Acta Scientiarum. Technology
ISSN: 1806-2563
eduem@uem.br
Universidade Estadual de Maringá
Brasil

Trevizan Possebom, Ayslan; Mendes da Silva Filho, Antonio; Pereira da Silva, Sérgio Roberto
Uma abordagem baseada em XML para construção de Interfaces de Usuário para Múltiplos Dispositivos
Acta Scientiarum. Technology, vol. 28, núm. 2, julio-diciembre, 2006, pp. 181-189
Universidade Estadual de Maringá
Maringá, Brasil

Uma abordagem baseada em XML para construção de Interfaces de Usuário para Múltiplos Dispositivos

Ayslan Trevizan Possebom[1*], Antonio Mendes da Silva Filho[2] e Sérgio Roberto Pereira da Silva[1]

[1]*Departamento de Informática, Universidade Estadual de Maringá, Av. Colombo, 5790, 87020-900, Maringá, Paraná, Brasil.* [2]*Centro de Estudos e Sistemas Avançados de Recife, Recife, Pernambuco, Brasil. *Autor para correspondência. e-mail: possebom@din.uem.br*

RESUMO. O desenvolvimento de aplicações para múltiplos dispositivos leva em consideração diferentes características e restrições de cada dispositivo. Linguagens de especificação de interfaces e modelos arquiteturais têm sido utilizados por projetistas de interfaces de usuário. Nesse contexto, este artigo apresenta a linguagem de especificação de interfaces de usuário UIML (UIML2 - *User Interface Markup Language 2.0*) e o modelo arquitetural MIM (*Meta-Interface Model*), que possuem papel importante na construção de interfaces de usuários para múltiplos dispositivos. Adicionalmente, discute-se o desenvolvimento de uma ferramenta de geração de software da interface de usuário a partir da especificação na linguagem UIML.

Palavras-chave: interfaces de usuário para múltiplos dispositivos, linguagens de especificação de interfaces, modelos arquiteturais, UIML, MIM.

ABSTRACT. A XML-based approach for building multi-device user interfaces. Multi-device application development takes into account several characteristics and constraints of each device. Interface specification languages and architectural models have been used by user interface designers. Within this context, this paper presents the user interface specification language UIML (UIML2 - User Interface Markup Language 2.0) and the architectural model MIM (Meta-Interface Model) that have important role in user interface design for multiple devices. Furthermore, this paper discusses the development of a user interface software generation tool from UIML specification.

Key words: multiple device user interfaces, interface specification language, architectural models, UIML, MIM.

Introdução

Atualmente, diversos dispositivos eletrônicos estão sendo projetados e para cada dispositivo existe a necessidade de softwares adequados a fim de fornecer aos usuários as funcionalidades desejadas. Tais dispositivos estão cada vez mais presentes na vida das pessoas e compreendem computadores pessoais, telefones celulares e pequenos computadores portáteis, como os PDAs (*Personal Digital Assistant* – assistente pessoal digital) e *Palmtops*. Apesar de esses dispositivos apresentarem softwares similares como, por exemplo, calculadoras, editores de texto e mensagens, correio eletrônico, agendas, *browsers* e jogos, esses dispositivos possuem características físicas diversas que os diferenciam uns dos outros, como: os mecanismos de entrada (teclado, mouse, canetas, comandos por voz, teclas numéricas e teclas de navegação, tela sensível ao toque, entre outros), a capacidade de armazenamento de dados, a largura de banda para transmissão de dados, a capacidade de processamento computacional e também as diferenças nas interfaces de usuário (resoluções e tamanho de display).

Ao se desenvolver uma aplicação para os computadores pessoais (*Desktops*), esta pode ser re-projetada para uso nos PDAs e nos telefones celulares, onde as interfaces (telas da aplicação) devem ser projetadas. O desenvolvimento de aplicações para múltiplos dispositivos produz um grande número de projetos adicionais para a mesma aplicação, devido às seguintes questões:

- Tempo necessário para o projeto da aplicação;
- Conhecimento necessário para a implementação das aplicações em cada dispositivo;
- Compatibilidade entre as plataformas, e
- Necessidade de projetos distintos de apresentação e navegação nas interfaces de usuário.

Com o objetivo de reduzir essas dificuldades na

especificação da interface de usuário, os projetistas passaram a fazer uso de algumas linguagens de descrição de interfaces para documentar quais informações fazem parte da interface de usuário. Através dessas linguagens de descrição de interfaces é possível mapear um projeto de interfaces em um código que pode ser executado em um dispositivo específico. Exemplos de linguagens de descrição de interfaces de usuário são a UIML (UIML2 - *User Interface Markup Language 2.0*) (Phanouriou, 2000), a XIML (*eXtensible Interface Markup Language*) (Puerta e Eisenstein, 2002) e a XUL (*XML User-Interface Language*) (Goodger *et al.*, 2004).

A UIML utiliza como base a XML (*Extensible Markup Language*) para descrever as interfaces de usuário. Sendo uma linguagem constituída por marcações e armazenada em arquivos de texto simples, pode ser empregada em diferentes plataformas computacionais. Desta forma, ela pode ser convertida para uma linguagem específica como HTML (*HyperText Markup Language*) (W3C, 1999), Java (Gosling *et al.*, 2005) ou mesmo VoiceXML (W3C, 2000). Com o uso da UIML, um projetista pode implementar interfaces para vários dispositivos sem a necessidade de conhecer linguagens específicas de uma plataforma, permitindo reduzir o esforço de desenvolvimento de interfaces para uma família de dispositivos e diminuindo a produção de diversos projetos distintos de interfaces (Harmonia, 2002).

