7
Conclusão

7.1.
Avaliação

A pesquisa descrita por esta dissertação teve por objetivo suprir o método SHDM com um formalismo para expressar o funcionamento das interfaces presentes nos sites da Web 2.0. Produziu como resultados a **Ontologia de Descrição** e a **Arquitetura SWUI de Interfaces Ricas,** ou seja, uma linguagem de descrição, em alto nível, do funcionamento de interfaces Web ricas e um processador para esta descrição, capaz de gerar o código executável da interface, junto com a respectiva máquina de *runtime* para executar as interfaces geradas.

O ambiente de modelagem e execução das interfaces RIA também introduz um protocolo assíncrono baseado em fila de mensagens como forma de implementar a comunicação entre as camadas de Modelo e Visão. Se a tecnologia Ajax permite a comunicação assíncrona entre cliente e servidor, de tal forma que diferentes componentes da interface possam ser atualizados de forma independente, as interações entre Visão e Modelo, quando mediadas por um sistema de fila de mensagens, tornam possível a atualização de interfaces com os resultados parciais do processamento de uma requisição.

Como ponto de partida da nossa pesquisa, revisamos os conceitos relacionados à matéria de Desenvolvimento Web Dirigido por Modelos, aplicada ao domínio das interfaces de usuário, e discutimos algumas iniciativas recentes da indústria com em foco solucionar, pelo menos em parte, as questões aqui levantadas, e outras, a saber: metadados para descrição de *layout*, composição e funcionamento de interfaces; autoria de aplicações independente de plataforma; suporte oferecido por *software* à construção de interfaces executáveis e interoperabilidade no desenvolvimento baseado em componentes.

Assim, como a abordagem adotada pela Ontologia de Descrição de Interfaces Ricas, proposta por este trabalho, o método **RUX-Model** divide a definição da interface em dois níveis: abstrato e concreto. O método, auxiliado pela ferramenta **RUX-Tool** também pressupõe a geração automática da interface final, com base em

PUC-Rio - Certificação Digital Nº 0721313/CA

transformações aplicadas aos componentes de interface com os quais os modelos são construídos.

As propriedades que caracterizam os componentes de interface da biblioteca RUX-Model variam de acordo com o nível de abstração. Por exemplo, no nível de interface abstrata, a propriedade **connectorid** relaciona os componentes ao modelo hipermídia. Na Ontologia de Descrição de Interfaces Ricas, o mapeamento visão – modelo é realizado pelas propriedades *fromClass e fromAttribute*. No Modelo RUX, a **capacidade** (*Capacity*) de um componente descreve a sua funcionalidade. Ela é definida por um **cabeçalho** (*Header*), compartilhado por todas as instâncias do componente, e um **corpo** (*Body*), específico de cada instância. O cabeçalho de um componente RUX é análogo às propriedades **dependencies**, **jsCode** e **cssCode** dos *DHTMLRichControls* na Ontologia de Descrição de Interfaces Ricas. Já a propriedade **htmlCode** é análoga ao corpo, mas só até certo ponto, visto que no Modelo RUX, o corpo de um componente pode definir valores para atributos de apresentação (tais como tamanho e cor) diferentes para cada instância. [11]

As recomendações do grupo **OpenAjax Alliance** focam no suporte à interoperabilidade das aplicações Web desenvolvidas sob a forma de *mashups*. O **OpenAjax Metadata** define um modelo de dados para a descrição de controles de interface na tecnologia DHTML (a combinação de HTML, CSS, DOM e Javascript). A semântica das classes da Ontologia de *Widgets* Concretos, do método SHDM, poderia ser definida em termos do vocabulário **OpenAjax Metadata**, visto que este foi projetado de forma a permitir a descrição, em baixo nível, de qualquer componente Ajax, e não somente isso, mas também componentes em um *mashup*.

