa área e que servirá de base para a UWE-R, a modelagem proposta nesta dissertação. Serão mostradas as 14 principais metodologias segundo (PRECIADO; LINAJE; SANCHÉZ, 2005). O referido estudo traz um resumo da avaliação da adequação dessas metodologias a RIA, no entanto não apresenta maiores detalhes de como chegou ao resultado da mesma. Com a presente seção, pretende-se abordar alguns desses detalhes e também fundamentar a UWE-R.

Duas metodologias citadas por (*PRECIADO; LINAJE; SANCHÉZ*, 2005) não foram detalhadas no presente trabalho: DMM+t e AHAM. A primeira por ser uma extensão de OMMMA-L, cujo estudo é feito neste trabalho. Tal extensão afeta um aspecto que não é relevante para RIA, como será explicado na seção de OMMMA-L. A segunda não foi inclusa pois, na verdade, se trata de uma solução para os chamados *Adaptive Hypemedia Systems* (AHS), ou sistemas que precisam ter seu conteúdo adaptado de acordo com um perfil de usuários modelados. É, portanto, uma problema diferente do que se pretende discutir nesta dissertação.

Apesar de não ser citada por (PRECIADO; LINAJE; SANCHÉZ, 2005), inclui-se uma outra metodologia, a *Web Application Extension* (CONALLEN, 2002) pela sua relevância tanto no mundo acadêmico quanto no mundo comercial. RUX também foi acrescentada, pois é a única, até o momento da escrita desse, que contempla aspectos de RIA.

O foco nesta seção está principalmente nos modelos de navegação e no projeto da interface com o usuário, uma vez que são esses os aspectos mais afetados com RIA. Portanto, apesar de fundamentais num contexto completo de um projeto real, não serão detalhadas outras fases como análise de requisitos, testes, implementação e modelos de dados (modelagem de objetos na WSDM ou RMDM na RMM e demais denominações em outras metodologias).

1.4.1.1 - HDM

O *Hypertext Design Model* (HDM) como definido por (*GARZOTTO; PAOLINI; SCHWABE*, 1993) é um modelo para descrever aplicações hipertexto. Nele, as informações são organizadas de modo hierárquico através de entidades, componentes e unidades. A relação entre esses é feito através de *links* que podem ser estruturais, de perspectiva ou de aplicações. Um *schema* é uma visualização esquemática das informações e seus *links*.

Uma entidade é uma estrutura de informação que representa um objeto no mundo real, como, por exemplo, um pedido de compra ou um mapa. Um componente é um desmembramento hierárquico de uma entidade, como, por exemplo, a lista de itens do pedido ou uma determinada camada do mapa. Uma unidade é a menor porção de informação representada. Um componente é feito de unidades, como um item de pedido faz parte de uma lista de itens ou uma feição específica é parte de uma camada do mapa. Há também o conceito de perspectiva, onde uma determinada porção da informação pode ser apresentada de vários modos. Um exemplo disso seria a representação em 2D ou 3D de uma determinada feição.

Os *links* servem tanto para demonstrar a relação existente entre as partes da informação quanto para permitir a navegação típica de um sistema hipertexto. Há *links* de perspectiva, que conectam unidades de um mesmo componente, como, por exemplo, um item de pedido e sua foto. Os *links* estruturais remontam a hierarquia da informação, como, por exemplo, do pedido como um todo para sua lista de itens. Os *links* de aplicação conectam entidades diversas, como um pedido e as suas informações de entrega.

O HDM foca devidamente os aspectos de uma aplicação hipertexto tradicional. Em especial no que se refere à estruturação da informação e a determinação de *links* implícitos. No caso de um SIG, a representação de um *schema* HDM com suas entidades, componentes, unidades e *links* poderia ser bastante útil no aspecto conceitual, mas não necessariamente no aspecto navegacional, uma vez que o mapa e grande parte das suas informações são apresentadas sob uma continuidade visual ao usuário. Não se trata de usar um conjunto de

hipertextos diferentes para camadas ou feições do mapa. Nem há *links*, no sentido navegacional, entre um mapa e sua camada. Essa continuidade visual é também uma característica típica de RIA, onde apenas porções da página são atualizadas.

Esse pressuposto do HDM fica explícito quando é afirmado: “a semântica *default* de navegação pressupõe que apenas unidades podem ser percebidas pelos leitores como 'nós-padrão, i.e., '*loci of navigation control*' e que apenas um nó é ativo em um dado instante” (*GARZOTTO; PAOLINI; SCHWABE*, 1993).

1.4.1.2 - WSDM

Web Site Design Method (WSDM) (DE TROYER; LEUNE, 1998) é uma metodologia centrada no usuário para modelagem de *sites web*. Trata-se de uma metodologia que difere de outras, como HDM que são orientadas aos dados que devem ser apresentados. Na WSDM o ponto de partida é o usuário. Entender seu perfil e suas necessidades é o primeiro passo dentro do seu conjunto de procedimentos.

A WSDM classifica os *web sites* em 2 tipos: “quiosque” e “aplicação”. O primeiro destina-se a mostrar de modo estruturado informação em páginas estáticas. O segundo prevê uma maior interação do usuário e a existência de páginas dinâmicas. O propósito da WSDM é cobrir o primeiro tipo. Essa é uma das suas limitações no que se refere a SIG-Web, em especial, às RIA, pois esses são exemplos do segundo tipo.

São as seguintes fases que o compõem: modelagem de usuário, projeto conceitual, projeto de implementação e implementação. Modelagem do usuário tem duas sub-fases: classificação de usuários e descrição das classes de usuários. Projeto conceitual tem duas sub-fases: modelagem de objetos e projeto navegacional.

Durante a classificação dos usuários são identificadas as atividades potenciais permitidas pelo *web site*. Apesar de o WSDM se focar em modelagem de sistes do tipo

quiosque, para fins de adequação ao tema, utilizaremos uma ilustração com aplicação (SIG). Por exemplo, para um SIG-Web hipotético, onde são possíveis as operações de planejamento de rotas, poder-se-iam listar os usuários especialistas em logística e usuário doméstico. Na fase de descrição das classes de usuários, cada um desses tipos seriam analisados e descritos em detalhes, tendo em vista suas necessidades diferentes. Nesse caso, a WSDM prevê a possibilidade de encarar ambos os usuários como herdando uma classe comum, mas como tem algumas características diferentes, teriam o conceito de "perspectiva" diferente. Uma perspectiva é um corte de diferenciação de uma classe de usuário a partir de uma base em comum.

A modelagem de objetos preocupa-se em analisar os dados a serem disponibilizados e suas relações de modo Orientado a Objetos. Assim, haveria uma classe Rota, com duas subclasses diferentes de acordo com as perspectivas de usuários definidas anteriormente. O projeto navegacional consiste na criação de diagramas que expressam as trilhas de navegação. Uma trilha é definida como a maneira pela qual usuários de uma determinada perspectiva navegam por um determinado conjunto de informações. O projeto de implementação é um modelo em que o *look and feel* do *site* é determinado. A implementação se ocupa em materializar esse último modelo.

Apesar de que há grande mérito em se colocar o usuário em primeiro plano, a WSDM possui limitações para RIA no que se refere à interação com o usuário e continuidade visual da interface, uma vez que não foi feita para esse propósito e sim para *web sites* tradicionais.

1.4.1.3 - RMM

Relationship Management Methodology (RMM) (*ISAKOWITZ; BALASUBRAMANIAN*, 1995) é uma metodologia para projetar sistemas hipermídia de modo estruturado. Trata-se de uma metodologia fortemente influenciada pelas metodologias de projeto de Sistemas de Bancos de Dados Relacionais. Define 7 conjuntos de procedimentos

em passos para sua execução: desenho do diagrama E-R, projeto das entidades (fatias), projeto navegacional, projeto do protocolo de conversão, desenho das telas da interface com usuário, desenho do comportamento em tempo de execução e construção. As 3 primeiras, no entanto, são tidas pelos seus próprios autores, como as mais fundamentais na RMM, portanto serão elas as tratadas aqui.

O *Relationship Management Data Model* (RMDM), produto central da RMM, descreve objetos de informação e mecanismos de navegação para aplicações hipermídia. Nesse modelo, são definidas entidades, atributos, relacionamentos, *links* e fatias. Entidades guardam o mesmo conceito desenvolvido em Sistemas de Bancos de Dados, ou seja, uma abstração para um objeto a ser modelado. Da mesma forma, atributos e relacionamentos são os mesmos conceitos: características das entidades e relação conceitual entre as entidades, delimitadas por multiplicidades (um-para-um, um-para-muitos, muitos-para-muitos), respectivamente. O conceito de fatias trata de um subconjunto dos atributos que interessam apenas a um determinado usuário. Assim, por exemplo, uma entidade CamadaDeMapaTematico poderia ser definida com uma série de atributos como nome, retângulo envolvente, tema, escala, projeção e *datum*. No entanto, para um usuário leigo, um *slice* CamadaDeMapaTematicoSimples poderia ser definido por uma fatia onde projeção e *datum* não seriam mostrados. Trata-se de uma idéia similar às perspectivas, estudadas em HDM e WSDM e ao conceito de visão em Sistemas de Bancos de Dados.

A fase de desenho do diagrama E-R segue os preceitos da disciplina de Sistemas de Bancos de Dados. A fase de desenho das entidades (fatias) segue a lógica que apresentamos no parágrafo anterior. No desenho navegacional, estão algumas contribuições interessantes da RMM, pois essa propõe a análise dos relacionamentos definidos no diagrama E-R para a criação dos *links* e apresenta tipos de *links* e uma notação específica. Há *links* do tipo: unidirecional, bidirecional, agrupamento, índice e *tour* guiado. Unidirecionais e bidirecionais são *links* que permitem a navegação entre fatias de uma mesma entidade. Um índice lista

entidades e permite acesso a cada uma delas. Um agrupamento permite o acesso a outros tipos de *links*, como se fosse um menu geral. Um *tour* guiado é um conjunto de *links* que modelam um assistente, ou seja, conjunto de páginas pré-determinadas com uma função específica e permite ao usuário ir para o passo seguinte ou retornar ao anterior. Os três últimos tipos podem ser condicionais, ou seja, um predicado deve ser satisfeito para que a navegação seja possível, por exemplo, CamadaDeMapaTematico (escala=”1:100.000”).

A RMM, apesar de meritória por concatenar um conjunto de procedimentos que servem de guia efetivo à construção de sistemas hipermídia, também não cobre os aspectos de RIA, assim como as metodologias apresentadas.

1.4.1.4 - HMBS

HMBS é a sigla para *Hypermedia Model Based on Statecharts* (OLIVEIRA; TURINE; MASIERO, 2001). Trata-se de um modelo que se baseia nos conceitos de máquinas de estados para descrever os sistemas *web*. Todos os conceitos existentes na definição formal de uma máquina de estados são aproveitados (conjunto de estados, função de hierarquia descrevendo estados dentro de estados, função de composição entre estados (*OR* e *AND),* função *default*, conjunto de expressões, conjunto de condições, conjunto de expressões de eventos, conjunto de ações, conjunto de rótulos, eventos/ações e conjunto de transições) e outros conjuntos e funções são acrescentados à tupla que define uma aplicação hipermídia: P, conjunto de páginas em que cada página é definida por conteúdo, título e conjunto de âncoras; m, função que mapeia os estados definidos na máquina de estados para páginas de P; ae, função que associa âncoras a eventos definidos na máquina de estados e N, que é o nível de visibilidade da aplicação para fins de controle durante a navegação nas páginas.

O HMBS tem uma grande vantagem de permitir, através da formalização exigida pelos conceitos de máquina de estados, algumas operações de modo automático, como verificação de alcançabilidade de uma página a partir de qualquer outra, e busca de ciclos de

navegação. Também permite representar atividades concorrentes, o que é particularmente útil no caso de RIA. No entanto, o modelo teria que ser estendido para compreender não somente reações a *links* (âncoras) e apresentações de páginas como quando trata da semântica de navegação. Na definição do HMBS, seria preciso estender o conceito de âncoras ou utilizar um termo melhor, que não remeta a *links*, para se referir a qualquer elemento de motivação de transição. A transição também não seria necessariamente entre páginas. Em RIA, uma transição pode ocorrer com o clique do *mouse* e a página ser mantida. Quanto à continuidade visual, há no HMBS, apenas uma estrutura que a permite: visões hierárquicas, mas é explicitamente utilizada apenas para navegação e não para composição em camadas mais genéricas. Uma solução para se ter essa continuidade no modelo seria utilizar a função de composição entre estados, utilizando o seu valor como *AND*, mas alterando sua semântica para que expresse a mesma continuidade que se tem em RIA.

1.4.1.5 - OOHDM

OOHDM (SCHWABE; ROSSI; BARBOSA, 1996) é um descendente direto do HDM. Aproveita, no entanto, os conceitos de Orientação a Objetos nas suas definições, daí seu nome. Nesta metodologia são definidas as seguintes fases: Projeto Conceitual, Projeto Navegacional, Projeto de Interface Abstrata e Implementação. No Projeto Conceitual, são definidas as classes, como abstrações dos dados a serem mostrados no sistema e suas relações. Um diagrama muito similar ao diagrama de classes é gerado, com a diferença de que um atributo pode ter mais de uma visão (camada de um mapa de terrenos mostrado com isolíneas ou com um mapa temático, por exemplo). Durante o Projeto Navegacional são expressos os esquemas de classes de navegação e de contexto de navegação. As classes de navegação mapeiam as classes do Projeto Conceitual para estruturas de navegação típicas de um sistema hipermídia, como nós, *links* e estruturas de acesso. O esquema de contexto de navegação expressa como o usuário navega no sistema entre os diferentes contextos de

navegação. Um contexto de navegação define as âncoras que estão disponíveis para um determinada classe de navegação. O Projeto de Interface Abstrata demonstra como as telas serão mostradas ao usuário final usando elementos de interface gráfica abstratos, ou seja, independente de qual seja a implementação do sistema (HTML, Toolbook ou outro). Seu principal produto é um diagrama de configuração contendo os *Abstract Data Views* (ADVs). Um ADV é um modelo formal orientado a objetos de elementos de interface gráfica. Podem ser agregados/compostos e especializados. Mostram em alto nível o *layout* da tela e como reagem a eventos de interação com o usuário. A Implementação preocupa-se, obviamente, com a transposição desses modelos em um conjunto de códigos que podem ser colocados em um servidor *web* e acessados por um navegador.

O OOHDM foi pensado para sistemas hipermídia antes de RIA e, portanto, está permeado de conceitos como âncoras e *links* que seriam os principais modos de navegar em um sistema hipermídia. No entanto, é possível, com algumas adequações, utilizá-la para RIA. Há, por exemplo, como expressar que um ou vários nós ficam ativos quando da ativação de um *link*. No entanto, o conceito de navegação precisa ser estendido. Literalmente no referido trabalho: "toda transformação navegacional causa uma transformação na interface". Isso claramente não é verdade em RIA. Essa é uma restrição da OOHDM por conta do modo como aplicações *web* tradicionais funcionam. É notável, entretanto, que de acordo com (ROSSI *et al.*, 1995) um diagrama de configuração de ADV, que representa as transições utilizando a idéia de máquinas de estado, é suficientemente abstrato para permitir a continuidade visual requerida por RIA. Bastaria explicitar quando as transições implicam em transformação na interface e quando não (esse último, o caso típico de RIA).

1.4.1.6 - UWE

UWE (KOCH *et al*., 2002) é uma metodologia que propõe a utilização da já conhecida *Unified Modeling Language* (UML) para a modelagem de sistemas *web*. Portanto,

diferentemente das demais metodologias vistas até agora, que propõem notações e diagramas específicos, a UWE se vale de um padrão já sedimentado tanto em meios acadêmicos quanto comerciais. São propostas extensões através de mecanismos já previstos na linguagem. Esses mecanismos são estereótipos, valores rotulados (*tagged values*) e restrições (*constraints*).

A metodologia se divide nas seguintes fases Especificação de Requisitos, Modelagem Conceitual, Modelagem Navegacional e Modelagem de Apresentação, Modelagem de Cenários, Modelagem de Tarefas e Modelagem de Implantação. As duas primeiras fases utilizam diagramas bem conhecidos em UML e sem alterações em relação à modelagem de sistemas não *web*: diagramas de casos de uso e de classes. Na modelagem navegacional são previstos 2 tipos de modelos: de espaço navegacional e de estrutura de navegação. O primeiro é usado com um nível maior de abstração durante a análise, enquanto o segundo mostra em detalhes os nós intermediários, índices e páginas de busca. Nesses, aparecem as primeiras extensões à UML tradicional: estereótipos são utilizados para representar classes de navegação, índices, menus e páginas para entrada de parâmetros de busca. As relações entre essas classes estereotipadas, em vez de denotar herança ou outras associações, indicam navegação (possibilidade de sair de um nó e chegar a outro), que nada mais é que uma associação estereotipada. Valores rotulados indicam se índices são ordenados. Na Modelagem da Apresentação, uma forma particular de diagramas de classes é utilizado. Trata-se de uma forma que permite mostrar a relação de composição entre as classes onde elas aparecem visualmente aninhadas, como numa representação esquemática de uma página e seus elementos de interface. Nesse diagrama, existem classes de apresentação (outro estereótipo) que representa a tela mostrada ao usuário. Para a Modelagem de Cenários, são utilizados diagramas de estado onde cada estado indica que uma classe de apresentação (da modelagem respectiva) foi apresentada ao usuário. A Modelagem de Tarefas se beneficia de diagramas de atividades para demonstrar como um usuário, utilizando o sistema *web* modelado, consegue executar uma determinada tarefa. Por fim, diagramas de *deploy* são utilizados para a

Modelagem de Implantação.

Nada indica nos diagramas de estrutura de navegação a continuidade visual requerida em aplicações RIA. No entanto, nada impede que sejam adicionadas essas informações, uma vez que a UWE não restringe estritamente a *links* no sentido tradicional de aplicações *web*. O outro ponto que mereceria adequação é a associação um para um entre estado e classe de apresentação que é feita na Modelagem de Cenários, ou seja, a transição indica que se saiu de uma classe de apresentação para outra, sendo que em RIA a classe poderia permanecer a mesma, tendo apenas alguns de seus atributos modificados. Indicações de uma interação assíncrona com o servidor *web* também poderiam ser benéficas para expressar melhor os mecanismos existentes em RIA.

1.4.1.7 - OO-H

OO-H ou *Object-Oriented Hypermedia* (GÓMEZ; CACHERO, 2003) é uma metodologia orientada a objetos para a modelagem de sistemas hipermídia. Possui um processo de projeto similar ao proposto pela UWE, mas acrescenta algumas características inovadoras, como: catálogo de padrões, diagrama de acesso navegacional (NAD) e modelo da camada de apresentação (diagrama de apresentação abstrata (APD) e diagrama de *layout* composto (CLD)). Esses últimos três diagramas representam uma extensão a metodologias tradicionais para expressar as necessidades específicas de aplicações *web*. No entanto, tais extensões utilizam UML, com uso de estereótipos e restrições assim como UWE.

Inicialmente são descritos os diagramas de casos de uso e de classes, como em uma modelagem tradicional. Os casos de uso são separados em alvos de navegação (NT), onde cada NT representa um grupo de classes de navegação similares. Classes de navegação (NC) são classes derivadas a partir do diagrama de classes tradicional que representam as entidades conforme elas serão apresentadas durante a navegação. Um exemplo de NT seria, num SIG-Web onde são mostradas as camadas de vários mapas provenientes de servidores WMS, a

Gerência de Servidores WMS. Esse NT teria todas as NC's relacionadas com inclusão e configuração dos servidores WMS e seus atributos escolhidos (projeções, formatos das imagens desejadas etc.). Um outro NT estaria relacionado com a Visualização e Manipulação dos mapas resultantes. São feitos NAD's para cada NT. Filtros na forma de restrições da UML são utilizados para denotar as propriedades de um *link* de navegação e por conseguinte determinar se uma NC é mostrada ou não. Um APD é derivado do NAD, expressando os elementos de interface presentes em cada elemento da interface gráfica, bem como sua localização e estilo sem, no entanto, definir detalhes de implementação.

OO-H traz alguns conceitos que são de interesse geral na modelagem de sistemas *web* e poderiam ser aproveitados e estendidos para RIA. Um deles é o uso de padrões de navegação (indexação, menus, *tours* guiados etc.) e um catálogo para os mesmos. Com isso, tem-se uma formalização dos possíveis modos de navegação e abre-se caminho para geração automática ou semi-automática de materializações dos APD's para HTML ou XML. Em algumas tecnologias RIA, o código que representa os elementos de interface são escritos em XML, como visto anteriormente. Há também definições de vários tipos de *links* de navegação. Uma possível extensão para RIA seria incrementar os *links* do tipo S e R (serviço e resposta) com a noção de assincronia.

1.4.1.8 - WebML

WebML (CERI; FRATERNALI; BONGIO, 2000) é uma metodologia que acompanha uma anotação própria a fim de modelar *web sites*. Ela define uma série de modelos: Modelo Estrutural, Modelo hipertexto (compreendidos de Modelo de Composição e Modelo de Navegação), Modelo de Apresentação e Modelo de Personalização. No Modelo Estrutural são expressos os dados que serão mostrados no sistema. Não define uma nova notação, mas alega ser compatível com modelo E-R e UML. No Modelo de Composição são definidas as partes dos conteúdos (expressos no Modelo Estrutural) que serão apresentados. Nesse modelo são

introduzidos os conceitos de unidades de conteúdo ou simplesmente, unidades. Uma unidade é uma abstração para um modo de apresentação de um conteúdo. São tipos de unidades: unidades de dados (uma instância de uma entidade de dados), multi-dados (várias instâncias da mesma entidade), índices (apontadores para as instâncias), filtros (permitem buscar as entidades a partir de chaves de busca), *scroller* (mostra entidades com referências de navegação: próximo, anterior, primeiro e último) e unidades diretas (referência não mostrada para uma entidade). Há uma notação gráfica para cada uma dessas unidades e uma expressão em XML, a fim de permitir uma geração automática de códigos de implementação.

O Modelo de Navegação nada mais é que um diagrama que usa a notação gráfica de cada uma das unidades definidas no Modelo de Composição e acrescenta uma notação para uma página ou *frames* (um retângulo com linha descontínua envolvendo as unidades) e setas para identificar os *links* entre as unidades. Os *links* são definidos como contextuais ou não contextuais. Os primeiros implicam na existência de relação estrutural (a representação gráfica de uma feição e seus atributos, por exemplo) entre as unidades ligadas e o segundo não.

É relevante notar que o conceito de cadeias de navegação apresentado nesta metodologia é muito similar ao de padrões de navegação do OO-H. Outro aspecto de interesse é a definição formal de relações entre unidades de dados, que permite checar a validade de um modelo de navegação automaticamente. Outras checagens de validade podem ser feitas, como alcançabilidade de uma página e existência de contexto para os *links* contextuais.

Do ponto de vista da adaptação para RIA, no Modelo de Composição o elemento "*alternative"* pode denotar a continuidade visual, desde que seja explicitado que o *link* não implica em recarga da página. Outro item a ser aproveitado é a chamada navegação escondida, em que uma determinada unidade de dados fica com seu valor pendente de ser mostrado. Seria necessário, no entanto, acrescentar uma quarta possibilidade dentre as apresentadas para definir quando o valor será mostrado: ao carregar assincronamente o dado,

como RIA permite.

1.4.1.9 - W2000

W2000 (BARESI; GARZOTTO; MARITATI, 2002) se auto entitula como sendo "a última herdeira do HDM". Ela é descrita através de *MetaObject Facility* (MOF, 2008). MOF é um padrão do *Object Management Group* (OMG) utilizado como metamodelo, ou seja, um modelo para fazer outros modelos. O MOF surgiu como uma necessidade de se formalizar a especificação de modelos como a própria UML. Curiosamente utiliza para tal apenas as notações disponíveis no diagrama de classes da UML. Uma dessas notações indica os pacotes, que são agrupamentos de classes. Uma classe é uma abstração para uma entidade real que está sendo modelada. No MOF da W2000 são definidos os seguintes pacotes, um para cada modelo da metodologia: Informação (com sub-pacotes de Elementos Comuns, Hiperbase e Estruturas de Acesso), Navegação, Apresentação e Comportamento Dinâmico (com os seguintes sub-pacotes: Diagramas de Estados, Cenários, Operações e Atividades).

No Modelo de Informação e seus sub-pacotes são definidos os elementos que são modelados na W2000, bem como as possíveis relações entre eles. Ele herda do HDM a noção de Entidade bem como as possíveis composições das mesmas em unidades menores, aqui denominadas Componentes e *Slots*. Associações semânticas entre as Entidades ou suas unidades menores, bem como entre coleções das mesmas são modeladas também.

No Modelo de Navegação, o conteúdo é organizado em nós. Um nó pode ser uma Entidade, suas menores unidades ou coleções e até mesmo as associações. A ligação entre nós se dá através de Relações de Acessibilidade, que podem ser inferidas das Associações Semânticas do Modelo de Informação.

O Modelo de Apresentação define Unidades de Apresentação como sua menor unidade. Seu agrupamento faz a Seção, cujo agrupamento forma uma Página. Cada um desses

componentes podem ligar-se entre si através de *links*. Há três tipos de *links*: foco (mantém o usuário na mesma página mas desloca sua área de visão para outra unidade de apresentação), intra-página (navegação entre instâncias da mesma Página) e página (navegação entre Páginas de tipos diferentes).

Há também um modelo de Comportamento Dinâmico onde se definem Estados relacionados com os elementos modelados e Operações que podem ser feitas para forçar a transição entre estados.

Analisando a adequação para RIA, nenhum dos tipos de *links* (foco, intra-página e página) refletem o que é possível com RIA. No entanto, na explicação dos pressupostos do pacote de Comportamento Dinâmico da W2000, há a possibilidade subliminar de os estados da aplicação hipermídia terem a interferência de sistemas externos. Apesar de não citar explicitamente o modo assíncrono, citam sistemas externos que o são (como um servidor de *Short Message System* (SMS), responsável por enviar mensagens de texto de celulares). Bastaria explicitar que o servidor HTTP que responde a uma requisição assíncrona é um desses sistemas externos. O Modelo de Operações, constante no pacote de Comportamento Dinâmico, fala de alguns tipos de operações quanto à multiplicidade de passos ou de seu aspecto transacional, mas não discrimina-as quanto à assincronia explicitamente.

1.4.1.10 - OMMMA-L

A sigla OMMMA-L (SAUER; ENGELS, 1999) significa *Object-Oriented Modeling of Multimedia Applications*. Como o próprio nome indica, trata-se de uma metodologia voltada para sistemas multimídia e não *web*. No entanto, as aplicações *web* tem convergido cada vez mais para terem aspectos de multimídia, portanto faz bastante sentido tratar as metodologias relacionadas nesta seção. No caso de SIG para *web*, por exemplo, a composição de mapas com vídeos e áudios não é uma novidade. Também como o nome indica, são usados os conceitos de Orientação a Objetos, mais especificamente, a notação UML. São utilizados

os seguintes diagramas nesta metodologia: classes, apresentação, seqüência (estendido) e estados.

