3
Abordagens para Especificações de Interfaces

Se o segundo capítulo evidencia a importância de se desenvolver interfaces isoladas do desenvolvimento da aplicação, este se ocupará em estabelecer uma forma precisa de especificar interfaces separadamente da lógica de programação, representando apenas os aspectos relacionados à troca de informação entre o usuário e a aplicação, ignorando detalhes relacionados à implementação. Desta maneira, a interface definida torna-se independente da linguagem e do ambiente de programação.

A abordagem mais comum para este problema é ter uma descrição da interface, em um nível bem abstrato, e um interpretador ou máquina virtual que, a partir do mesmo arquivo, interprete essa descrição gerando interfaces para diversos dispositivos como PDA's, computadores portáteis, PC's, celulares, dentre outros. Este capítulo discute algumas abordagens apresentadas na literatura e que se referem às mesmas linhas expostas nesta proposta, ilustrando assim a gama de abstrações possíveis para o tratamento desta questão. Algumas dessas abordagens tratam mais das definições de geração de interface abstrata, semelhantes às definições proposta pela SHDM. Normalmente, essas definições são descritas em um nível menor de abstração (i.e., mais próximos do ambiente final de execução), correspondendo assim à ontologia de *widgets* concretos proposta nesta dissertação.

3.1.
Abordagens associadas a métodos de projeto

Aqui serão apresentadas algumas abordagens baseadas em modelos para a criação da interface de usuário. Das abordagens existentes, pode-se mencionar: Tereza XML [21] e XIML [19], onde cada uma possui uma linguagem específica de marcação com base em XML para definir a interface.

O modo como o método WebML realiza a modelagem de uma interface de usuário, descrito no capítulo 2, será brevemente comentado a seguir, no tópico 3.1.3.

3.1.1. Teresa XML

Teresa XML [21] é uma ferramenta que fornece um ambiente completo, semi-automático, para modelagem e geração de interface para aplicações, com base na linguagem XML. Essa ferramenta considera três níveis de abstração para a geração de suas interfaces: o modelo de tarefa, o modelo de interface abstrato e o modelo de interface concreto; para cada um desses níveis uma linguagem específica é definida e utilizada. Uma de suas desvantagens é que o nível menos abstrato, o que trata da interface concreta, é dependente de plataforma. Assim, existem diferentes variantes que devem ser consideradas para cada plataforma, no desenvolvimento da interface.

3.1.1.1. Modelo de Tarefa

O modelo de tarefa do Teresa XML especifica todas as tarefas que o usuário poderá realizar na aplicação e, a partir daí, o sistema realiza uma análise das relações de tempo dessas tarefas. Dessa forma são identificados os conjuntos de tarefas que são habilitados em um mesmo período de tempo (ETS - *enabled task sets*), de acordo com as restrições indicadas no modelo. Assim, as técnicas de interação que apóiam as tarefas, que por sua vez pertencem ao mesmo conjunto ETS, são logicamente classificadas para fazer parte da mesma apresentação. Algumas dessas restrições indicadas no modelo de tarefa podem estar relacionadas também às plataformas nas quais serão executadas tarefas específicas.

Uma vez que os ETS tenham sido definidos, é necessário especificar algumas regras para reduzir o número de apresentações, ou seja, deve-se juntar dois ou mais ETS em uma mesma apresentação, definindo assim conjuntos de apresentação (PSs - *Presentation Sets*). Com os PSs definidos é possível iniciar a construção da interface abstrata.

3.1.1.2. Modelo de Interface Abstrata

O conjunto de PSs obtido é a entrada inicial para a construção da interface abstrata. Ela será composta basicamente de elementos do tipo *iteractor* (objetos abstratos de interação), que devem ser associados com as tarefas dos PSs. Esses

iteractors são objetos de interação de alto nível, classificados em um primeiro momento de acordo com o tipo de tarefa a ser executada e com o tipo e a cardinalidade dos elementos a serem manipulados; e, posteriormente, pelos aspectos de apresentação. É desta forma que a estrutura de apresentação começa a ser construída, pois os relacionamentos entre as tarefas são refletidos através dos diferentes relacionamentos entre esses *iteractors*.