Com a UIML é possível uma separação entre conteúdos e apresentação dos componentes da interface de usuário. Essa separação entre a lógica da aplicação e da apresentação pode ser organizada com o uso de modelos arquiteturais de interfaces, como os modelos MVC (*Model-View-Controler*) (Krasner e Pope, 1988) e MIM (*Meta-Interface Model*) (Phanouriou, 2000). O modelo MIM é utilizado em conjunto com a linguagem UIML e foi criado com o propósito de descrever interfaces que podem ser mapeadas para múltiplos dispositivos. Dentro desse contexto, este artigo discute o uso da linguagem de especificação de interfaces de usuário UIML juntamente com o modelo arquitetural MIM, objetivando a construção de interfaces de usuário para múltiplos dispositivos mediante o desenvolvimento de um protótipo de uma ferramenta de geração de software de interfaces de usuário que efetua o mapeamento da linguagem UIML para códigos Java e HTML.

As sessões seguintes apresentam algumas linguagens de especificação de interfaces de usuário para múltiplos dispositivos, modelos arquiteturais de interfaces de usuários e os passos necessários ao desenvolvimento de uma ferramenta de geração de interfaces de usuário a partir da linguagem UIML.

Linguagens de especificação de interfaces de usuários

As linguagens de especificação de interfaces de usuários são utilizadas para descrever as funcionalidades e comportamentos do sistema. Essas linguagens permitem especificar a interface do usuário por meio de diferentes perspectivas como, por exemplo, as linguagens orientadas à análise de tarefas e comportamentos dos usuários, as linguagens orientadas ao processo de desenvolvimento de softwares e as linguagens de marcação para interfaces de usuário.

No desenvolvimento de interfaces de usuário, as linguagens de marcação possuem um papel de suma importância. Elas oferecem suporte à descrição, por exemplo, da aparência da interface em qualquer dispositivo, utilizando-se dos elementos de interface disponíveis em uma plataforma computacional. As linguagens de marcação de interfaces de usuários mais proeminentes compreendem a XUL (*XML User-Interface Language*), a XIML (*eXtensible Interface Markup Language*) e a UIML (*User Interface Markup Language*).

O principal objetivo para a utilização de linguagens de marcação é manter a consistência das interfaces de usuário entre plataformas distintas, garantindo a portabilidade das interfaces para os diferentes dispositivos.

A linguagem XUL (Goodger *et al.*, 2004) foi projetada com o objetivo de descrever interfaces de usuário em múltiplas plataformas. Ela foi desenvolvida para ser utilizada com o navegador Mozilla e apresenta todas as características disponíveis na XML (*Extensible Markup Language*). Qualquer dispositivo como, por exemplo, *Desktop*, *Palmtops* ou quaisquer outros que possuam navegadores baseados no Mozilla ou Netscape 6 podem utilizar essa notação para a criação de interfaces de usuários. A XUL permite criar os elementos mais utilizados nas interfaces gráficas modernas como, por exemplo, controle de entradas de dados através de caixas de textos e *checkboxes*, barra de ferramentas com botões e outros componentes, barras de menus, árvores hierárquicas ou informações tabulares, dentre outras.

A linguagem XIML (Puerta e Eisenstein, 2002; 2004) também faz uso da XML e tem como um dos principais objetivos a padronização de dados que definem e relacionam todos os elementos de uma interface de usuário. A XIML é uma linguagem de especificação de interface que provê suporte necessário à modelagem, projeto e geração de

interfaces em múltiplas plataformas. Essas interfaces em múltiplas plataformas se baseiam em uma representação comum de dimensões abstratas e uma representação específica de dimensões concretas para cada tipo de plataforma. Por exemplo, um elemento de domínio pode possuir três valores (alto, médio e baixo). Esse elemento se associa a uma apresentação intermediária que indique "seleção única" e automaticamente é correspondida por uma apresentação em *radio buttons* em páginas de internet ou *scrollable list* em telefones celulares.

A linguagem UIML (Phanouriou, 2000) tem sido utilizada em diversas aplicações, pois apresenta uma documentação completa, além de ser uma linguagem de padrão aberto. Um dos objetivos da criação dessa linguagem é o de que ela seja uniforme para a construção de interfaces de usuário, podendo ser empregada no projeto de interfaces de usuário para qualquer dispositivo, qualquer sistema operacional, representar qualquer metáfora de interface e possa também ser mapeada para qualquer linguagem da aplicação como Java, HTML, WML etc. Por ser uma linguagem de marcação baseada em XML, possibilita uma rápida prototipação da interface e é portável entre as plataformas. Adicionalmente, a UIML permite a separação do código da interface do usuário (apresentação) do código lógico da aplicação.

Um documento UIML deve incluir os estilos de apresentação apropriados de cada dispositivo, em que a interface é desenvolvida e executada. O projetista deve criar descrições de interfaces separadas para cada plataforma, utilizando um vocabulário específico. Todos os componentes da interface de usuário são representados através de um conjunto de *Tags*.

A Figura 1 apresenta a estrutura da linguagem UIML. O elemento raiz é o <uiml> e apresenta quatro elementos filhos: <head>, <interface>, <peers> e <template>. O <head> é composto pelo cabeçalho que contém informações de identificação do documento, enquanto que o <interface> identifica todos os componentes da interface de usuário; o <peers> indica qual o vocabulário utilizado no documento. Este vocabulário é composto pelos nomes das classes e eventos apropriados e o <template> define quais partes da interface ou comportamentos podem ser reutilizados por outras interfaces.

