2
Aplicações Web Ricas

O termo ***Rich Internet Application*** (RIA) surge em março de 2002 [3], em um documento publicado pela Macromedia, para designar um novo modelo de aplicações para a Internet, que combina as conveniências que a computação baseada na Web oferece aos desenvolvedores com as funcionalidades de interação e diversidade de mídias presentes no ambiente desktop [4]. Algumas das características que diferenciam as aplicações Web ricas das tradicionais são: camada de apresentação executando no cliente, em vez de no servidor; interfaces sofisticadas para representar dados e processos complexos; metáforas de interação semelhantes às disponíveis nos aplicativos desktop; armazenamento e processamento de dados no cliente e, daí, a possibilidade de funcionamento *offline* e redução do uso de banda. [3, 5, 6].

Yu et al. [6] define alguns conceitos-chave para o projeto de RIAs. Um **modelo de interface** é a representação da interface que o usuário percebe e com a qual interage. Os modelos são expressos em alguma **linguagem de interface**, tais como XUL (*XML User Interface Language*), XAML (*eXtensible Application Markup Language*), UIML (*User Interface Markup Language*) [3] e SUL (*Simple User Interface Language*) [7]. Um modelo consiste tipicamente em uma hierarquia de **componentes ou controles de interface** (tais como botões, painéis e menus) e um conjunto de **eventos** (como o clique de um *mouse*). Uma **definição de interface** é a descrição, especificamente, de seus componentes e suas propriedades. Em geral, esta definição é codificada em linguagens declarativas tais como XUL e HTML. A **lógica de interface**, por outro lado, é codificada em uma linguagem de programação, tal como Java ou Javascript, manipula os eventos de interface e se comunica com a lógica de negócios da aplicação.

2.1.
Plataformas RIA

Diversas tecnologias têm sido propostas para suportarem o desenvolvimento e a execução de aplicações Web ricas. Segundo Bozzon et al. [3], as aplicações RIA podem ser classificadas em quatro categorias:

[3] http://uiml.org/

a) **Baseadas em script**: a lógica que executa no cliente é codificada em linguagens tais como Javascript, enquanto que o conteúdo e a aparência das interfaces são representados por uma combinação de HTML e CSS;
b) **Baseadas em *plugin***: os navegadores suportam as funcionalidades de renderização avançada e processamento de eventos via *plugins* que interpretam arquivos de programa ou mídia. Exemplos desta abordagem são as tecnologias ActiveX, Flash, Flex e Laszlo;
c) **Baseadas em navegador**: nesta categoria, os navegadores dão suporte nativo à interação rica. Este é o caso dos navegadores que interpretam linguagens de definição de interfaces tais como XUL;
d) **Tecnologias para desktop baseadas na Web**: nesta categoria, as aplicações são disrtibuídas pela Internet, mas executadas à parte do navegador, como acontece com as tecnologias Java Web Start, .NET Smart Client e Adobe AIR.

A partir de agora, apresentaremos um sumário das principais plataformas que dão suporte ao desenvolvimento e à execução de aplicativos Web ricos.

O termo **DHTML** (*Dynamic HTML*) se refere ao uso da combinação HTML DOM[4] + Javascript + CSS para manipular o conteúdo, a estrutura e o estilo de um documento Web em tempo de carregamento ou interação do usuário com a página. [8, 9] Em 2005, Garrett [10] introduziu o nome **Ajax** (*Asynchronous Javascript and XML*), definindo-o como um modelo de aplicação Web que agrega ao DHTML o uso da interface XMLHttpRequest, do Javascript, para enviar requisições HTTP assíncronas. O principal benefício do emprego do Ajax na comunicação cliente-servidor é não bloquear a interação dos usuários durante o processamento das requisições. [6, 11]

Embora esta abordagem proporcione uma interação contínua, ela possui algumas desvantagens. Por exemplo, a falta de padronização no suporte dos navegadores às APIs Javascript e HTML DOM, o que cria a necessidade de escrever código para garantir a compatibilidade *cross-browser*. Além disso, em geral, a manutenção e a depuração de código Javascript em larga escala constituem um desafio. [6] Todavia, o surgimento de *frameworks* tais como Prototype[5], Dojo Toolkit[6], Google Web Toolkit (GWT)[7] e ASP.NET AJAX[8] visa tornar a programação Ajax mais suave para os desenvolvedores.

[4] DOM (Document Object Model): especificação W3C que padroniza a forma de acessar e modificar os elementos de um documento eletrônico. [8]
[5] http://www.prototypejs.org/
[6] http://www.dojotoolkit.org/
[7] http://code.google.com/webtoolkit/
[8] http://www.asp.net/(S(vovsvx454o5rex452c4ypcy3))/ajax/

A tecnologia **Adobe Flash**[9] (anteriormente conhecida como Macromedia Flash) foi criada com o objetivo de acrescentar animação e interatividade às páginas Web e desempenhou um papel chave na criação do conceito de clientes Web ricos [4]. A plataforma **Adobe Flex**[10], uma evolução do Flash, é um *framework* de código aberto que promete alta produtividade no desenvolvimento e manutenção de aplicativos Web expressivos. As aplicações Flex são escritas em MXML e ActionScript. [12]

MXML é uma linguagem declarativa baseada em XML, utilizada para descrever o *layout* dos componentes da interface e também aspectos não-visuais da aplicação, por exemplo, a ligação de dados (*databinding*) e conexões com serviços Web. A linguagem MXML oferece *tags* que representam componentes avançados, tais como *datagrids*[11], árvores, menus, *accordions*[12] e navegadores com abas, além de efeitos de animação. **ActionScript** é uma linguagem de programação orientada a objetos, que provê funcionalidades não presentes em MXML, tais como controle de fluxo e manipulação de objetos [13], e é utilizada para customizar o comportamento de uma aplicação Flex.

As aplicações escritas em MXML e ActionScript são compiladas em arquivos SWF e renderizadas pelo Adobe Flash Player ou Adobe AIR[13]. A Adobe oferece também uma ferramenta de desenvolvimento, o Adobe Flex Builder, que inclui compiladores, biblioteca de componentes ricos e depuradores.

XUL[14] é uma linguagem de interface para aplicativos multiplataforma, que podem funcionar conectados ou não à Internet. Diferentemente da linguagem HTML, que tem por objetivo a definição de documentos, XUL é orientada ao desenvolvimento de GUIs (*Graphical User Interfaces*); por isso, seu conjunto de *tags* representa conceitos tais como janelas, rótulos e botões, em vez de páginas, níveis de cabeçalho e *links* de hipertexto. [14] Documentos XUL podem ser vinculados a arquivos Javascript para incluir lógica procedural no código declarativo. [15]

Uma aplicação XUL pode ser carregada em um navegador com suporte à linguagem ou pode ser executada pelo sistema operacional da máquina cliente. Este último é o caso dos aplicativos da suíte Mozilla (da qual fazem parte o navegador Firefox e o cliente de correio eletrônico Thunderbird), cujas interfaces são codificadas em XUL e desenhadas pelo motor de *layout* Gecko. [16].

[9] http://www.adobe.com/products/flash/
[10] http://www.adobe.com/products/flex/
[11] *Datagrid*: controle de interface que apresenta uma visão tabular de um conjunto de dados.
[12] *Accordion*: controle de interface no qual as diferentes seções de um documento podem ser expandidas ou contraídas.
[13] http://www.adobe.com/products/air/
[14] https://developer.mozilla.org/En/XUL

Microsoft Silverlight[15] é uma tecnologia de apresentação na Web que dá suporte à criação de interfaces elaboradas, endereçando questões como transmissão de áudio e vídeo com alta qualidade. Assim como o WPF (*Windows Presentation Foundation*)[16], Silverlight se baseia na linguagem **XAML**, uma linguagem declarativa, orientada a objetos, usada para definir controles gráficos, comportamento e animações. [18] XAML é baseada em XML e as suas *tags* mapeiam classes .NET. A lógica para manipulação de eventos é usualmente escrita em C# e o todo o código (declarativo e procedural) é compilado em classes .NET, para ser executado por um *plugin* no navegador da Web ou instalado no desktop [15, 19, 20].