No sistema Teresa uma interface abstrata é composta de um ou mais objetos do tipo *presentation*. Cada *presentation* possui duas partes: a primeira (*connection*), relacionada ao comportamento dinâmico da *presentation*, e a segunda, relacionada com a organização dos diferentes elementos que compõem a *presentation*. Pode-se ter, para cada *presentation*, zero ou mais objetos do tipo *connection*, onde cada *connection* identifica o elemento que permite mover-se para uma outra *presentation*.

Na segunda parte, descreve-se a organização estática dos diferentes elementos dentro da mesma *presentation,* que pode ser composta de dois tipos de elementos: um *iteractor_composition* e um *iteractor*. Na Figura 1 pode-se visualizar o diagrama de classes que representa a estrutura utilizada para o desenvolvimento da interface abstrata.

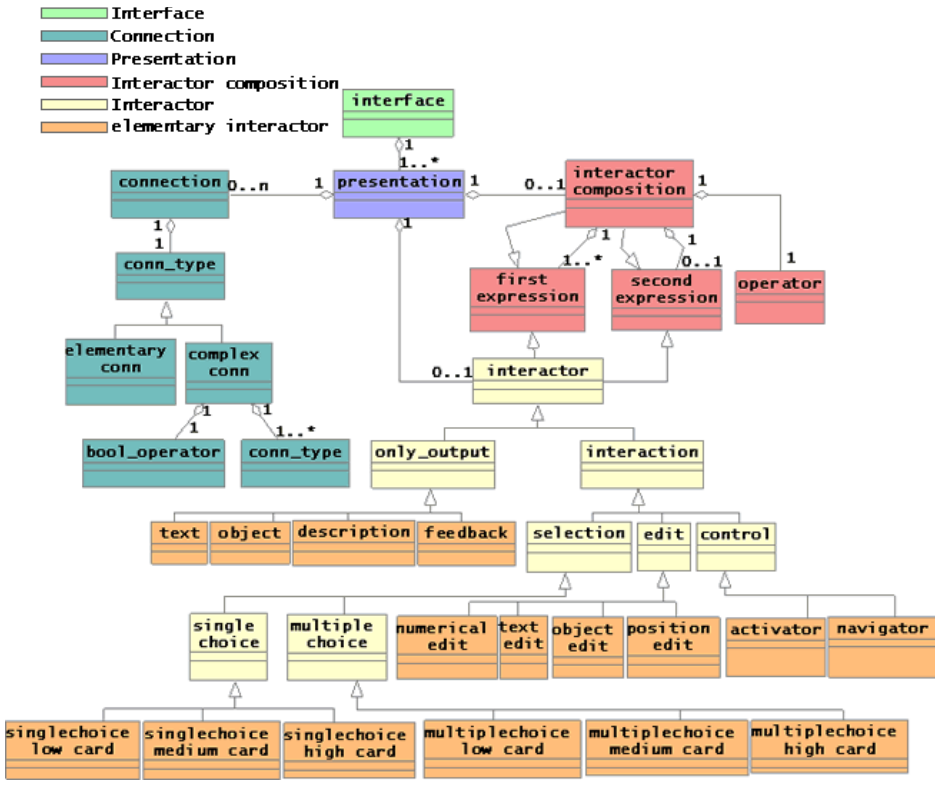


Figura 1 - Diagrama de classes do desenvolvimento da interface abstrata do Teresa XML.

Um objeto de interação abstrato, que representa uma interação entre o usuário e a aplicação, é definido por um *interactor*. Tal interação pode ser de diferentes tipos, visto que depende do tipo de tarefa a ser executada. Com esse objetivo, o sistema Teresa possui elementos do tipo:

- *selection*, que permitem selecionar um elemento entre um conjunto de elementos;
- *edit*, que permitem editar um elemento;
- *control*, que permite ativar um evento dentro da interface. Cada um desses elementos é, por sua vez, divido em sub elementos.