O elemento <interface>, por sua vez, tem quatro elementos filhos. O elemento <structure> define todas as partes ou componentes que irão ser exibidos de forma hierárquica na interface. O elemento <style> contém todas as propriedades que definem o estilo de apresentação de uma determinada parte. O elemento <content> define o conteúdo de uma parte e o <behavior> contém todas as ações e eventos que podem ser gerados quando o usuário interage com alguma parte da interface.

```
<?xml version="1.0"?>
<!DOCTYPE uiml PUBLIC "-//Harmonia//DTD UIML 2.0 Draft//EN"
   "UIML2_0g.dtd">
<uiml>
        <head> . . . </head>
        <interface>
                <structure>  <part> <style> </style> </part  </structure>
                <style>  <property> . . . </property> </style>
                <content>  <constant/>  </content>
                <behavior>
                        <rule>
                                <condition> <event/> </condition>
                                <action> . . . </action>
                        </rule>
                </behavior>
        </interface>
        <peers> . . . </peers>
        <template> . . . </template>
</uiml>
```

Figura 1. Estrutura de códigos em UIML.

A Figura 2 apresenta um fragmento de código básico em UIML que define um formulário, dois botões e um rótulo, além dos eventos gerados ao clicar nos botões. Quando o usuário clica em algum dos botões, um evento "*actionPerformed*" é gerado. Assim, as ações decorrentes dessa ocorrência causam a alteração do texto do elemento "Rotulo" de acordo com o botão pressionado. Cada plataforma computacional para a qual o código UIML é gerado requer um vocabulário específico. Esse vocabulário define o conjunto de elementos de interface com suas respectivas propriedades e comportamentos, ou seja, define o conjunto de *tags* que são válidas para representar os elementos de interface em cada plataforma.

```
<?xml version="1.0"?>
<!DOCTYPE uiml PUBLIC "-//Harmonia//DTD UIML 2.0 Draft//EN"
"UIML2_0g.dtd">

<uiml>
  <interface>
    <structure>
      <part id="Janela" class="JFrame"><style>
          <property name="title">Janela simples com componentes eventos</property>
        </style>
        <part id="Rotulo" class="JLabel"><style>
            <property name="text">Pressione um botao</property>
          </style></part>
        <part id="BotaoEsquerdo" class="JButton"><style>
            <property name="text">Esquerdo</property>
          </style></part>
        <part id="BotaoDireito" class="JButton"><style>
            <property name="text">Direito</property>
          </style></part>
      </part>
    </structure>
    <behavior>
      <rule><condition>
          <event class="actionPerformed" part-name="BotaoEsquerdo"/>
        </condition><action>
          <property part-name="Rotulo" name="text">Botao Esquerdo Pressionado</property>
        </action>
      </rule>
      <rule><condition>
          <event class="actionPerformed"  part-name="BotaoDireito"/>
        </condition><action>
          <property part-name="Rotulo" name="text">Botao Direito Pressionado.</property>
        </action>
      </rule>
    </behavior>
  </interface>
  <peers>
    <presentation base="Java_1.3_Harmonia_1.0"/>
  </peers>
</uiml>
```

Figura 2. Exemplo de código em UIML.

Modelos arquiteturais de interfaces de usuários

Os modelos arquiteturais são utilizados pelos projetistas de software para organizar os componentes (elementos computacionais que possuem uma funcionalidade específica) pertencentes a um determinado módulo do sistema, tendo uma visão mais clara e abstrata do sistema como um todo.

Os modelos arquiteturais de interface de usuário são modelos que possuem um foco em sistemas interativos e são divididos, basicamente, em três domínios: um para as definições de entrada e saída, outro para a aplicação e outro para a seqüência de diálogos (Hussey e Carrington, 1996).

Um conjunto dos modelos de interface de usuário mais utilizados pelos projetistas compreende: Seeheim, Arch/Slink, MVC (*Model-View-Control*) e MIM (*Meta-Interface Model*).

O modelo Seeheim (Green, 1985) é bem adaptado para ser utilizado com interfaces de linhas de comando ou formulários. Ele divide a interface em três componentes: (1) *componente de apresentação,* que descreve operações de entrada e saída; (2) *componente de controle de diálogo,* que descreve as ações físicas; (3) *componente de interface com a aplicação,* que descreve como e quando os métodos da aplicação são chamados.

Já o modelo Arch/Slink (Bass *et al.*, 1992) é apropriado para interfaces que utilizam manipulação direta, porém não possue abstração que são adequadas para múltiplos dispositivos. Esse modelo apresenta um conjunto de cinco componentes: *componente de domínio,* que se comunica com a aplicação; *componente de adaptador de domínio,* que é utilizado para manipular os dados exibidos na interface; *componente controlador de diálogo,* que faz a comunicação da aplicação com os objetos de interação; *componente de apresentação,* que fornece objetos como campos de texto e botões; e *componente de toolkit,* que lida com as operações de entrada e saída, além de controle dos eventos.

O modelo MVC (Krasner e Pope, 1988) foi utilizado inicialmente como modelo arquitetural de interfaces em *Smalltalk*. A interface de usuário é modelada com base em três componentes: o componente *model,* que representa a estrutura de dados da aplicação; o componente *view,* encarregado de apresentar os dados na tela através de botões, textos e janelas; e o componente *control,* responsável pela entrada dos dados dos usuários, além de servir como interface entre o *model* e *view*. A implementação de uma interface de usuário utilizando o modelo MVC não é uma tarefa fácil, sendo que o grupo *Model-View-Controler* pode tornar-se uma hierarquia de difícil manutenção e difícil de ser reutilizada. (Silva Filho, 2002).