O **OpenAjax Hub** também provê um *framework* para criação de ferramentas para desenvolvimento e composição de *widgets* Ajax, ou seja, o problema que se deseja solucionar é a implementação segura de *mashups*. Diferentemente, a Arquitetura SWUI descrita neste trabalho tem por alvo a modelagem e a construção de aplicações em que a lógica de negócio não é programada no componente cliente da aplicação. No caso dos *mashups*, cujo funcionamento resulta das interações entre seus *widgets*, componentes de diversos provedores são integrados na camada de Visão.

A especificação **XForms** se propõe a atender, pelo menos em parte, a necessidade de uma linguagem de alto nível para especificação de interfaces Web. Visto que os navegadores atuais conseguem oferecer suporte ao **XForms** via *plugins* e espera-se que esta nova tecnologia se torne um padrão integrante da especificação XHTML 2.0, trata-se de uma alternativa vantajosa em relação aos tradicionais formulários HTML. No entanto, o vocabulário da linguagem oferece um conjunto

PUC-Rio - Certificação Digital Nº 0721313/CA

limitado de controles de interface. Além disso, a especificação **XForms** herda o modelo orientado à página do HTML, ou seja, a resposta à submissão de um formulário substitui o formulário corrente ou toda a página que o contém, não sendo possível a atualização de trechos de conteúdo da página, como no modelo Ajax. Assim, **XForms** fornece recursos limitados para a construção de interfaces ricas. [6]

Collage é um modelo de programação declarativa, aplicável à construção de sistemas reativos. Sua maior contribuição é propor a utilização de uma linguagem uniforme, baseada no modelo de dados RDF, durante todo o ciclo de desenvolvimento Web. A mesma linguagem é utilizada tanto para definir o comportamento da aplicação como também para descrever, em qualquer nível de abstração, a interface de usuário. Este refinamento sucessivo da especificação de interfaces é suportado pelo conceito de MVC recursivo, generalizado pelo **Collage**.

A linguagem **Collage** oferece primitivas abstratas para especificar a interação com o usuário. Apesar de estas se basearem na especificação **XForms**, constituem apenas um subconjunto destes últimos. Sua semântica refere-se à funcionalidade dos elementos de interface, enquanto que a aparência com a qual tais primitivas serão renderizadas é determinada pelos chamados *renderkits*, que descrevem a estrutura, o comportamento e a aparência (*look-and-feel*) da interface em uma tecnologia em particular: XHTML, XUL, WML, VoiceXML, ou outra. Os *renderkits* providos pelo ambiente de execução **Collage** dão a aparência *default* às primitivas abstratas, mas é possível que os programadores escrevam seus próprios *renderkits* customizados.

Os **XForms Form Controls**, assim como as primitivas de interação abstrata do modelo **Collage**, definem controles de interface de forma agnóstica quanto a plataforma e dispositivo, o que corresponde à Ontologia de *Widgets* Abstratos, do método SHDM. No entanto, a especificação **XForms** não oferece ao desenvolvedor nenhum mecanismo de mapeamento entre definição abstrata e representação concreta: a renderização dos controles fica a critério do dispositivo de interface. No modelo de programação Collage, esta limitação é resolvida pelo mecanismo de *renderkits*, componentes substituíveis do ambiente de execução das aplicações.

Assim como o modelo de programação **Collage** introduz um ambiente de execução próprio, o escopo deste trabalho incluiu, além das ontologias relativas à descrição de interfaces, uma ferramenta de geração de interfaces e um ambiente para a execução destas interfaces. A característica que mais distingue este ambiente de execução é a forma como ele resolve a atualização da camada de Visão: a comunicação baseada em fila de mensagens com o Modelo. Com isso, o mesmo grau de paralelismo dos *mashups*, que combinam diversas fontes de dados na camada de Visão, pode ser alcançado por aplicações que integram serviços na camada de

PUC-Rio - Certificação Digital Nº 0721313/CA

Modelo. A vantagem inerente desta solução de projeto é permitir o reaproveitamento do componente cliente independente do domínio da aplicação, uma vez que toda lógica no cliente ocupa-se apenas com a apresentação dos dados recuperados do Modelo.