O diagrama de classes é usado para explicitar a estrutura lógica da aplicação. Além disso, mostra as classes que se são mídias e seus relacionamentos com classes que são da lógica da aplicação de modo explícito (por exemplo, uma classe para Feição e outra para VideoDaFeicao).

O diagrama de apresentação é um diagrama novo, ou seja, não existente na UML tradicional que mostra o *layout* de objetos de visualização e de interação. Um objeto de visualização é um objeto passivo como um texto, imagem ou vídeo. Já um objeto de interação são elementos de interface como botões, campos de texto barras de rolagem entre outros.

O diagrama de seqüência estendido mostra a linha do tempo com marcadores, diferentemente do diagrama tradicional. Também há uma série de refinamentos para expressar atrasos entre mídias sendo tocadas, bem como a composição entre elas. A relação entre as classes de mídia e as classes de negócio é reforçada neste diagrama.

O diagrama de estados é aproveitado para demonstrar os estados da aplicação e suas transições. Uma alteração sintática feita na UML tradicional é que uma ação interna de um estado pode estar relacionada com um diagrama de seqüência e não só a métodos. Isso foi necessário, segundo os autores, para expressar o controle interativo de uma mídia com as partes não interrompíveis de uma aplicação multimídia.

Para uma possível adaptação a RIA, esta é a primeira metodologia estudada que faz uso de UML a mencionar a utilização de mensagens síncronas e assíncronas, como já é padrão em UML nos diagramas de seqüência. Sob o ponto de vista de continuidade visual, a aplicação multimídia tem maior similaridade com uma RIA, no entanto a riqueza de elementos de interação do diagrama de apresentação deveria ser aumentada consideravelmente a fim de se expressar o poder das tecnologias RIA.

No trabalho de (PRECIADO; LINAJE; SANCHÉZ, 2005) há a referência ao que seria uma metodologia à parte, a DMM+t (*HAUSMANN; HECKEL; SAUER*, 2001). No entanto, o estudo nos mostra que na verdade trata-se de uma extensão ao OMMMA-L, que permite uma semântica mais precisa para representação da sincronização entre diferentes mídias (por exemplo, sincronização entre um áudio e seu respectivo vídeo).

1.4.1.11 - DEMAIS

DEMAIS (BAILEY; KONSTAN; CARLIS, 2001) é a sigla para *DEsigning Multimedia Applications with Interactive Storyboards*. Trata-se de uma ferramenta baseada em desenhos de diagramas (*sketches*). O objetivo é capturar as idéias do projetista (*designer*) da aplicação multimídia o mais cedo possível.

O projetista usa a ferramenta desenhando traços à mão livre com um periférico adequado (*pen tablet*). A ferramenta tenta reconhecer os traços como objetos ou comportamentos. Os objetos podem ser mídias, quando o traçado se assemelha a um retângulo ou textos, quando o usuário pressiona e segura a caneta no *tablet* por alguns instantes. Há uma série de símbolos usados para descrever comportamentos, principalmente relacionados com sincronização de mídias. Textos, se de acordo com uma gramática bem simples, podem ser transformados automaticamente em comportamentos gerenciados pelo sistema, como por exemplo: “Quando este áudio terminar, navegue para o nó X”.

Apesar de ter um grande apelo para a construção de aplicações multimídia em um estágio inicial, não há de fato uma metodologia, no sentido de um conjunto de artefatos que são gerados em uma determinada ordem seguindo determinadas regras para levar a produção de um sistema final. Não oferece, por exemplo, uma visão hierarquizada do conteúdo ou dos nós de navegação.

1.4.1.12 - HMT

Hypermedia Modeling Technique ou HMT (SPECHT; *ZOLLER*, 2000) baseia-se na RMM. A HMT estende a RMM principalmente no que se refere a primitivas de projeto para expressar sincronização de tempo, consultar e manter os dados na camada de modelo.

Como uma metodologia completa, divide-se em 6 passos: Análise de Requisitos, Projeto Entidade/Relacionamento, Projeto Conceitual de Hipermídia, Projeto de Autorização, Projeto Lógico de Hipermídia e Projeto de *Layout*.

Na Análise de Requisitos, o domínio do problema é descrito e são identificados os usuários. Uma especificação do uso e das funcionalidades do sistema é gerada. No Projeto Entidade/Relacionamento, um diagrama E/R é gerado tendo em vista as entidades do domínio e suas relações. O Projeto Conceitual de Hipermídia é o cerne da HMT. Nesse passo, os dados definidos no diagrama E/R são agrupados em documentos e a navegação é definida. O Projeto de Autorização define papéis e restrições de acesso aos documentos definidos na fase anterior. O Projeto Lógico de Hipermídia define as restrições temporais, questões de sincronização, bem como os rótulos e descrições dos elementos. No Projeto de *Layout* a interface, que será percebida pelo usuário, é definida.

Para fins deste estudo, o diagrama de navegação, discutido na fase de Projeto Conceitual de Hipermídia merece alguns comentários adicionais. Nele, o conceito de documento é a unidade básica. É derivada do diagrama E/R, podendo expressar uma entidade, um relacionamento ou agrupamentos desses. No diagrama de navegação, há primitivas relacionadas com *links* estruturais, índices, *tours* guiados e a novidade em relação ao RMM: o *slide show*. A diferença do *slide show* para o *tour* guiado é que aquele tem um tempo pré-determinado após o qual o nó seguinte é automaticamente mostrado no lugar do nó anterior. Há também primitivas que indicam a sincronicidade, ou seja, ao executar uma navegação pode-se indicar se uma apresentação de uma mídia é interrompida ou não. Essas primitivas são baseadas no conceito de contexto de *link*, que estende a noção de *link* estrutural pura do

RMM.

É proposto um conjunto de primitivas, no Projeto Lógico de Hipermídia, que definem uma série de relações temporais entre os nós do sistema *web*, tais como: execução seqüencial, execução após atraso, execução paralela com sincronização inicial ou final etc.

Do ponto de vista de RIA, o modelo apresentado sobrecarrega várias noções: relações estruturais entre os dados, *layout* e navegação em um mesmo diagrama. O ideal seria separar isso, como outras metodologias estudadas aqui o fazem. Sem tal separação, uma descrição de interface com a riqueza típica de uma RIA seria de difícil entendimento. Além disso, apesar da idéia de sincronização ser explicitada e formalizada, não é discutida como representar a requisição assíncrona de dados para a camada de modelo.

1.4.1.13 - HFPM

Hypermedia Flexible Process Modeling ou HFPM (OLSINA, 1998) é uma metodologia que pode ser usada em conjunto com outras já citadas aqui, mas se preocupa com alguns aspectos do desenvolvimento de uma aplicação *web*, como, por exemplo, o uso de padrões de projetos, gerência, qualidade, entre outros.

Define uma série de tarefas a serem realizadas: Modelagem dos Requisitos de *Software*, Planejamento de Projeto, Modelagem conceitual, Modelagem Navegacional, Modelagem de Interfaces Abstratas, Emprego de Padrões de Projeto, Edição e Captura de Dados multimídia, Modelagem/Integração Física, Validação/Verificação, Garantia de Qualidade, Gerência e Coordenação de Projeto e Documentação. Muitas delas, como se percebe, tem a ver com aspectos gerenciais ou de boas práticas e não necessariamente apenas com as atividades de engenharia *web* propriamente ditas. Outros itens listados, na verdade são preocupações que devem fazer parte do processo de desenvolvimento como um todo.

Por ter essa preocupação mais abrangente com o projeto de desenvolvimento do

sistema *web*, a HFPM se refere a outras metodologias para as partes relacionadas aos artefatos de análise e projeto. Por exemplo, RMM ou HDM para o modelo conceitual. No entanto, usa o um diagrama de classes/pacotes da UML e cita o modelo conceitual da OOHDM como aquele que deve ser usado. Para o diagrama de navegação também usa OOHDM.

Do ponto de vista de RIA (conceitos de *links* com acesso assíncrono, modelo de *layout* de telas com componentes sofisticados, entre outros) não acrescenta nenhum conceito que outras metodologias não tenham antes descrito. Portanto as mesmas sugestões e críticas cabem aqui.

1.4.1.14 - WAE

Web Application Extension (WAE) é uma extensão da UML proposta por (CONALLEN, 2002) para se modelar aplicações *web*. Utiliza-se dos já referidos mecanismos de estereótipos, valores rotulados e restrições existentes na UML para promover essa extensão.

Como metodologia apoia-se no *Rational Unified Process* (RUP) para definir as fases, ou seja, preconiza as fases: Modelagem de Negócios, Levantamento de Requisitos, Análise e Projeto, Implementação, Testes e Implantação, de modo iterativo e não em cascata.

Do ponto de vista de artefatos usados na fase de Análise e Projeto não há diagramas específicos para modelar a navegação e o *layout* das páginas, como em outras metodologias apresentadas nesta seção. Para essas modelagens são aproveitados os diagramas de classes e de seqüência, com classes estereotipadas. Essas classes estereotipadas modelam páginas no servidor (como JSP, PHP, ASP e outras) e páginas no cliente (estáticas ou com *scripts* e *applets*). Associação estereotipada <<*build*>> marca a geração de uma página cliente por uma página servidora. Há também associação <<*redirect*>> para marcar redirecionamento entre páginas servidoras. A ligação entre duas páginas é representada por uma associação

estereotipada <<*link*>>. Há também outros estereótipos para representar formulários e *frames*.

Há vários exemplos de utilização do diagrama de classes, que é um diagrama estático, para representar situações dinâmicas, que seriam mais bem representadas com diagramas de seqüência. Os diagramas de seqüência permitem uma visão temporal e comportamental da aplicação.

Do ponto de vista de RIA, apesar da WAE prever execuções de parte da aplicação no lado do cliente, não há uma discussão mais aprofundada de como representar seus aspectos com as extensões descritas. Um desses aspectos é o *link* que faz uma requisição de modo assíncrono ao servidor. Isso poderia ser representado acrescentando um valor rotulado no estereótipo <<*link*>>, proposto pela WAE. Uma outra solução seria utilizar mensagens assíncronas nos diagramas de seqüência, como já previsto na especificação da UML.

1.4.1.15 – RUX

Rich User eXperience Method (RUX) (*PRECIADO; LINAJE; SANCHÉZ*, 2008) e (*PRECIADO; LINAJE; SANCHÉZ*, 2007) é uma metodologia do tipo MDD (*Model-Driven Development*) para permitir a modelagem da adaptação de aplicações *web* tradicionais para RIA. Ela tem aspectos muito interessantes pois permite a utilização de outras metodologias já existentes como WebML ou UWE e através de uma série de transformações dos modelos dessas outras metodologias, chega a representações úteis para RIA. Inicialmente, é feita uma transformação para um modelo de interface abstrata. Em seguida, deve ser modelada a interface concreta através de um conjunto de modelos. Entre esses modelos, há representações temporais, espaciais e de interação com o usuário. Por fim, uma transformação da interface concreta para a interface final gera o código necessário para alguma tecnologia RIA específica (Laszlo, Flex, Ajax etc.).

Utiliza-se de vários outros padrões e linguagens para representar os modelos de

interação, temporal e espacial, como por exemplo: *Extensible User Interface Components Language* (XICL) para a definição dos componentes de interface gráfica, *User Interface Modeling Language* (UiML) para representação da interface abstrata, *Synchronized Multimedia Integration Language* (SMIL) para representar a relação de lógica temporal entre os componentes da interface e *XML Events* para representar a interação entre usuários e a aplicação.

Diferentemente das metodologias analisadas até agora, ela não foi feita para aplicações *web* tradicionais, mas utiliza-se de metodologias já existentes para criar modelos de aplicação RIA. Portanto, não faz sentido discutir sua extensão para RIA, mas sim a razão de não a utilizarmos e propor uma extensão própria. Isso será feito na seção 3.

2 – ESTUDO COMPARATIVO DE TECNOLOGIAS RIA APLICADAS A SIG-Web

2.1 – Introdução

Nesta seção, é feito um estudo que compara a utilização de 3 tecnologias RIA para um conjunto de operações básicas de um SIG-Web. Inicialmente, será apresentado o padrão *Scalable Vector Graphics* (SVG), utilizado neste estudo. As tecnologias escolhidas serão apresentadas e suas escolhas justificadas. Também serão apresentados alguns *sites* que utilizam as tecnologias RIA citadas para discutir sobre a hipótese de uso das mesmas em SIG-Web.

Citando (DAVIS; LAENDER, 1999) as operações de um SIG podem ser divididas nos seguintes grupos: geométricas, generalização de mapas, análise espacial e auxiliares. Tendo em vista que o objeto de estudo são os SIG-Web, muitas dessas operações não são aplicáveis, pois o usuário neste contexto não é um fotogrametrista ou cartógrafo ou outro especialista em Geomática. Não são consideradas, portanto, as operações relacionadas com produção de mapas, correção ou aquisição de dados e outras operações mais sofisticadas. Serão objeto do estudo as seguintes operações: 1 – Colapsamento e ampliação (*zoom*): diminuição ou aumento do detalhamento do mapa sem alteração na janela de visualização, 2 – Deslocamento: visualização de uma outra porção do mapa, 3 – Simbolização: associação de um símbolo a uma feição ou porção de uma feição do mapa, 4 – Sobreposição de polígonos: sobreposição de feições ou porções de feições em um mapa e 5 – Seleção: seleção feita pelo usuário a uma feição ou porção de uma feição um mapa.

2.2 – O padrão SVG

A fim de discutir o suporte a operações no formato vetorial para SIG-Web, o padrão

Scalable Vector Graphics (SVG) será considerado. O SVG (SVG, 2008) é um padrão XML do *World Wide Web Consortium* (W3C) utilizado para descrever gráficos 2D. SVG é um formato bastante flexível que permite descrever linhas, pontos curvas (elipses, circunferências, *beziers*) e polígonos. É também possível definir gradientes, filtros, transformações, animações, preenchimentos, contornos, códigos em *script* e metadados. Como um exemplo simples, segue um extrato de SVG ilustrando o seu uso para descrever a camada de um mapa:

```
    <svg width="100%" height="100%" viewBox="-189 -87.97 378
182.31">
    <path style="fill:white;stroke:red;stroke-width:2"
id="countries_2133" attrib:__gid="38291" attrib:cat="32"
attrib:fibs="BR" attrib:name="Brazil" attrib:f_code="FA001"
attrib:total="184101109" attrib:male="91044573"
attrib:female="93056536" attrib:ratio="97.799999999999997"
d="M-60.74 -5.21l.31 .03l.23 -.09l.23 ...>
```

A *viewBox* pode ser utilizada para descrever o retângulo envolvente relacionado a coordenadas latitude-longitude. A *tag* "*path*" é utilizada para descrever os atributos da feição. Existem atributos de preenchimento e contorno bem como um identificador (id) para ela. Todos os atributos "*attrib*", que são opcionais, podem ser personalizados para oferecer dados específicos da aplicação para uma feição específica. Obviamente esses dados viriam de um banco de dados geográfico. O atributo "d" descreve os dados do "*path*", ou seja, a geometria da feição. Nesse caso, move-se (vide letra "M" para aquele comando) para a posição -60.74 e -5.21 (respectivamente x e y) e de lá, desenha-se uma linha (vide a letra "l") para 0.31 e 0.3. Depois, a partir daquele ponto, outra linha é desenhada para 0.23 e -0.9, e assim por diante.

Claro que é possível desenvolver um SIG-Web vetorial sem suporte de SVG, e serão discutidas algumas estratégias quando SVG não estiver disponível. No entanto, uma vez que

se trata de um padrão do W3C e vários servidores *Web Map Service* (WMS) geram camadas de mapas em SVG, esse será o formato preferido neste estudo. Existem ainda outras razões que levaram a essa decisão. Inicialmente porque entende-se que é fundamental separar dados geográficos da lógica de negócios (código da aplicação SIG-Web). Misturar dados com lógica no mesmo lugar é péssimo *design* e torna extensibilidade, manutenibilidade e documentação quase impossíveis em um projeto de *software*. Especialmente se for considerado que tipicamente uma feição de interesse tem uma geometria bastante complicada. Portanto, escrever código que descreva a geometria dentro da aplicação deve ser evitado. Eis que SVG se mostra útil, pois permite tal separação e faz com que o desenvolvedor se concentre na lógica do SIG-Web. Essa também é a razão para se considerar SVG como um formato de dados e não como uma tecnologia RIA completa, apesar de ser possível escrever a lógica da aplicação em *Javascript* dentro do mesmo arquivo SVG.

Outras razões que demonstram o interesse em utilizar SVG para SIG-Web são: SVG é suportado nativamente em navegadores como Firefox e podem ser abertos utilizando um *plugin* da Adobe no *Internet Explorer* – apesar de que para os propósitos deste trabalho, as próprias tecnologias RIA devem ser capazes de renderizar SVG. Existem em (CARTO.NET, 2008) *scripts* que geram SVG a partir de arquivos no formato SHAPE da ESRI. Naquela mesma referência é possível encontrar uma série de outros trabalhos que mostram como SVG pode ser útil em SIG-Web. Existe até uma iniciativa no Japão (SVG MAP LAB, 2008) que se propõe à criação de software e infra-estrutura para utilização de SVG em SIG.

Não obstante isso, há também alguns problemas com SVG. Em alguns casos, quando há uma grande quantidade de feições complexas pode-se levar muito tempo para renderizá-las (em um computador Pentium Dual *Core*, 2.8 Ghz e 2 GB de RAM, houve instâncias de SVG que levaram aproximadamente 30 segundos para serem mostradas). Alguns arquivos SVG não foram renderizados pelas tecnologias RIA utilizadas por não estarem conformes ao padrão. No que se refere a questões geográficas, SVG faz uma representação duplicada das linhas de polígonos vizinhos, o que acrescenta um *overhead* desnecessário. Também falta

informação de topologia geométrica (vizinhança de polígonos, ponto dentro de polígono, polígono dentro de polígono), como descrito em (BURROUGH; MCDONNELL, 1998).

2.3 – Tecnologias RIA escolhidas

Existem inúmeras tecnologias RIA disponíveis. Muitas delas são baseadas em AJAX (ZAKAS; MCPEAK; FAWCETT, 2007). Algumas requerem uma grande quantidade de código a ser escrito em *Javascript* e outras oferecem um contexto de *framework* a fim de evitar código em linguagem *script* tanto quanto possível. Linguagens *script* são mais difíceis de se desenvolver em projetos maiores uma vez que a compilação não é possível, testes unitários são dificultados (se não impossibilitados) e o suporte em IDEs é mais fraco do que em linguagens compiladas. *Javascript* também tem outras desvantagens como a tipagem fraca e problemas de compatibilidade entre diversos navegadores. Portanto, apesar de existir uma grande variedade de tecnologias RIA baseadas estritamente em *Javascript*, elas não serão consideradas nesta dissertação. São consideradas as seguintes: (GWT, 2007), (FLEX, 2007) e (OPENLASZLO, 2007). Muitas das razões pelas quais elas foram escolhidas estão explicadas em 1.3.

Existem algumas tecnologias para integrar *Yahoo! Maps* ou *Google Maps* a Flex, Openlaszlo e GWT. Para o propósito deste trabalho, elas não foram consideradas. O objetivo aqui é focar em tecnologias RIA para um uso mais geral em SIG-Web. Um uso que permita ao desenvolvedor incorporar dados geográficos próprios a uma aplicação específica. Utilizar aquelas APIs força também a utilização dos dados (fotos de satélite e outros daqueles provedores), além dos seus esquemas de licenças restritivas.

A ESRI desenvolveu uma API baseada em Flex 2.0.1 para acessar o *ARCWeb Services: ArcWeb Explorer Flex*. Apesar de ser uma API poderosa, não é considerada aqui, pois só funciona com o referido produto da ESRI, que não está sob licença aberta ou livre.

XForms é um padrão do W3C para definir, independente de dispositivo, formulários e

suas interações. É uma tecnologia RIA, mas é basicamente limitada a formulários, por isso, não é considerada neste estudo, uma vez que SIG-Web imprescinde da interação com mapas, guardados em formato *raster* ou vetorial.

WPF da Microsoft não é estudado aqui pois só funciona no navegador Internet Explorer. XUL, também desconsiderada, é uma poderosa linguagem de definição de interfaces que permite a criação de RIA, mas é limitada a navegadores como Firefox, Mozilla e Netscape. Seu funcionamento no *Internet Explorer* não é trivial para usuários comuns.

Applets Java também podem ser usados como uma tecnologia RIA, mas tem comportamento não uniforme em diferentes navegadores. Há também uma desvantagem para SIG-Web: um *applet* tem que ser assinado digitalmente caso tente fazer uma conexão de rede a um servidor diferente daquele de onde veio. Faz parte da *sandbox* de segurança de um *applet* que possui outras restrições. Isso é particularmente problemático se considerarmos a possibilidade de fazer requisições a servidores WMS e WFS, localizados em domínios diversos.

Algumas tecnologias RIA são ainda mais recentes e encontram-se em beta, no momento da escrita desta dissertação. Elas também têm em comum o objetivo de rodarem em plataformas diferentes da *Web*, como o *desktop* e até celulares. Vale a pena manter-se informado acerca dos seus desenvolvimentos, mas são anotados aqui apenas para fins de completude. JavaFX é a tentativa da Sun Microsystems de igualar, no lado do cliente, o poder que sua linguagem Java tem no lado do servidor. Trata-se de uma linguagem de *script* para construir RIA em ambientes Java. Apollo (também conhecido como AIR) é um projeto da Adobe que permitirá basicamente fazer com que uma aplicação escrita em Flex rode transparentemente no navegador ou no *desktop*. Silverlight (conhecida anteriormente como WPF/E) é a iniciativa da Microsoft de melhorar o WPF e fazê-lo rodar em vários navegadores. Orbit é um projeto colaborativo entre a Sun Microsystems e Laszlo Systems para fazer com que o Openlaszlo rode na plataforma Java Micro Edition (JME), ou seja, em celulares, telefones, *pagers*, *set-top boxes* e afins.

Todas as tecnologias RIA escolhidas aceitam os formatos de imagens mais populares, como PNG, GIF e JPG. Elas também possuem formulários para entradas de dados e possuem uma API rica e de fácil uso para componentes como *tooltips*, *tabbed panes*, *accordeon panes*, vídeos, entre outros.

2.3.1 – Flex

Flex é a tecnologia RIA da Adobe. É preciso conhecer MXML (um extensão de XML) e ActionScript para o desenvolvimento das aplicações. A Adobe anunciou que o SDK (sem o IDE) será software livre. Há suporte para SVG, pré-carregamento e *caching* de imagens (particularmente útil para carregar e manter porções de um mapa no navegador do cliente sem necessidade de nova requisição ao servidor). Como gera arquivos Flash, tratam-se de aplicações bastante atraentes ao usuário final (FLEX SAMPLE APPLICATIONS, 2007). Não se deve esquecer que apenas arquivos Flash são gerados em tempo de execução, necessitando de *plugins* para tal. Pesquisas independentes acerca da penetração do *plugin* Flash não foram encontradas. Apenas no próprio site da Adobe foi encontrada uma estatística afirmando que esse *plugin* encontra-se em 98% dos navegadores. Deve-se considerar também que muitos *firewalls* podem bloquear conteúdo em Flash.

Nesta dissertação, foi utilizada a versão 3.0 Beta do FlexBuilder para Linux. Flex oferece suporte a SVGZ (formato SVG comprimido). Também oferece um suporte melhor a SVG, uma vez que mais instâncias de arquivos SVG foram corretamente renderizados do que em Openlaszlo. O desempenho (tempo para renderizá-los) também foi sensivelmente melhor do que com Openlaszlo. Entretanto, suporte à manipulação dinâmica de SVG não existe. Não é possível, por exemplo, carregar dinamicamente um SVG proveniente de um servidor WMS. Os dados SVG têm que estar disponíveis em tempo de compilação.

2.3.2 – GWT

A sigla GWT significa *Google Web Toolkit*. É um *kit* de desenvolvimento AJAX onde

o desenvolvedor escreve código em Java e obtém, de modo transparente, um código DHTML/AJAX como *runtime*. Há algumas chamadas de API para manipulação de imagens (como pré-carregamento). Google Maps e GMail foram desenvolvidos com essa tecnologia. Possui algumas vantagens como aproveitamento do conhecimento já existente em Java e a compatibilidade entre diferentes navegadores sem necessidade de *plugins*. Por outro lado, para fazer uma interface gráfica tão atraente quanto em Flash requer muito mais trabalho. Também devido à sua pouca maturidade há alguns *bugs* na sua API.

A versão 1.4.61 foi utilizada neste trabalho. Não há suporte para SVG na versão 1.4 do GWT, nem mesmo com o uso de extensões até o momento. É possível, com uso de extensões de terceiros, fazer desenhos vetoriais (sem carregamento de SVG), mas assim como qualquer extensão desse tipo, há severas restrições. Por exemplo, a Canvas Widget API que se baseia em uma *tag* HTML suportada em poucos navegadores (*Internet Explorer* não incluso). Uma outra extensão, GWT Widget Library, tem suporte limitado a SVG e foi descontinuada na versão 1.4 do GWT.

2.3.3 – Openlaszlo

Openlaszlo é uma plataforma de código aberto para desenvolver RIA. Um programa Openlaszlo é escrito em LZX (um tipo de XML) e *Javascript*. Escreve-se pouco em *Javascript* porque há um contexto de *framework* que simplifica sobremaneira o trabalho de desenvolvimento. O código é compilado para DHTML/AJAX ou Flash (apenas na versão 4 do Openlaszlo). Provê recursos de pré-carregamento e *caching* de imagens. A aplicação final pode ser bastante atraente (LASZLO SHOWCASE, 2008), especialmente se Flash for escolhido como o *runtime*.

A versão 4.0.2 foi utilizada com o editor Emacs. Há um *plugin* para Eclipse (ide4laszlo), mas ele possui muitos *bugs* e seu desenvolvimento foi descontinuado. Em relação a suporte a SVG, há uma extensão de terceiros chamada svgview. Infelizmente ela tem algumas limitações: não é capaz de renderizar qualquer tipo de SVG, por exemplo, SVG

com valores negativos para o retângulo envolvente, como o mostrado em 2.2. O desempenho da renderização é pior do que em Flex. Sem utilizar SVG, é possível fazer desenhos de feições utilizando o componente drawview nativo da plataforma Openlaszlo. Trata-se de uma implementação parcial do elemento Graphics da especificação (WHATWG, 2008) do W3C. O componente drawview permite descrever geometrias complexas, mas tem a desvantagem de misturar dados com lógica, como foi criticado em 2.2.

2.4 – Operações SIG-Web implementadas com tecnologias ria

As operações são aquelas apontadas em 2.1. Apenas a porção principal do código (classes, *scripts* e XML) são apresentados. Requere-se do leitor um conhecimento básico de cada uma dessas tecnologias a fim de que possa usar o código abaixo e colocá-lo em uso efetivo.