O elemento *selection* é dividido em dois tipos diferentes, o que vai caracterizar o número de elementos que serão selecionados: somente um de um conjunto (*single_choice*) ou muitos de um conjunto (*multiple_choice*). Tanto o *single_choice* como o *multiple_choice*, estão divididos em três tipos, classificados pela cardinalidade do conjunto em: baixa, média e alta.

O elemento *edit* é divido em quatro tipos diferentes, de acordo com o tipo de elemento que será editado: texto (*text_edit*), objeto (*object_edit*), valor numérico (*numerical_edit*) e posição (*position_edit*). O elemento *control* é divido entre dois elementos que realizam eventos particulares na interface: *navigator* e *activator*.

Uma vez definida a organização estática da interface abstrata, é necessário especificar o comportamento dinâmico. Para especificar esse comportamento é necessário se utilizar uma regra denominada *transition task*, na qual para cada conjunto de tarefas T, define-se um conjunto de regras de transição dessas tarefas, especificando o comportamento de cada uma delas.

Após a aquisição do modelo de organização estática e do comportamento dinâmico da interface, obtém-se a interface abstrata da aplicação gerada em XML. Assim, pode-se finalmente mapear a interface abstrata para uma interface concreta em uma linguagem específica.

3.1.1.3. Modelo de Interface Concreta

Nesta fase, cada *interactor* abstrato é mapeado em um objeto de interação concreto. Esse mapeamento leva em consideração o tipo de plataforma, a mídia disponível e o número de atributos que define mais concretamente a aparência e o comportamento dos *interactors*.

A geração da interface concreta é completamente subordinada à plataforma, visto que é necessário considerar propriedades específicas do dispositivo alvo. Por isso os *interactors* são mapeados em técnicas de interações, que são executadas considerando algumas configurações dos dispositivos particulares (sistema operacional, kit de ferramentas, entre outros). É necessário, também, especificar a linguagem na qual a interface abstrata será implementada; por exemplo, deve-se especificar se um *pull-down menu* em uma plataforma de telefone celular será exibido através de WML, XHTML *Mobile Profile*, ou outra linguagem.

A partir da descrição dessa linguagem pode-se notar que a mesma realiza uma descrição em detalhes de todos os aspectos que envolvem a interface. Seus elementos de interface abstrata incorporam funcionalidades, interações que levam em consideração parte da lógica da tarefa, mas também aspectos concretos como, por exemplo, a quantidade de itens a serem selecionados.

Desta forma, os seus elementos ficam muito específicos, ou seja, mais voltados para o nível concreto. Por exemplo, a noção de "edição" de um elemento implica, na verdade, em uma semântica de aplicação, e não puramente de interface.

Pode-se afirmar que o sistema Teresa trata a definição de interface abstrata em um nível mais próximo da implementação do que a proposta dessa dissertação; já que esta se preocupa em descrever apenas os aspectos relevantes para a realização das tarefas, considerando somente as trocas de informações entre o usuário e a aplicação.

3.1.2. XIML

O XIML (*eXtensible Interface Markup Language*) [19] é uma linguagem de representação baseada em XML que visa o suporte universal de funcionalidade da aplicação ao longo de todo o ciclo de vida de uma interface de usuário. O foco principal da sua abordagem é o desenvolvimento fundamentado em modelos. XIML oferece um mecanismo padrão para o intercâmbio de dados entre ferramentas e aplicações, desde o projeto até a operação e manutenção da interface. Isto é, permite que os modelos de projeto possam ser transformados em soluções de implementação multiplataformas.

A descrição de uma interface utilizando essa linguagem é composta de alguns detalhes em um nível abstrato e de alguns em um nível mais concreto, voltados a detalhes de implementação da interface. O XIML define três modelos que são utilizados para o desenvolvimento de interfaces: o modelo de tarefa, o modelo de plataforma e o modelo de apresentação.