A plataforma **Sun JavaFX**[17] permite a criação de aplicativos com conteúdo expressivo para desktops, navegadores Web, ou quaisquer outros dispositivos que executem a máquina virtual Java. [21] As aplicações JavaFX são escritas em JavaFX Script, uma linguagem baseada na plataforma Java e caracterizada por uma sintaxe declarativa, modelo de ligação de dados, suporte à animação e efeitos visuais. [22]

OpenLaszlo Framework[18], lançado em 2002 sob o nome de Laszlo Presentation Server pela Laszlo System, Inc[19], é uma plataforma para desenvolvimento e distribuição de aplicações Web ricas. [15] As interfaces ricas são descritas por meio da linguagem LZX e daí compiladas em aplicações.OpenLaszlo. O processo de compilação produz não somente o código específico da plataforma alvo selecionada pelo projetista – o código Flash ou DHTML gerado incorpora também o cliente OpenLaszlo, um conjunto de classes e serviços que provê uma camada de abstração para o verdadeiro ambiente de execução (Flash ou DHTML).

LZX é uma linguagem de marcação orientada a objetos, projetada para descrever clientes Web ricos. [15] LZX combina código declarativo com ECMAScript[20]: utilizam-se *tags* XML para descrever a estrutura e Javascript para definir a lógica do cliente. [11] A linguagem permite integração com a linguagem CSS e oferece APIs que implementam animações, ligação de dados, manipulação de eventos e comunicação com o servidor.

OpenXUP[21] é um *framework* de desenvolvimento Web baseado no protocolo XUP (eXtensible User Interface Protocol)[22], que oferece um conjunto de APIs para o

[15] http://www.microsoft.com/SILVERLIGHT/
[16] Subsistema gráfico do .NET Framework 3.0 e dos sistemas operacionais Microsoft a partir do Windows XP Service Pack 2 e Windows Server 2003. [17]
[17] http://javafx.com/
[18] http://www.openlaszlo.org/
[19] http://www.laszlosystems.com/
[20] Linguagem de programação que serviu de base para a criação do Javascript e também do ActionScript. [12]
[21] http://openxup.org/

desenvolvimento rápido de aplicações Web ricas (XUP Server) e um cliente magro (XUP Client), que dispensa o uso de navegadores. [6] O cliente possui basicamente duas funções: gerenciar a interação com o usuário (exibir controles e capturar eventos de interface) e realizar a comunicação com o servidor (enviar mensagens informando a ocorrência de eventos e receber as atualizações a serem desenhadas na interface). O servidor provê o ambiente de execução para as aplicações XUP, que processam os eventos comunicados pelo cliente e retorna as atualizações correspondentes.

Por ocasião da realização desta pesquisa, a implementação do *framework* **OpenXUP** estava baseada na plataforma ASP .NET. Futuramente, o *toolkit* OpenXUP também deverá estar disponível como um *servlet* Java e o cliente, como uma aplicação Java Swing.

No ambiente Web, a existência de inúmeros *frameworks* tanto impõem padrões de arquitetura, quanto poupam o desenvolvedor de codificar as funcionalidades não específicas do domínio da aplicação. Somente na plataforma Java, por exemplo, alguns dos *frameworks* populares são: Hibernate (para persistência de dados)[23], Spring MVC[24] (para aplicações corporativas), Java Server Faces[25] e Google Web Toolkit (estes dois últimos, para apresentação dos dados). No entanto, visto que não existem linguagens para descrição de interfaces Web que sejam padrões de fato, os *frameworks* voltados para a camada de visão apresentam, cada qual, uma solução particular para o problema da modelagem. Ademais, na maioria das situações, a especificação da composição e da ligação de dados (*data binding*) dos *widgets* é bastante dependente da tecnologia, o que dificulta ou impossibilita o reuso dos modelos.

2.2. Projeto de Interfaces RIA Baseado em Modelos

Os modelos utilizados no desenvolvimento de software evoluíram de meros mecanismos de abstração e ferramentas de documentação e comunicação de requisitos para poderosas linguagens de especificação, que, aliadas a técnicas de transformação, podem aliviar muito da carga de trabalho envolvida no projeto e implementação de sistemas. A tarefa de gerar código de infra-estrutura, tediosa e repetitiva, é agora auxiliada por *software*, graças a ferramentas de suporte ao

[22] Protocolo SOAP que implementa a comunicação entre os componentes cliente e servidor da camada de apresentação do modelo MVC. Com XUP, as modificações na interface são enviadas do servidor para o cliente como atualizações incrementais; ao mesmo tempo, os eventos de interface são comunicados do cliente para o servidor de forma assíncrona. [6]
[23] https://www.hibernate.org/
[24] http://www.springsource.org/
[25] https://javaserverfaces.dev.java.net/

desenvolvimento que conhecem a semântica dos modelos e os transformam em código executável.

Na abordagem da **Arquitetura Dirigida por Modelos** (*Model Driven Architecture* – MDA), a funcionalidade de um sistema é especificada através de um Modelo Independente de Plataforma (*Platform Independent Model* – PIM), o qual poderá ser traduzido em um ou mais Modelos Específicos de Plataforma (*Platform Specific Model* – PSM), pela aplicação de um conjunto de Regras de Transformação. As transformações são efetuadas de modo iterativo até culminarem na geração de código para a plataforma-alvo selecionada [23].

De acordo com Preciado et al. [24], as metodologias para o desenvolvimento Web – WebML (*Web Modelling Language*)[26], UWE (*UML-based Web Engineering*)[27], OO-H (*Object-Oriented Hypermedia*)[28] e OOHDM (*Object-Oriented Hypermedia Design Method*)[29] – falham em contemplar a modelagem do que caracteriza os aplicativos RIA, a saber: a interação dos usuários; a composição síncrona ou assíncrona de múltiplas fontes de dados; a representação de objetos multimídia (imagem, áudio e vídeo); a continuidade visual (suavidade na execução de atualizações de interface); internacionalização; localização; acessibilidade; suporte a múltiplos dispositivos e à colaboração em tempo real.

RUX-Model (*Rich User eXperience Model*) é um método dirigido por modelos aplicável ao projeto de interfaces ricas. Embora Preciado et al. [5] descreva a sua integração com WebML, conceitualmente, RUX-Model pode ser utilizado com outras metodologias Web. [23] Sendo uma proposta multidisciplinar, o método divide a especificação de interfaces em três níveis:

a) Interface Abstrata: modelo reutilizável, pois especifica os conceitos de interface comuns a todos às plataformas e dispositivos RIA;
b) Interface Concreta: especifica a apresentação e o comportamento da interface. É um modelo otimizado para uma classe de dispositivos, mas comum a todas as plataformas RIA;
c) Interface Final: descreve a interface de acordo com a plataforma-alvo selecionada (Flex, Laszlo, Ajax + DHTML) e provê a geração de código para a aplicação modelada.