O aspecto mais relevante da abordagem UML para a modelagem de interfaces é a utilização de uma linguagem que já se tornou um padrão de fato no desenvolvimento de software. Haubold et al. [32] propõe estender a linguagem UML com uma notação capaz de modelar os aspectos dos controles de interface baseados na tecnologia Ajax. Os aspectos que definem um *widget* no perfil UML descrito no artigo também são considerados na Ontologia de Descrição de Interfaces Ricas, tema desta dissertação.

Entretanto, apontamos algumas lacunas na solução apresentada por Haubold et al. [32]. Por exemplo, pressupõe-se a existência de uma ferramenta para transformar os modelos de interface descritos em código executável, mas não se discute a sua implementação. Além disso, o aspecto *dojoAttachPoint*, do perfil UML, é dependente da plataforma-alvo da geração de código: o *toolkit* Ajax Dojo. E, finalmente, na solução atual, o controlador no lado cliente precisa ser implementado manualmente.

Em geral, o público alvo dos *toolkits* Ajax são programadores; por isso, em sua maioria, eles oferecem somente novas APIs Javascript e algum açúcar sintático. Diferentemente, a linguagem **Jitsu XML** representa uma abstração direcionada a tornar mais confortável o trabalho dos *designers* – o conceito do *framework* é estender a linguagem XHTML com marcadores capazes de definir controles de interface que informem não somente como devem ser apresentados, mas também como se comportam. Por características de comportamento, nos referimos a *handlers* de eventos, mapeamento com o modelo de dados da aplicação e animações, entre outros. Todas estas questões são endereçadas pelo *framework*. Quanto à eficiência e eficácia das soluções **Jitsu**, o projeto ainda não atingiu maturidade suficiente para que possa determiná-las.

A proposta possui, ainda, a virtude de fornecer uma ferramenta de geração automática de código. Entretanto, visto que os documentos fontes utilizados na compilação aceitam a mesclagem de código XHTML, podemos dizer que o nível de abstração da **Jitsu XML** ainda é dependente da tecnologia.

Existem muitos pontos em comum entre o *framework* **Jitsu** e a Arquitetura SWUI. Além da óbvia comparação entre as plataformas tecnológicas (HTML + CSS + DOM + Javascript + JSON), assim como o **Jitsu**, o presente trabalho também propõe a existência de uma linguagem de especificação, um compilador, uma biblioteca de controles Ajax e uma máquina de execução no lado cliente, escrita em Javascript.

PUC-Rio - Certificação Digital Nº 0721313/CA

7.2. Contribuições

A primeira contribuição desta pesquisa foi para a evolução do método SHDM, ao estender o seu formalismo para a modelagem de interfaces. Atualmente, é difícil conceber uma aplicação Web que não explore técnicas avançadas de programação Javascript para enriquecer a experiência dos usuários. Outras plataformas e linguagens, tais como o Adobe Flex, Microsoft Silverlight e Sun JavaFX, também ganharam popularidade porque conseguem criar na Web uma experiência de interação similar a que está disponível nos aplicativos *desktop*. É tão-somente natural que a aplicação das Ontologias de *Widgets* Abstratos e Concretos tenha se tornado cada vez mais limitada. A Ontologia de Descrição de Interfaces Ricas, proposta neste trabalho, visa atualizar as suas predecessoras, preenchendo uma de suas lacunas de expressividade: a modelagem do comportamento dos elementos de interface.

Entretanto, o principal objetivo e contribuição deste trabalho foi viabilizar a geração de interfaces ricas dirigida por modelos. A comparação com outras abordagens conhecidas para a questão da modelagem e projeto de interfaces revela que a nossa pesquisa não se limitou a propor apenas um novo esquema de metadados para a descrição das interfaces na era da Web 2.0. Preocupamo-nos, principalmente, em oferecer uma solução de *software* com a capacidade de auxiliar a criação de instâncias da ontologia e viabilizar a geração automática de código com base nos modelos construídos. Ao mesmo tempo, por desenvolvermos a implementação de referência da Arquitetura SWUI integrada ao HyperDE, contribuímos para auxiliar, por *software,* a aplicação do método SHDM durante todo o ciclo de autoria de aplicações hipermídia.