2.4.1 – Usando GWT

Pelas razões descritas em 2.3.2, é considerado que GWT não oferece suporte a operações no modo vetorial. Portanto apenas o modo *raster* será discutido.

2.4.1.1 – Colapsamento e ampliação (*raster*)

É bastante simples efetuar esta operação. Basta configurar o objeto *Image* com os valores dos atributos *width* e *heigth* de modo adequado. Um detalhe importante: o valor inicial da largura e altura da imagem é obtida depois que a imagem é carregada através do método *onLoad* da interface *LoadListener*. Sem fazer uso desse artifício, o mecanismo sugerido não funciona. Segue um extrato do código (declaração da classe, *imports* e declaração do método *onModuleLoad* não são mostrados):

```
Button zoomIn = new Button("Zoom In");
Button zoomOut = new Button("Zoom Out");
final Image rioMap = new
      Image("rio_de_janeiro_alta_resolucao.jpg");

rioMap.addLoadListener(new LoadListener() {
         public void onError(Widget sender) {
           GWT.log("An error occurred while loading.",
                   null);
         }

         public void onLoad(Widget sender) {
           imageWidth = rioMap.getWidth();
           imageHeight = rioMap.getHeight();
         }
       });


     zoomIn.addClickListener(
           new ClickListener() {
                public void onClick(Widget sender) {
                     imageWidth *= 1.1;
                     imageHeight *= 1.1;
                     rioMap.setWidth(imageWidth +"px");
                     rioMap.setHeight(imageHeight
                                      +"px");
                  }
               }
     );

     zoomOut.addClickListener(
           new ClickListener() {
```

```
                    public void onClick(Widget sender) {
                        imageWidth *= 0.9;
                        imageHeight *= 0.9;
                        rioMap.setWidth(imageWidth +"px");
                        rioMap.setHeight(imageHeight
                                        +"px");
                    }
                }
        );
    AbsolutePanel mapPanel = new AbsolutePanel();
    FlowPanel buttonPanel = new FlowPanel();

    buttonPanel.add(zoomOut);
    buttonPanel.add(zoomIn);
    mapPanel.add(rioMap);
    RootPanel.get().add(buttonPanel);
    RootPanel.get().add(mapPanel);
```

2.4.1.2 – Deslocamento (*raster*)

É possível utilizando um *MouseListenerAdapter* (*DragMouseListener*), cujo código foi escrito baseado em uma mensagem postada em (FÓRUM GWT, 2008). Algumas adaptações ao código visto nessa mensagem foram necessárias para generalizá-lo: fazer com que seja possível utilizá-lo para qualquer *Panel* que tenha como pai um *AbsolutePanel* e não somente o *RootPanel*. É também importante colocar dois *Panels* envolventes à imagem que representa o mapa. Um que será deslocado junto com a imagem (ao tentar mover a imagem dentro do *Panel*, não se obtém o efeito desejado) e outro para ser a área de *clipping*. Esse último deve ser maior que o primeiro, que deve ser posicionado alguns *pixels* de distância da origem (0,0). Dependendo da imagem, o redesenho para cada movimento do *mouse* pode exigir muito da CPU, e a resposta visual ao usuário pode ser muito lenta. O extrato do código

segue abaixo. É muito similar ao anterior, e deve-se usar aquele código em conjunto com este, exceto a partir da linha que declara o *mapPanel*. A partir daquela linha o código deve ser:

```
    AbsolutePanel wrapperPanel = new AbsolutePanel();
    AbsolutePanel mapPanel = new AbsolutePanel();
     DragMouseListener        dragListener        =        new
DragMouseListener(mapPanel);

    rioMap.addMouseListener(dragListener);

    FlowPanel buttonPanel = new FlowPanel();

    buttonPanel.add(zoomIn);
    buttonPanel.add(zoomOut);

    mapPanel.add(rioMap);
    wrapperPanel.setSize("900px", "900px");
    wrapperPanel.add(mapPanel, 10, 10);



    RootPanel.get().add(buttonPanel);
    RootPanel.get().add(wrapperPanel);
```

E o código para o *DragMouseListener* é:

```
public class DragMouseListener extends MouseListenerAdapter {
    private boolean dragging;
    private int baseX, baseY, diffX, diffY, posX, posY;
    private Widget contentWidget;
    private AbsolutePanel wrapperWidget;

    public DragMouseListener(Widget contentWidget) {
```

```
     this.contentWidget = contentWidget;
    }

    public void onMouseDown( Widget w, int x, int y ){
        if( !dragging ){
            dragging = true;
        }


         DOM.setCapture( w.getElement() );
         baseX = contentWidget.getAbsoluteLeft();
         baseY = contentWidget.getAbsoluteTop();
         diffX = x;
         diffY = y;
         posX = baseX - diffX;
         posY = baseY - diffY;
    }

    public void onMouseMove( Widget w, int x, int y ){
        if( dragging ){
            baseX = contentWidget.getAbsoluteLeft();
            baseY = contentWidget.getAbsoluteTop();

                posX = baseX-diffX+x;
                posY = baseY-diffY+y;

              AbsolutePanel              parent              =
(AbsolutePanel)contentWidget.getParent();
                  parent.setWidgetPosition(contentWidget,posX,
posY );
        }
    }
```

```
    public void onMouseUp( Widget w, int x, int y ){
        if( dragging ){
                posX = baseX-diffX+x;
                posY = baseY-diffY+y;
     AbsolutePanel                    parent                    =
(AbsolutePanel)contentWidget.getParent();
                   parent.setWidgetPosition(contentWidget,posX,
posY );

            dragging = false;
        }
    DOM.releaseCapture( w.getElement() );
    }
}
```

2.4.1.3 – Sobreposição de polígonos (*raster*)

Esta operação é bem simples de ser implementada. O *AbsolutePanel* deve ser utilizado e as imagens devem ser adicionadas nas posições corretas. As imagens devem ser transparentes para se obter o efeito desejado. Os retângulos envolventes e as projeções devem ser as mesmas. Abaixo está o código onde as imagens são obtidas através de servidores WMS com os parâmetros apropriados:

```
     Image bordersMap = new
Image("http://www2.demis.nl/mapserver/Request.asp?SERVICE=WMS&
VERSION=1.1.1&REQUEST=GETMAP&LAYERS=Borders&SRS=EPSG:4326&BBOX
=-180,-
60,176,75&WIDTH=1000&HEIGHT=500&format=image/png&TRANSPARENT=T
RUE");
         Image coastsMap = new
Image("http://www2.demis.nl/mapserver/Request.asp?SERVICE=WMS&
VERSION=1.1.1&REQUEST=GETMAP&LAYERS=Coastlines&SRS=EPSG:4326&B
BOX=-180,-
60,176,75&WIDTH=1000&HEIGHT=500&format=image/png&TRANSPARENT=T
RUE");
```

```
    Image citiesMap = new
Image("http://clearinghouse4.fgdc.gov/cgi-
bin/services/global?SERVICE=WMS&VERSION=1.1.1&REQUEST=GETMAP&L
AYERS=cities&SRS=EPSG:4326&BBOX=-180,-
60,176,75&WIDTH=1000&HEIGHT=500&format=image/png&TRANSPARENT=T
RUE");


        AbsolutePanel mapPanel = new AbsolutePanel();
        mapPanel.add(bordersMap);
        mapPanel.add(coastsMap);
        mapPanel.setWidgetPosition(coastsMap,0,0);
        mapPanel.add(citiesMap);
        mapPanel.setWidgetPosition(citiesMap,0,0);
        RootPanel.get().add(mapPanel);
```

2.4.1.4 – Simbolização (*raster*)

Uma vez que manipulação de *pixels* não é possível, então não há como mudar o preenchimento ou contorno das feições à guisa de simbolização. Só é possível adicionar outras imagens que servem como símbolos e sobrepô-las às imagens que representam as feições, utilizando a mesma técnica mostrada no código da seção anterior. Para um único mapa estático as posições desses símbolos podem ser configuradas (usando o método *setWidgetPosition*, como visto anteriormente, e passando as coordenadas dos centróides das feições como parâmetros). Tais posições devem ser mantidas em uma estrutura de dados a fim de serem reconfiguradas sempre que o mapa for deslocado, colapsado ou ampliado.

2.4.1.5 – Seleção (*raster*)

Esta operação só é possível com GWT se utilizada em conjunto com a estratégia de simbolização: a seleção de uma feição será possível se à imagem que a representa (símbolo) for associado um objeto de *ClickListener*. No método *onClick* estaria o tratamento para a seleção. Não é possível adicionar painéis invisíveis no formato de um polígono complexo

para servir de máscaras de seleção (o objeto *ClickListener* seria associado a esse painel invisível se houvesse). Essa é a estratégia usada em Flex, por exemplo. Caso o mapa seja estático, pode-se considerar a colocação de uma imagem completamente transparente, com a mesma geometria da feição para servir de máscara de seleção.

2.4.2 – Usando flex

Todas as operações em Flex funcionam quase exatamente da mesma forma independente de se usar o modo *raster* ou vetorial. A principal diferença é a *tag* "@Embed" que deve existir para SVG, i.e., o dado em SVG tem que estar carregado em tempo de compilação como explicado em 2.3.1 . A outra diferença é que a *tag* <mx:Image> deve ter seu atributo *source* referenciando a um arquivo SVG ou de imagem *raster*.

2.4.2.1 – Colapsamento e ampliação (*raster* e vetorial)

Simplesmente criam-se funções em ActionScript que vão mudar a altura e largura da imagem, dentro de uma Box (Hbox ou Vbox com barra de rolagem). O código ActionScript é:

```
public function zoomOut(): void {
     myimg.width = myimg.width * 0.9;
     myimg.height = myimg.height * 0.9;
}

public function zoomIn(): void {
     myimg.width = myimg.width * 1.1;
     myimg.height = myimg.height * 1.1;
}
```

E o código MXML relativo é:

```
<mx:HBox>
          <mx:Button label="Zoom out" click="zoomOut();"/>
```

```
                <mx:Button label="Zoom in" click="zoomIn();"/>
     </mx:HBox>

    <mx:Canvas id="v1"
        width="1000" height="1000"
        borderStyle="solid"
        backgroundColor="#DDDDDD">

        <mx:Image id="myimg"
            source="@Embed(source='../images/so_municipios.svg
')" />
    </mx:Canvas>
```

Uma outra opção é utilizar a funcionalidade de "*Behaviors*", utilizando a *tag* <mx:Zoom> que pode ser referenciada em um método *mouseDownEffect*.

2.4.2.2 – Deslocamento (*raster* e vetorial)

O mapa contido em <mx:Image> deve ser envolvido em um <mx:Canvas> com as funções *dragEnter* (que indica o aceite do arrasto) e *dragDrop* (onde a imagem será posicionada depois de solta) definidas. Na <mx:Image> correspondente deve haver referência a uma função *mouseMove* que será responsável por configurar o iniciador do arrasto, adicionar os dados da imagem e requisitar ao *DragManager* para efetuar o arrasto. Não deve ser utilizado o *imageProxy*, que é uma imagem de referência em miniatura que é mostrada enquanto o *mouse* é arrastado. Isso provocaria uma grande lentidão na operação. Abaixo o código em ActionScript:

```
public var xOff:Number;
     public var yOff:Number;

     // Responsável por tratar o event de início de
     // arrasto, chamado pelo evento mouseMove da imagem
     private function dragMe(event:MouseEvent, img1:Image,
```

```
            format:String):void {
     var dragInitiator:Image=Image(event.currentTarget);
     var ds:DragSource = new DragSource();
     ds.addData(img1, format);
     DragManager.doDrag(dragInitiator, ds, event);
}

// Função chamada pelo evento dragEnter do canvas;
// Habilita a soltura da imagem
private function doDragEnter(event:DragEvent):void {
     DragManager.acceptDragDrop(Canvas(event.target));
}

// Função chamada pelo evento dragDrop do canvas;
// Seta a posição da imagem "largando-a"
// no seu novo lugar
private function doDragDrop(event:DragEvent,
           target1:Canvas, format:String):void {
     myimg.x = target1.mouseX - xOff
     myimg.y = target1.mouseY - yOff
}

// Função auxiliar chamada pelo evento de arrasto
// do mouseMove, à medida em que a imagem é
// arrastada pelo Canvas. Atualiza os valores
// de xOff e yOff para mostrar a posição corrente
// do mouse
private function myoffset(img:Image):void {
     xOff = img.mouseX
     yOff = img.mouseY
}
```

E o código MXML relacionado:

```
<mx:Canvas id="v1"
        width="1000" height="1000"
        borderStyle="solid"
        backgroundColor="#DDDDDD"
        dragEnter="doDragEnter(event);"
        dragDrop="doDragDrop(event, v1, 'img');">

        <mx:Image id="myimg"
          source="@Embed(source='../images/so_municipios.svg')
"
          mouseMove="dragMe(event, myimg, 'img');
myoffset(myimg);"/>
    </mx:Canvas>
```

2.4.2.3 – Simbolização (*raster* e vetorial)

A mesma estratégia de 2.4.1.4 será usada aqui por não ser possível manipulação dinâmica de *pixels* (para *raster*) nem dos dados do SVG (para vetorial). A diferença em relação ao GWT é que esses símbolos podem ser desenhados dinamicamente usando o objeto Graphics que pertence ao *Canvas* onde está a tag <mx:Image>. Isso será mostrado na seção 2.4.2.5. Depois que o símbolo é carregado ou desenhado, basta colocá-lo no *Canvas*. Sempre que colapsamento, ampliação e deslocamento ocorrerem, as posições dos símbolos têm que ser ajustadas. Para tal, se baseado no código em 2.4.2.2 basta mudar no método *doDragDrop* configurando as propriedades x e y de todos os símbolos como é feito com xOff e yOff do mapa propriamente dito.

2.4.2.4 – Sobreposição de polígonos (raster e vetorial)

Simplesmente adiciona-se tantas *tags* <mx:Image> quantas camadas existirem no mapa. As mesmas considerações feitas em 2.4.1.3 no que se refere à transparência das

imagens, projeções e retângulos envolventes aplicam-se aqui. É possível combinar imagens *raster* e SVG como camadas, mas a camada contendo o SVG tem que ser a primeira camada ou seu estilo (valor da *tag "fill-opacity")* tem que ser configurado com o valor 0.0. Abaixo o código em MXML:

```
     <mx:Image id="cities"
source="http://clearinghouse4.fgdc.gov/cgi-
bin/services/global?SERVICE=WMS&amp;VERSION=1.1.1&amp;REQUEST=
GETMAP&amp;LAYERS=cities&amp;SRS=EPSG:4326&amp;BBOX=-180,-
60,176,75&amp;WIDTH=1000&amp;HEIGHT=500&amp;format=image/png&a
mp;TRANSPARENT=TRUE"/>
     <mx:Image id="borders"
source="http://www2.demis.nl/mapserver/Request.asp?SERVICE=WMS
&amp;VERSION=1.1.1&amp;REQUEST=GETMAP&amp;LAYERS=Borders&amp;S
RS=EPSG:4326&amp;BBOX=-180,-
60,176,75&amp;WIDTH=1000&amp;HEIGHT=500&amp;format=image/png&a
mp;TRANSPARENT=TRUE"/>
     <mx:Image id="coasts"
source="http://www2.demis.nl/mapserver/Request.asp?SERVICE=WMS
&amp;VERSION=1.1.1&amp;REQUEST=GETMAP&amp;LAYERS=Coastlines&am
p;SRS=EPSG:4326&amp;BBOX=-180,-
60,176,75&amp;WIDTH=1000&amp;HEIGHT=500&amp;format=image/png&a
mp;TRANSPARENT=TRUE"/>
```

2.4.2.5 – Seleção (*raster* e vetorial)

Esta operação pode ser implementada de um modo mais sofisticado do que com GWT. Pelo fato de ser possível desenhar formas arbitrariamente complexas com o objeto *Graphics*, e essas formas poderem ser *clicáveis* e transparentes, essa estratégia será mostrada aqui. Inicialmente, cria-se um componente personalizado (*ClickableMapCanvas*), herdando de *Canvas*. Esse componente terá dois componentes-filho: uma *Image* e um *UIComponent*. Na aplicação principal MXML, depois que a imagem é carregada, um conjunto de formas

(objetos da classe *Shape*[1]), representando as feições selecionáveis serão desenhadas em suas posições. Para fins de simplificação, no exemplo abaixo, as feições são simplesmente retângulos, mas com o *Graphics* é possível desenhar formas muito mais complexas. Essas formas devem ser adicionadas como filhas do *UIComponent* no *ClickableMapCanvas*. Um objeto de tratamento de clique de *mouse* é adicionado ao UIComponent. A função *handleClickMap* usará o método *hitTestPoint* para checar qual das feições foi *clicada* pelo usuário. O Código do *ClickableMapCanvas* é:

```
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml"
width="400" height="300">
     <mx:Image id="mapImage"/>
     <mx:UIComponent id="clickableLayer">

     </mx:UIComponent>
</mx:Canvas>
```

O código da aplicação principal é:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute"
     xmlns:MapComp="*" applicationComplete="afterLoad();">

     <mx:Script>
        <![CDATA[
          import mx.controls.Image;
          import flash.display.Shape;
          import mx.controls.Alert;

          private var shapes:Array = new Array();

          public function afterLoad():void {
```

1 Essa classe Shape não se refere ao formato SHAPE da ESRI.

```
            rioMap.mapImage.load("../images/rio_de_janeiro_al
ta_resolucao.jpg");
            var shape1:Shape = new Shape();

            shape1.graphics.beginFill(0xFFCC00, 0);
            shape1.graphics.drawRect(0,0,50,100);
            shape1.graphics.endFill();

            var shape2:Shape = new Shape();
            shape2.graphics.beginFill(0,0);
            shape2.graphics.drawRect(50,0,50,100);
            shape2.graphics.endFill();

            rioMap.clickableLayer.addChild(shape1);
            rioMap.clickableLayer.addChild(shape2);
            shapes.push(shape1);
            shapes.push(shape2);

rioMap.clickableLayer.addEventListener(MouseEvent.MOUSE_DOWN,h
andleMapClick);
        }

          private function handleMapClick( event:MouseEvent ) :
void {
              event.stopPropagation();
              for(var i:int=0; i < shapes.length; i++) {
                   var sh:Shape = shapes[i] as Shape;
                   if( sh.hitTestPoint(event.stageX,
event.stageY, true) ) {
                        Alert.show("You clicked on "+i);
                        // Put feature selection handling
                        // code here
                        break;
```

```
                        }
                    }
                }
            ]]>
        </mx:Script>
         <MapComp:ClickableMapCanvas id="rioMap"/>
</mx:Application>
```

O código acima foi inspirado em uma mensagem colocada em (ADOBE FORUM, 2008) no tópico "*Add clickable image to Flex*".

Para obter um *design* melhor, esse componente personalizado em MXML não deveria ter o atributo *source* de <mx:Image> configurado dentro dele, já que pretende ser reutilizável. Nesse caso, a imagem deveria ser carregada dinamicamente, o que impediria o uso de SVG.

Uma outra alternativa, ao invés de se usar objetos *Shape* diretamente, é fazer uso de uma subclasse de *Shape* que seria criada tendo um id e se relacionaria com a feição geográfica respectiva. Assim, ao ser selecionada, obter-se-iam facilmente os dados da feição.

2.4.3 – Usando openlaszlo

2.4.3.1 – Colapsamento e ampliação (*raster*)

É possível configurando uma view externa de *clipping* e uma *view* interna que mostra a imagem. Nessa segunda *view* o atributo "*stretches"* deve estar com o valor *"both"*. Depois disso, basta adicionar métodos para aumentar ou diminuir a altura e largura da *view* mais interna. O código segue:

```
 <canvas height="100%" width="100%" debug='true'>
     <view name="outerView" clip="true" width="250"
height="200" x="5" y="20" >
     <view name="imageView" stretches='both'
resource="images/rio_de_janeiro_alta_resolucao.jpg"/>
          <method name='ZoomIn'>
```

```
                var  curZoom =
this.imageView.width/imageView.resourcewidth;
                curZoom += 0.10;
          this.imageView.setWidth(this.imageView.resourcewidth
* curZoom);
     this.imageView.setHeight(this.imageView.resourceheight *
curZoom);
          </method>
          <method name='ZoomOut'>
                var  curZoom =
this.imageView.width/imageView.resourcewidth;
                curZoom -= 0.10;
     this.imageView.setWidth(this.imageView.resourcewidth *
curZoom);
     this.imageView.setHeight(this.imageView.resourceheight *
curZoom);
          </method>
    </view>
    <button onclick="this.parent.outerView.ZoomIn()"
text="Zoom +" x="96" y="0" width="50" height="20" />
    <button onclick="this.parent.outerView.ZoomOut()"
text="Zoom -" x="144" y="0" width="50" height="20" />
</canvas>
```

2.4.3.2 – Colapsamento e ampliação (vetorial)

O componente svgview não aceita o atributo *stretches="both"*, apesar de herdar do componente *view*. O componente drawview não pode ser usado pois não possui atributos de largura e altura para serem alterados.

2.4.3.3 – Deslocamento (*raster*)

Pode ser conseguido com um esforço extra. Um componente personalizado deve ser

criado usando a *tag "class"*. Esse componente implementará o suporte ao arrastar e soltar do Openlaszlo definindo os atributos *onmousedown* e *onmouseup* com os métodos *dragger.apply* e *dragger.remove*, respectivamente. Dentro desse componente, uma *view* com a referência da imagem do mapa será colocada. Este é o código do componente:

```
     <class name="mywindow"
onmousedown="dragger.apply()" onmouseup="dragger.remove()"
     bgcolor="blue" clip="true">
     <dragstate name="dragger"/>
     <view name="mycontent" x="10" y="10" bgcolor="white"
width="${parent.width-20}" height="${parent.height-20}"
stretches="both" resource="tests/svg_examples/lion.jpg">
     </view>
</class>
```

O código da aplicação principal terá a mesma *view* externa de 2.4.3.1 e uma instância do componente acima como sendo sua *view* mais interna. Os métodos *ZoomIn* e *ZoomOut* devem ser alterados para configurar os tamanhos de ambas as *views* corretamente. Eis o código da aplicação principal:

```
     <view name="outerView" clip="true" width="250"
height="200" x="5" y="20" >
          <method name='ZoomIn'>
               var  curZoom =
this.my.mycontent.width/this.my.mycontent.resourcewidth;
               curZoom += 0.10;
               var outerZoom =
this.my.width/this.my.mycontent.width;
               outerZoom += 0.10;
     this.my.mycontent.setWidth(this.my.mycontent.resourcewidth
* curZoom);
     this.my.mycontent.setHeight(this.my.mycontent.resourceheig
ht * curZoom);
     this.my.setWidth(this.my.width * outerZoom);
     this.my.setHeight(this.my.height * outerZoom);
```

```
          </method>
          <method name='ZoomOut'>
               var  curZoom =
this.my.mycontent.width/this.my.mycontent.resourcewidth;
               curZoom -= 0.10;
               var outerZoom =
this.my.width/this.my.mycontent.width;
               outerZoom -= 0.10;
     this.my.mycontent.setWidth(this.my.mycontent.resourcewidth
* curZoom);
     this.my.mycontent.setHeight(this.my.mycontent.resourceheig
ht * curZoom);
     this.my.setWidth(this.my.width * outerZoom);
     this.my.setHeight(this.my.height * outerZoom);
          </method>
       <mywindow id="w" name="my" width="300" height="250">
       </mywindow>
     </view>


      <button onclick="this.parent.outerView.ZoomIn()"
text="Zoom +" x="96" y="0" width="50" height="20" />
 <button onclick="this.parent.outerView.ZoomOut()" text="Zoom
-" x="144" y="0" width="50" height="20" />
```

2.4.3.4 – Deslocamento (vetorial)

A mesma solução aplicada em 2.4.3.2 é aplicada para este caso.

2.4.3.5 – Sobreposição de polígonos (*raster* e vetorial)

Ambos os modos (*raster* e vetorial) funcionam da mesma forma: simplesmente adicione as *views* com os mesmos valores de x e y. As mesmas considerações de 2.4.1.3 em relação à transparência das imagens, projeções e retângulos envolventes aplicam-se aqui.

Abaixo o código:

```
     <canvas height="600" width="1200" debug='true'
bgcolor="#ffffff">


     <view name="coastsView" x="0" y="0"
resource="images/coasts.png"/>
     <view name="bordersView" x="0" y="0"
resource="images/borders.png"/>
     <view name="citiesView" x="0" y="0"
resource="images/svg_examples/cities.png"/>
</canvas>
```

No exemplo, acima as imagens estão referenciadas como arquivos PNG estáticos, mas poderiam ser referenciadas como URLs apontando para servidores WMS, como nos exemplos com GWT e Flex. Para o propósito único de sobreposição de polígonos, svgview e drawview funcionam igualmente.

2.4.3.6 – Simbolização (*raster*)

Se imagens *raster* forem utilizadas para representar o mapa, a mesma estratégia apresentada anteriormente com as demais tecnologias RIA pode ser usada aqui: usar outras imagens como símbolos e sobrepô-las às feições. A sobreposição é simples como demonstrado em 2.4.3.5, mas os mesmos esforços descritos em 2.4.1.4 são necessários para manter a posição dos símbolos correta após operações de colapsamento, ampliação e deslocamento.

2.4.3.7 – Simbolização (vetorial)

Utilizar svgview requer um grande esforço: consultas em *XPath* devem ser utilizadas para obter o valor de preenchimento ou contorno de uma determinada feição e em seguida

mudá-lo. Com drawview é bem mais fácil alterar tais valores. Um código de exemplo estará em 2.4.3.9. É importante não se esquecer das desvantagens de se usar drawview no que tange à separação de lógica de aplicação e dados, como discutido anteriormente.

2.4.3.8 – Seleção (*raster*)

Há duas alternativas: 1 – aplicar a mesma estratégia mostrada até agora, ou seja, utilizar imagens como símbolos e associar a eles um tratador de eventos de clique de *mouse*. Veja 2.4.3.9 para saber como fazer essa associação. 2 – utilizar tantos drawviews quantas feições houver, fazendo dessa drawview uma máscara de seleção com um tratador de eventos de clique de *mouse*. Bastaria desenhar uma drawview por feição, sobrepondo a cada uma delas e sem chamar o método *stroke*, para fazê-lo invisível. A vantagem dessa segunda abordagem é que a seleção fica disponível em qualquer parte da feição e não apenas sobre a imagem que lhe serve de símbolo. Por outro lado, dados e lógica se misturam e dependendo da complexidade das feições, torna-se muito difícil de desenvolver essa solução. Essa idéia é bem similar à apresentada em 2.4.2.5, onde objetos de Shape eram usados como máscaras invisíveis das feições.

2.4.3.9 – Seleção (vetorial)

Se as feições são representadas com svgview, trata-se de uma operação muito difícil de ser implementada: define-se uma máscara para cada feição com retângulos envolventes e utiliza-se o método *convertCoords* (de svgview) para obter as coordenadas do *mouse*. Depois verifica-se se confere com algum dos retângulos. Caso positivo, trata a seleção de modo adequado. Abaixo segue o código usando drawview:

```
<canvas height="200" proxied="false" debug="true">
      <drawview name="polygon1">
```

```
        <handler name="oninit">
         this.moveTo(100,100);
         this.lineTo(200,100);
         this.quadraticCurveTo(120, 200, 300, 100)
         this.closePath()
         this.stroke()
        </handler>
      </drawview>
      <drawview name="polygon2" >
       <handler name="oninit">
          this.moveTo(100,100);
          this.lineTo(200,100);
          this.lineTo(150,50);
          this.closePath();
          this.stroke();
       </handler>
       <handler name="onclick">
            this.strokeStyle="#FF0000";
            this.stroke();
            // Adicione código para tratar a seleção
       </handler>
      </drawview>
  </canvas>
```

2.4.4 – Resumo das operações

Como resultado será mostrado um gráfico onde, para cada operação, é dada uma nota de 0 a 3. O valor 0 significa nenhum suporte e 3, suporte total sem restrições. O valor 1 significa algum suporte com restrições severas e 2, algum suporte com alguma restrição. Por "severas" entende-se que 2 ou mais restrições aplicam-se. Por "alguma" restrição entende-se apenas uma. A restrição pode ser uma das seguintes: *design* ruim (por exemplo, misturar

lógica e dados), uso de API's ou extensões de terceiros, complexidade do código, interação com o usuário não ideal (como para a operação de seleção, quando requer que o usuário clique em um determinado símbolo para selecionar uma feição, ao invés de poder clicar em qualquer porção da mesma) e não carregamento dinâmico de dados (como no suporte ao SVG pelo Flex). Por exemplo: todas as operações de seleção foram descritas, exceto no Flex, utilizando uma imagem como símbolo. Portanto essas receberam no máximo a nota 2. Se combinadas com o uso de uma API de terceiro, como no caso da solução do Openlaszlo para o modo vetorial, a nota cai para 1. A solução do Flex não tem a restrição de interação não ideal com o usuário, mas é considerada complexa porque para cada feição é necessário desenhar um objeto *Shape* como máscara. Eis porque obteve uma nota 2. GWT não oferece nenhum suporte ao modo vetorial, nem mesmo com API's e extensões de terceiros que sejam suportadas em navegadores populares. Isso leva GWT a ter nota 0 nessas operações.

O resultado final é:

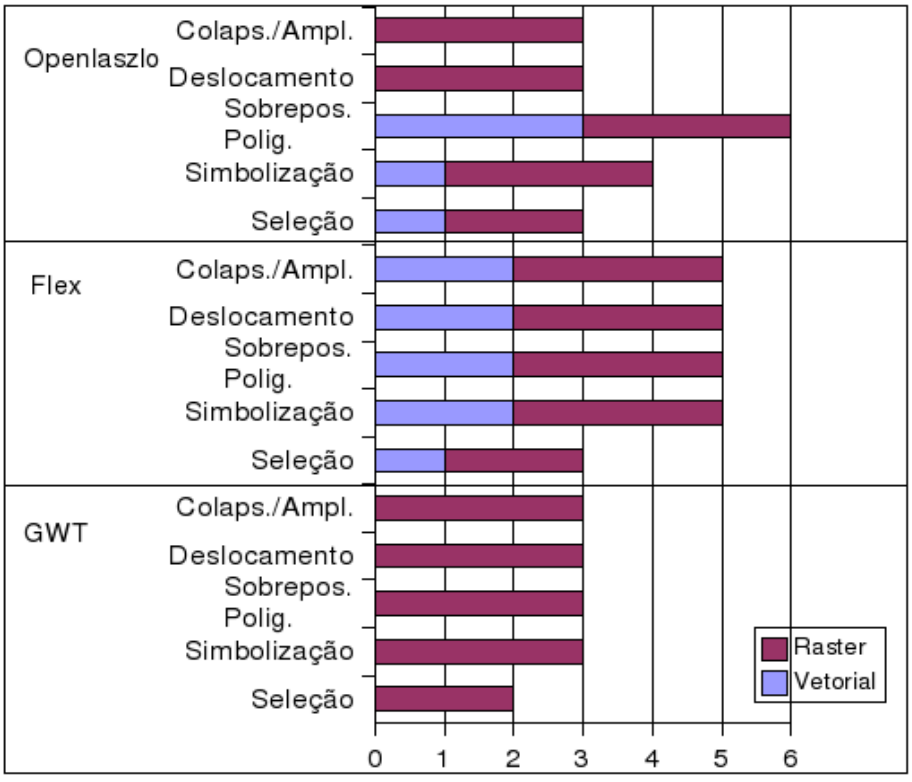

Figura 1 – Resumo das Operações

2.5 – *Sites web* com SIG utilizando tecnologias RIA

Há vários exemplos de *sites* com funcionalidades de um SIG-Web feitos com estas e outras tecnologias RIA. Escolheu-se (GEABIOS, 2007) feito com Openlaszlo, (GOOGLE MAPS, 2007) feito com GWT e (YAHOO! MAPS, 2007) desenvolvido com Flex. Tratam-se de SIG-Web suficientemente complexos e de uso comum para que possam servir de base ao nosso estudo, por isso foram escolhidos.

2.5.1 - GEABIOS

GeaBios é um conjunto de serviços e aplicações *Web* gratuito disponibilizado por uma empresa eslovena chamada *Advanced Computer Aided Design Engineering & Manufacturing Agency* (Academa). O nome do *site* é um acrônimo para "*Geo Enabled And Better Internet Oriented Services*".

São várias aplicações disponíveis no *site* GeaBios e usando várias tecnologias RIA. Pode ser bastante confuso navegar entre elas para o usuário iniciante. A aplicação de que trata este estudo, desenvolvida em Openlaszlo, é "*World Map & Satellite Images*", disponível ao acessar "*Maps & Related Info*" a partir do menu no topo da página ou através do *link* "*Other Maps & Related Info*".

Não é o caso aqui de criticar a aplicação do ponto de vista de interface com o usuário, por certo um aspecto que mereceria alguma atenção neste *site*. O que importa para este estudo é que as 5 operações básicas discutidas na seção anterior estão aqui representadas (vide barra de ferramentas com o título "*View*" na figura 2) e combinadas com outras operações mais complexas, por exemplo, medição de distâncias. A sobreposição de polígonos é feita com dados vindos de servidores WMS e WFS, cuja configuração pode ser visualizada. Não é possível acrescentar um outro servidor WMS ou WFS. A fonte dos dados é vetorial. É bem provável que eles desenvolveram uma API própria para manipulação e renderização desses dados. Algumas janelas flutuantes ficam disponíveis ao usuário e podem ser manipuladas

(fechadas e arrastadas). A informação da latitude e longitude é atualizada em tempo real com o reposicionamento do cursor. Esses são aspectos típicos que apenas uma tecnologia RIA pode oferecer, levando a uma experiência mais rica ao usuário. É bem verdade que não se beneficia de uma estratégia mais sofisticada para deslocamento, por exemplo, o pré-carregamento dos dados do mapa em torno da área visualizada no momento. Tal estratégia é utilizada em outros SIG-Web, como o Google Maps, e oferece uma resposta muito mais rápida ao usuário. Pela mensagem de *log* que é mostrada, a cada operação de deslocamento, uma nova requisição ao servidor é feita. Até a resposta retornar e o sistema renderizar o mapa, o tempo passado é maior do que se poderia esperar de uma tecnologia RIA que pode utilizar-se de requisições assíncronas.

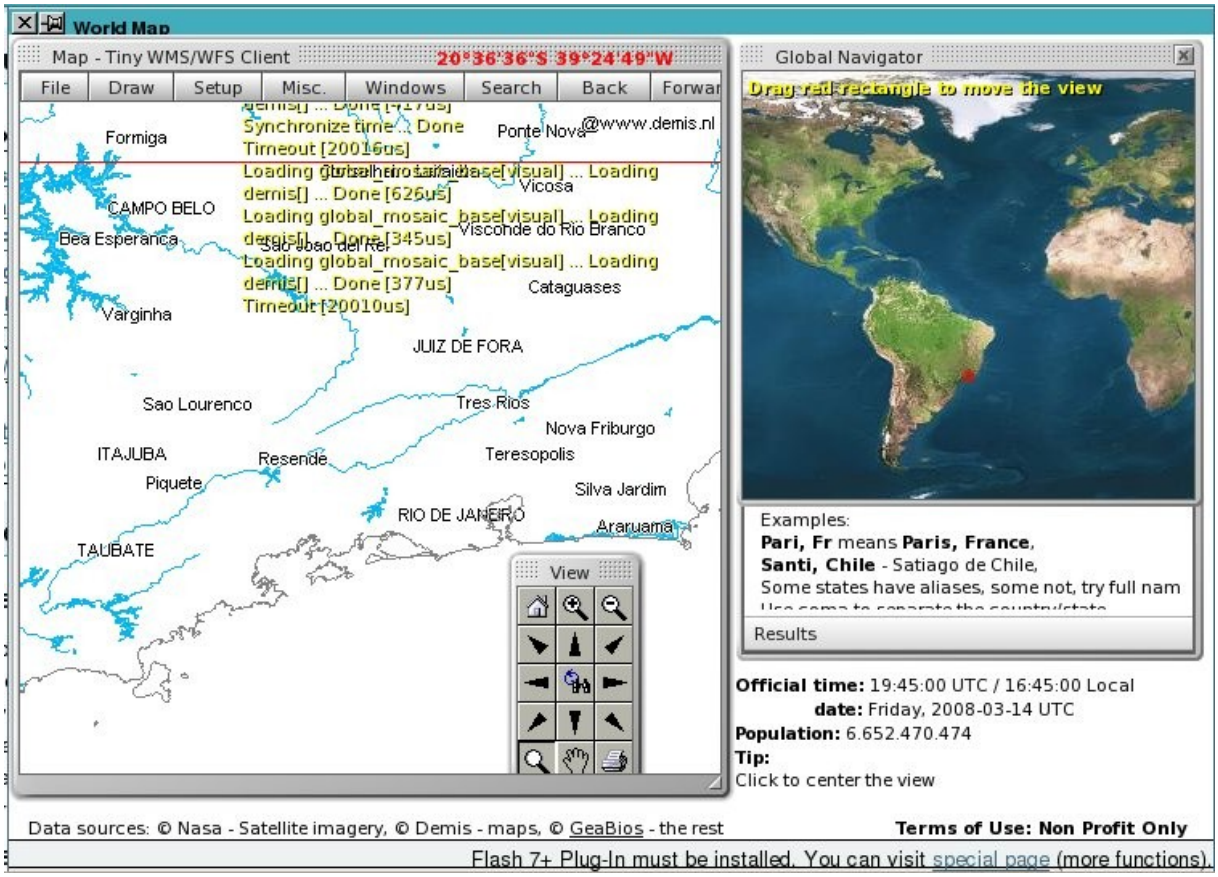

Figura 2 – Tela do GeaBios

2.5.2 - Yahoo! Maps

O Yahoo! é o um dos principais *sites* de serviços na Internet e um dos mais antigos também. O serviço de mapas é relativamente recente, de maio de 2007 e foi desenvolvido por uma empresa chamada Cartifact, que também provê os dados dos mapas. Tais dados estão em 2 formatos, um vetorial contendo informações das ruas, estradas e feições afins e outro em formato de imagens de satélites. Tais dados podem ser sobrepostos ou vistos de modo separado. Todas as demais operações discutidas nesta seção estão presentes neste SIG-*Web*, combinadas com inúmeras outras funcionalidades (geolocalização a partir de endereço, rota entre 2 ou mais localidades, informações de tráfego em vias públicas, entre outros.). Infelizmente no momento da escrita desta dissertação havia um *bug* neste *site* que fazia o navegador (Firefox no Linux) fechar toda a vez que se tentava acessar a funcionalidade de rota entre 2 localidades. Como o objetivo deste SIG-Web é um público mais geral, as informações sobre latitude e longitude, bem como eventuais servidores WMS ou WFS utilizados não são mostrados, como ocorre no GeaBios. No entanto, há uma melhor utilização da capacidade de requisições assíncronas de uma RIA neste *site*: ao executar um deslocamento para regiões mais próximas do mapa, percebe-se que algumas das respectivas imagens já são carregadas, se tiver havido um tempo de inatividade razoável do usuário, antecipando a sua interação. Uma outra limitação para usuários brasileiros é que os dados dos mapas referentes ao nosso país se restringem a apenas as imagens de satélites. Logo não é possível fazer geolocalização a partir de endereços nas cidades brasileiras.

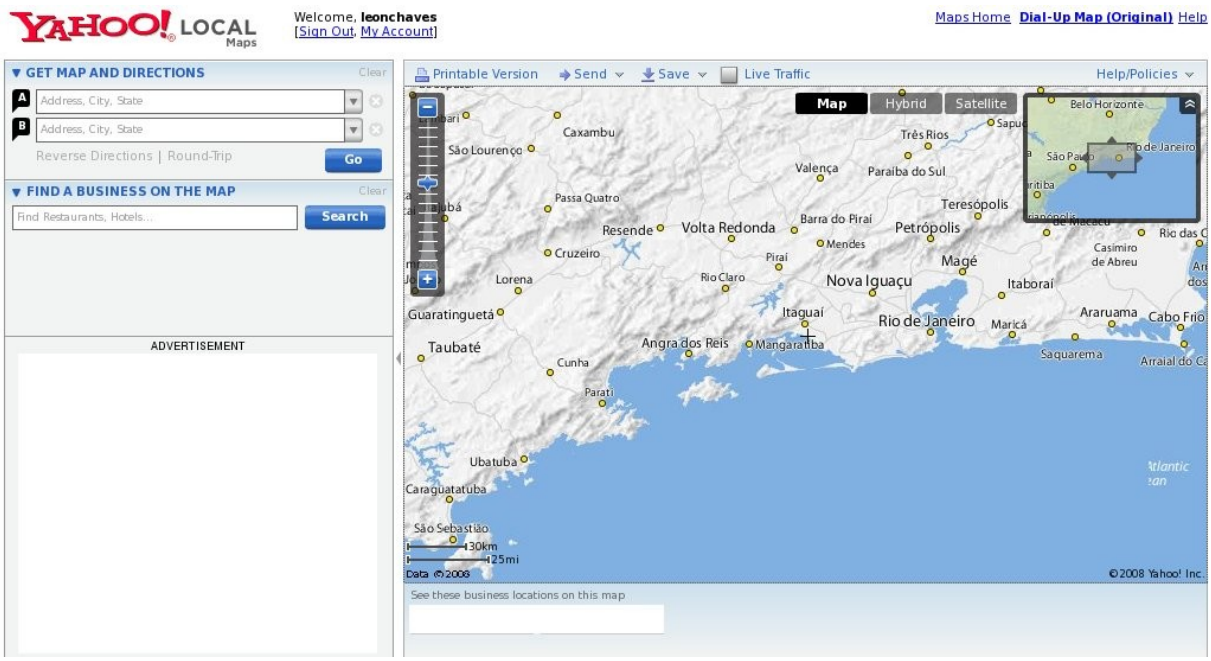

Figura 3 – Tela do Yahoo! Maps

2.5.3 - Google Maps

A Google é a maior empresa do ramo da Internet e seus serviços abarcam desde *email* a telefonia, passando por *marketing* e propaganda, suíte Office e naturalmente SIG. O programa Google *Earth* é um grande sucesso da companhia e popularizou muito o uso de SIG. No entanto, como se trata de um software *desktop*, está fora do escopo deste estudo. Para este estudo usa-se o Google *Maps,* que é um aplicativo para *web*. Trata-se de um serviço um pouco mais maduro do que o do Yahoo!, mas na mesma linha de servir a usuários comuns. Daí também não oferecer as funcionalidades mais sofisticadas do GeaBios, como comentando anteriormente. Novamente as 5 operações de que trata este estudo estão contempladas aqui. Além delas, as mesmas do Yahoo! *Maps* (geolocalização, rotas e informações de tráfego) acrescidas de uma camada que mostra o terreno. Uma outra funcionalidade interessante é o "*My Maps*" que permite ao usuário cadastrado gravar as visualizações de determinados mapas, acrescentar anotações (textos, fotos e outros) sobre as mesmas e compartilhar com outros usuários. Os dados também são oferecidos em formato vetorial (pela TeleAtlas e

NAVTEQ) e em imagens *raster* de satélite (pela DigitalGlobe, além de outras fontes não discriminadas). Ao contrário do Yahoo! *Maps* há informações suficientes sobre a região do Brasil. A interação com o usuário é bastante rica e mal se percebe que se trata de um *runtime* em *Javascript* (o desenvolvimento em GWT, que gera *Javascript*). A operação de deslocamento é bastante suave, o que só é possível com a utilização da estratégia de pré-carregamento de imagens de modo assíncrono.

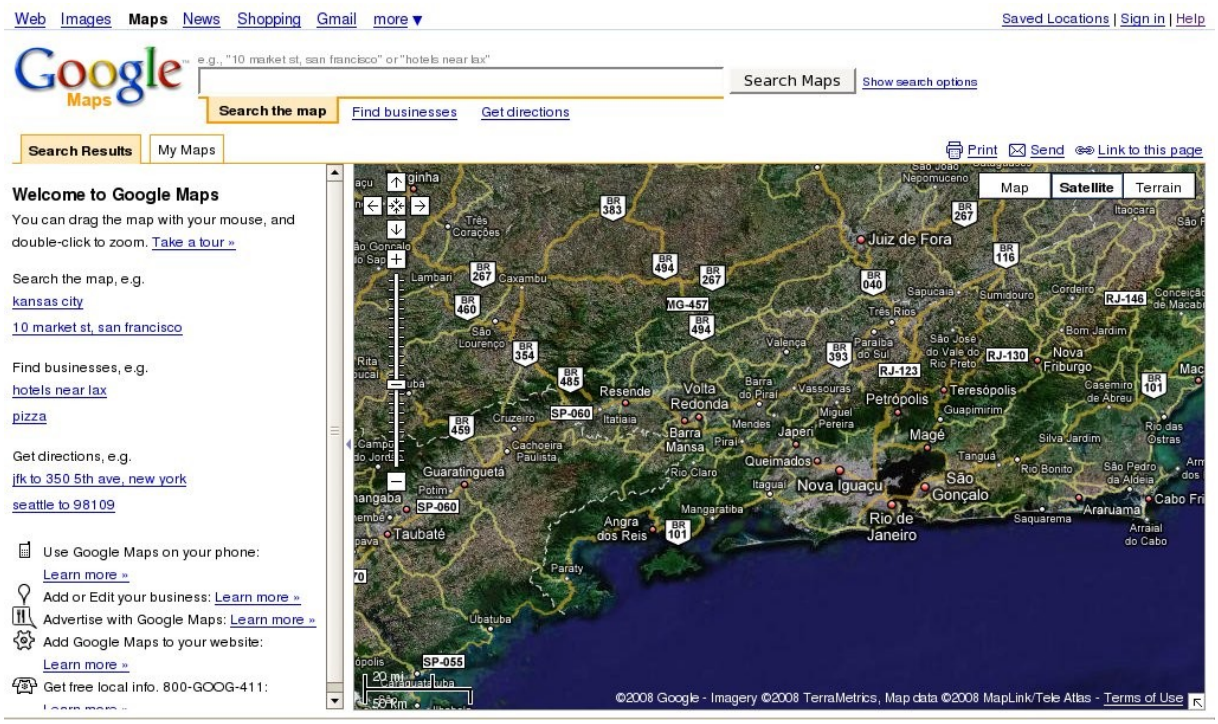

Figura 4 – Tela do Google Maps

3 – UWE-R

3.1 – Introdução

Nesta seção é apresentada a extensão proposta para uma das metodologias existentes, a UWE, a fim de dar suporte às características de RIA e aos aspectos de um SIG-Web. Inicialmente é justificada a escolha de UWE. Em seguida é aprofundada a descrição da UWE, que foi apenas delineada na seção 1. Por fim UWE-R, a extensão proposta, é descrita.

3.2 – Justificativa da escolha da UWE

A UWE-R não pretende ser uma metodologia completa de modelagem de SIG-Web no contexto de RIA. Seu foco é nos aspectos de navegação e organização dos elementos de interface gráfica na tela do usuário (apresentação). Isso não só porque esses aspectos são os mais peculiares à RIA, mas também porque os demais aspectos de uma modelagem (como análise de requisitos e modelagem de dados) já se encontram suficientemente cobertos em outras abordagens e na própria UWE. A limitação de escopo imposta por um trabalho de mestrado é outra razão para essa limitação da presente dissertação.

Também é objetivo deste trabalho aproveitar tanto quanto possível conceitos e linguagens existentes. Acredita-se que a profusão de metodologias para Engenharia *Web* é uma das razões para a sua não adoção ou adoção não uniforme no desenvolvimento de sistemas *Web*. A própria Orientação a Objetos só teve uma adoção universal quando da unificação das metodologias de Booch, Rumbaugh e Jacobson na UML. Portanto, por mais que seja necessário criar metodologias novas para conceitos de implementação novos (como RIA), é necessário fazê-lo tendo como base os avanços já obtidos e dominados pela comunidade de desenvolvedores, sob pena de se tornar mais uma publicação sem possibilidade de adoção prática.

Um desses conceitos, largamente utilizado na modelagem de sistemas em geral, é a Orientação a Objetos. É por isso que HMBS, RMM, HDM e HMT não foram consideradas para este trabalho. HFPM foi desconsiderada por não ser de fato uma metodologia que traz contribuições próprias nos aspectos de navegação e *layout*. DEMAIS não foi considerada por se tratar mais de uma ferramenta do que uma metodologia *per se*.

Das que restam (WAE, OMMMA-L, W2000, WebML, OO-H, UWE, OOHDM e WSDM) foi usado o critério de repercussão no meio acadêmico. Para materializar tal critério, uma pesquisa no Portal da ACM (PORTAL ACM, 2008) foi realizada buscando as metodologias com maior quantidade de artigos e citações (tanto ao artigo principal, ou do livro, como no caso da WAE, como quanto artigos derivados do principal). Dessas, WebML, OOHDM e UWE destacaram-se das demais com 79, 69 e 60 resultados respectivamente (data da pesquisa: 17 de junho de 2008). Utilizando o critério de continuidade de pesquisas, buscou-se a data de publicação do último artigo de cada uma dessas três metodologias: WebML e UWE em 2008 e OOHDM em 2006. Apesar de uma notável diferença nesse aspecto, ainda faltava um critério de desempate e a aderência a UML foi utilizada. Como dessas apenas UWE tem uma aderência total, por não definir diagramas diferentes para nenhum modelo, ela foi a escolhida. UML é uma linguagem de modelagem universalmente aceita e adotada. Não é necessário, pois exigir do modelador nenhum conhecimento novo para que dela possa fazer uso. Bastará entender as extensões propostas em UWE e UWE-R para tal.

Na tabela 1 seguem os critérios resumidos. Alguns deles não foram expostos para algumas metodologias, pelo fato de essas já terem sido desconsideradas por critérios anteriores:

Tabela 1 – Comparação entre metodologias de Engenharia *Web*

Metodologias	Orientada a Objetos	Referência no Portal da ACM	Data de publicação do último artigo	Aderência a UML
HMT	Não	-	-	-
HMBS	Não	-	-	-
RMM	Não	-	-	-
HDM	Não	-	-	-
WAE	Sim	11	-	-
OMMMA-L	Sim	4	-	-
W2000	Sim	14	-	-
WebML	Sim	79	2008	Parcial
OO-H	Sim	2	-	-
UWE	Sim	69	2008	Total
OOHDM	Sim	60	2006	Não aderente
WSDM	Sim	24	-	-

Apesar de inegáveis méritos da RUX, optou-se por propor uma metodologia alternativa a ela, pois essa não utiliza UML. Como exposto em 1.4.1.15, ela se utiliza de uma série de outros padrões e linguagens (para diagramas dos modelos de interface concreta) que, se por um lado talvez sejam mais expressivos, implicam em uma curva de aprendizado que será mais custosa ao modelador e desenvolvedor. Com a extensão proposta neste trabalho, basta o conhecimento de UML e algumas extensões expressas em mecanismos da própria UML: os estereótipos, restrições e valores rotulados.

3.3 – Aprofundamento da UWE

3.3.1 - Generalidades

Na seção 1.4.1.6 do presente trabalho foi feita uma apresentação em alto nível da UWE. Nesta seção, serão aprofundados seus conceitos tendo como referências (KOCH *et al.*,

2008) e (KROIβ ; KOCH, 2008).