Em qualquer nível de abstração, os modelos são construídos a partir de uma Biblioteca de Componentes de Interface (*Interface Components Library*). Um

[26] http://www.webml.org
[27] http://uwe.pst.ifi.lmu.de/
[28] http://gplsi.dlsi.ua.es/iwad/ooh_project/
[29] http://www.tecweb.inf.puc-rio.br/oohdm

Componente de Interface é definido por seu nome, identificador e um conjunto de propriedades, métodos e eventos. Cada componente pertence somente a um nível de interface (Abstrata, Concreta ou Final). A biblioteca também armazena as regras de transformação entre componentes de níveis adjacentes. [11] O método está representado graficamente na figura a seguir:

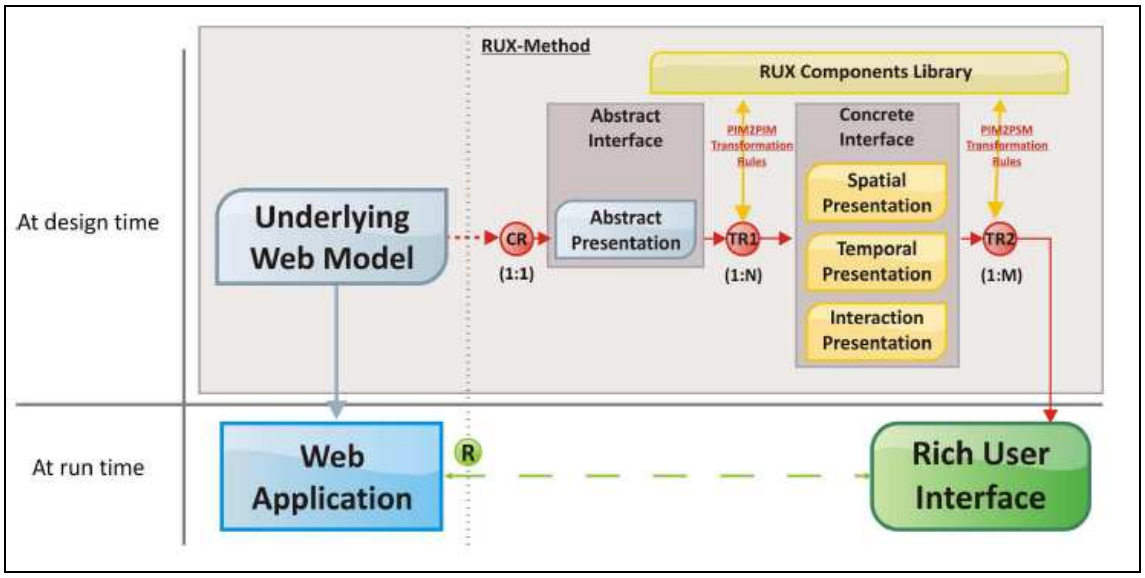

Figura 1 – Método RUX-Model

A Interface Abstrata em RUX-Model é obtida a partir do modelo hipermídia (WebML, OO-H, UWE ou outro) e é definida pelos seguintes conceitos [23]:

a) Conectores (*Connectors*): fazem o mapeamento entre os componentes de interface e o modelo de dados;
b) Mídia (*Media*): um objeto de informação, categorizado como discreto (texto e imagem) ou contínuo (vídeo, áudio e animação);
c) Visões (*Views*): agrupam a informação que será exibida simultaneamente para o cliente. Para este fim, RUX-Model provê quatro tipos de contêineres: visões simples, alternativas, replicadas e hierárquicas.

A Interface Concreta, por sua vez, divide a apresentação em três dimensões adicionais:

a) Apresentação Espacial: responsável por especificar o posicionamento, a dimensão e a aparência (*look-and-feel*) dos componentes de interface;
b) Apresentação Temporal: representa o tipo de comportamento que independe da interação do usuário e que requer uma sincronização no tempo. É útil para modelar animações e chamadas à lógica de negócios baseadas em eventos temporais;
c) Apresentação de Interação: captura e representa os comportamentos disparados pela interação do usuário.

RUX-Model emprega XML Events[30] para capturar eventos. Em RUX-Model, evento é a representação de alguma ocorrência assíncrona associada a um componente de interface concreta. Ação é uma resposta a um evento; *handlers* especificam tais ações e *listeners* são o mecanismo de associação entre *handlers* e eventos. Os *handlers* de eventos em RUX-Model seguem o Modelo Evento-Condição-Ação (ECA). Desta forma, é possível definir regras com ações disparadas quando certas condições são satisfeitas. Este modelo de comportamento é suficiente para expressar tanto as alterações na interface quanto as chamadas à lógica de negócios.

Os componentes de interface concreta podem ser classificados em três categorias: (a) Controle, para entrada e saída de informação; (b) *Layout*, para organização do conteúdo e (c) Navegacional, para percurso da estrutura que organiza a informação. A Biblioteca de Componentes provê um conjunto de componentes RIA nativos, selecionados entre os disponíveis nas plataformas Laszlo, Flex, Flash and XUL. Este conjunto, entretanto, pode ser modificado ou estendido pelo projetista.

A última etapa do método RUX-Model é a tradução da Interface Concreta para a Interface Final, cujo código é obtido automaticamente e depende da plataforma selecionada. A aplicação do método é facilitada por uma ferramenta CASE chamada RUX-Tool[31], que atualmente suporta a geração automática de código para as plataformas Flex, OpenLaszlo e Ajax. A Biblioteca de Componentes e geradores de código da ferramenta RUIX-Tool possuem uma arquitetura baseada em *plug-ins* que permite a inclusão de novos componentes e plataformas-alvo. [11] A RUX-Tool trabalha em conjunto com WebRatio[32], a ferramenta CASE WebML, mas há trabalho em andamento também com as ferramentas CASE UWE e OO-H. [23]

Koch et al. [26] propõe uma abordagem diferente para a modelagem RIA. Em resumo, defende a aplicação de padrões no nível abstrato de especificação e o uso do formalismo de máquinas de estado para a representação dos padrões. Um padrão RIA é definido como uma solução reutilizável para um problema recorrente no projeto de aplicações RIA. Em tese, os padrões RIA podem ser integrados em quase todas as metodologias Web existentes, desde que sejam definidos pontos de extensão na metodologia para permitir a inclusão de referências às funcionalidades RIA. O artigo ilustra a integração dos padrões RIA à metodologia UWE.

[30] Sintaxe padronizada pelo consórcio W3C para as linguagens baseadas em XML; especifica *listeners* e *handlers* de forma integrada ao sistema de eventos do modelo DOM 2. [25]
[31] http://www.ruxproject.org/ruxtool.html
[32] http://www.webratio.com

A partir de agora, discutiremos esquemas de metadados recentemente propostos para a descrição de interfaces RIA e o suporte que as abordagens apresentadas oferecem à construção, por *software*, de interfaces executáveis.

2.3. Metadados para Descrição de Interfaces RIA

2.3.1. OpenAjax

OpenAjax é um esforço do grupo OpenAjax Alliance que tem como objetivo principal promover a interoperabilidade entre componentes Ajax[33] . [27] O advento da tecnologia Ajax traçou o caminho para a popularização dos *mashups*, todavia, questões de interoperabilidade e segurança no desenvolvimento deste tipo de aplicação ainda são preocupações recorrentes a cada novo projeto. O OpenAjax visa propor padrões de indústria que facilitem o desenvolvimento de ferramentas para desenvolvedores, tais como IDEs Ajax e aplicações para montagem de *mashups* (*Mashup Assembly Applications*), capazes de lidar eficazmente com essas questões.

As principais tecnologias/ especificações propostas como parte do OpenAjax são:

- **OpenAjax Hub**: uma biblioteca Javascript que visa atender aos requisitos de interoperabilidade e segurança do ambiente de execução Ajax e implementa as funcionalidades de carga e registro de componentes Ajax e notificação de eventos baseada no paradigma *publish/subscribe*; [27, 28]
- **OpenAjax Registry**: uma autoridade de registro de componentes Ajax, centralizada e de alcance mundial, que visa prevenir colisão de nomes entre componentes de diversos fornecedores;
- **OpenAjax Metadata**: um esquema de metadados para descrição de componentes Ajax. [29]

O desenvolvimento de componentes Ajax em conformidade com estes padrões promete viabilizar a integração sem emendas de tais componentes.

[33] Componentes Ajax são trechos de código HTML/Ajax, que representam desde controles de interface tais como barras de progresso, gráficos e carrosséis de imagem, até mini aplicações combinadas em um *mashup*.