Uma característica importante da Ontologia de Descrição de Interfaces Ricas é que ela preserva do trabalho de Moura [2] a estratégia de dividir a modelagem de interfaces em dois níveis de abstração: o primeiro nível, independente de plataforma, e o segundo nível, que mapeia a descrição abstrata para a tecnologia DHTML (HTML + CSS + DOM + Javascript). Esta divisão contribui fundamentalmente para tornar a linguagem extensível. Embora este trabalho não tenha discutido a geração de interfaces Web para plataformas diversas, um dos princípios de projeto da Ontologia de Descrição de Interfaces Ricas foi acomodar o desenvolvimento de compiladores para outras linguagens de representação de interfaces. Analogamente, o presente trabalho, antes de documentar a implementação da Arquitetura SWUI como um módulo do HyperDE, forneceu a sua descrição de forma genérica.

PUC-Rio - Certificação Digital Nº 0721313/CA

Inspirados pelo desafio de modelar o paralelismo dos *mashups*, que combinam diversas fontes de dados e serviços, idealizamos uma arquitetura que implementa no lado servidor o assincronismo que a tecnologia Ajax já havia introduzido na camada cliente. O conceito de Transação torna possível a comunicação assíncrona entre cliente e servidor na Arquitetura SWUI. As transações utilizam filas de mensagens, uma substituição ao tradicional ciclo de requisição/resposta, como um mecanismo para comunicar, à camada de visão, a ocorrência de eventos na camada de modelo. Utilizando as filas das transações, o Modelo é capaz não apenas de retornar o resultado final do processamento de uma requisição, mas também de atualizar a Visão com quaisquer resultados parciais assim que se tornem disponíveis. Isto é especialmente útil para a categoria de aplicações em que é possível tirar partido do paralelismo no processamento de uma requisição do usuário. Este é o caso dos sites que oferecem acesso a diversos serviços Web de forma centralizada, como o site Alibabuy.com, que compara os preços de passagens aéreas praticados por diferentes agências de viagens.

7.3. Críticas e Trabalhos Futuros

Ainda restam alguns aspectos da Arquitetura SWUI que precisam ser mais bem avaliados, por exemplo: o desempenho do Interpretador de Interfaces Ricas, suporte a múltiplos navegadores, usabilidade e segurança.

Outra questão sem resposta é como os serviços de indexação de conteúdo da Internet serão afetados pelo fato de que os *widgets* percebidos pelos usuários na interface são construídos somente em tempo de carregamento da página. Sabe-se, porém, que as interfaces desenhadas pelo HyperDE em sua versão modificada não suportam as funcionalidades de Histórico e Favoritos dos navegadores.

Sugerimos como uma possível otimização para o funcionamento da arquitetura redistribuir a função de desenhar os *widgets* que mapeiam objetos no modelo. Na implementação atual, o Renderizador de Interfaces Ricas desenha apenas os gabaritos (*templates*) para os *widgets*, deixando a construção dos *widgets* em si para o Interpretador de Interfaces Ricas. Esta arquitetura foi concebida para atender a situação em que os objetos navegacionais a serem representados não são conhecidos no momento de renderização da interface. Este é o caso das interfaces de transição, em que os dados a serem exibidos dependem da execução de uma operação no modelo. No entanto, para diminuir a carga de processamento no navegador e o tempo de carregamento da página, os *widgets* que representam nós em contexto ou

PUC-Rio - Certificação Digital Nº 0721313/CA

estruturas de acessos poderiam ser desenhados no servidor e enviados junto com o restante da interface para o cliente.