Além de ser um perfil (*profile*) de UML ou uma extensão leve (i.e., baseada apenas em estereótipos, valores rotulados (*tagged values*) e restrições (*constraints*)) a UWE aplica o conceito de separação de preocupações (*concerns*). Tal conceito implica na utilização de diagramas diferentes para expressar visualizações diferentes (apresentação, navegação e conteúdo) em fases diferentes (análise, projeto e implementação) e que abordam aspectos distintos da mesma (estrutural e comportamental). Esquematicamente tem-se:

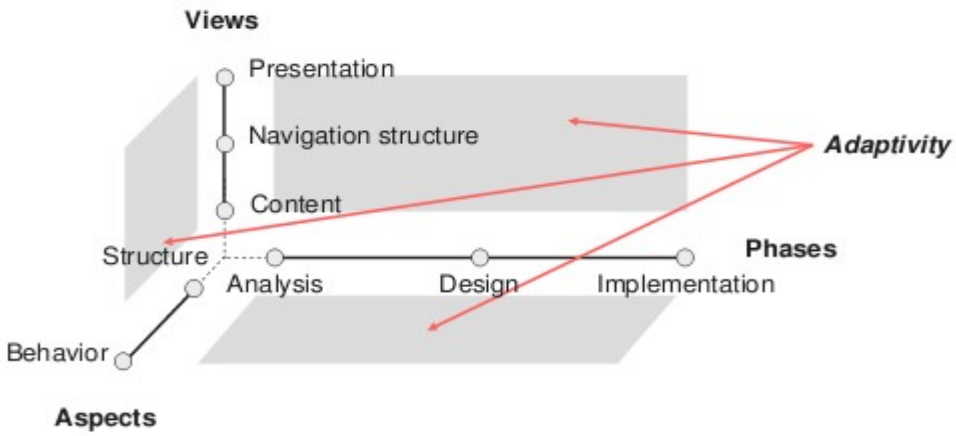

Figura 5 – UWE: separação de preocupações (KOCH *et al*., 2008)

O conceito de personalização (*adaptivity*), ou seja, comportamento diferenciado da aplicação *Web* baseado em contexto ou no usuário que a acessa, é uma preocupação transversal aos demais eixos, como uma quarta dimensão. Na UWE é tratada utilizando Orientação a Aspectos.

Outro conceito importante na UWE é sua orientação a modelos, ou *Model Driven Development* (MDD). Tal orientação é baseada na *Model Driven Architecture* (MDA), um padrão da OMG que pretende oferecer um modo de desenvolver software que parta da utilização de modelos e uma série de transformações automáticas ou semi-automáticas para se chegar ao sistema propriamente dito (MUKERJI; MILLER, 2003). Os *Computational Independent Model* (CIM) são utilizados para especificação de requisitos. Dos CIM são

derivados *Platform Independent Models* (PIM). A partir dos PIMs são obtidos os *Platform Specific Models* (PSM) no qual são baseados os códigos efetivos da aplicação. No caso da UWE, o diagrama de casos de uso é usado como CIM, enquanto os diagramas de conteúdo, navegação, apresentação e processos são exemplos de PIM. No grupo de desenvolvimento da UWE já existem esforços envidados para a criação das transformações necessárias.

A UWE, por ser um perfil de UML, pode ser utilizada em qualquer ferramenta CASE que suporte estereótipos, valores rotulados e restrições. Isso é uma outra grande vantagem desta metodologia, pois não implica na utilização de uma ferramenta diferente da que se esteja acostumado. Basta utilizar os estereótipos definidos pela UWE. É claro que o suporte de uma ferramenta que já incorpore tais extensões e seja capaz de checar automaticamente as restrições impostas facilita sobremaneira o trabalho. Tal ferramenta existe: ArgoUWE, uma extensão *open source* feita pelos criadores da UWE à ferramenta ArgoUML. Além de outras funcionalidades, esta ferramenta implementa algumas das transformações requeridas pela MDA.

3.3.2 – Metamodelo

Como extensão à UML, a UWE expressa seus estereótipos e diagramas na forma de um metamodelo. O metamodelo é uma expressão usando a própria UML para demonstrar novos conceitos não cobertos por essa linguagem originalmente. A UWE é uma extensão conservadora da UML 2.0. Por "conservadora" entenda-se que nenhum dos elementos originais da UML (Classe, Associação etc.) são alterados. Todos os elementos do metamodelo da UWE são referenciados como extensões (mecanismo semelhante à herança entre classes) de elementos existentes na UML. Para definir a semântica estática desses novos elementos é utilizada a OCL (*Object Constraint Language*). Como é compatível com MOF (*Meta-Object Facility*), um modelo escrito baseado no metamodelo da UWE podem ser lidos em outras ferramentas capazes de tratar o formato XMI (*XML Metadata Interchange*).

No metamodelo da UWE existem dois pacotes de alto nível (*Core* e *Adaptivity*), que expressam a já referida separação de preocupações. Dentro do pacote *Core* encontram-se os principais subpacotes da UWE:

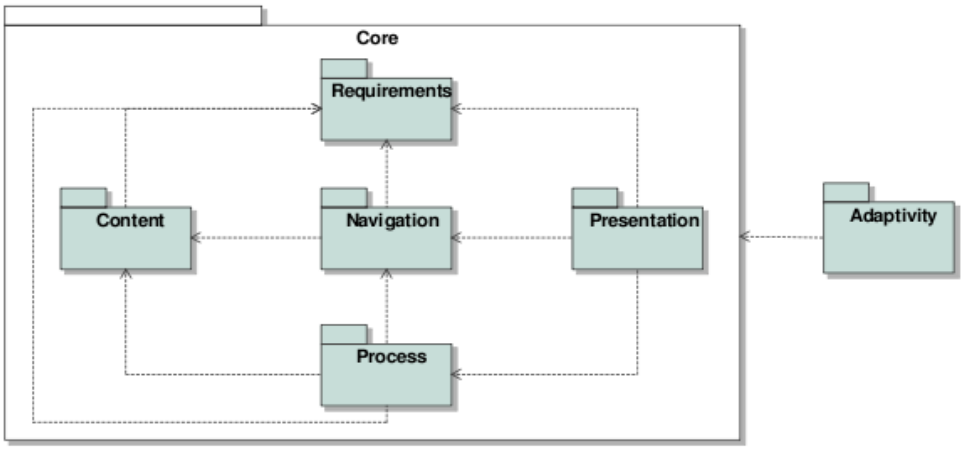

Figura 6 – UWE: visão geral do metamodelo (KROIβ; KOCH, 2008)

No pacote de requisitos (*Requirements*) está o diagrama de casos de uso com algumas pequenas modificações: há uma diferenciação entre casos de uso de navegação, processo e personalização. Casos de uso de navegação são relacionados a buscas, saídas para *sites* externos e visualização de dados de índices e menus. Casos de uso de processo indicam tarefas específicas relacionadas ao domínio do problema em questão. Casos de uso de personalização indicam que a aplicação se comporta de modo diferente (apresentando dados ou menus diferentes) dependendo do contexto ou do usuário.

O pacote de conteúdo (*Content*) não difere da modelagem com UML para sistemas não *web*. Devem ser usados todos os diagramas (notadamente o de classes e seqüência) para expressar as entidades do domínio do problema atacado.

3.3.2.1 – Pacote de Navegação

No pacote de navegação (*Navigation*) começam a surgir os elementos de maior interesse para modelagem de aplicações *web*. A figura 7 ilustra-os:

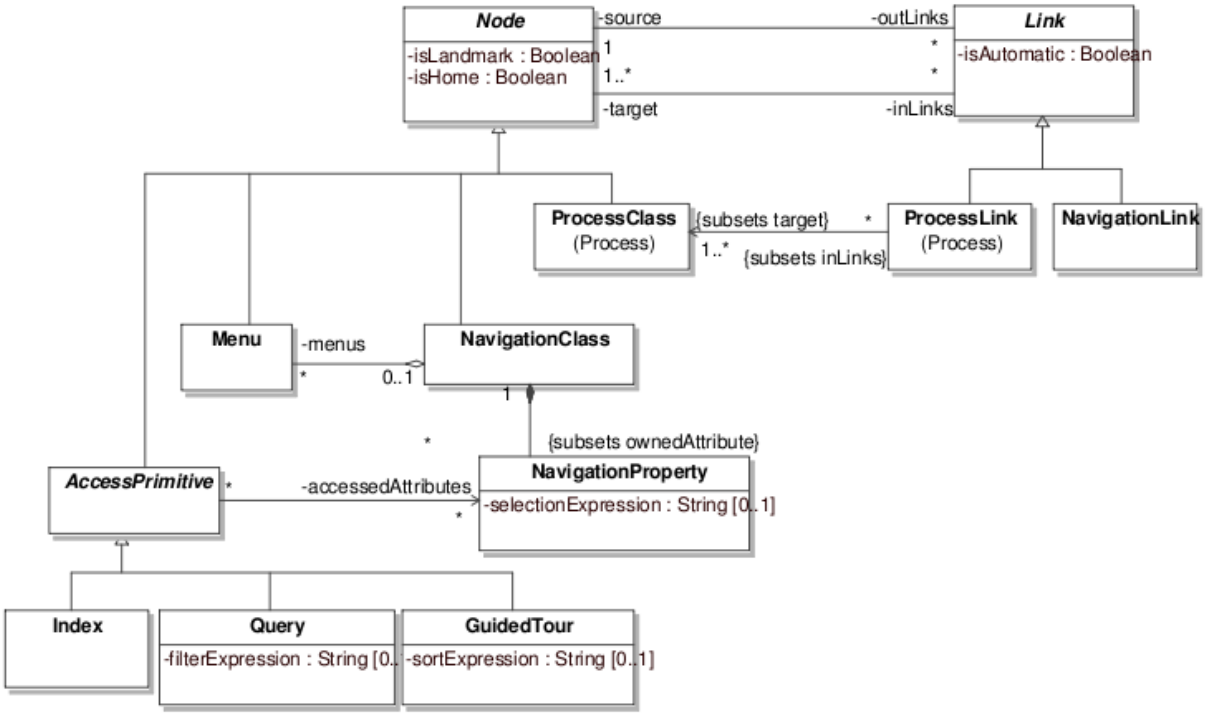

Figura 7 – UWE: pacote de navegação do metamodelo (KROIβ; KOCH, 2008)

Um nó (*Node*) é uma metaclasse abstrata que representa qualquer ponto do grafo de navegação. Quando um nó é alcançado, um conjunto de informações é apresentado ao usuário e opcionalmente esse pode tomar algumas ações. *isLandmark* é um atributo que indica que tal nó pode ser alcançado por todos os demais nós. O atributo *isHome* é um atributo que indica aquele nó como sendo a origem do grafo de navegação. Tem como superclasse no metamodelo da UML a metaclasse *Class*.

Um *link* (*Link*) é uma aresta do grafo de navegação que conecta 2 nós. Trata-se de uma metaclasse abstrata. Não necessariamente representa uma âncora cuja ação feita por um usuário promove a transição de nós. Isso vai depender da organização das classes de apresentação, como será explicado mais adiante. A metaclasse da UML na qual se baseia *Link*

é *Association*. Portanto, em um modelo UWE, uma seta simples será usada para representar um *link*.

Uma metaclasse de navegação (*NavigationClass*) é um tipo de nó (herda de *Node*) e está associada a uma classe de conteúdo, do pacote de conteúdo. Representa no grafo de navegação a entidade do domínio do problema em questão.

Propriedades de navegação (*NavigationProperty*) são atributos da metaclasse de navegação. São eles que definem os conteúdos dos elementos da interface gráfica. Tanto pode ser exatamente um atributo da classe de conteúdo relacionada quanto uma derivação daqueles.

Link de navegação (*NavigationLink*) conecta dois nós desde que nenhum deles seja uma classe de processo (*ProcessClass*).

Menu é usado para tratar alternativas entre caminhos de navegação. Não confundir o Menu do pacote de navegação com a idéia que se faz de menu em uma interface gráfica. Pode-se escolher implementar um Menu com vários nós sendo mostrados ao mesmo tempo, ou seja, o usuário não precisa tomar nenhuma ação para visualizar tais nós.

Primitiva de acesso (*AccessPrimitive*) é uma metaclasse abstrata utilizada para abstrair possíveis modos de acesso a metaclasses de navegação. São subclasses dela: índice (*Index*), consulta (*Query*) e *tour* guiado (*GuidedTour*). O índice mostra um conjunto de referências a metaclasses de navegação baseado em uma seleção anterior. A consulta permite ao usuário entrar com parâmetros que serão usados para obter as metaclasses de navegação a partir de uma fonte de dados. O *tour* guiado provê um conjunto ordenado de instâncias de uma metaclasse de navegação. Com essa primitiva de acesso é possível ao usuário ir para a próxima instância ou a anterior.

3.3.2.2 – Pacote de Apresentação

No pacote de apresentação (*Presentation*) estão os elementos do modelo de apresentação da UWE. Tal modelo, que está baseado no modelo de navegação, é uma visualização abstrata da interface gráfica a ser apresentada ao usuário. Isso implica que

aspectos concretos da interface, como por exemplo, cor, fontes, posição exata, decorações e estilos não são apresentados neste modelo. Nele é descrito apenas a estrutura básica da interface gráfica com seus elementos (âncoras, campos de texto, imagens etc.). A real implementação desses elementos também é abstraída, portanto uma âncora tanto pode ser uma *tag* <*a*> do código html quanto um botão.

O diagrama da figura 8 ilustra o cerne deste pacote:

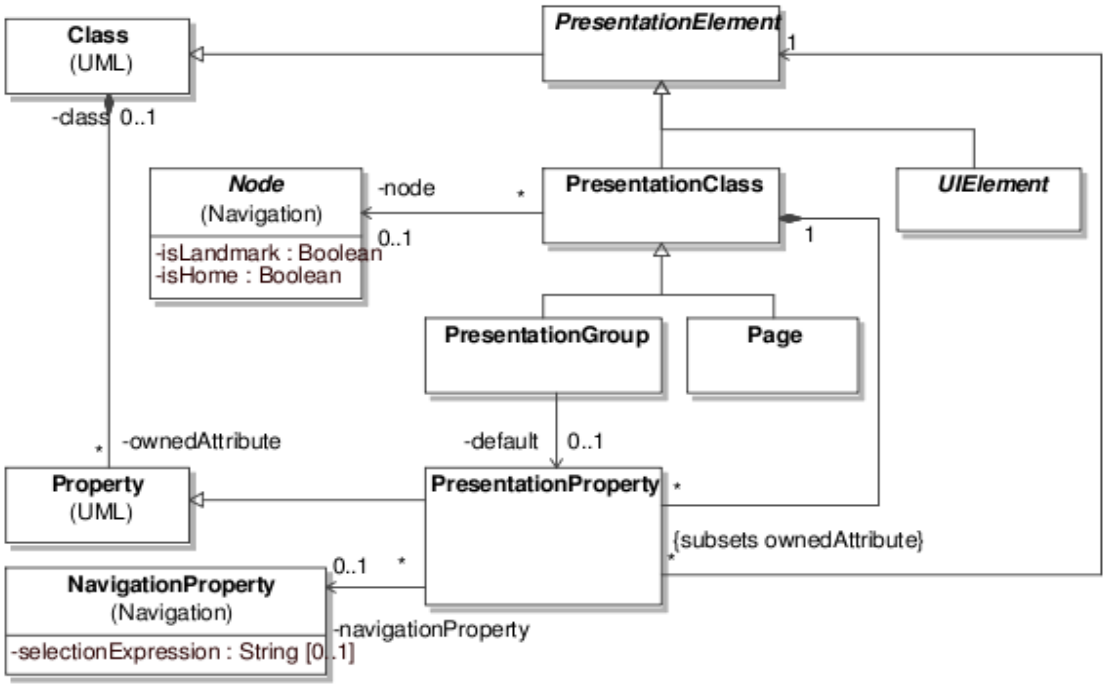

Figura 8 – UWE: pacote de apresentação (KROIβ; KOCH, 2008)

Notar que de acordo com o documento de superestrutura da UML, quando se estende de uma metaclasse da UML para uma metaclasse de um perfil, a relação entre elas é *Extension* e não *Generalization*. Essa última existe para denotar o conceito de herança e, portanto, pode relacionar metaclasses entre si ou classes normais. Na generalização, a representação como ícone é diferente da primeira: a ponta da seta não é preenchida. Na figura Figura 16 será dado um exemplo correto do símbolo de *Extension* (seta preenchida). A diferença é que o mecanismo de herança é entre duas classes ou metaclasses onde a superclasse não está num dos pacotes orginais da UML. Se este for o caso, trata-se de

extensão.

A metaclasse básica é a metaclasse de apresentação (*PresentationClass*). Uma metaclasse de apresentação está relacionada a nós do modelo de navegação. Classes de apresentação podem conter outras classes de apresentação, pois possuem propriedades de apresentação (*PresentationProperty*) cujos tipos podem ser elementos de apresentação (*PresentationElement*). Através dessa composição de classes de apresentação (chamadas de árvores de inclusão), mais de uma delas podem ser mostradas ao mesmo tempo (*links* "seguidos automaticamente"). Caso duas classes de apresentação não estejam compostas, então a transição entre elas deve ser feita através de um *link* ativado pelo usuário. Já um grupo de apresentação (*PresentationGroup*) define um conjunto de classes de apresentação que são mostradas de modo alternativo. Isso quer dizer que se um nó de navegação associado a uma dessas classes é atingido, o conteúdo dessa classe substitui o conteúdo da classe anterior na tela apresentada ao usuário.

Já uma página (*Page*) não pode ser inclusa dentro de uma outra classe de apresentação. Ela define, pois, a raiz de uma composição de classes de apresentação. Uma página não tem que estar associada a um nó de navegação desde que inclua ao menos uma classe de apresentação que tenha referência ao modelo de navegação.

Os elementos de interface gráfica são definidos no seguinte diagrama:

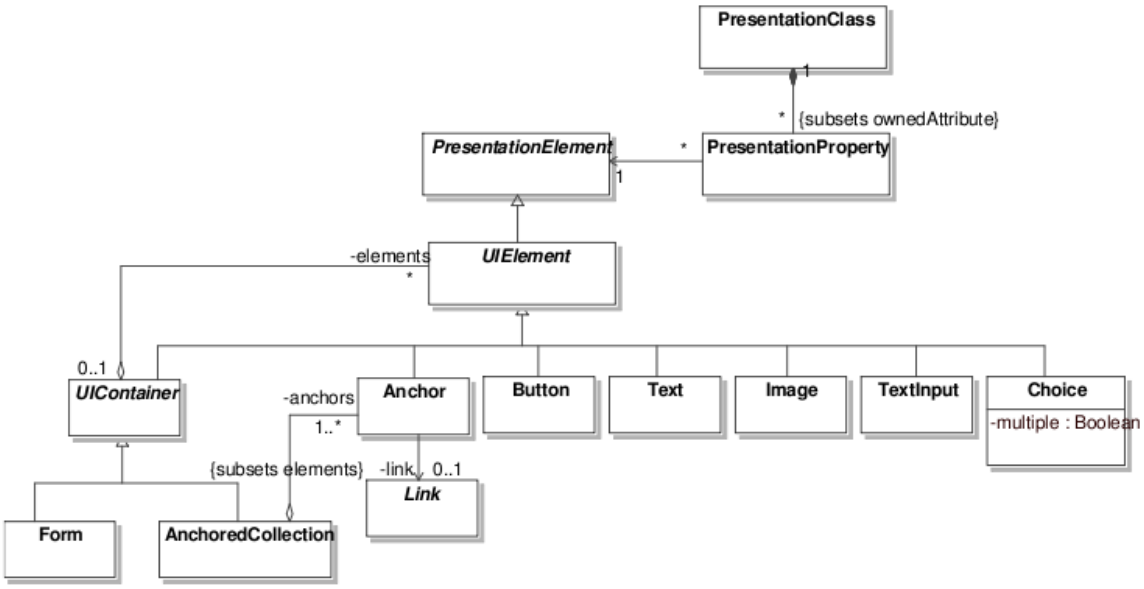

Figura 9 – UWE: elementos de interface gráfica (KROIβ; KOCH, 2008)

Elemento de interface gráfica (*UIElement*) é a super-metaclasse abstrata para apresentação de elementos através dos quais conteúdo pode ser apresentado ou editado. Todo elemento de interface gráfica tem que ser incluso em uma classe de apresentação que está associada a um nó de navegação. São de 4 tipos: 1) contêineres (*UIContainer*), que contêm outros *UIElements*, 2) estáticos, como textos e imagens (*Text* e *Image*), 3) de entrada de dados, como campos de texto ou de escolha (*TextInput* e *Choice*) e 4) gatilhos de transição, como âncoras e botões (*Anchor* e *Button*). Interessante notar que está expresso no diagrama da figura 9 um padrão de projeto conhecido como *composite*: *UIElement* faz o papel de componente, *UIContainer* o *composite* e os demais elementos que não herdam de *UIContainer*, são folhas. Como UIContainer ao mesmo tempo herda de UIElement e é feito (agregado) desse, pode-se ter contêineres dentro de contêineres, além de instâncias das folhas dentro dos contêineres. Notar que entre os possíveis contêineres, há apenas formulários (*Form*) e coleção de âncoras (*AnchoredCollection*). Para SIG-Web usando RIA falta um contêiner que seja capaz de guardar um mapa (em formato *raster* ou vetorial) e que seja capaz de receber eventos de *mouse*.

Notar que já existe na superestrutura da UML (OMG, 2007) uma metaclasse chamada *Image*. Como na documentação da UWE não consta se veio do mesmo pacote (*Profiles*), pode-se considerar isso como uma duplicidade não desejável. O correto seria referir-se à metaclasse já existente, caso ela tenha tudo de que se necessita, ou estendê-la, caso contrário.

4.3.2.3 – Pacote de Processo

Este pacote oferece elementos de modelagem que integram os processos de negócio ao modelo de aplicações *web* da UWE. Na figura 10 o diagrama do Pacote de Processo:

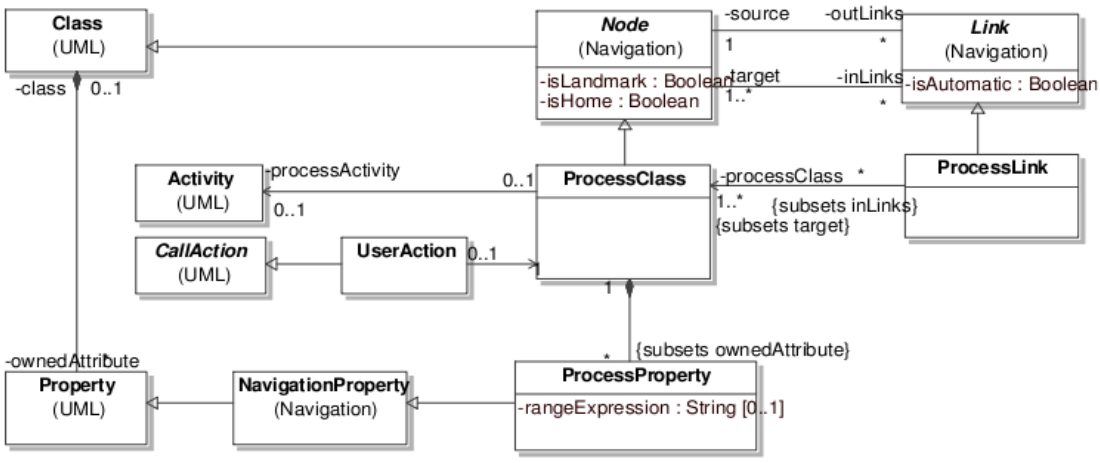

Figura 10 – UWE: Pacote de Processos (KROIβ; KOCH, 2008)

São três as tarefas neste pacote: 1) integração entre processos de negócio e modelo de navegação, 2) definição de uma interface de usuário para dar suporte ao processo e 3) definição do comportamento.

A tarefa 1) é realizada através das metaclasses *ProcessClass* e *ProcessLink*. Elas herdam de *Node* e *Link* respectivamente. Com elas é possível, no modelo de navegação, relacionar uma página ou grupo de páginas com um processo de negócio e ligar de volta esse processo (usando *ProcessLink*) à página ou grupo de páginas apresentadas a seguir ao usuário. Desse modo, pode-se descrever, por exemplo, que primeiro o usuário informa o

login e senha em uma página que tem um formulário adequado, e um processo de *login* é invocado. Esse processo de *login* tem as regras de negócio necessárias para validar as credenciais informadas pelo usuário. A página de resposta (sucesso ou fracasso) é apresentada seguindo um *link* de processo que liga o processo de *login* a uma página com a mensagem apropriada.