2.3.2. OpenAjax Hub

O grupo OpenAjax Alliance produziu a especificação OpenAjax Hub e também sua implementação de referência. [34] Os principais serviços oferecidos pelo OpenAjax Hub são:

- **Carga e descarga** segura de componentes em um *mashup*, para prevenir injeção de código malicioso;
- **Mediação da comunicação** entre os componentes, por meio de APIs para assinatura e publicação de mensagens. A aplicação como um todo ou seus componentes individualmente podem notificar (publicar) e ser notificados da ocorrência de eventos (assinar);
- Diferentes níveis de **segurança na comunicação** entre componentes. O Unmanaged Hub (Hub Não-Gerenciado) oferece o nível mais fraco de segurança. Nele, os componentes podem comunicar-se livremente entre si e com a aplicação. Esta configuração é adequada quando são utilizados componentes chamados "confiáveis", ou seja, desenvolvidos em conformidade com as políticas de segurança do departamento de TI da empresa. O Managed Hub (Hub Gerenciado) inclui a habilidade de isolar componentes não-confiávies, fornecidos por terceiros, em seguras *sandboxes* ("caixas de areia") e garantir que as informações enviadas por e para cada um destes componentes trafeguem sobre um canal de mensagens seguro e mediado. A Figura 2 ilustra o cenário típico de uma aplicação *mashup*, onde a convivência segura de componentes confiáveis e não-confiáveis precisa ser garantida;
- **Canais privados de comunicação** entre a aplicação e cada um de seus componentes. Isto previne código malicioso de monitorar ou forjar mensagens de outros componentes.

[34] Projeto *open source* hospedado em http://sourceforge.net/projects/openajaxallianc/

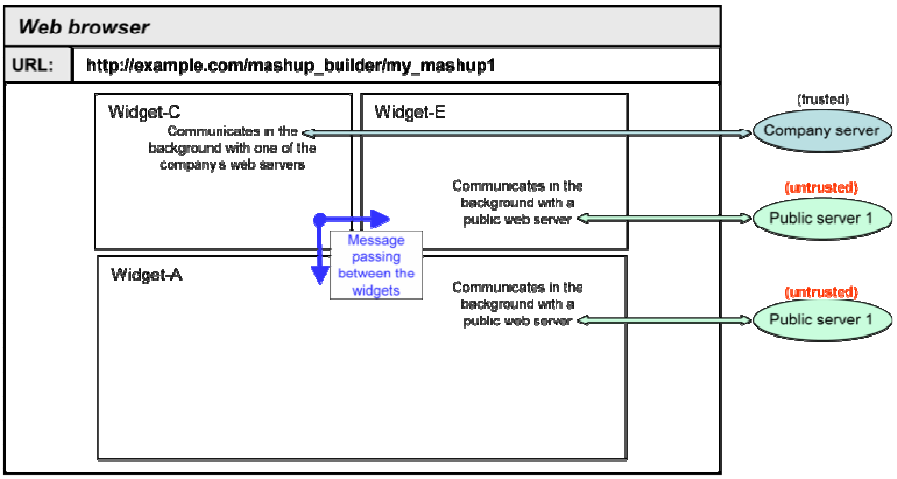

Figura 2 – Componentes confiáveis e não-confiáveis em um *mashup*

2.3.3. OpenAjax Metadata

O vocabulário OpenAjax Metadata define o termo OpenAjax Widget de duas formas diferentes:

- Controles de interface fornecidos por bibliotecas Ajax (barras de menu; controles deslizantes, carrosséis, etc...);
- Componentes em um *mashup* (conhecidos como *widgets* ou *gadgets*)

A descrição de um OpenAjax Widget utiliza uma linguagem de marcação XML. O elemento raiz de um documento de descrição é a *tag* **<widget>**, cujos atributos contêm informação do tipo: nome, identificador único, versão e nome da classe Javascript usada para instanciar o *widget* (quando usado em *mashup*).

Alguns outros elementos usados na linguagem de descrição dos *widgets* são:

- **<javascript>**: especifica o bloco de código Javascript que contém a lógica necessária para o widget ser executado. O código Javascript pode estar contido no corpo do elemento <javascript> ou a sua localização pode estar referenciada no atributo 'src' do elemento;
- **<library>** e **<require>**: referenciam arquivos externos necessários ao funcionamento do *widget*, tais como scripts ou folhas de estilo;
- **<preload>**: elemento opcionalmente aninhado em <library>. Especifica código Javascript que deve ser executado antes do carregamento do recurso referenciado pelo elemento <library>;
- **<postload>**: elemento opcionalmente aninhado em <library>. Especifica código Javascript que deve ser executado após o carregamento do recurso referenciado pelo elemento <library>;

- **<property>**: define quais são as propriedades, ou parâmetros configuráveis, de um *widget*, se tais propriedades devem ser definidas em tempo de projeto ou em tempo de execução, além do tipo da propriedade: Object ou Function.

OpenAjax Widget APIs definem classes e métodos que darão suporte à interação entre *widgets* em um *mashup*. Estes incluem APIs para as seguintes funcionalidades: carregar e descarregar *widgets* (controle de ciclo de vida); recuperar e atribuir valores de propriedades; assinar e publicar mensagens via OpenAjax Hub; recuperar, alterar e receber notificações de mudanças no tamanho dos *widgets*, entre outras.

Qualquer *widget* que vá ser utilizado para compor um *mashup* precisa incluir o atributo **jsClass** no elemento <widget> de sua descrição. O valor do atributo jsClass é o nome da classe Javascript que deverá prover as APIs da especificação OpenAjax.

2.3.4. XForms

XForms é uma recomendação W3C para manipulação de entrada de dados na Web, uma evolução dos tradicionais formulários HTML, com as seguintes vantagens: separação entre conteúdo e apresentação; uso das linguagens XML para definição e transporte de dados e XHTML para exibição dos dados; tipagem de dados; funções lógicas, aritméticas e de data pré-definidas e suporte a funções de usuário; independência de plataforma e de dispositivo de interface. [30]

XForms não é um novo tipo de documento. Ao invés disso, funciona de forma integrada a outras linguagens de marcação, tais como XHTML ou SVG. Um **Processador XForms**, embutido no navegador, é responsável pela submissão dos dados em formato XML.

2.3.4.1. O *Framework* XForms

Usando XForms, um formulário é descrito sob dois aspectos diferentes:

a) **XForms Model**: definição dos dados. Descreve o que o formulário realiza;
b) **XForms User Interface**: definição dos controles de interface que exibem os dados. Descreve como o formulário é apresentado.

2.3.4.2. XForms Model

O elemento **XForms Model** define um *template* para os dados a serem coletados pelo formulário. O exemplo a seguir mostra o que poderia ser um formulário utilizado em uma aplicação de comércio eletrônico, junto com o respectivo código XForms que define o seu modelo de dados. Neste formulário, o usuário é solicitado

informar a forma de pagamento – dinheiro ou cartão de crédito – e também o número e data de expiração do cartão.

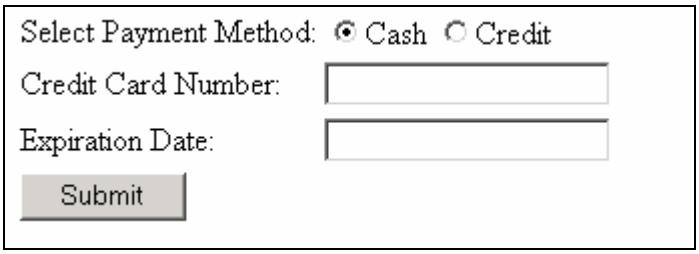

Figura 3 – Exemplo de formulário em uma aplicação de comércio eletrônico

```
      <xforms:model>
        <xforms:instance>
          <ecommerce xmlns="">
            <method/>
            <number/>
            <expiry/>
          </ecommerce>
        </xforms:instance>
        <xforms:submission action="http://example.com/submit" method="post"
id="submit" includenamespaceprefixes=""/>
      </xforms:model>
```

Figura 4 – Modelo de dados XForm correspondente ao formulário

O elemento **xforms:instance** no XForms Model define o documento XML que irá conter os dados coletados pelo formulário. Para o exemplo acima, o documento XML correspondente aos dados inseridos pelo usuário será algo como:

```
      <ecommerce>
            <method>cc</method>
            <number>1235467789012345</number>
            <expiry>2001-08</expiry>
      </ecommerce>
```

Figura 5 – Dados coletados pelo formulário em formato XML

O elemento **xforms:submission**, por sua vez, define um formulário e descreve como os dados serão submetidos. No exemplo acima, o formulário é definido pelo id “submit”, enquanto que os atributos **action** e **method** informam qual o endereço alvo do formulário e o método de envio, respectivamente.