Por outro lado, o modelo MIM foi desenvolvido com o intuito de descrever interfaces de usuário que podem ser mapeadas para múltiplos dispositivos. Esse modelo divide a interface em três componentes: *lógica*, *apresentação* e *interface*. O componente de lógica fornece uma forma canônica de comunicação entre a interface de usuário e a aplicação, escondendo os detalhes envolvidos nessa comunicação, como os protocolos, transformações de dados, métodos, servidores etc. O componente de apresentação descreve as informações sobre os componentes da interface, suas propriedades e manipulação de eventos, utilizando um vocabulário específico para cada plataforma. O componente de interface faz a comunicação do usuário com a aplicação utilizando-se de um conjunto de partes abstratas (estrutura, estilo e conteúdo), eventos abstratos (como, por exemplo, clicar em um botão) e chamadas de métodos (de acordo com o evento) apropriados (Phanouriou, 2000).

A Figura 3 ilustra o modelo MIM. Nesse modelo, o componente de *interface* é ainda dividido em quatro subcomponentes: estrutura (organização dos elementos), estilo (forma de apresentação), conteúdo (informações que são apresentadas) e comportamento (as interações com a interface). Esse modelo arquitetural mostra-se adequado para ser utilizado em conjunto com a linguagem UIML, garantindo a independência de dispositivo e a portabilidade, porque consegue representar todas as metáforas de uma interface de usuário de um dispositivo.

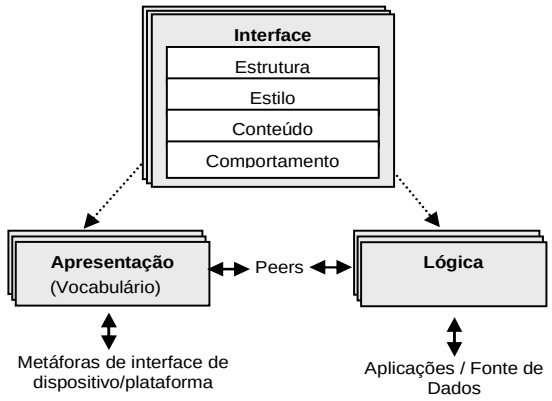

Figura 3. Modelo MIM.

De acordo com a especificação da interface em UIML apresentada na Figura 2, o modelo MIM é capaz de organizar os elementos da interface de usuário por meio do componente de Interface através dos elementos de estrutura, estilo, conteúdo e comportamento, conforme a Figura 4.

Estrutura		
Parte: Janela		Classe: JFrame
	Parte: Rotulo	Classe: JLabel
	Parte: BotaoEsquerdo	Classe: JButton
	Parte: BotaoDireito	Classe: JButton
Estrutura		
Janela		
	Posição X da tela:	100
Rotulo		
	Fonte:	Arial
Conteúdo		
Janela		
	Titulo:	Janela simples com componentes
Rotulo		
	Texto:	Pressione um botão
.....		
Comportamento		
BotãoEsquerdo		
	Classe:	actionPerformed
	Ação:	texto: Botão Esquerdo Pressionado
...		

Figura 4. Componente de Interface do modelo MIM.

Geração do software de interfaces

A UIML foi criada com o objetivo de facilitar o desenvolvimento de interfaces de usuário para múltiplos dispositivos e pode ser utilizada com o modelo arquitetural MIM proposto por Phanouriou (2000). A utilização da linguagem UIML e do modelo MIM neste trabalho se devem ao fato de que ambos auxiliam o processo de geração de software de interface de usuário para múltiplos dispositivos. Ambos são utilizados para desenvolver aplicações que podem ser convertidas para diversas plataformas computacionais e também garantem a portabilidade da aplicação.

A empresa Harmonia (www.harmonia.com) disponibiliza aos clientes uma ferramenta chamada LiquidUI, que faz o mapeamento da linguagem UIML para outras linguagens, como Java, HTML, WML, VoiceXML, entre outras. Porém, o custo para a aquisição dessa ferramenta é considerado elevado. O LiquidUI é capaz de ler códigos UIML e apresentar a interface de usuário em tempo de execução. Fornece também uma classe em Java que pode ser instanciada pela aplicação e, por meio de alguns parâmetros, é capaz de exibir a interface em tempo de execução. Assim, o artigo apresenta o processo de desenvolvimento de um protótipo de software que realize o mapeamento da linguagem UIML para a linguagem Java e HTML, conforme ilustra a Figura 5.

Figura 5. Gerador de software de interface.

O gerador de software de interface foi implementado na linguagem Java e validado mediante a realização de mapeamentos dos exemplos de códigos UIML que acompanham o LiquidUI.

Como a linguagem UIML é baseada em XML, também é necessária a utilização de um *parser* XML para conseguir manipular os dados da UIML. Os *parsers* utilizados são o SAX (*Simple API for XML*) (Saxproject, 2004) e o DOM (*Document Object Model*) (W3C Architecture Domain, 2004):

- O SAX lê um arquivo XML seqüencialmente gerando eventos segundo a instrução lida no momento, por exemplo, início e fim de documento, início e fim de *tags* de elementos, caracteres, entre outros. Esse *parser* é utilizado na leitura do arquivo UIML para obter os dados da interface de usuário.