Durante os testes com os componentes da arquitetura, identificamos também as seguintes necessidades não contempladas na versão atual:

- Modelar a substituição de um elemento da interface corrente com o resultado de uma operação (por exemplo, retornar visualmente a informação de que foi executada a operação de inserir ou remover itens em um carrinho de compras, sem atualizar a página inteira);
- Modelar eventos que não se classifiquem como eventos DOM. Os eventos DOM incluem os eventos `onclick`, `onmouseover` e `onload` utilizados em scripts. Todavia, controles ricos não raro têm a sua própria coleção de eventos customizados. Por exemplo, a biblioteca Yahoo! UI define os eventos **slideEnd** e **slideStart** para o controle **YAHOO.widget.Slider**. Embora o projetista possa registrar *handlers* para estes eventos no código da função Javascript que define o controle, não se pode usar “slideEnd” como valor para a propriedade ***concreteEvent*** na descrição retórica de uma interface;
- Modelar a execução de ações na interface (tais que não sejam decorações) em resposta a eventos de usuário. O código contido na propriedade ***jsCode*** de um *DHTMLRichControl* pode ser utilizado para registrar *listeners* e *handlers* de eventos. No entanto, do ponto de vista conceitual, a Ontologia de Descrição de Interfaces Ricas é capaz de descrever, por exemplo, a execução de um código que realiza um efeito de animação na interface, mas não possui vocabulário para modelar a validação da entrada de usuário quando uma caixa de texto perde o foco;
- Modelar classes de controles ricos. A classe *RichControl* não possui propriedades cujos valores possam ser customizados para cada aplicação ou mesmo para cada instância do controle na aplicação;
- Suporte à personalização. As opções do usuário relativas à apresentação (ordenação de listas ou tabelas, estado de visibilidade de *accordions*, etc...) devem poder ser persistidas no modelo da aplicação;
- Suporte à representação de objetos multimídia (áudio e vídeo);
- Suporte a múltiplos dispositivos, acessibilidade, internacionalização e localização;
- Executar transições de interfaces que especifiquem a inserção de um elemento da interface de destino antes da remoção de qualquer elemento da interface de origem. Atualmente, toda a saída da interface de origem tem de ser completada antes que se inicie a entrada da interface de destino.

PUC-Rio - Certificação Digital Nº 0721313/CA

Os seguintes problemas são dignos de serem sugeridos como temas para futuras pesquisas:

- Estender a Ontologia de Descrição de Interfaces Ricas para dar suporte à geração de código em outras linguagens de representação de interfaces, por exemplo, na plataforma Flex/ ActionScript. Embora a linguagem tenha sido projetada para acomodar futuras extensões, adaptar o compilador e o ambiente de execução da Arquitetura SWUI a mudanças na linguagem é uma tarefa que certamente demandará um esforço considerável;
- Criar implementações da Arquitetura SWUI para outros ambientes de desenvolvimento Web;
- Substituir a camada de execução de interfaces concretas, que utiliza uma solução particular, baseada em XHTML/RDFa/JSON e Javascript, por uma padronizada. Embora não seja um padrão (e justamente pela ausência deles), apontamos o *framework* Jitsu como candidato à plataforma de implementação da camada concreta da Arquitetura SWUI. As razões incluem o fato de ser um projeto *open source* e, principalmente, as similaridades que possui com a solução desenvolvida por essa pesquisa. Jitsu propõe uma sintaxe XML para representação das interfaces concretas (um nível de abstração acima das interfaces reais, executáveis) e uma máquina de execução para as interfaces concretas, escrita em Javascript;
- Substituir a implementação de transações, que atualmente utiliza a técnica de *polling*, por uma baseada em **WebSockets**, uma tecnologia disponível no HTML 5. WebSockets, uma especificação W3C ainda em andamento (*working draft*), habilitam a comunicação bidirecional nos navegadores de última geração[79], vencendo as limitações da natureza assimétrica da comunicação sobre o protocolo HTTP, em que o cliente pode invocar o servidor, mas não o oposto. [46, 47]

[79] A interface WebSockets já está implementada no Google Chrome 4.x, no Safari/WebKit e no Firefox 3.7 [45]

PUC-Rio - Certificação Digital Nº 0721313/CA