2.3.4.3. XForms User Interface

Na especificação XForms, os elementos de interface são chamados **Form Controls**, um conjunto de controles de entrada (*input controls*) abstratos. Visto que tais controles são definidos de forma agnóstica quanto à plataforma e dispositivo, é deixado a cargo do navegador decidir como estes deverão ser exibidos. Desta forma, os formulários XForms podem ser executados em todos os tipos de dispositivo:

computadores pessoais, telefones celulares, PDAs e dispositivos para portadores de necessidades especiais.

Os Form Controls são declarados com o uso de *tags* especializadas e seus atributos. Abaixo estão listados os principais controles XForms, suas respectivas funções e exemplos de utilização:

- **Label** – rótulo para o controle;
- **Input** – entrada de texto em uma linha;

```
<input ref="name/fname">
 <label>First Name</label>
</input>
```

- **Secret** – entrada de senhas;

```
<secret ref="name/password">
 <label>Password:</label>
</secret>
```

- **Textarea** – entrada de texto em múltiplas linhas;

```
<textarea ref="message">
 <label>Message</label>
</textarea>
```

- **Submit** – submissão do formulário;

```
<submit submission="form1">
 <label>Submit</label>
</submit>
```

- **Trigger** – disparo de ação pelo usuário;

```
<trigger ref="calculate">
 <label>Calculate!</label>
</trigger>
```

- **Output** – exibição de dados, não permitindo a entrada ou a edição;
- **Upload** – envio de arquivos para o servidor;

```
<upload bind="name">
 <label>File to upload:</label>
 <filename bind="file"/>
 <mediatype bind="media"/>
</upload>
```

- **Select1** – seleção de um item em uma lista;

```
<select1 ref="status">
 <label>Status:</label>
 <item> <label>Male</label> <value>M</value> </item>
 <item> <label>Female</label> <value>F</value> </item>
</select1>
```

- **Select** – seleção de múltiplos itens em uma lista;

```
<select ref="languages">
<label>Languages:</label>
<item> <label>English</label> <value>E</value> </item>
<item> <label>French</label> <value>F</value> </item>
<item> <label>Spanish</label> <value>S</value> </item>
```

```
<item> <label>German</label> <value>G</value> </item>
</select>
```

- **Range** – seleção de um valor em uma faixa. No exemplo a seguir, o usuário pode escolher qualquer múltiplo de 5 entre 0 e 100:

```
<range ref="length" start="0" end="100" step="5">
 <label>Length:</label>
</range>
```

Os controles de interface são ligados ao Model pelo mecanismo de *binding* do XForms. Há duas maneiras em que o *binding* pode ser indicado: por meio dos atributos **ref** ou **bind** do respectivo elemento de interface. O atributo **ref** em um controle de interface é a referência XPath de um elemento no modelo, enquanto que o atributo bind é um ponteiro para um elemento do tipo **bind** definido do modelo. Para exemplificar os dois tipos de *binding*, considere o modelo a seguir:

```
<model>
  <instance>
    <person>
      <name>
        <fname/>
        <lname/>
      </name>
    </person>
  </instance>
  <bind nodeset="/person/name/fname" id="firstname"/>
  <bind nodeset="/person/name/lname" id="lastname"/>
</model>
```

Figura 6 – Exemplo de Modelo de dados XForm

O *binding* pode ser declarado por referência direta ao XPath dos elementos do modelo no atributo **ref** dos elementos de entrada, como exemplificado abaixo:

```
<input ref="/person/name/fname"><label>First Name</label></input>
<input ref="/person/name/lname"><label>Last Name</label></input>
```

Figura 7 – *Binding* usando o atributo `ref`

Ou via referência aos elementos **bind firstname** e **lastname**:

```
<input bind="firstname"><label>First Name</label></input>
<input bind="lastname"><label>Last Name</label></input>
```

Figura 8 – Binding usando o atributo `bind`

2.3.4.4. XForms Actions

XForms Actions provêm um mecanismo flexível e uma linguagem declarativa de alto nível para se especificar o processamento de eventos DOM, via XML Events. Alguns dos elementos XForms que representam ações em resposta a eventos são:

- Load: esta ação carrega o endereço alvo de um *link*;
- Message: esta ação encapsula a exibição de uma mensagem para o usuário;

- Send: esta ação inicia a submissão do formulário;
- Setvalue: esta ação atribui um valor a uma determinada instância de informação definida no modelo.

No exemplo a seguir, a mensagem "Input Your First Name" será exibida como um *tool tip* (atributo level=“ephemeral”) quando o campo de entrada receber o foco (atributo event=“DOMFocusIn”)

```
<input ref="fname">
  <label>First Name</label>
  <message level="ephemeral" event="DOMFocusIn">
  Input Your First Name
  </message>
</input>
```

Figura 9 – Exemplo de declaração de uma ação XForm.

2.3.5. IBM Collage Programming Model

Collage é um modelo de programação e ambiente de execução proposto pela IBM para o desenvolvimento e distribuição de aplicações construídas a partir de componentes Web. [31] Hoje em dia, as aplicações fracamente acopladas, distribuídas por diferentes redes e organizações, são cada vez mais comuns. Collage é a resposta da IBM aos novos requisitos deste tipo de desenvolvimento. Suas principais características são: modelo de dados RDF e programação declarativa.

2.3.5.1. Modelo de dados

Collage utiliza os conceitos da linguagem RDF, tais como recurso, tripla, propriedade e classe, para expressar modelos de dados distribuídos. O modelo Collage/RDF vem unificar diversos outros modelos de dados populares, conforme ilustrado na tabela a seguir, que descreve o mapeamento entre os conceitos Collage/RDF e os de seus concorrentes:

Collage/RDF	Entity-Relationship	UML	Relational	XML
class	entity class	class	table	---
resource	entity instance	object	row	element, attribute
value property	attribute	attribute	column	parent-child relationship
value tree	composite attribute	---	---	XML (sub)-tree
non-value property	---	association	PK/FK	---

Tabela 1 – Comparação entre os modelos de dados Collage/RDF, Entidade-Relacionamento, UML, Relacional e XML.

No modelo Collage/RDF, os recursos suportam classificação múltipla e dinâmica. As classificações são armazenadas como triplas com a propriedade *rdf:type*. As

classes, assim como os recursos, são identificados por URIs. O suporte RDF à classificação múltipla torna possível a composição de diferentes artefatos Collage em um único programa, independentemente de tais artefatos classificarem recursos de forma diversa. Além disso, as aplicações Collage podem, com base na classificação, associar comportamento a um recurso.

2.3.5.2. Modelo de execução

O modelo de execução Collage é reativo e centralizado em dados, ou seja, o funcionamento da aplicação é definido em termos de reações a eventos (externos ou internos) e de evolução de estado. O estado de execução de um programa é dado pelo conjunto de triplas RDF contidas em um repositório de triplas. A linguagem declarativa oferece duas classes de construções para atualizar o estado de execução:

- **Bind <c:bind>**: Define como o valor de um recurso (saída) varia em função das alterações em um ou mais outros recursos (entrada). Um conceito-chave para o modelo de execução são as **atualizações** de valor em um recurso. A computação do recurso de saída pode ser disparada pela ocorrência de uma atualização em pelo menos um dos recursos de entrada. As entradas são ativas ou passivas, dependo se as suas atualizações dispararam ou não a execução de um *bind*;
- **Let <c:let>** e **Create <c:create>**: Criam novos recursos e modificam os relacionamentos entre eles (criação e destruição de triplas)

A figura a seguir representa, como elipses, os conceitos do modelo de dados Collage, como retângulos, os conceitos do modelo de execução, e, como arcos, as relações entre eles.