- O DOM possui algumas características diferentes do SAX. Ele carrega o arquivo XML para a memória, e constrói uma árvore de elementos, na qual é possível navegar diretamente para um elemento específico. Esse *parser* é utilizado na leitura do arquivo de vocabulário, porque sempre que uma instrução de interface é lida, ela precisa ser validada no arquivo de vocabulário. Desta forma, é possível ler o arquivo de interface seqüencialmente e seguir validado conforme o vocabulário.

O processo de geração do software de interface de usuário a partir da especificação em UIML consiste em três etapas: (1) a leitura do arquivo UIML para obter os dados necessários para a criação da interface, (2) a validação de propriedades e (3) a escrita do arquivo final na linguagem destino.

Leitura do arquivo UIML

O processo de leitura do arquivo contendo a especificação em UIML para a obtenção dos dados da interface e a geração do software de interface de usuário é apresentado na Figura 6.

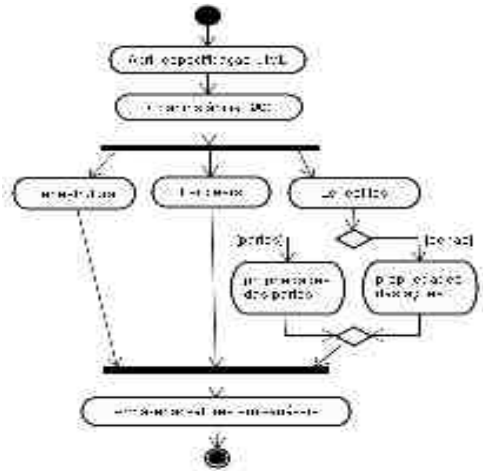

Figura 6. Leitura da especificação UIML.

Inicialmente é feita a identificação do arquivo de interfaces em UIML e criada uma instância do *parser* SAX para realizar a leitura do código da especificação. Nessa leitura, deve-se identificar as *tags* <part>, <style>, <behavior> e <peers>.

As *tags* <part> indicam a existência de um novo elemento de interface. Ao ler esse elemento, deve-se armazenar seus valores em uma estrutura de variáveis semelhante a uma árvore de elementos.

```
Nome do componente[0]: Janela
Tipo do componente[0]: Jframe
Nome do componente pai[0]:(nulo)

Nome do componente[1]: Rotulo
Tipo do componente[1]: JLabel
Nome do componente pai[1]: Janela

...
```

As *tags* <style> indicam o início das propriedades dos componentes. As propriedades são expressas por <property>. Os valores de cada componente são armazenados em outra estrutura de variáveis.

```
Nome do componente[0]: Janela
Nome da Propriedade[0]: title
Valor da propriedade[0]: "Janela simples com
componentes"
Nome do componente[1]: Janela
Nome da Propriedade[1]: bounds
Valor da propriedade[1]: 100,100,400,100
Nome do componente[2]: Rotulo
Nome da Propriedade[2]: text
Valor da propriedade[2]: "Pressione um botão"
...
```

O conteúdo de componentes como tabelas, árvores ou listas são apresentados através de *tags* <constant> e deve ser tratado de forma diferenciada. Tabelas devem apresentar a posição (linha e coluna) das células e seus valores. Árvores mostram o nível da árvore em relação ao elemento principal e seu respectivo valor. As listas indicam uma seqüência de valores.

Os comportamentos da interface indicados pela *tag* <behavior> apontam que um evento será gerado pelo usuário na interface da aplicação. Um <behavior> sempre vem acompanhado de uma seqüência de <rule>, a qual, por sua vez, é constituída por uma <condition> e uma <action>. Essa *tag* pode ser interpretada como sendo um comportamento a que a interface deve se adaptar quando o usuário interage com algum componente. Esses comportamentos são organizados com a criação de regras, sendo que cada regra possui uma condição para que um evento seja gerado (clique do mouse, pressionar uma tecla, etc.) e as ações a serem tomadas caso ocorra o evento (alterar alguma propriedade, chamar funções externas, etc.). No exemplo apresentado, a estrutura que armazenará os comportamentos deve ser formada por:

```
Regra [0]:
        Condição [0]:
                Evento na parte: BotaoEsquerdo
                Evento da classe: actionPerformed
        Regra [0]:
                Nome da parte a ser gerada: Rotulo
                Propriedade a ser alterada: text
                Novo valor: Botão Esquerdo
Pressionado
Regra [1]:
        Condição [1]:
                Evento na parte: BotaoDireito
                Evento da classe: actionPerformed
        Regra [1]:
                Nome da parte a ser gerada: Rotulo
                Propriedade a ser alterada: text
                Novo valor: Botão Direito Pressionado
```

O vocabulário utilizado para a validação dos dados e propriedades é definido na *tag* <peers>. Essa *tag* pode apresentar dois elementos filhos: <presentation> e <logic>. O elemento <presentation> indica um vocabulário já definido para cada linguagem na qual a UIML será mapeada. O elemento <logic> permite definir novos componentes e seus métodos pelos elementos <d-component> e <d-method>. É importante incluir o elemento <presentation> para identificar qual vocabulário será utilizado no mapeamento da especificação.

Validação das Propriedades

Após a conclusão da leitura do documento UIML, o *parser* SAX gera o evento *endDocument*. Nesse evento são feitas todas as manipulações dos dados lidos, validação dos valores no vocabulário e geração do código fonte da aplicação na linguagem destino. A validação das propriedades é apresentada na Figura 7.