A tarefa 2) realiza-se com a utilização dos elementos herdeiros de *ProcessProperty* (propriedades de processo). São eles que determinam quais elementos da interface gráfica alimentam os processos com dados. Como vários dados podem ser necessários em um processo, uma *ProcessClass* apenas fica no modelo de navegação (processo principal) e, ao se definir o comportamento do processo (vide próximo parágrafo), são criados processos para cada passo do processo principal. Cada um desses "sub-processos" terão classes de apresentação específicas. Como exemplo, temos o processo principal como sendo de envio de *email* e tendo como sub-processos a confirmação do envio e outro de validação dos dados.

Para a tarefa 3) há um diagrama de atividades relacionado ao processo principal. Nesse diagrama uma ação de usuário (*UserAction*) denota um ponto de controle onde o usuário deve entrar com dados. No sub-processo (atividade) que segue um *UserAction*, é denotado por um "pino" a entrada de dado no processo (relacionada a uma propriedade do processo) e a saída obtida pela ação do usuário:

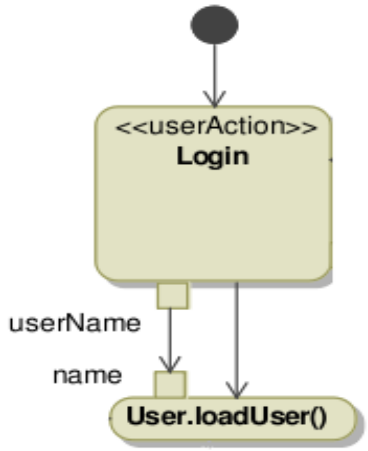

Figura 11 – UWE: Excerto de diagrama de atividade (KROIβ; KOCH, 2008)

3.4 – UWE-R

3.4.1 - Generalidades

Como não faz parte do escopo deste trabalho, os pacotes de Requisitos, Conteúdo e Personalização da UWE não serão alterados. Os pacotes de Apresentação, Navegação e Processo receberão alterações na forma de perfis (*profile*) da UML. Isso implica, de acordo com o documento de superestrutura da UML (OMG, 2007), que as classes já existentes na UWE não podem ser modificadas. Todas as modificações se resumiram em acréscimos de metaclasses para incorporar os conceitos de RIA e SIG-Web.

3.4.2 – Alterações no pacote de Navegação

A UWE já prevê uma metaclasse para navegação (*NavigationClass*), mas na sua definição há restrições típicas de aplicações *web* tradicionais, uma vez que lá refere-se a "estrutura de hipertexto" e há uma relação direta dela para a classe de conteúdo respectiva, pois ela serve de representação daquela no modelo de navegação. No caso de RIA e em especial de SIG-Web, esse cenário pode ser mais complexo. Quando se está em um ponto da navegação de um *site* como o Google Maps, a estrutura não é toda em hipertexto e há várias classes de conteúdo representadas ali. Portanto, é preciso que se crie uma nova metaclasse que represente tal fato. Essa metaclasse tem que herdar de *Node*, para manter a estrutura da UWE e também terá uma relação de composição com *NavigationProperty*. Se não fossem as restrições de perfis da UML, poderia ser aproveitada a classe *NavigationClass*, se seu conceito fosse alterado. Tal nova metaclasse foi chamada de *RichNavigationClass*.

Além disso, é preciso um novo conceito de *Link*. *NavigationLink* é capaz de ligar apenas nós diferentes de processos. No caso de RIA, alguns links podem implicar na execução de processos locais ou remotos, cujos resultados modificam um nó de navegação retornando ao mesmo nó que já era apresentado, ou seja, existe uma ligação entre um nó (*RichNavigationClass*), um processo e de volta ao mesmo nó, com alterações de alguns

atributos. Além disso, há características de assincronia que são importantes de serem modeladas neste novo tipo de *link* e que fazem com que não possa ser aproveitado o já existente *ProcessLink*. Criou-se então *RichNavigationLink*.

Na figura 12 segue o diagrama das novas metaclasses (preenchidas). São apresentadas apenas as metaclasses com que elas se relacionam diretamente.

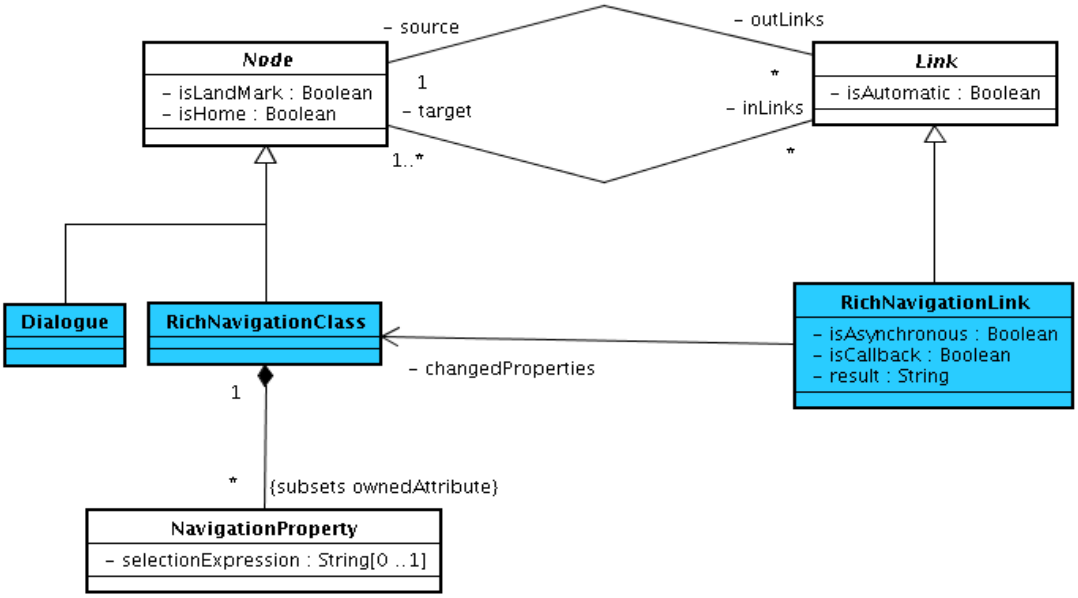

Figura 12 – UWE-R: novas metaclasses de apresentação

Uma classe para representar um diálogo (também conhecida como janela de *pop-up*) foi acrescentada para representar esse elemento de navegação que se sobrepõe de maneira especial aos nós já existentes.

Um exemplo de utilização dessas metaclasses em um modelo de navegação seria:

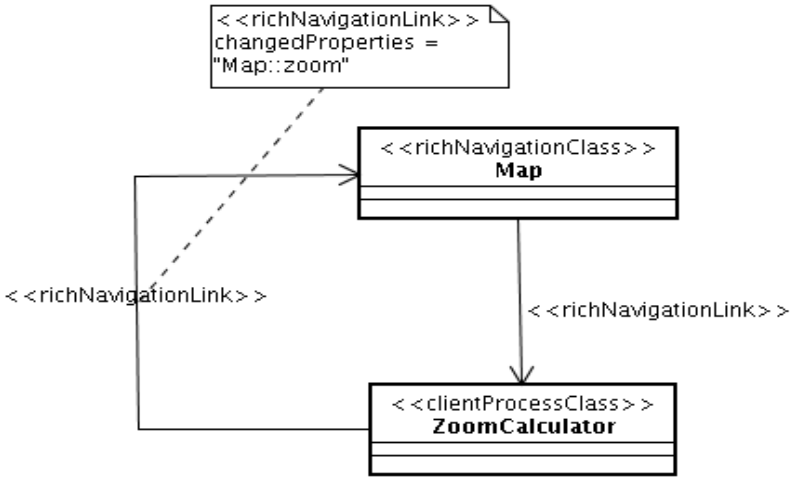

Figura 13 – UWE-R: exemplo de trecho de diagrama de navegação

Como preconizado no referido documento de superestrutura da UML, no modelo de exemplo, uma metaclasse do metamodelo se transforma em um estereótipo, um atributo da metaclasse se transforma em um valor rotulado (*tagged value*) e um papel (como *changedProperties*) transforma-se em uma restrição (*constraint*) expressa com OCL (no diagrama da figura 13, descrita dentro de uma nota).

Já foi utilizado no exemplo anterior, uma metaclasse (*ClientProcessClass*) que faz parte da extensão aqui proposta e será mais bem detalhada nas seções seguintes. No entanto, é possível entender que se trata de um processo que ocorre no lado do cliente (navegador). No caso em que o processo ocorre no servidor, o *link* de navegação pode ser detalhado com a expressão de assincronia, como se vê na figura 14:

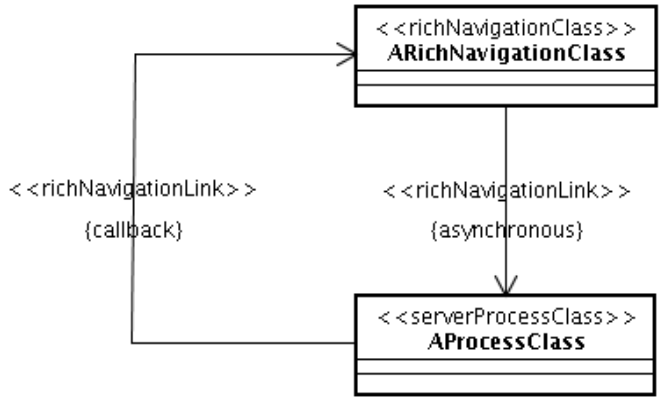

Figura 14 – UWE-R: exemplo de assincronia na navegação

Como a requisição ao processo foi feita de modo assíncrono, então a resposta tem que ser do tipo *callback*. Aparentemente há uma sobrecarga de informações no diagrama (estereótipos e valores rotulados), que em um caso real pode dificultar a sua legibilidade. Assim, sendo, algumas dessas notações podem ser suprimidas ou subentendidas, como a descrita na figura 14. Algumas ferramentas de desenho UML permitem colocar todas as informações e configura-se quais delas são visíveis em um determinado momento. O que importa é que esteja claro o conceito para os modeladores e desenvolvedores. Uma outra boa prática na utilização dessa notação é a criação de diagramas de navegação orientados a casos de uso. Se em uma aplicação *web* tradicional já é possível a partir de um nó de navegação (um menu, por exemplo com vários *links*) escolher caminhos de navegação diferentes e complexos, com RIA isso se torna ainda mais crítico. A recomendação é, portanto, ter diagramas de navegação separados por casos de uso.

Na seção 4 deste trabalho são apresentados exemplos mais completos e complexos para ilustrar o uso da extensão proposta.

3.4.3 – Alterações no pacote de Apresentação

Do ponto de vista da Apresentação algumas extensões são necessárias para representar a riqueza de elementos de interface gráfica de RIA e os de representação de mapas em SIG-

Web. No entanto, não é necessário criar nenhuma metaclasse nova que herde de *PresentationClass* (vide Figura 8). Essa já prevê em sua definição o aninhamento (através das chamadas árvores de inclusão) de vários elementos de apresentação (e portanto com *UIElements* que herdam de *PresentationElement*) através da composição com *PresentationProperty*, que por sua vez está associado a um *PresentationElement*.

Assim sendo foram criadas metaclasses que herdam de UIContainer (*Canvas*, *Panel* e suas subclasses: *AccordeonPanel*, *TabbedPanel* e *TreePanel*) e de UIElement (*Audio*, *Video* e *DialogueWindow*). Segue o diagrama da Figura 9 apenas com as classes novas (preenchidas) e as suas relações diretas:

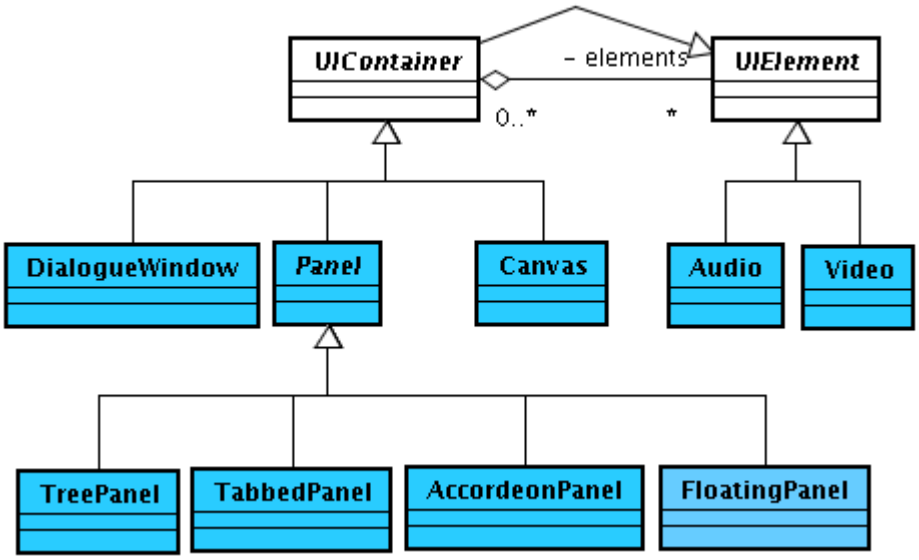

Figura 15 – UWE-R: novas metaclasses de apresentação

A metaclasse *Canvas* é a principal tanto do ponto de vista de RIA quanto de SIG-Web. A palavra canvas em inglês, segundo o dicionário Merriam-Webster, tem o significado de "um pedaço de tecido apoiado ou emoldurado em uma superfície para se pintar". É esse o seu sentido aqui, mas em um contexto computacional, ou seja, é uma área livre de desenho onde eventos de *mouse* são capturados e outros elementos de interface (imagens, outros *Canvas* etc.) podem ser sobrepostos. Até antes desta extensão, a UWE só provia dois tipos de

UIContainer: *Form* e *AnchoredCollection*. Tais elementos fazem com que a expressão de mapas em uma aplicação RIA não seja possível de se modelar. Com o *Canvas* isso se resolve. Um *Canvas* pode conter uma ou várias imagens (sendo que essas imagens podem ser vetoriais ou *raster*) e informações como projeção, escala e *datum* podem ser compostas no próprio mapa. Diferentes camadas de um mesmo mapa podem ser agrupadas utilizando composição entre *Canvas*. Naturalmente, a mesma metaclasse pode ser utilizada para modelar jogos. Um jogo pode ser generalizado como possuindo várias camadas (representadas por *Canvas*), em que se tem uma camada de fundo e cada elemento móvel da tela seria uma outra camada, todas capturando eventos de *mouse*.

A metaclasse abstrata *Panel* foi criada com o intuito de oferecer um *container* que não tivesse, necessariamente uma aparência visual, mas que fosse capaz de sugerir uma ordenação visual dos elementos de interface gráfica na tela. Algumas subclasses de exemplo foram criadas para representar essa organização como um painel com abas, em formato de árvores, acordeão (painéis que se sobrepõem como pastas) ou painéis flutuantes.

Muito da riqueza presente em aplicações RIA vem da possibilidade de se ter vídeo e áudio nas páginas, por isso a criação das metaclasses *Audio* e *Video*. *DialogueWindow* completa as extensões neste pacote, representando uma janela de diálogo.

Apenas uma observação neste pacote e que será detalhada com o auxílio das extensões propostas pela UWE-R no pacote de processo: de acordo com a definição da UWE, quando elementos de interface pertencem à mesma árvore de inclusão, os respectivos nós de navegação são apresentados como se os *links* entre si tivessem sido seguidos automaticamente. No entanto, quando não pertencem é necessária uma ação do usuário (*UserAction*) que gatilhe essa apresentação. Como será visto na próxima seção, há um tipo de ação para a qual isso não é verdade. Afinal, em RIA, podem ocorrer eventos relacionados com *callbacks* e eventos do navegador que não são ações do usuário. Tal ação será modelada na metaclasse *AutonomousAction*.

3.4.4 – Alterações no pacote de Processo

Neste pacote três são as extensões: diferenciação entre processo cliente e processo servidor, uso de diagrama de seqüência com uma mensagem de tipo específico (*ControlMessage*) e criação de um tipo de ação autônoma (*AutonomousAction*) para modelar ações que independem do usuário. Na figura 16 um extrato do diagrama de metaclasses desse pacote, apenas com as novas metaclasses (preenchidas):

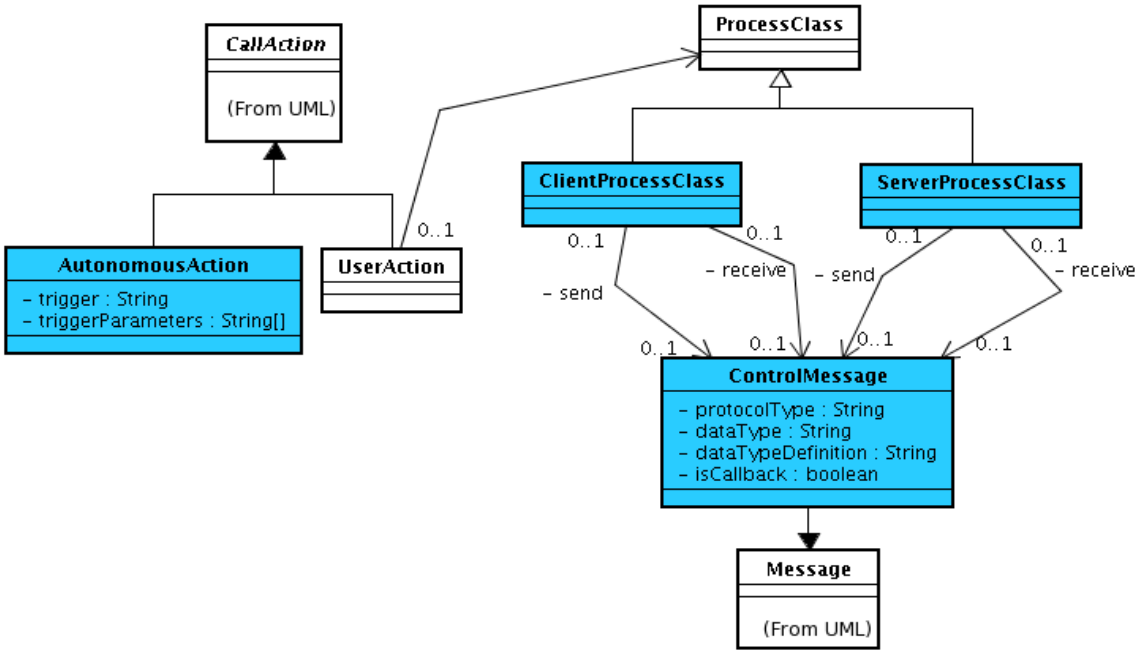

Figura 16 – UWE-R: as novas metaclasses do pacote de processo

A diferenciação entre processo cliente (que roda no navegador através de *script* ou *plugin*) e processo servidor (que roda no servidor *web* ou servidor de aplicação) é muito relevante no contexto de RIA. Como para esse tipo de aplicação há uma porção relevante do código que fica no lado do cliente, é muito importante ter mecanismos de indicar isso durante a modelagem do sistema. Talvez no nível de análise, possa-se usar a metaclasse já proposta pela UWE, mas quando se projeta o sistema tendo em vista a sua implementação, essa diferenciação é bastante importante. Idealmente *ProcessClass* deveria se tornar uma metaclasse abstrata, uma vez que não haveria instâncias concretas diretamente dela em uma

modelagem realista. Entretanto, como a proposta aqui é não modificar em nada a UWE, seguindo as restrições de perfil da UML, ela foi mantida tal como definida. A contribuição da UWE-R neste pacote é o acréscimo do modelo de interação expresso através de um diagrama de seqüência, como descrito nos próximos parágrafos.

O pacote de processo da UWE tem três tarefas como descrito em 3.3.2.3: 1) integração entre processos de negócio e modelo de navegação, 2) definição de uma interface de usuário para dar suporte ao processo e 3) definição do comportamento. No caso de RIA, como existem comunicações assíncronas com o servidor e os dados muitas vezes não são simples parâmetros enviados por HTTP GET ou POST, mas sim em formato XML (usando ou não SOAP), JSON e outros, faz-se necessária a utilização de diagramas de seqüência que explicitem essas comunicações. Além disso, é necessária um tipo de mensagem específica que tenha essas informações. Criou-se a *ControlMessage* que estende o conceito da metaclasse *Message* (da UML, pacote *BasicInteractions*). Tem esse nome, por se tratar de uma mensagem entre a camada de controle e de modelo (do padrão de projetos *Model-View-Controller* - MVC). *Message* já possui um atributo (*messageSort*) que indica se uma mensagem é assíncrona ou não, portanto não é preciso criar esse atributo. Os novos atributos de *ControlMessage* são: isCallback (apenas será denotada quando se tratar de um *callback* do servidor para o cliente), *protocolType* (tipo do protocolo. Valores reservados: "LOCAL" para chamadas locais no cliente, "HTTP" para simples GET ou POST, "FTP", "REST" e "SOAP"), *dataType* (tipo do dado enviado ou recebido. Valores reservados: "JSON", "XML","KML", "GML" e "STRING") e *dataTypeDefinition* (definição do tipo: elemento do *Schema* ou *DTD* que esquematiza o XML ou referência da *EBNF* que formata o *JSON*[2]). Um exemplo de sua utilização é dada pelo modelo de interação da figura 17:

2 Schemas e DTDs são padrões do W3C para definição de tipos ou elementos em XML. JSON é um tipo de codificação de dados de modo estruturado, sem o uso de XML, utilizado em aplicações *web*. O modo de definir um tipo JSON é utilizando *Extendend Backus-Naur Form* (EBNF), também utilizada na definição de gramáticas de linguagens de programação. *Simple Object Access Protocol* (SOAP) é um protocolo definido pelo W3C que encapsula mensagens XML em um envelope, que possui cabeçalho e corpo. Já com *Representation State Transfer* (REST) não há a necessidade de enviar esse envelope, mas apenas o XML sobre HTTP.

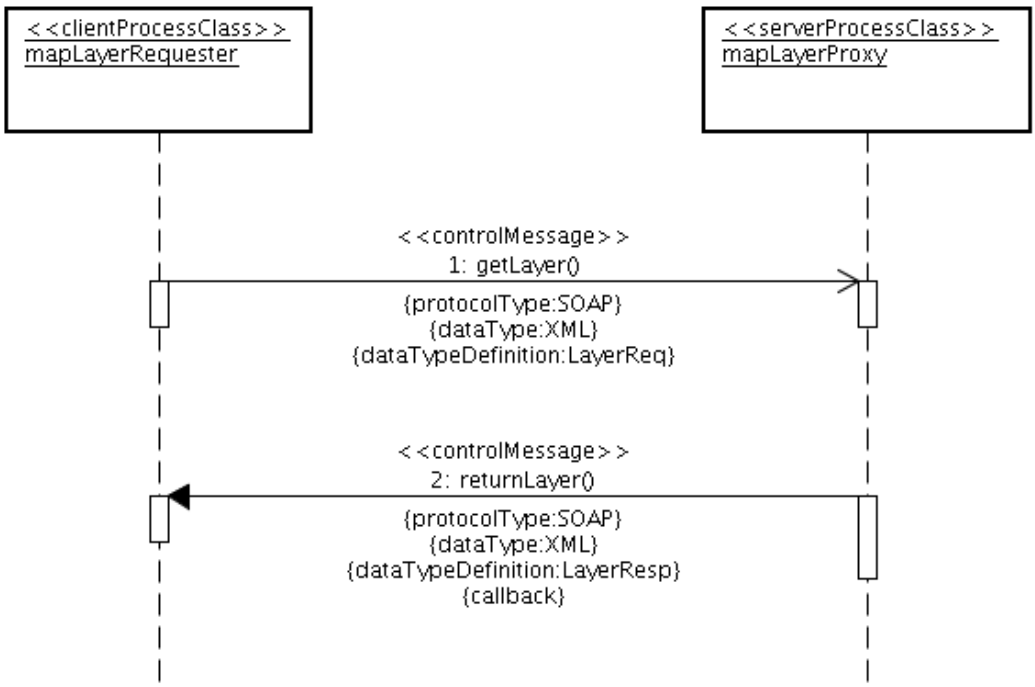

Figura 17 – UWE-R: Modelo de interação