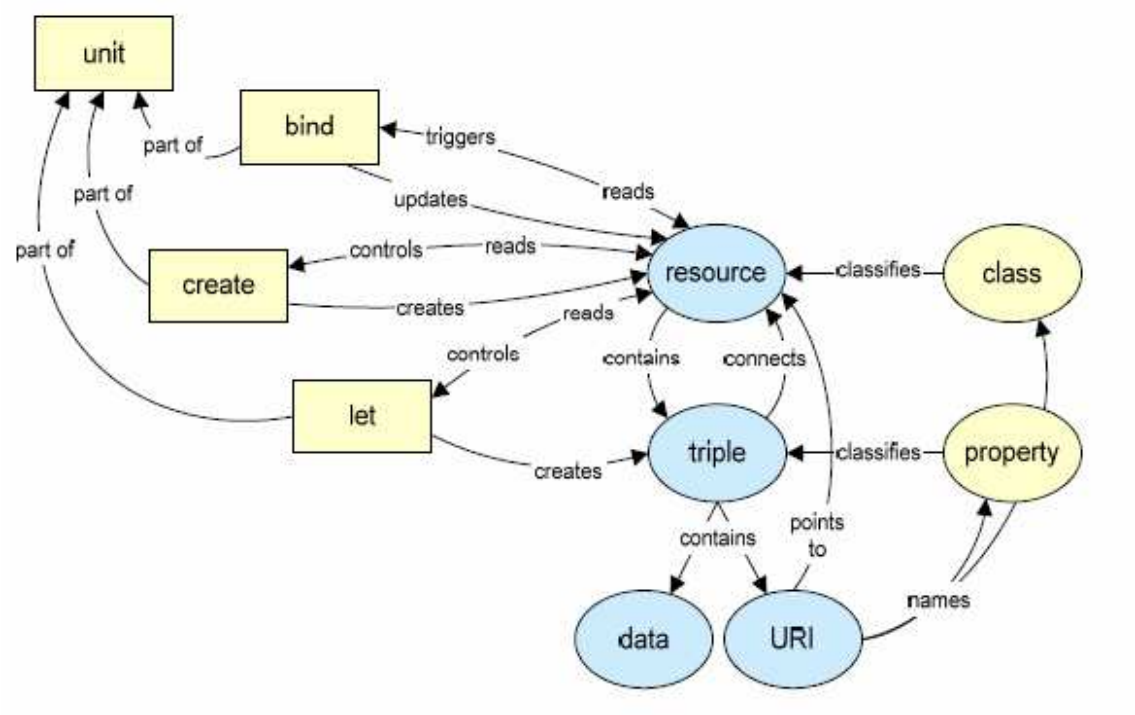

Figura 10 – Modelo de Execução Collage

Um ciclo de execução inicia-se com a ocorrência de uma atualização assíncrona em um recurso. A partir disso:

a) Um conjunto de construções do tipo *bind* é escolhido para ser executado, com base no estado atual do modelo de dados (recursos e triplas existentes);
b) Os *binds* são executados em ordem de dependência;
c) Construções do tipo *let* e *create* são executadas com base nos valores dos recursos atualizados para então criar novos recursos e triplas.

2.3.5.3. Modelo de interação

O Collage suporta o chamado padrão MVC recursivo para descrever interfaces de usuário sob diferentes níveis de abstração. Aqui, cada instância do padrão MVC é chamada de **unidade de interação abstrata**. O modelo (*Model*) é representado por um recurso Collage; a visão (*View*), por um conjunto de recursos associados ao modelo, instanciados declarativamente por meio da construção <create>; e o controlador (*Controller*), por um conjunto de construções <bind> que atualizam o recurso do modelo em resposta a atualizações nos recursos da visão, e vice-versa.

Os recursos que compõem uma visão do modelo, por sua vez, podem servir eles mesmos como modelos para outras visões, suportando desta forma, o MVC recursivo. Além disso, os recursos que compõem a visão, juntamente com o <bind> no papel do controlador, também podem ser vistos como um detalhamento da abstração

representada pelo modelo. De forma inversa, o modelo pode ser encarado como um encapsulamento da funcionalidade provida pela visão. (Figura a seguir)

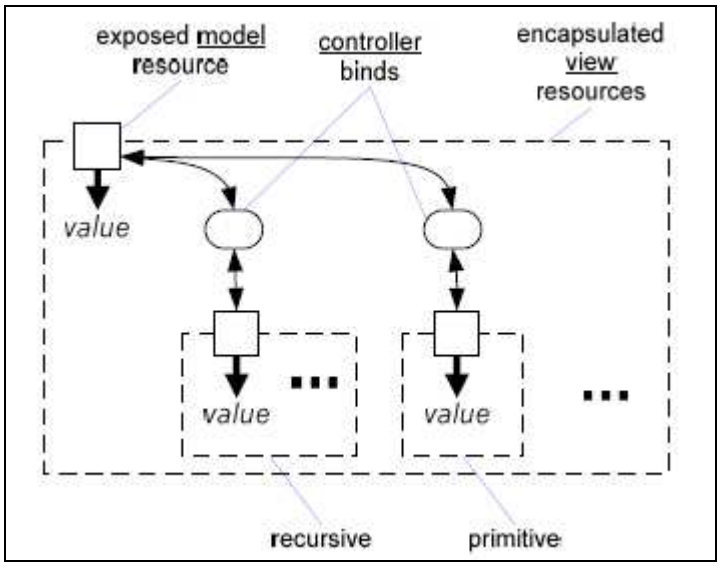

Figura 11 – Unidade de interação abstrata do modelo Collage

As **primitivas de interação abstrata** formam a base para a especificação de interfaces. Estas primitivas são classes de recursos e descrevem tanto a apresentação de uma informação no modelo subjacente quanto a resposta do sistema a uma possível interação do usuário com esta informação.

A primitiva **c:INPUT** é usada, por exemplo, para coletar um simples pedaço de informação tal como um literal RDF. Ela representa um campo de entrada de texto em uma linha e consiste de três recursos:

- O recurso c:INPUT;
- Um recurso dependente conectado ao c:INPUT pelo predicado c:label;
- Um recurso dependente conectado ao c:INPUT pelo predicado c:hint.

A semântica da estrutura dependente de c:INPUT (as triplas em que c:INPUT figura como sujeito) é a seguinte:

- O valor associado ao recurso label será apresentado ao usuário para indicar que tipo de resposta é esperado para esta unidade de interação;
- O valor associado ao recurso hint será apresentado ao usuário como ajuda;
- O valor de c:INPUT sempre representa o valor corrente do campo de entrada conforme visualizado pelo usuário: inicialmente, o valor default do campo, e subseqüentemente, a resposta do usuário para esta unidade de interação.

A figura a seguir exibe o código que declara um campo de entrada de dados utilizando a primitiva c:INPUT, a representação gráfica das triplas RDFs correspondentes e uma possível renderização para o elemento de interface.

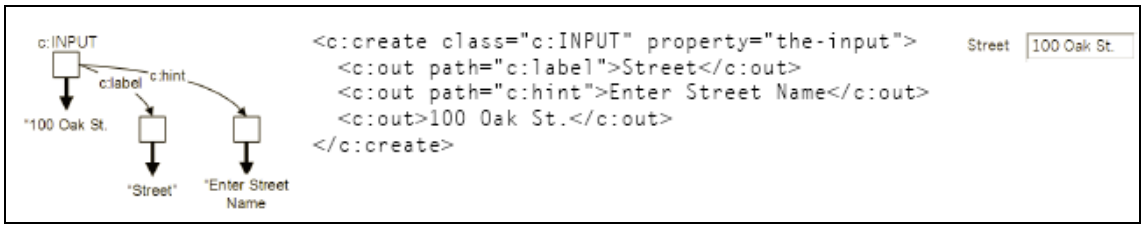

Figura 12 – Exemplo da declaração de um campo de entrada de dados no modelo Collage

As primitivas de interação do modelo Collage encontram correspondência nos Form Controls da especificação XForms, cuja semântica está resumida a seguir:

- c:INPUT - entrada de texto em uma linha;
- c:TEXTAREA – entrada de texto em múltiplas linhas;
- c:SECRET – entrada de senhas;
- c:OUTPUT – exibição de dados;
- c:SELECT1 – seleção de um item em uma lista;
- c:SELECT – seleção de múltiplos itens em uma lista;
- c:TRIGGER - disparo de ação pelo usuário.