O modelo de tarefa é uma representação estruturada das tarefas que usuário poderá realizar na aplicação. Esse modelo é decomposto hierarquicamente em sub tarefas que contém informações relacionadas aos objetivos, as pré-condições e as pós-condições de cada uma delas. O modelo de plataforma representa todos os elementos de cada plataforma específica, e cada elemento, por sua vez, contém atributos que descrevem suas características e restrições.

O modelo de apresentação detalha a aparência visual da interface. Nele são incluídas informações que descrevem a hierarquia dos elementos que compõem a interface, conhecidos como *widgets*. Esses *widgets* são definidos de acordo com uma relação entre AIOs (objetos de interação abstrata) e CIOs (objetos de interação concreta). Os AIOs são elementos que permitem ao usuário de uma aplicação visualizar e manipular a informação, enquanto os CIOs são elementos de interação que são executáveis em alguma plataforma com um processamento adicional. É interessante observar que os AIOs não são executáveis em nenhuma plataforma e, por isso, eles são completamente portáveis, permitindo a implementação de uma mesma interface em inúmeras plataformas.

Na Figura 2 é possível visualizar o exemplo de um mapeamento de um AIO (*Push Button*) em três CIOs e a plataforma correspondente a cada CIO.

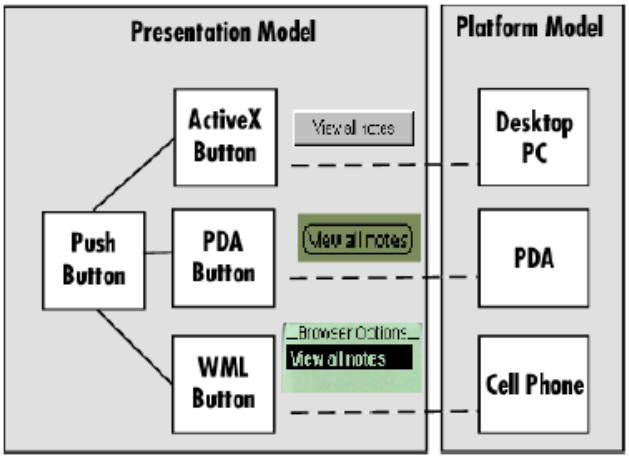


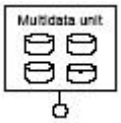
Figura 2 - Representação do mapeamento de um AIO em CIOs.[26]

Um modelo que representa uma interface abstrata em XIML é composto de vários *widgets*, que são representados pelos AIOs. Então, para se gerar a interface concreta, o sistema busca os AIOs e verifica o seu mapeamento para o CIO, procurando a informação sobre a plataforma alvo para descobrir o CIO correspondente.. A definição dos AIOs não é tão abstrata, pois está muito próxima da "mecânica" da interface, ou seja, alguns AIOs já especificam o tipo de aplicação do elemento. Por exemplo, o *Push Button* já indica a forma como será ativado um evento.

3.1.3. WebML

As especificações de interface do WebML [6] também são baseadas em XML e são independentes dos dispositivos de saída e da linguagem de programação. A representação dos elementos, que é descrita em XML, é utilizada para a realização de uma implementação e manutenção automática de sites. Utilizando a tecnologia XSL, as especificações do WebML são transformadas em *templates* de páginas, sendo traduzidos para uma linguagem específica que pode ser escolhida entre as que são suportadas atualmente.

A WebML utiliza unidades de elementos visuais para representar as informações contidas em uma página. Essas informações foram descritas no "modelo de dados" dessa linguagem e podem representar tanto o conteúdo quanto as operações. Como exemplos têm-se: uma unidade de dados, que representa um único objeto; uma unidade de dados múltiplos, que apresenta vários objetos juntos; uma unidade de índice, que apresenta objetos múltiplos de uma lista; uma unidade de hierarquia de múltiplas escolhas; uma unidade de entrada de dados; uma operação de *login* ou de *logout* da página; uma operação de enviar e-mail, dentre outros. Cada unidade possui propriedades específicas. A Tabela 2 apresenta um exemplo de algumas unidades do WebML.