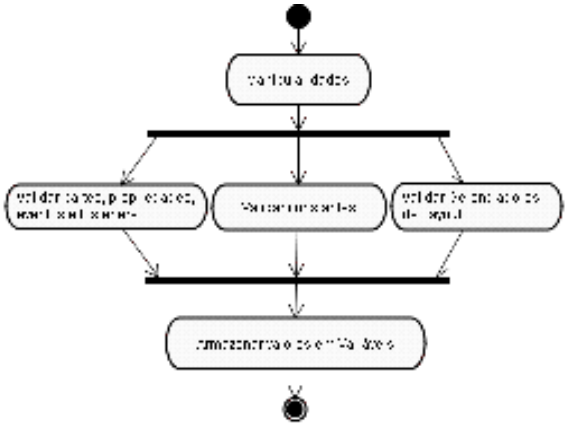

Figura 7. Manipulação dos dados.

Inicialmente é feita a validação de todas as partes, propriedades, eventos e *listeners* do elemento da interface. Essa validação é realizada por meio da criação de uma instância do *parser* DOM e o acesso ao vocabulário definido na *tag* <peers>. Com o uso do vocabulário é possível obter todos os dados necessários para a geração da interface de usuário na linguagem destino, como, por exemplo, os nomes das classes, parâmetros e possíveis eventos para as classes. A Figura 8 ilustra um fragmento do código UIML e seu relacionamento com o a linguagem Java.

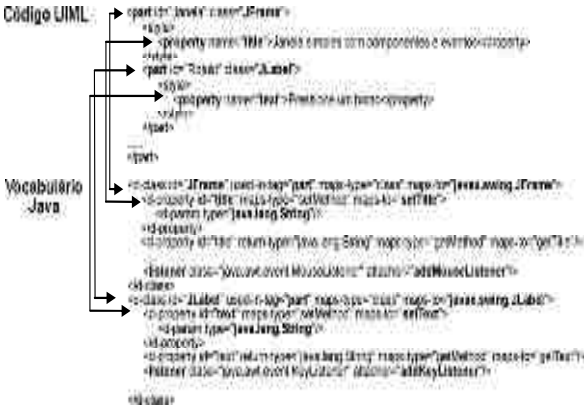

Figura 8. Relacionamento do código UIML com o vocabulário em Java.

Conforme ilustra a Figura 8, a definição de uma classe *JFrame* no código UIML é mapeado para a classe "javax.swing.JFrame" e sua propriedade *title* é mapeada para o método "setTitle", passando um parâmetro do tipo *string*. Da mesma forma ocorre para o elemento JLabel, que é mapeado para a classe "javax.swing.JLabel" e sua propriedade *text* é mapeada para o método "setText". O vocabulário ainda é constituído pelos eventos associados a cada elemento de interface e esses eventos são descritos pela *tag* <listener> no vocabulário.

O vocabulário utilizado para outras linguagens segue a mesma estrutura, contendo todas as partes da interface, suas propriedades e eventos. A Figura 9 demonstra um fragmento de código de interface na linguagem UIML e um fragmento do vocabulário para a linguagem correspondente. O código UIML define a estrutura de uma tabela com as propriedades de alinhamento e borda.

Após a validação das partes, propriedades, eventos e *listeners*, também devem ser validados os elementos que fazem uso de constantes. O uso de constantes é utilizado na definição de elementos de List, JList, Choice e JComboBox na linguagem Java.

Os gerenciadores de *layout* também devem ser tratados de forma cuidadosa. Ao se definir a propriedade *BorderLayout* em alguma das propriedades dos elementos da interface, esses elementos devem ser incluídos em posições específicas na tela como norte, sul, leste, oeste ou centro. O mapeamento da linguagem UIML para a linguagem Java oferece apoio para os gerenciadores: *BorderLayout, FlowLayout, GridLayout* e *GridBagLayout*. Outras linguagens como a HTML não fazem uso de gerenciadores de *layout* para dispor os elementos na interface.

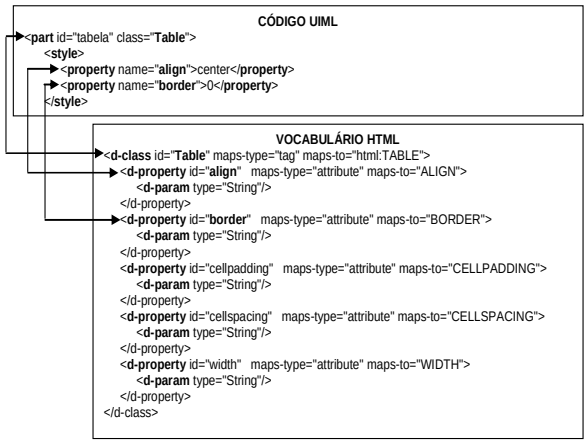

Figura 9. Relacionamento do código UIML com o vocabulário em HTML.

Criação do arquivo da aplicação

Nesse ponto do mapeamento, o arquivo de interfaces em UIML já foi lido e todos os dados já estão em memória, algumas propriedades já foram tratadas e o código já foi validado no vocabulário a procura de classes, métodos, parâmetros e *listeners*. A geração do código fonte da aplicação depende da linguagem destino a ser utilizada. Para a linguagem Java, essa geração é apresentada na Figura 10. A geração da aplicação na linguagem Java pode ser realizada conforme a seqüência de ações:

- Inicialmente o arquivo texto que conterá a codificação da interface é criado, o qual deve possuir a extensão .java.