Na figura 17 é feita uma requisição assíncrona de uma classe do cliente para requisitar uma camada do mapa (elemento LayerReq definido em um *Schema*) , e a resposta (elemento LayerResp definido em um *Schema*) vem de modo síncrono, do servidor. Assim, durante a modelagem do projeto da aplicação tem-se uma noção muita clara de como os dados serão trocados, de que tipo e com que freqüência. Isso é particularmente importante para que a aplicação RIA tenha de fato, não só do ponto de vista de aparência da tela, mas do ponto de vista de interação do usuário, a "riqueza" que carrega em seu nome. Isso é dito aqui, pois muitos *sites* desenvolvidos atualmente com tecnologias RIA claramente carregam em seus projetos as mesmas idéias das aplicações *web* tradicionais. Muitas delas não se beneficiam das interações assíncronas e a experiência do usuário acaba sendo pobre. Com um modelo como o da figura 17, fica evidenciado se a aplicação em questão aproveita-se dessas interações ou não.

Outras duas vantagens na utilização desse modelo é para testes e para avaliação de desempenho da aplicação. Um dos objetivos de se modelar um sistema é permitir que os

testes possam começar o quanto antes, garantindo uma maior qualidade do sistema. Assim, com os casos de uso já é possível fazer casos de teste. Do mesmo modo, se a interação é explicitada como no diagrama da figura 17, testes unitários e de integração interna já podem ser definidos. Do ponto de vista de desempenho, patenteia-se a quantidade de interações entre cliente e servidor e o volume de dados trafegados (pelo tipo de dados definido). Portanto, gargalos de desempenho podem ser antecipados e trabalhados ainda na fase de projeto.

A última extensão proposta neste pacote é a criação da *AutonomousAction*. Tal ação diferencia-se da *UserAction* da UWE, pois essa última implica na interação do usuário. Ora, em RIA, uma determinada ação pode ser executada por um evento do navegador (ao carregar uma página, por exemplo), por um temporizador ou pela chegada de uma mensagem de *callback*. São atributos desta metaclasse: *triggerType* (o tipo de gatilho para sua execução. Valores reservados: "EVENT" para eventos do navegador, "TIMER", para temporizador e "ONCALLBACK" para recebimento de *callbacks*), *triggerName* (nome do evento do navegador ou da função de *callback*. Não é utilizado para temporizador) e *triggerParameters* (série de parâmetros para os eventos do navegador ou da função de *callback* se for necessário, ou valor do tempo no caso de "TIMER"). Evitou-se usar um nome como "*EventAction*", pois já existe uma *AcceptEventAction* na superestrutura da UML, o que poderia gerar confusão.

4 – EXEMPLOS DE APLICAÇÃO DA UWE-R

4.1 – Introdução

Nesta seção, a UWE-R é aplicada a 3 diferentes SIG-Web disponíveis na Internet que foram feitos com tecnologias RIA: (GEABIOS, 2007) feito com Openlaszlo, (GOOGLE MAPS, 2007) feito com GWT e (YAHOO! MAPS, 2007) feito com Flex. Como não se dispõe dos códigos fonte de nenhuma delas, suas modelagens são abordadas através de um processo de engenharia reversa, observando seus comportamentos. É importante que o leitor tenha aberto em navegadores cada um desses *sites* enquanto lê as modelagens aqui propostas. O objetivo é verificar o poder expressivo da UWE-R na modelagem de SIG-Web com RIA.

Serão apresentados modelos de navegação, apresentação, processo e atividades para pelo menos um caso de uso, dando ênfase aos elementos novos trazidos pela UWE-R.

4.2 – UWE-R aplicada ao Google Maps

Para este SIG-Web foi escolhido o caso de uso de busca de endereço. O diagrama de navegação segue na figura 18:

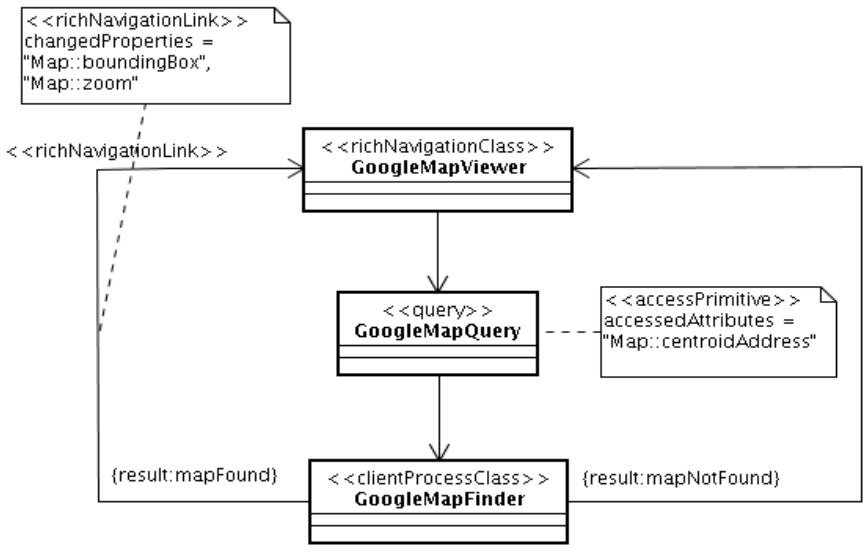

Figura 18 – UWE-R para Google Maps: modelo de navegação

De imediato nota-se a presença de uma classe de navegação rica (*GoogleMapViewer*). Neste caso de uso (e em quase todos os demais) para fins de navegação, todos os *links* retornam a ela mesma, demonstrando a continuidade visual típica de aplicações RIA. Em seguida, usa-se uma classe com o estereótipo <<query>>, para denotar a busca a ser realizada. Tal estereótipo é da própria UWE. Já a classe seguinte (*GoogleMapFinder*) implementa o estereótipo <<clientProcessClass>> proposto pela UWE-R, ou seja, explicita que é um processo que age no navegador, com possível interação com o servidor (a ser explicitada nos diagramas posteriores). Em seguida, o <<richNavigationLink>> presente denota o retorno à mesma classe de navegação, tendo o atributo do mapa daquela alterado para refletir o resultado da busca (retângulo envolvente e zoom adequados). Ambas as situações (mapa encontrado ou não) são retratadas utilizando o valor rotulado "result" de <<richNavigationLink>>. Como se trata de uma requisição síncrona (o usuário tem que esperar terminar a busca para ter seu mapa mostrado) o <<richNavigationLink>> em questão não tem o valor rotulado de *callback*.

Seguindo a metodologia proposta pela UWE, apenas o processo principal (relativo à classe GoogleMapFinder) fica exposto no diagrama de navegação. É preciso um modelo de processos (representado por um diagrama de atividades) que mostre como de fato o mapa é encontrado. Antes disso, porém, é usado um diagrama de classes para representar os demais processos que fazem parte deste processo principal:

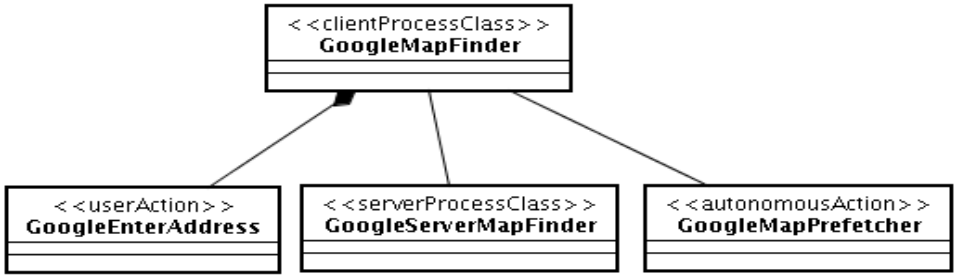

Figura 19 – UWE-R para Google Maps: classes de processos

No diagrama da figura 19 são utilizados os estereótipos para representar uma ação do

usuário (*GoogleEnterAddress*) com um estereótipo já previsto na UWE e uma classe de processo que é executada no servidor, proposta pela UWE-R. O diagrama de atividades da figura 20, parte do modelo de processo, ilustra o funcionamento deste caso de uso:

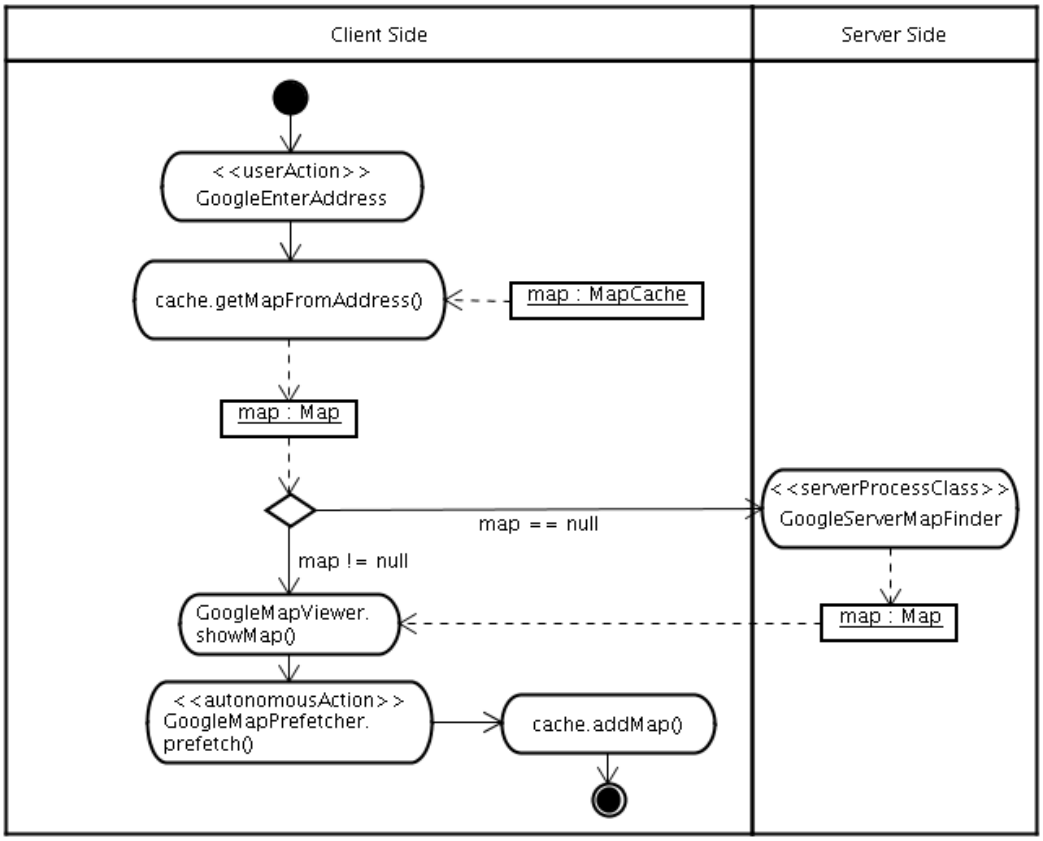

Figura 20 – UWE-R para Google Maps: parte do modelo de processos

Neste diagrama foram utilizadas raias (*swimlanes*) para patentear a interação existente entre o lado cliente e servidor da aplicação. Inicialmente, no lado do cliente busca-se um mapa que esteja no cache e que atenda a consulta do usuário. Como toda boa aplicação RIA, o Google Maps demonstra esse comportamento de só acessar o servidor quando é realmente necessário. Por isso, nesta modelagem foi considerado algum mecanismo de cache para mapas. Apenas se o mapa não existe no cache, é que o processo servidor (*GoogleServerMapFinder*) é invocado. Por fim, um outro comportamento típico de RIA e apresentado pelo Google Maps é a busca de mapas no entorno do mapa encontrado. Trata-se

de uma ação autônoma (proposta pela UWE-R), que nesse caso, foi invocada após a apresentação do mapa buscado, mas pelo observado no *site*, é também invocado temporalmente sempre que o usuário não interage com a aplicação. Eis um exemplo de como uma aplicação *web* provê a ilusão de uma aplicação *desktop*: ao deslocar o mapa para regiões vizinhas, ele poderá ser apresentado sem interação com o servidor e, portanto, imediatamente, como se todos os dados estivessem desde sempre na máquina do usuário. No modelo de interação referente àquela ação autônoma ficará representado que o mapa encontrado é colocado no cache.

Uma outra contribuição da UWE-R encontra-se no fato de explicitar as interações dos processos entre si com o modelo de interação. Eis o exemplo neste caso:

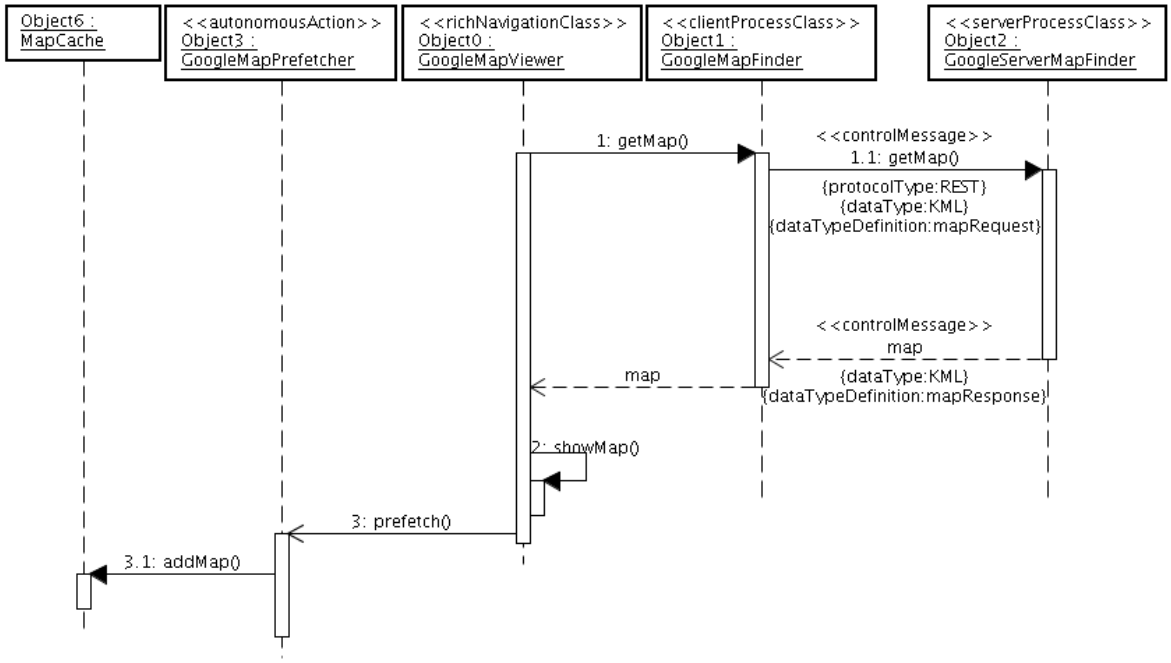

Figura 21 – UWE-R para Google Maps: modelo de interação

Na figura 21 fica mostrado o tráfego de mensagens entre as classes de processo servidor e cliente. Foi utilizado o estereótipo <<controlMessage>> proposto pela UWE-R, neste caso no modo síncrono. Os valores rotulados deixam claro quais são os dados e protocolos utilizados. Aqui pressupõe-se a utilização de KML, por ser esse o formato usado

pela Google. No entanto, trata-se de uma pressuposição apenas, que em nada prejudica a ilustração de expressividade da notação proposta.

Também foi utilizado um modelo de interação para ilustrar o funcionamento detalhado da <<autonomousAction>> *GoogleMapPrefetcher*:

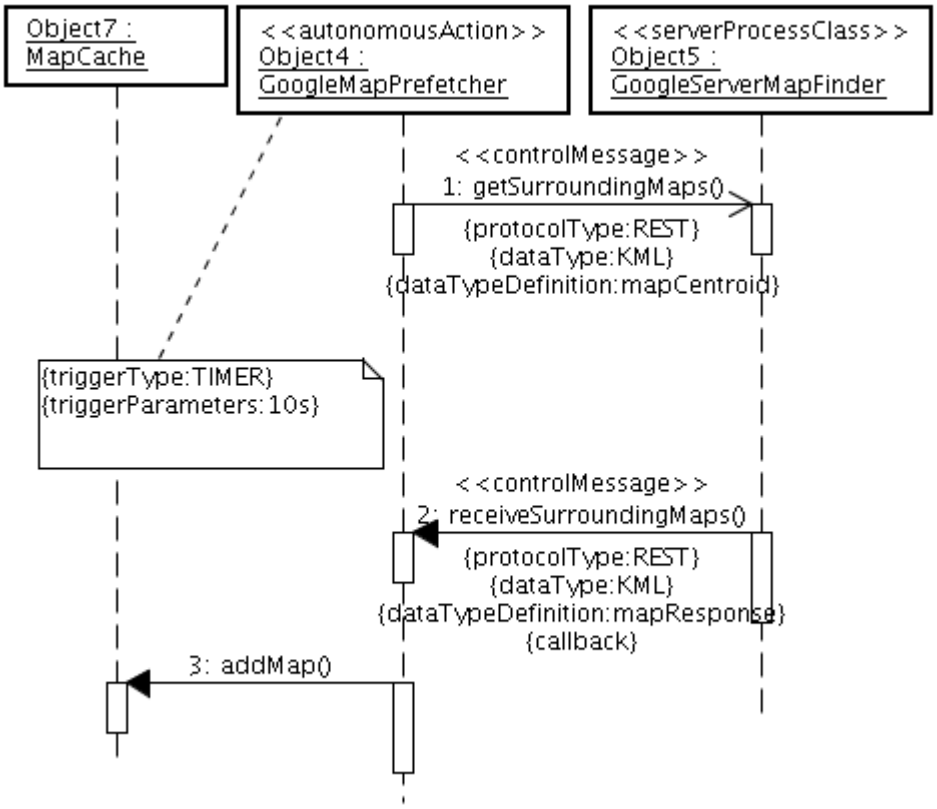

Figura 22 – UWE-R para Google Maps: Modelo de interação para *prefetch*

Na figura 22 são mostrados os valores rotulados que detalham como a ação autônoma em questão é invocada: através de um evento temporal (a cada 10s de inatividade). Neste caso, trata-se de uma chamada assíncrona ao servidor, portanto *receiveMapSurroundings()* é uma *callback* no cliente (foi utilizada a restrição definida na <<controlMessage>> como definida pela UWE-R) .

Por fim, segue o modelo de apresentação:

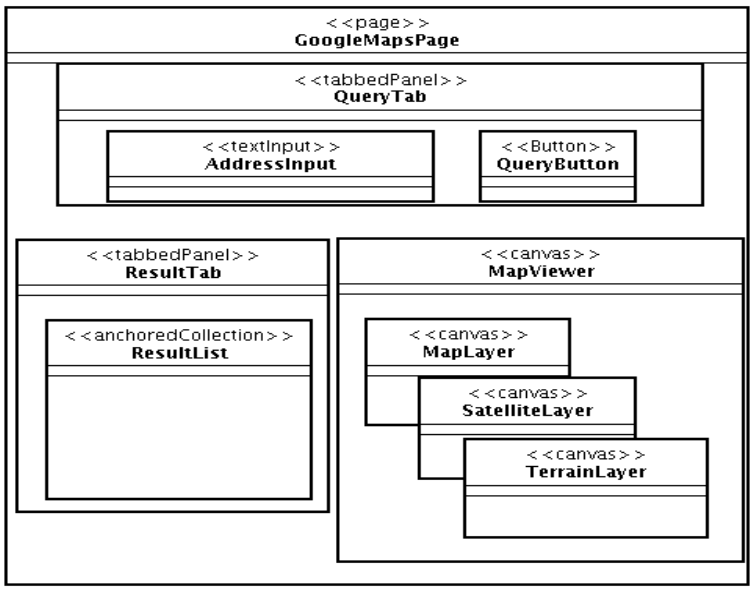

Figura 23 – UWE-R para Google Maps: modelo de apresentação

A modelagem feita abstrai os elementos que estão em torno dos demais painéis (*links*, textos e imagens) para melhor entendimento visual. Caso se quisesse fazer uma modelagem mais literal, bastaria colocar uma classe de apresentação dentro de GoogleMapsPage, acrescentando tais elementos. Notar a conveniência dos painéis com abas que de fato estão no *site* em questão e a utilização de <<canvas>> para representação da região do mapa e suas diversas camadas.

4.3 – UWER-R APLICADA AO GEABIOS

Para este site escolheu-se a funcionalidade de busca de localidade. É importante notar que este *site* apresenta restrições nessa funcionalidade. Nele é possível apenas encontrar localidades como cidades, regiões, territórios, montanhas e não endereços específicos de uma cidade, como no caso do Google Maps.

Para obter os dados e imagens dos mapas o GeaBios acessa servidores WMS e WFS. Tais servidores são fixos na aplicação, mas dentre eles pode-se escolher quais camadas desejadas para apresentação na opção “Setup -> Layers”.

Segue seu modelo de navegação:

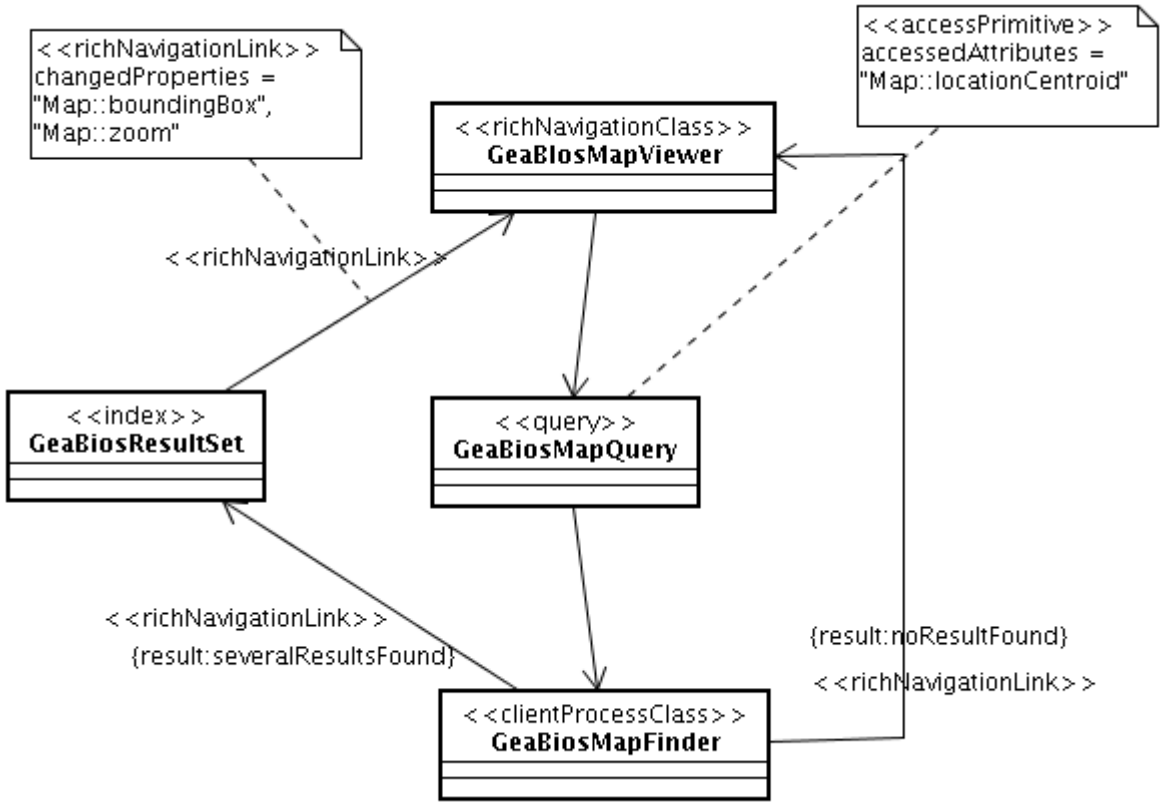

Figura 24 – UWE-R para GeaBios: modelo de navegação

Notar que há uma maior quantidade de resultados possíveis, já que não se entra com um endereço específico para a consulta. Por isso um nó intermediário de resultados encontrados foi adicionado. De resto a modelagem é bem parecida com a do Google Maps.

Este *site* é um exemplo de uso de tecnologia RIA com a modelagem seguindo idéias da *web* tradicional: cada vez que é feita uma requisição a um mapa contíguo ou a um novo que já havia sido mostrado, uma nova requisição é realizada. Isso fica evidenciado pela presença de um *log* das ações que indicam acesso ao servidor de mapas. Tal *log* fica visível ao usuário o tempo todo. As classes de processo, portanto não tem um *prefetcher*:

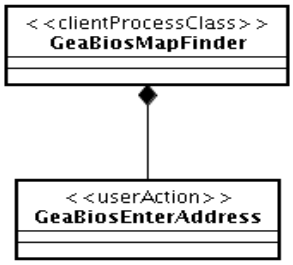

Figura 25 – UWE-R para GeaBios: classes de processos

Por conseguinte, o diagrama de atividades que modela esse processo vai evidenciar o caráter síncrono dessa implementação (não há uso de nenhum mecanismo de cache). A classe de processo cliente responsável pela busca da localidade atua apenas como um requisitante de mapas para os servidores de WMS e WFS cadastrados:

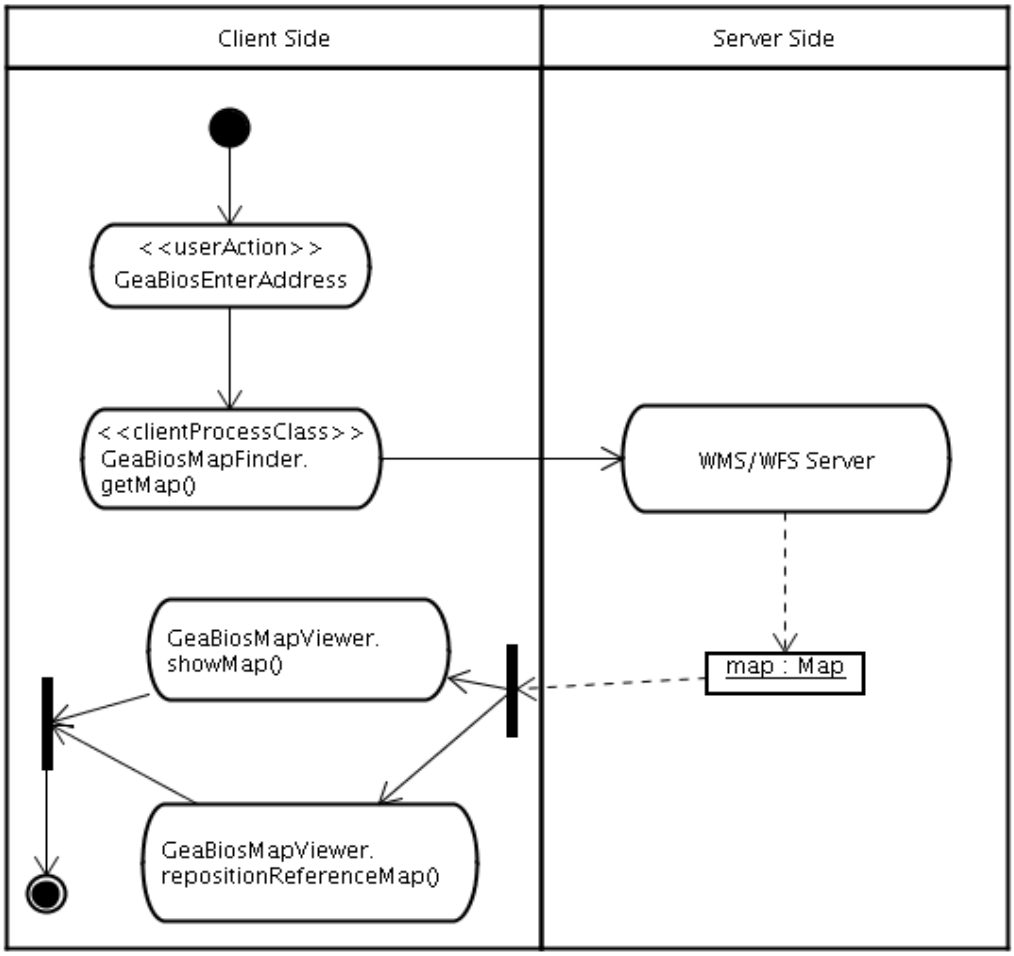

Figura 26 – UWE-R para GeaBios: modelo de processos

Notar que há uma atividade adicional de reposicionamento do mapa de referência, que será refletida na interface ao usuário. Tal reposicionamento ocorre em paralelo com a apresentação do mapa principal.

Para o modelo de interação, notar a natureza síncrona da operação. Também notar que, como se trata de uma requisição a um servidor WMS, o mapa vem como uma imagem *raster*, tipicamente. Por isso o tipo de dado é "STRING" e não há *dataTypeDefinition*, pois não há Schema ou EBNF contra o qual validar a requisição.

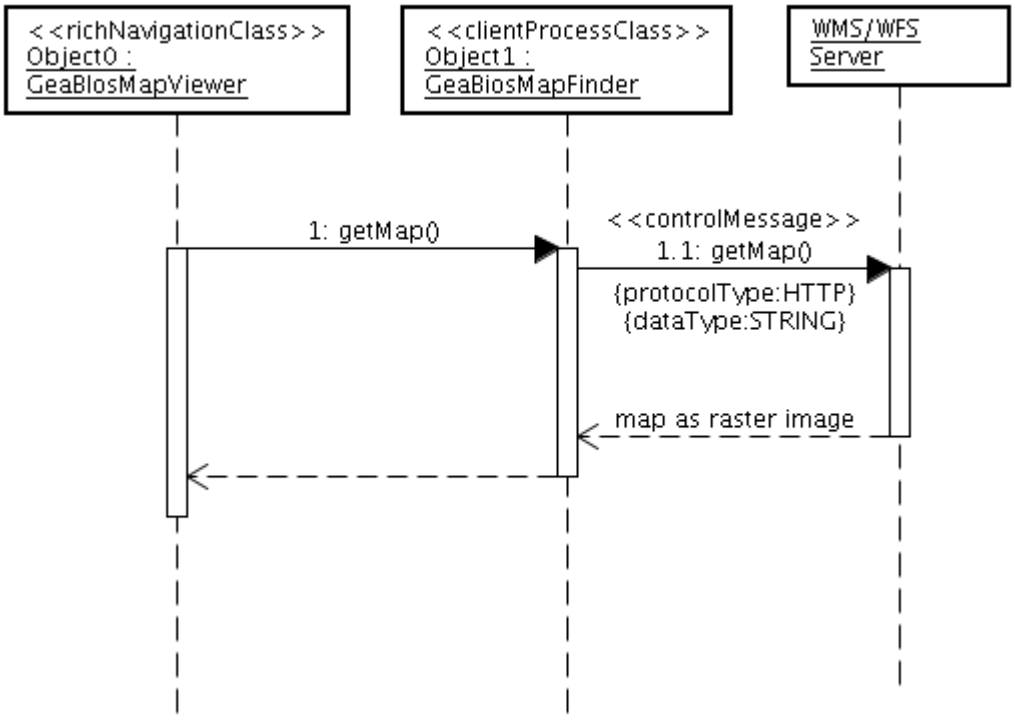

Figura 27 – UWE-R para GeaBios: Modelo de interação

Para a modelagem de apresentação, tem-se o seguinte diagrama:

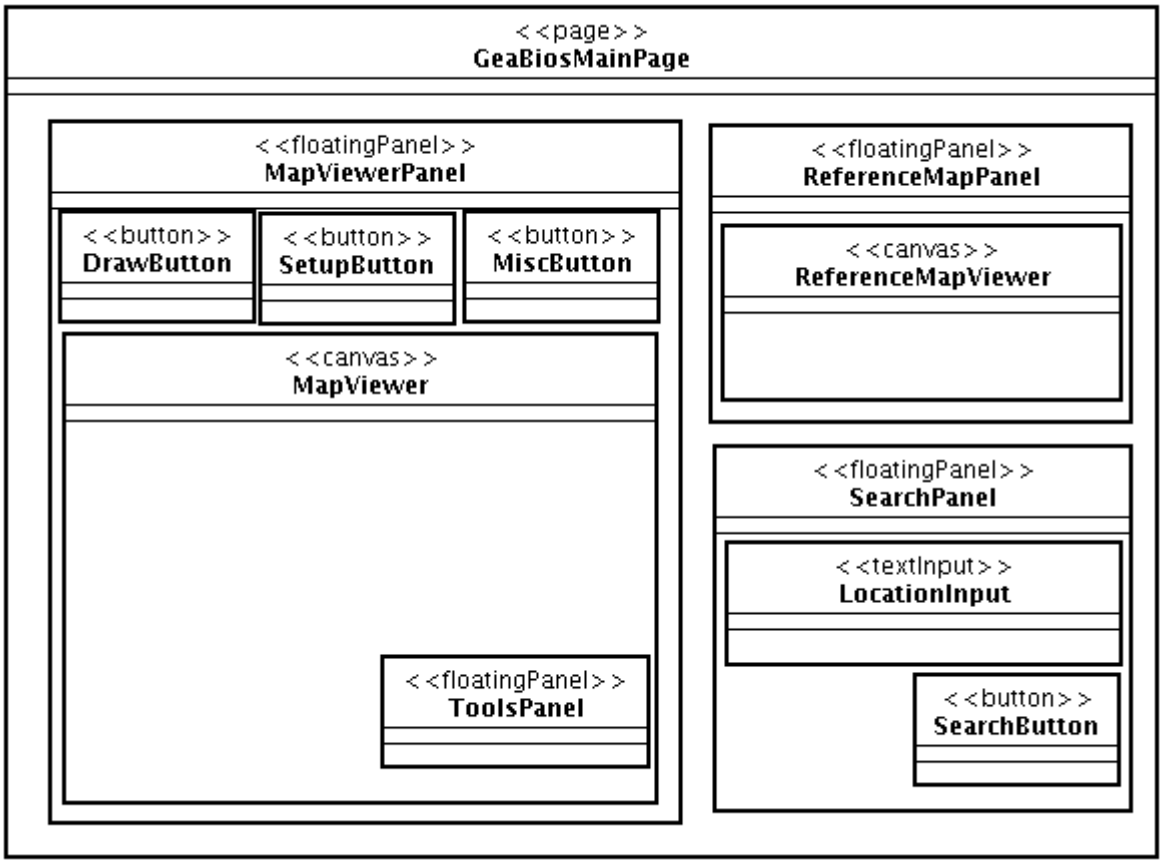

Figura 28 – UWE-R para GeaBios: modelo de apresentação

Notar o uso de painéis flutuantes (estereótipo <<floatingPanel>>), extensão proposta pela UWE-R. Para fins de simplificação, apenas os principais botões de *MapViewerPanel* foram colocados.

4.4 – UWE-R aplicada ao Yahoo! Maps

Para este *site* foi escolhido um caso de uso diferente: obter trajeto entre 2 endereços. O usuário informa 2 endereços e o sistema mostra-os no mapa com uma marcação da trajetória, bem como uma descrição textual dos trechos da mesma. A seguir o modelo de navegação:

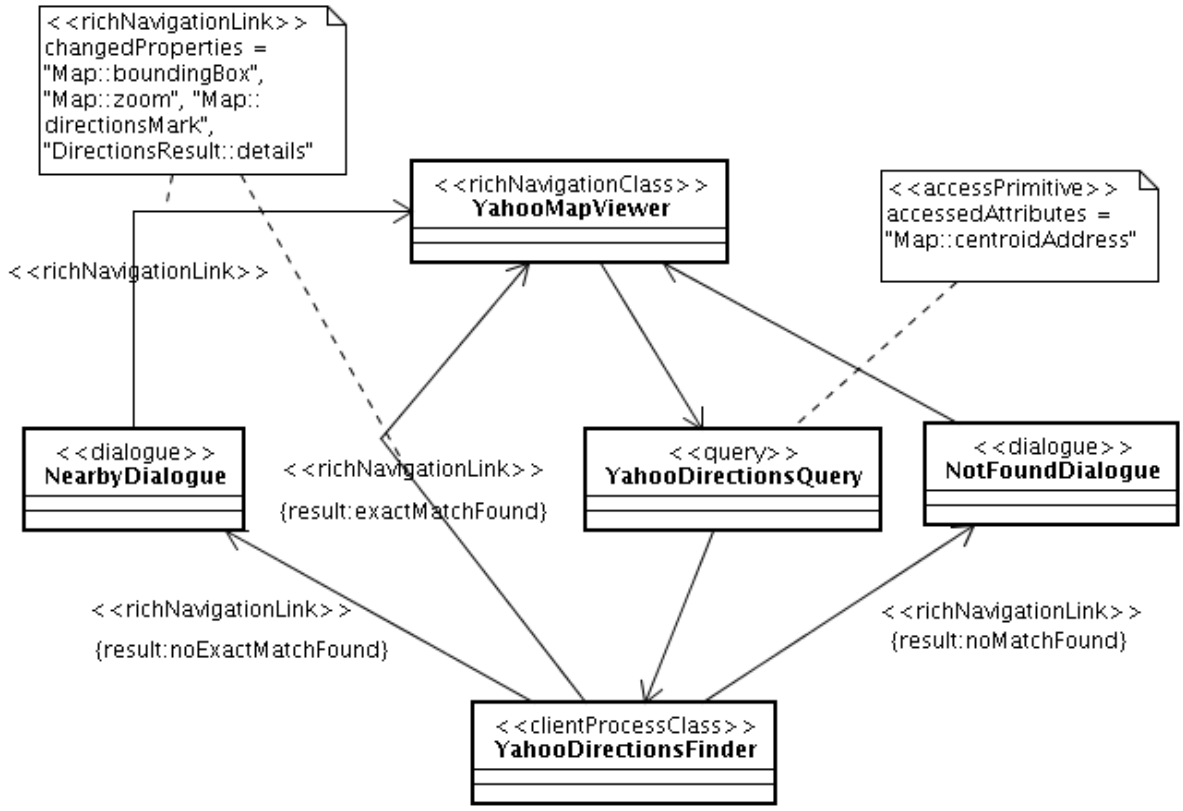

Figura 29 – UWE-R para Yahoo! Maps: modelo de navegação

Notar que há alguns comportamentos diferenciados em relação aos demais *sites*: o Yahoo! Maps tenta achar endereços com a grafia mais próxima daquela digitada pelo usuário e com a maior proximidade geográfica da cidade ou país discriminados. Tal tentativa ocorre para ambos os endereços digitados. Isso é feito com freqüência resultando em poucos casos onde o endereço não é encontrado. Tanto no caso de endereço próximo encontrado quanto no caso de endereço não encontrado, um diálogo é mostrado ao usuário notificando desse resultado. Tais diálogos não bloqueiam a navegação: são apresentados ao mesmo tempo em que o <<richNavigationLink>> seguinte é seguido. Importante perceber a modelagem feita para as várias atualizações que são realizadas no mapa quando se encontram resultados exatos ou próximos: não apenas o mapa muda de retângulo envolvente e zoom, mas uma marca com a trajetória encontrada é colocada no mapa e os detalhes textuais dela são acrescentados. Esses últimos são colocados em um outro atributo (lista de resultados de trajetória) e não no

mapa, por isso a alteração em *DirectionsResult* e não em *Map*.

Para o modelo de processos, as classes existentes são muito similares aos do Google Maps, uma vez que o Yahoo! Maps apresenta o mesmo mecanismo de cache e pré-carga de mapas:

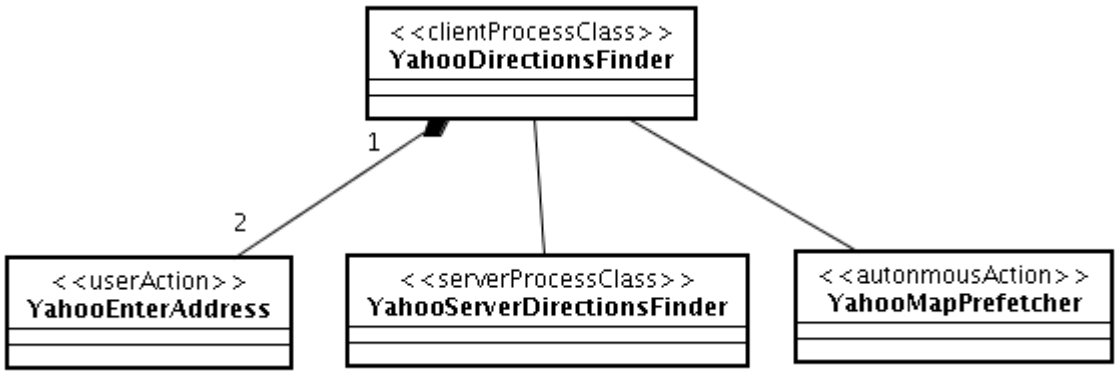

Figura 30 – UWE-R para Yahoo! Maps: classes de processos

A única diferença é pela explicitação feita quanto à multiplicidade existente para a <<userAction>> YahooEnterAddress: são 2 os endereços buscados.

Dentro do modelo de processos tem-se o diagrama de atividades da figura 31:

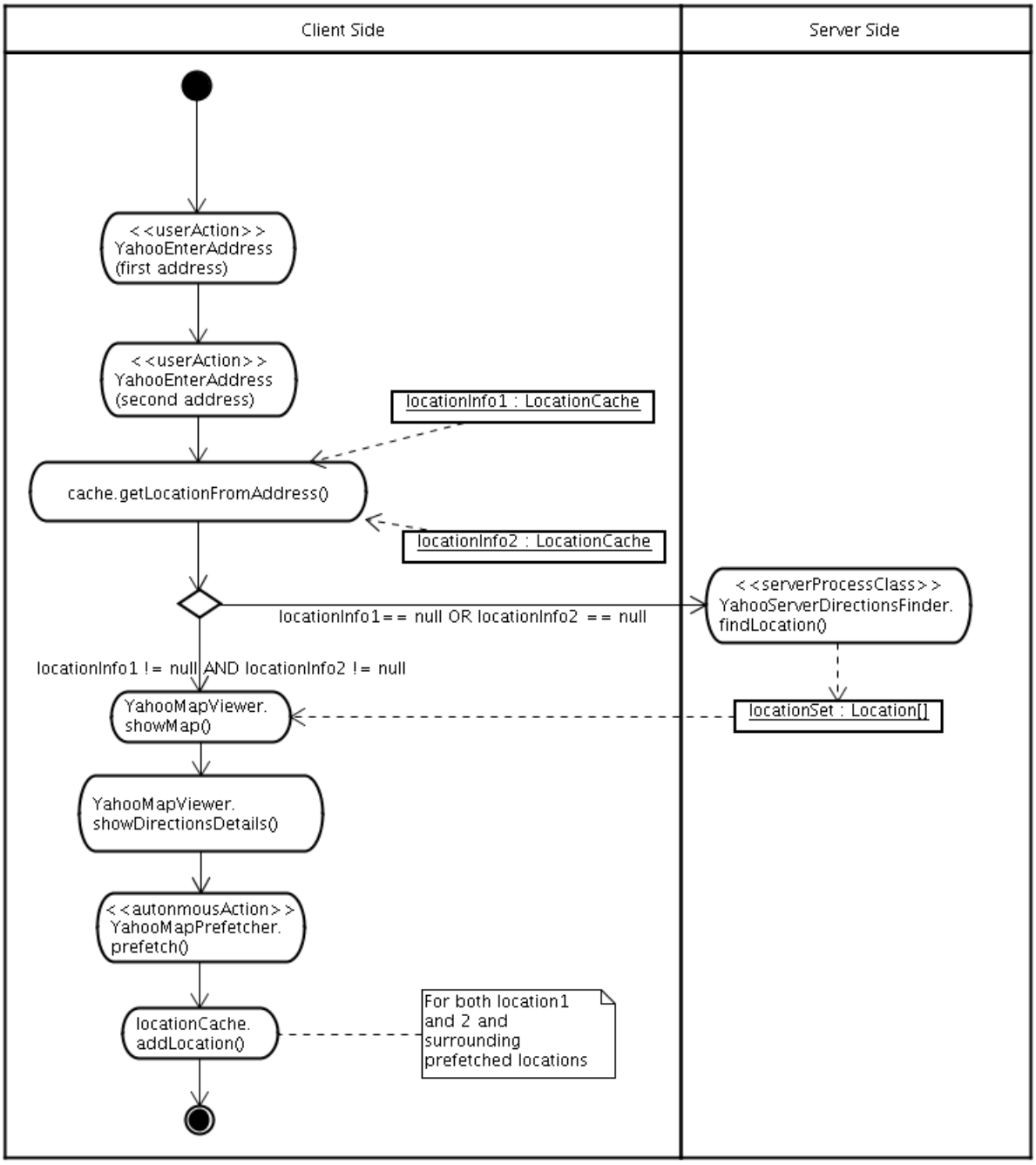

Figura 31 – UWE-R para Yahoo! Maps: modelo de processos

locationInfo1 e locationInfo2 são objetos que trazem informações vetorizadas dos mapas e das localizações permitindo o desenho da trajetória no *YahooMapViewer*. De resto é muito similar ao diagrama respectivo do Google Maps, inclusive no que se refere à pré-carga de mapas no entorno do mapa mostrado (usando a <<autonomousAction>>

YahooMapPrefetcher). O modelo de interação é quase idêntico ao do Google Maps, por isso não será mostrado aqui. Apenas o tipo de dados é diferente, uma vez que KML é um formato usado pelo Google.

Para o modelo de apresentação tem-se o seguinte resultado:

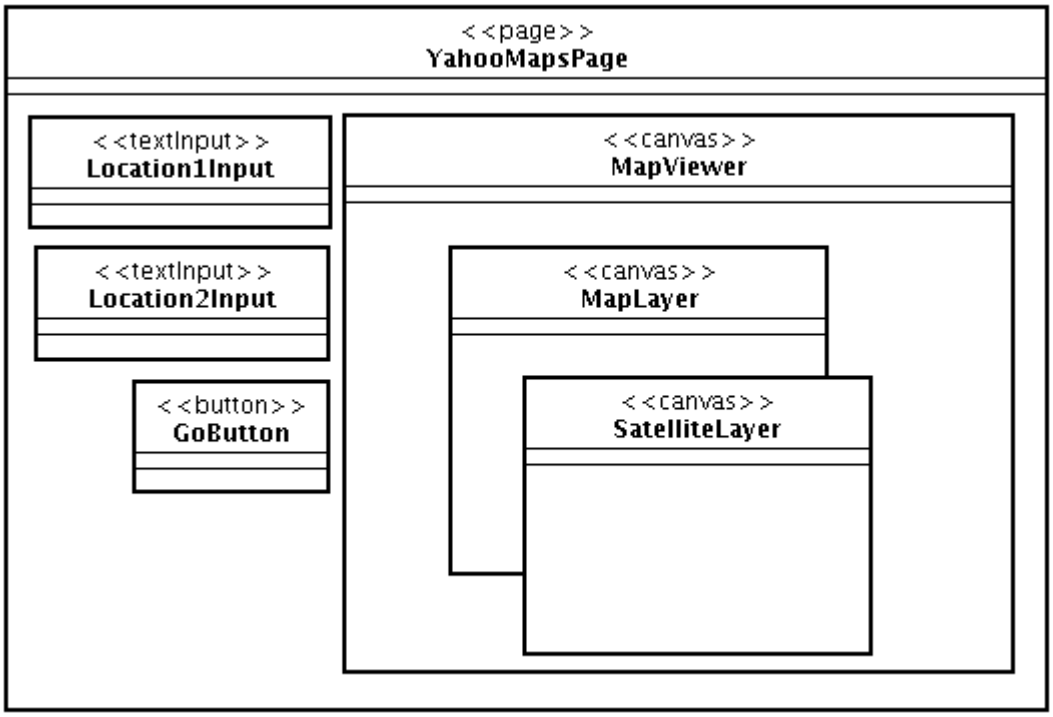

Figura 32 – UWE-R para Yahoo! Maps: modelo de apresentação

Notar que as únicas duas camadas disponíveis são de imagens de satélites ou mapa de limites (países, estados, cidades e ruas). Não há camada de terreno, como no Google Maps.

5 – CONCLUSÕES

Cabe nesta seção avaliar a corroboração ou o afastamento das duas hipóteses levantadas neste trabalho: a adequação de tecnologias RIA para desenvolvimento de SIG-Web e da UWE-R para a sua modelagem. Sugestões de trabalhos futuros também são discutidas.

De acordo com a Figura 1 apresentada na seção 2 podemos concluir que, com diferentes graus de facilidade e estratégias variadas, é possível executar todas as operações básicas escolhidas. Especialmente se o formato *raster* é utilizado. Com formatos vetoriais, há necessidade de um esforço maior para o desenvolvimento de códigos próprios não disponíveis de imediato nos *kits* de desenvolvimento das tecnologias escolhidas. No entanto, o estudo feito em 2.5 mostra que os *sites* estudados desenvolveram esses códigos próprios, pois muitas das interações envolvidas só são possíveis com o uso de um formato vetorial. Exemplo disso é a alteração da trajetória através de um arraste de *mouse* no Yahoo! Maps. A criação desses códigos, no formato de bibliotecas de componentes, é uma sugestão de trabalho de interesse para a comunidade SIG-Web. Um outro trabalho que poderia ser feito é uma análise, do ponto de vista de interação humano-computador, das vantagens de se usar tecnologias RIA em comparação com o modelo da *web* tradicional. Isso serviria para trazer uma maior atenção dos desenvolvedores dessa área para RIA.

Quanto à segunda hipótese, a seção 4 mostra que a UWE-R consegue capturar os principais elementos de uma modelagem fidedigna aos sistemas como foram implementados. Os conceitos de RIA (notadamente continuidade visual, riqueza de elementos gráficos e interação assíncrona) são explicitamente anotados de um modo simples, mas nem por isso menos poderoso e passível de extensões. Os *sites* modelados são referências em SIG-Web e possuem as funcionalidades típicas de sistemas do gênero. A possibilidade de usar qualquer ferramenta de modelagem já disponível, como foi o caso para este trabalho, é outra vantagem relevante para a UWE-R. Os próprios criadores da UWE, de onde a UWE-R deriva, foram

contactados e expressaram a sua aprovação quanto à extensão proposta. A aceitação da comunidade científica na forma das publicações conseguidas, também atestam a validade da UWE-R, como proposta. Trata-se, pois, de uma contribuição relevante, principalmente se considerar-se que poucas são as metodologias que cobrem os aspectos de RIA.

Considera-se ainda ser necessário sua utilização em uma maior quantidade de *sites* e casos de uso para se explorar ainda mais suas possibilidades. Durante o desenvolvimento mesmo desses exemplos, algumas pequenas alterações foram feitas na concepção da UWE-R.

Um outro desdobramento futuro deste trabalho é a criação de uma maior quantidade de metaclasses, em especial para o modelo de apresentação, refletindo ainda mais a riqueza dos elementos de interface com o usuário disponível em RIA. Há relevância também, como trabalho futuro, na formalização do metamodelo da UWE-R, aqui apresentado de modo informal. Por fim, uma sugestão de continuação deste é a incorporação de transformações automáticas como preconizado em MDA. Assim poder-se-ia partir de um modelo em UWE-R para a geração de código em algumas das tecnologias RIA existentes.

REFERÊNCIAS

ADOBE**.** Rich Internet Applications. Disponível em: <http://www.adobe.com/resources/business/rich_internet_apps/>. Acesso em: 28/03/2007.

ADOBE FORUM**.** Flex General Discussion Forum. Disponível em: <http://www.adobe.com/cfusion/webforums/forum/categories.cfm?forumid=60&catid=585>. Acesso em 15/02/2008.

BAILEY, B. P.; KONSTAN, J. A.; CARLIS, J.V. *DEMAIS: Designing Multimedia Applications with Interactive Storyboards.* 9th ACM international conference on Multimedia. Ottawa. ACM Press. 2001., páginas 241-250.

BARESI L.; GARZOTTO, F.; MARITATI, M. . *W2000 as a MOF Metamodel.* 6th World Multiconference on Systemics Cybernetics and Informatics - Web Engineering track. Orlando. 2002. Vol. I.

BURROUGH, P. A.; MCDONNELL, R. A. . *Principles of Geographical Information Systems.* 1a. Edição. Nova Iorque. Oxford University Press. 1998. Caps. 1 e 2.

CARTO.NET. Cartographers on the net. Disponível em <http://www.carto.net/>. Acesso em 04/01/2008.

CERI, S.; FRATERNALI, P.; BONGIO, A. . *Web Modeling Language (WebML): a Modeling Language for Designing Web Sites.* 9th International WWW Conference. Amsterdam. 2000. Páginas 137 - 157. ISSN:1389-1286.

COUCLELIS, H.. *People Manipulate objects (but cultivate fields): beyond the raster-vector debate in GIS,* apud A. U. Frank, I. Campari, e U. Formentini (eds.). Theories and Methods of Spatio-Temporal Reasoning in Geographic Space. Editora Springer Verlag. 1992. Berlin.

CONALLEN, J.. *Building Web Applications with UML.* 2a. Edição. Boston. Addison-Wesley Longman Publishing Co., Inc.. 2002. ISBN:0201730383.

DAVIS, C.; LAENDER, A.. *Multiple Representations in GIS: Materialization through Map*

Generalization, Geometric, and Spatial Analysis Operations. Proceedings of the 7th ACM international symposium on Advances in geographic information systems. Kansas City. 1999. Páginas. 60-65.

DE TROYER, O.; LEUNE, C.. *WSDM: A User Centered Design Method for Web Sites.* Proceedings of the 7th International World Wide Web Conference. 1998. Páginas 85 – 94.

DRIVER, M.; VALDES, R.; PHIFER, G.. *Rich Internet, Applications Are the Next Evolution of the Web.* Technical report, Gartner Group apud Bozzon, A., Comai, S., Fraternali, P., e Carughi, G. T. 2006. "Capturing RIA concepts in a web modeling language.". Proceedings of the 15th International Conference on World Wide Web, Edimburgo. 2005.

FLEX. Adobe Flex 2. Disponível em: <http://www.adobe.com/products/flex/>. Acesso em: 10/10/2007.

FLEX SAMPLE APPLICATIONS. Flex Developer Center. Disponível em <http://www.adobe.com/devnet/flex/?tab:samples=1>. Acesso em: 12/06/2007.

FÓRUM GWT. Drag and Drop post. Disponível em <http://groups.google.com/group/Google-WebToolkit/browse_thread/thread/39a371fb0b215af>. Acesso em 08/02/2008.

GARZOTTO, F.; PAOLINI, P; SCHWABE, D.. *HDM — A Model-Based Approach to Hypertext Application Design.* ACM Transactions on Information Systems. 1993. Vol. 11. Páginas 1-26.

GEABIOS. *GeaBIO – Tiny WMS/WFS Client.* Disponível em: <http://www.geabios.com/>. Acesso em: 12/03/2007.

GWT. *Google Web Toolkit – Building AJAX apps in the Java language*. Disponível em: <http://code.google.com/webtoolkit/>. Acesso em: 13/06/2007.

GÓMEZ, J. ; CACHERO, C.. *OO-H Method: extending UML to model web interfaces*, Idea Group Publishing. 2003. ISBN:1591-40050-3

GOOGLE MAPS. Google Maps. Disponível em: <http://maps.google.com/>. Acesso em: 01/12/2007.

HAUSMANN, J.H.; HECKEL, R.; SAUER, S.. *Towards Dynamic Meta Modeling of UML Extensions: An Extensible Semantics for UML Sequence Diagrams.* Human Centric Computing Languages and Environments. IEEE Computer Society. Itália, 2001. ISBN:0-7695-0474-4.

ISAKOWITZ, T.; BALASUBRAMANIAN, P.. *RMM, A Methodology for Structured Hypermedia Design.* Communications of the ACM. ACM Press. 1995. Vol 38, is. 8. Páginas 33-44.

KLEIN, N.; CARLSON, M.; MCEWANN, G.. *Laszlo in Action*. 1a. Edição. Londres. Manning Publications. 2007.

KOCH, N.; KRAUS, A.. *The Expressive Power of UML-based Engineering*, in Second International Workshop on Web Oriented Software Techonlogy, CYTED, Páginas 105-119.

KOCH, N.; KNAPP, A.; ZHANG, G.; BAUMEISTER, H.. *UML-Base Web Engineering – An approach based on standards.* Capítulo 7 de Web Engineering: Modelling and Implementing Web Applications. Springer Verlag. Londres. 2008. Páginas 143-177. ISBN 978-1-84628-922-4.

KROIβ, C.; KOCH, N.. *The UWE Metamodel and Profile – User Guide and Reference*, Technical Report 0802. Disponível em: <http://www.pst.informatik.uni-muenchen.de/projekte/uwe/download/UWE-Metamodel-Reference.pdf>. Acesso em 05/08/2008.

LASZLO SHOWCASE. Showcase – Sample Applications. Disponível em <http://www.laszlosystems.com/showcase/samples>. Acesso em 08/03/2008.

MACHADO, L. C. ; BERNARDO FILHO, O.; RIBEIRO, João Araújo. *RIA Technologies Comparative Study Applied to GIS*. Seminário de Informática – SEMINFO. Torres. Anais do Seminário de Informática - SEMINFO, 2007.

MACHADO, L. C. ; BERNARDO FILHO, O.; RIBEIRO, João Araújo. *UWER: uma extensão de metodologia em Engenharia Web para Rich Internet Applications*. II Simpósio de Informática e VIII Mostra de Software da PUCRS Campus Uruguaiana, RS. 2008.

Revista Hifen v. 32, n. 62. Páginas 205 a 212, ISSN 1983-6511.

MACHADO, L. C. ; BERNARDO FILHO, O.; RIBEIRO, João Araújo. *Typical Web GIS client side operations using RIA technologies*. III Semana de Geotecnologias em Santarém – SIGES 2008. 2008a.

MACHADO, L. C. ; BERNARDO FILHO, O.; RIBEIRO, João Araújo. *UWE-R: an extension to a Web Engineering methodology for Rich Internet Applications.* WSEAS Journal Transactions on Information Science and Applications. Issue 4, Volume 6. 2009. ISSN: 1709-0832. Disponível em http://www.worldses.org/journals/information/information-2009.htm.

MOF. OMG MetaObject Facility Specification. Disponível em <http://www.omg.org/spec/MOF/2.0/>. Acesso em 15/05/2008.

MUKERJI, J; MILLER, J. MDA Guide Version 1.0.1. 2003. Disponível em: <http://www.omg.org/cgi-bin/apps/doc?omg/03-06-01.pdf> . Acesso em 05/08/2008.

OLIVEIRA**,** M. Cristina Ferreira de; TURINE, M. Augusto Santos; MASIERO, P. Cesar. *A Statechart-Based Model for Hypermedia Applications.* ACM Transactions on Information Systems. 2001. Vol. 19, No. 1. Páginas 28–52.

OLSINA, L.. *Building a Web-based information system applying the hypermedia flexible process modeling strategy.* 1st ACM International Workshop on Hypermedia Development with Hypertext. Pittsburgh. 1998.

OMG. *OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2*. 2007. Disponível em: <http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF/>. Acesso em 19/08/2008.

OPENLASZLO. Open-source platform for Rich Internet Applications. Disponível em: <http://www.openlaszlo.org>. Acesso em: 22/06/2007.

PORTAL ACM. The ACM Portal. Disponível em <http://portal.acm.org>. Acesso em 30/07/2008.

PRECIADO, J.; LINAJE, M.; SANCHÉZ, F.. *Necessity of methodologies to model Rich Internet Applications*. Proceedings of the 2005 Seventh IEEE International Symposium on Web Site Evolution (WSE'05). Budapeste. 2005.

PRECIADO, J.; LINAJE, M.; SANCHÉZ, F.. *Enriching Model-based Web Applications Presentation.* Journal of Web Engineering. 2008. Vol. 7, Número 3. Páginas 239-256.

PRECIADO, J.; LINAJE, M.; SANCHÉZ, F.. *Engineering Rich Internet Application User Interfaces over Legacy Web Models.* IEEE Internet Computing. IEEE Computer Society. 2007. Vol 11 número 6. Páginas 53 a 59.

ROSSI, G.; SCHWABE, D.; LUCENA, C.J.P de; COWAN, D.D.. *An Object-Oriented Model for Designing the Human-Computer Interface of Hypermedia Applications.* Proceedings of the International Workshop on Hypermedia Design (IWHD'95). Springer Verlag. Workshops in Computing Series. 1995. Montpellier, França. Páginas 123-143.

SAUER, S; ENGELS, G.. *Extending UML for Modeling of Multimedia Applications.* IEEE Symposium on Visual Languages. IEEE Computer Society. Tokyo. 1999. Páginas 80-87.

SCHWABE, D.; ROSSI, G.; BARBOSA, S.. *Systematic Hypermedia Design with OOHDM.* 7th ACM International Conference on Hypertext. ACM Press. Washington. 1996. Páginas 116 – 128.

SPECHT, G.; ZOLLER, P.. *HMT: Modeling Temporal Aspects in Hypermedia Applications.* 1st International Conference on Web-Age Information Management. Springer-Verlag. Shanghai. 2000. Páginas 256-270. ISBN:3-540-67627-9.

SVG. Scalable Vector Graphics (SVG) Specification. Disponível em: http://www.w3.org/TR/SVG11/. Acesso em 04/10/2008.

SVG MAP LAB. SVG Map Lab. Disponível em <http://blog.svg-map.com/2007/03/about_establish.html>. Acesso em: 04/01/2008.

YAHOO! MAPS. Yahoo! Maps. Disponível em: <http://maps.yahoo.com/>. Acesso em: 01/12/2007.

WHATWG. Web Hypertext Application Technology Working Group. Disponível em <http://www.whatwg.org/>. Acesso em 07/01/2008.

ZAKAS, N. C.; MCPEAK, J.; FAWCETT, J.. *Professional AJAX.* 2a. Edição. Wiley Publishing, Inc.. Indianápolis. 2007.