2.3.6. Modelagem UML

Haubold et al. [32] descreve uma abordagem para a construção de interfaces Web ricas segundo o paradigma do Desenvolvimento Dirigido por Modelos e utilizando a linguagem UML como ferramenta de modelagem de interfaces.

Como principal contribuição, o artigo propõe um perfil[35] UML para modelar *widgets* e uma arquitetura baseada no *framework* Java Spring e no *toolkit* Javascript Dojo[36] para realizar as aplicações.

Os aspectos da interface que o perfil UML se preocupa em modelar incluem: a composição (aninhamento) de *widgets*; o seu conteúdo (estático ou dinâmico) e as ações executadas em resposta a eventos de interface (por exemplo, eventos de teclado ou *mouse*).

A figura a seguir mostra o perfil UML desenvolvido.

[35] Um perfil é uma extensão à linguagem UML padrão, que define, agrupa e dá semântica a novos elementos de notação [33].
[36] http://www.dojotoolkit.org/

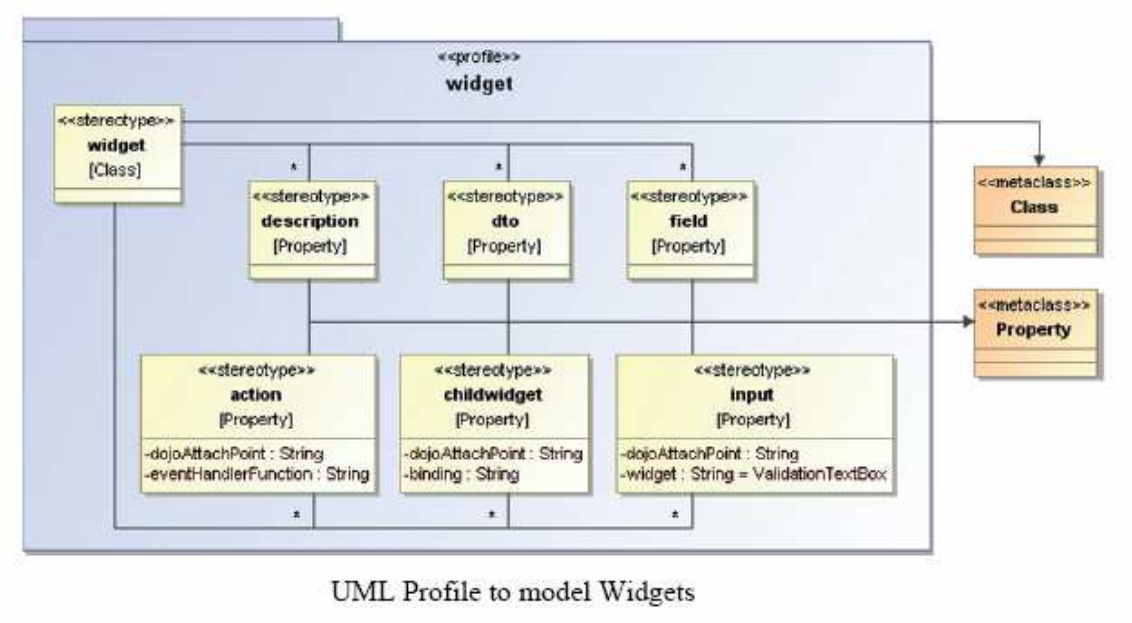

Figura 13 – Perfil UML para modelagem de *Widgets*.

O perfil define o estereótipo *widget* como subclasse da meta-classe UML **Class** e um conjunto de outros estereótipos subclasses da meta-classe **Property**, que representam os aspectos que compõem um *widget*.

A abstração utilizada para fazer o *binding* entre a camada de apresentação e a de persistência são os Objetos de Transferência de Dados (*Data Transfer Objects* – DTO). Os atributos com o estereótipo `dto` representam partes do contexto que o *widget* utiliza para acessar conteúdo dinâmico. Por outro lado, o conteúdo estático de um *widget* é descrito no modelo UML por meio dos atributos com o estereótipo `description`.

Os estereótipos `input` e `field` são utilizados para especificar *widgets* que, respectivamente, exibem conteúdo editável e não editável. O estereótipo `action` denota ações, enquanto que `childwidget` é usado para construir composições de *widgets*.

A aplicação que realiza um modelo desenvolvido com este perfil é concebida sobre a seguinte arquitetura:

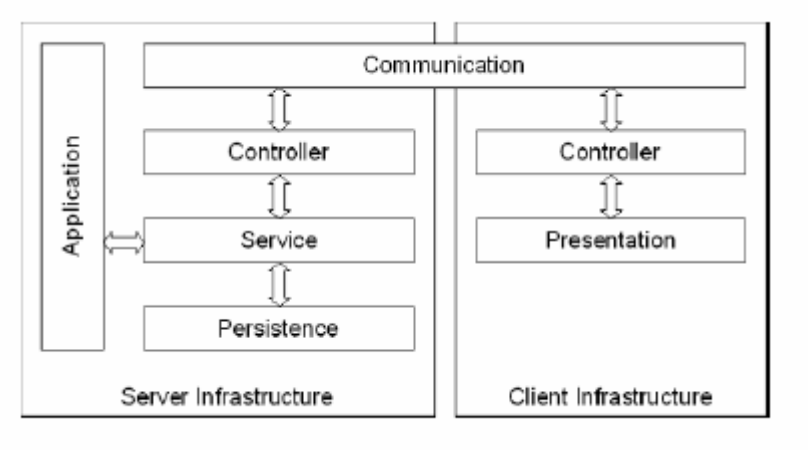

Figura 14 – Arquitetura associada ao perfil UML

Conforme mostrado na figura acima, a arquitetura pressupõe a existência de um controlador no lado cliente para, entre outras funções, tratar os eventos de usuário, recuperar os dados do servidor web e ligá-los aos *widgets*. Na camada de apresentação, propõe-se a utilização do conceito de *widget* definido pela biblioteca Dojo. Um *widget* dojo é descrito sob três aspectos: (a) um objeto Javascript, para definir o seu comportamento; (b) um *template* HTML, para definir o seu conteúdo e (c) um *template* CSS, para definir a sua aparência.

Digno de nota, o artigo cita dois exemplos de abordagens ao Desenvolvimento Dirigido por Modelos: os projetos UWE (UML based Web Engineering)[37] e CATWALK, ambos baseados no *framework* Apache Cocoon[38]. O primeiro provê uma linguagem específica de domínio, baseada em UML, uma metodologia de desenvolvimento orientada a modelos e o suporte de uma ferramenta para a geração (semi-) automática de aplicações Web. O segundo é um *framework* orientado a componentes que interpreta modelos em tempo de execução para gerar aplicações Web adaptáveis e sensíveis ao contexto. [34]. Todavia, os autores do artigo declaram não conhecer trabalhos relacionados que combinem os tópicos Desenvolvimento Dirigido por Modelos e interfaces Web Ajax.

2.3.7. Jitsu

O *framework* **Jitsu** oferece um conjunto de ferramentas para a construção de aplicações Web ricas, incluindo linguagem de marcação XML, compilador, motor de ligação de dados e de motor de animações. [35] Desenvolvido e mantido pela ATTAP Technologies39 desde 2006, Jitsu inova como *framework* Ajax por promover o uso da

[37] http://uwe.pst.ifi.lmu.de/
[38] http://cocoon.apache.org/
[39] http://attap.com/

linguagem XML tanto na especificação dos controles de interface quanto na descrição dos dados para apresentação. Por ocasião da escrita deste trabalho, Jitsu estava sendo oferecido em sua versão 0.1, apenas para propósito de avaliação.