- Em seguida, todas as bibliotecas dos elementos da interface, suas propriedades e seus eventos devem ser declarados no início do programa. Esses valores são obtidos na validação do código no vocabulário. Bibliotecas adicionais também podem ser necessárias como, por exemplo, bibliotecas para *BoxLayout* ou *DefaultTreeModel*, caso existam propriedades de *layout* do tipo BoxLayout ou elementos de árvores.

- Inicialização da classe com o nome do arquivo, declaração e inicialização dos componentes da interface. A classe se refere ao atributo **class** e o nome da parte se refere ao atributo **id** pertencente a uma <part>.

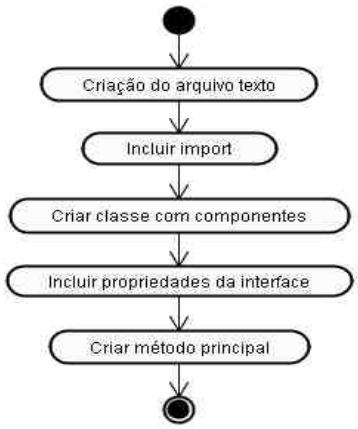

Figura 10. Geração da aplicação Java.

- Inicialização do método da classe. Esse método recebe o mesmo nome dado ao arquivo que recebe o código fonte, com excessão da extensão .java.

- Inclusão das propriedades de cada componente na seqüência hierárquica da interface. O método utilizado é apresentado no vocabulário através do atributo **maps-to** e seu valor foi definido no arquivo UIML.

- Inclusão dos manipuladores de eventos que foram definidos em <behavior> no arquivo UIML. De acordo com a validação do valor do atributo **class** apresentado em <event> obtém-se o método *Listener* e *Event* correspondente.

- Emparelhamento dos componentes, ou seja, adicionar o elemento na estrutura hierárquica da interface.

- Fechamento do corpo do método da classe.

- Inclusão da função principal.

- Inclusão da herança da classe.

- Inclusão de eventos para fechar a janela principal.

- Inclusão do método **show()** para exibir a interface.

- Fechamento do corpo da função principal e da classe.

Através dessas é possível efetuar a leitura da especificação da interface em UIML e gerar o código fonte em Java com o software de interface. O mapeamento do código UIML da Figura 2 para a linguagem Java é apresentado na Figura 11.

A geração do código fonte da aplicação para outras linguagens está diretamente ligada à estrutura da codificação da linguagem destino; portanto, cada linguagem deve apresentar uma seqüência específica para a codificação da interface. No caso da linguagem HTML, as *tags* devem ser incluídas utilizando a ordem hierárquica das partes da interface e cada propriedade se refere a um atributo. Os <behaviors> são definidos normalmente com a seqüência de <condition> e <action>, sendo que o elemento <peers> deve apresentar o código na linguagem JavaScript que deverá ser inserido no código HTML gerado.

```
import java.awt.event.WindowAdapter;                          }
import java.awt.event.WindowEvent;                            |
import javax.swing.JButton;                                   |
import javax.swing.JFrame;                                    |
import javax.swing.JLabel;                                    |
import java.awt.event.FocusListener;                          |
import java.awt.event.MouseListener;                          |
import java.awt.event.MouseMotionListener;                    |
import java.awt.event.KeyListener;                            } 2
import java.awt.Rectangle;                                    |
import java.lang.String;                                      |
import java.awt.Font;                                         |
import java.awt.event.FocusAdapter;                           |
import java.awt.event.MouseAdapter;                           |
import java.awt.event.KeyAdapter;                             |
import java.awt.event.ActionListener;                         |
import java.awt.event.ActionEvent;                            |
import java.awt.Container;                                    }
public class EventButton2{                                    }
  private JFrame Janela = new JFrame( );                      |
  private JLabel Rotulo = new JLabel( );                      } 3
  private JButton BotaoEsquerdo = new JButton( );             |
  private JButton BotaoDireito = new JButton( );              }
  public EventButton2() {                                     } 4
    Janela.setTitle("Janela simples com componentes e         }
                    eventos");                                |
    Janela.setBounds(100,100,400,100);                        } 5
    Rotulo.setText("Pressione um botao");                     |
    Rotulo.setFont( new Font("Arial", Font.BOLD, 14) );       }

    Rotulo.setHorizontalAlignment( JLabel.CENTER );
    Janela.add(Rotulo, "Center");                             } 7
    BotaoEsquerdo.setText("Esquerdo");                        } 5
    BotaoEsquerdo.addActionListener( new ActionListener(){    }
      public void actionPerformed(ActionEvent ev) {           }
          Rotulo.setText("Botao Esquerdo Pressionado.");      } 6
      }                                                       |
    });                                                       }
    Janela.add(BotaoEsquerdo, "West");                        } 7
    BotaoDireito.setText("Direito");                          } 5
    BotaoDireito.addActionListener( new ActionListener(){     }
      public void actionPerformed(ActionEvent ev) {           }
          Rotulo.setText("Botao Direito Pressionado.");       } 6
      }                                                       |
    });                                                       }
    Janela.add(BotaoDireito, "East");                         } 7
  }                                                           } 8
  public static void main (String args[]) {                   } 9
    EventButton2 app = new EventButton2();                    } 10
    app.Janela.addWindowListener(                             }
      new WindowAdapter(){                                    |
          public void windowClosing(WindowEvent e) {          } 11
          System.exit(0); }                                   |
      }                                                       |
    );                                                        }
    app.Janela.show();                                        } 12
  }                                                           }
                                                              } 13
```

Figura 11. Mapeamento do código UIML para Java.