Entre as principais motivações para o desenvolvimento deste *framework* está a necessidade de se reduzir o grau de acoplamento da lógica Javascript com o código HTML e o modelo de dados da aplicação. Os principais requisitos tecnológicos do projeto são: autoria de interfaces utilizando linguagem XHTML estendida com *tags* XML, para expressar funcionalidades mais ricas; ligação de dados no lado do cliente e uso de tecnologias *open source*.

Os principais componentes do *framework* são:

- **Jitsu XML**: uma combinação de código XHTML e elementos XML; descreve tanto o modelo de apresentação quanto o modelo de dados;
- **Compilador Jitsu**: traduz a marcação Jitsu XML em código e objetos Javascript;
- **Biblioteca de controles de interface Jitsu**: controles Ajax, tais como **sliders** e **popups**; controles de formulário (**TextBox**, **Button**, **ComboBox**) dotados de funcionalidades avançadas tais como ligação de dados, animação, validação, etc..;
- **Máquina de Execução Jitsu**: biblioteca Javascript compatível com os navegadores Firefox, Internet Explorer 6 e Safari; implementa a ligação de dados, a manipulação de eventos e animação de controles.

2.3.7.1. Aplicações Jitsu

Uma aplicação Jitsu é qualquer trecho de página delimitado pela tag **<j:App>**. Pode conter texto, elementos XHTML e elementos Jitsu, tais como controles, animações e conjuntos de dados.

Quando uma página é compilada, o resultado da compilação é separado em dois arquivos: todo o código fonte externo às aplicações Jitsu será simplesmente copiado para o arquivo **HTML Gerado**. Este arquivo não conterá nenhuma *tag* Jitsu e todo o conteúdo das aplicações Jitsu será traduzido para uma representação em Javascript, a ser armazenada no arquivo **Javascript Gerado**. A seguir, no navegador do cliente, a máquina de execução Jitsu irá interpretar o Javascript gerado para executar a aplicação.

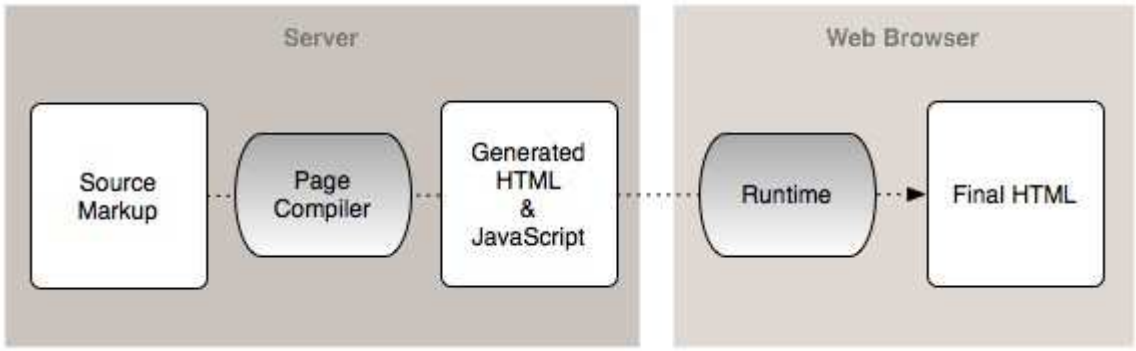

Figura 15 – Geração e execução das aplicações Jitsu

O arquivo HTML resultante da compilação instrui o navegador a carregar também o código da máquina Jitsu e o arquivo Javascript gerado pela compilação. A máquina Jitsu irá então executar este *script* e renderizar a aplicação, o que equivale a construir em tempo de carregamento da página os elementos XHTML que representam o conteúdo da aplicação Jitsu.

2.3.7.2. Ligação de Dados (*Databinding*)

No *framework* Jitsu, cada página contém um ou mais conjuntos de dados, ou *DataSets*, implementados como objetos Javascript e que representam os dados da aplicação. Jitsu suporta dois tipos de *dataSets*:

- **Estáticos**: representam informação somente de leitura, por exemplo uma lista de códigos postais em uma interface de cadastro de endereço. Na listagem a seguir, a definição Jitsu XML de um *dataSet* chamado “folders”:

```
<j:DataSet id="folders">
    <j:di>InBox</j:di>
    <j:di>OutBox</j:di>
    <j:di>Favorites</j:di>
</j:DataSet>
```

- **Ativos**: representam informação que o usuário pode editar. A declaração de *Datasets* ativos tipicamente possui um parâmetro src que indica o ponto de entrada da fonte de dados, como no exemplo abaixo:

```
<j:DataSet id="activefolders" src="http://www.myweb.com/foo/bar" />
```

Para ilustrar o modo Jitsu de ligar dados aos controles de interface, considere a seguinte marcação:

```
<ComboBox items="#bind(//data.folders)"/>
```

A semântica da propriedade `items="#bind(//data.folders)"` é instruir ao ambiente de execução Jitsu a construir os nós `<option>` da *tag* `<select>` com os itens no *dataset* “folders”.

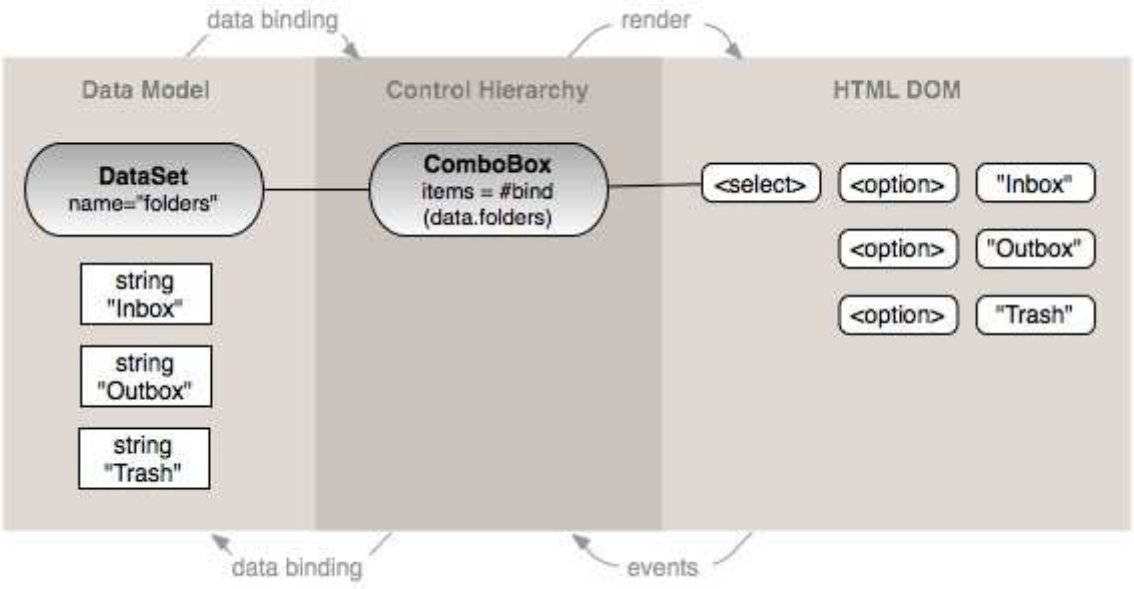

Figura 16 – *Databinding* no *framework* Jitsu

As facilidades de ligação de dados no *framework* Jitsu propagam alterações entre *datasets* e *controles* em ambas as direções. No exemplo acima, se o *dataset* "folders" for alterado, o motor de ligação de dados notificará o controle ComboBox das mudanças que, em resposta, irá gerar código HTML atualizado. No sentido inverso, as interações do usuário com a caixa de seleção serão propagadas de volta ao *dataset*. Se este for do tipo ativo, as alterações serão propagadas de volta ao servidor. Jitsu utiliza chamadas Ajax e o protocolo JSON[40] para a comunicação entre cliente e servidor.

[40] JSON é o acrônimo para *JavaScript Object Notation*.