Conclusão

O trabalho apresentou uma breve descrição sobre a linguagem UIML (UIML2) e o modelo MIM utilizado no desenvolvimento de interfaces para múltiplos dispositivos. A linguagem UIML possui uma estrutura que se torna capaz de organizar a interface de usuário através de quatro componentes: estrutura, estilo, conteúdo e comportamento. Utilizando a UIML juntamente com o modelo MIM é possível criar especificações de interfaces de

usuários independentes de dispositivos e que podem ser mapeadas para múltiplas plataformas.

Apresentaram-se também os passos para o desenvolvimento de um gerador de software de interfaces, em que uma especificação UIML é lida e mapeada de acordo com o vocabulário específico para a linguagem Java. Com o uso de um gerador de software de interfaces, os projetistas de sistemas interativos podem criar as especificações da interface na linguagem UIML e efetuar o mapeamento da especificação para códigos Java. Desta forma, os programadores poderiam modificar os elementos de interface de usuários e eventos diretamente no código fonte da aplicação em Java, sem a necessidade de recorrer à especificação UIML ou utilização de instâncias de classes externas para a exibição da interface de usuário em tempo de execução. No desenvolvimento desse gerador de software de interfaces não foram considerados a utilização de partes, estilos, conteúdos ou comportamentos reutilizáveis e chamadas de métodos externos. Códigos reutilizáveis são definidos no elemento <template> e as chamadas a métodos externos são definidas no elemento <peers> em UIML.

O mapeamento da especificação de interfaces em UIML para códigos Java se torna possível através do uso do vocabulário apropriado. Esse vocabulário é formado por metáforas de interfaces que representam elementos de interação utilizando Java AWT e Java Swing. Por meio da seqüência descrita para a obtenção dos dados do código UIML e os passos para a validação no vocabulário, pode-se mapear os códigos UIML para outras linguagens como HTML, WML ou mesmo VoiceXML desde que utilize o vocabulário apropriado para cada linguagem.

Para facilitar o projeto de interfaces, uma ferramenta gráfica de geração de interfaces de usuários onde o projetista possa desenhar a interface através de "*drag and drop*" pode ser desenvolvida. A criação da especificação em UIML através de textos e marcações pode gerar problemas como erros em nomes de variáveis, nome de propriedades e valores possíveis para cada propriedade.

Referências

GOODGER, B. *et al. XML User-Interface Language (XUL) 1.0*. Disponível em: <*http://www.mozilla.org/projects/xul/xul.html*>. Acesso em: 30 jun. 2004.

GOSLING, J. *et al. The Java™ language specificatoin*. 3. ed. Editora: Addison-Wesley, 2005.

GREEN, M. *Report on dialogue specification tools*. *In:* GÜNTHER E.P. (Ed.). *User interface management systems.* New York: Springer-Verlag, 1985. p. 9-20.

HARMONIA. *User interface markup language (UIML) specification*. Language Version 3.0, 2002. Disponível em: <*http://www.uiml.org/specs/uiml3/DraftSpec.htm*>. Acesso em: 23 maio 2004.

HUSSEY, A.; CARRINGTON, D. *Comparing two user-interfaces architectures: MVC and PAC*. FAHCI'96, Springer Verlag, 1996. Disponível em: <*http://www.itee.uq.edu.au/~sse/SVRC-TRS/tr95-33.pdf*> Acessado em: 27 jul. 2004.

KRASNER, G.E.; POPE, S.T. A Cookbook for using the model view controller user interface paradigm in smalltalk-80. *J. Object Orient. Program.*, Framingham, v. 1, n. 3, p. 26-49, 1988.

PHANOURIOU, C. *UIML: A Device-independent user interface markup language*. 2000. Tese (Doutorado)–Virginia Polytechnic Institute, Blacksburg, Virginia, 2000.

PUERTA A.; EISENSTEIN, J. XIML: A common representation for interaction data. *In:* INTERNATIONAL CONFERENCE ON INTELLIGENT USER INTERFACES. 7., 2002. Santa Fe. *Proceedings…* New York: ACM Press pages. 2002. p. 69-76.

PUERTA, A.; EISENSTEIN, J. *XIML: A universal language for user interfaces*. Disponível em: <*http://www.ximl.org/Docs.asp*> Acessado em: 30 jun. 2004.

SAXPROJECT. *SAX: Simple API for XML*. Disponível em: <*http://www.saxproject.org/*> Acessado em: 16 dez. 2004.

SILVA FILHO, A.M. *Arquitetura de software: desenvolvimento orientado para arquitetura.* 1. ed. Rio de Janeiro: Ed. Campus, 2002. Cap. 7, p. 127-144.

BASS, L.; FANEUF, R. *et al.* A metamodel for the runtime architecture of an interactive system. *ACM SIGCHI Bulletin*, New York, v. 24, n. 1, p. 32-37, 1992.

W3C. *HTML 4.01 Specification*. W3C recommendation 24/12/1999. Disponível em: <*http://www.w3.org/TR/REC-html40/*>. Acessado em: 20 abr. 2005.

W3C. *HTML 4.01 Voice extensible markup language (VoiceXML)*. W3C Note 05/05/2000. Disponível em: <*http://www.w3.org/TR/2000/NOTE-voicexml-20000505/*>. Acessado em: 20 abr. 2005.

W3C Architecture domain. *Document object model (DOM)*. Disponível em: <*http://www.w3.org/DOM/*>. Acesso em: 16 dec. 2004.

Received on November 01, 2005.
Accepted on September 12, 2006.