Unit	Notação Visual	Descrição
Multidata		Representa vários objetos que representam dados de uma entidade.
Hierarchical Index		Uma variante de índice no qual as entradas de índice são organizadas em uma árvore de multi-níveis.

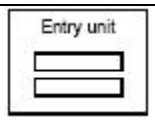
Data Entry		Representa um formulário que possui dados de entradas.

Tabela 2 - Exemplo de alguns tipos de Unit do WebML.

Um exemplo de representação de uma interface utilizando essa notação pode ser visualizado na Figura 3.

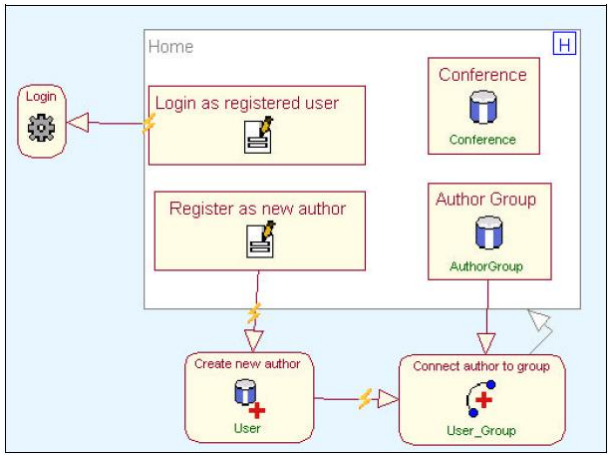


Figura 3 - Representação gráfica de uma interface abstrata, modelada em WebML.[27]

O site descrito na Figura 3 representa uma página principal de uma conferência, composta por quatro unidades que representam dados e três unidades que representam operações. A unidade *Conference* apresenta dados sobre a conferência e a unidade *Author Group* é uma unidade que apresenta dados sobre um grupo de autores e possui a operação *Connect author to group.* A unidade *Login as registered User* representa um formulário (entrada de dados) e possui uma operação de *Login*, da mesma forma que a unidade *Register as new autor* também representa um formulário. Ao ser ativada essa última unidade, uma nova instância de *User* é criada (operação *Create a New User*) e conectada a um grupo de autores (operação *Connect User to Group*).

As interfaces descritas utilizando a linguagem WebML são independentes dos dispositivos de saída e da linguagem de programação. Essas interfaces são descritas por meio de uma sintaxe XML abstrata, que realiza uma descrição abrangente dos elementos e das operações. As especificações de WebML são traduzidas para um arquivo JSP para gerar os *templates* das interfaces. A aparência gráfica das unidades é feita em uma outra linguagem (HTML, WML, dentre outras) e as consultas aos dados são definidas a partir de *Tag Libraries*. Os aspectos sobre a posição das unidades que compõem cada página devem ser

especificados para a geração desses *templates*, ou seja, deve-se criar um *Style Sheet* para essas unidades.

3.2. Abordagens não associadas a métodos de projeto

Algumas linguagens de marcação com base em XML, voltadas para a definição de uma interface de usuário, serão aqui apresentadas. Essas linguagens não são fundamentadas em modelos para a criação de uma interface e normalmente descrevem as interfaces em um nível menor de abstração. Das linguagens existentes, pode-se mencionar o XUL (XML *User Interface Language*) [7], o Sistema *Laszlo* [9], o UIML *(User Interface Markup Language*) [1], o XAML (*Extensible Application Markup Language*) [14] e o projeto PIMA (*Platform Independent Model for Applications*) [2], da IBM.

3.2.1. XUL

O XUL é uma linguagem que descreve interfaces, tendo sua base em XML. Desta forma, todas as características e vantagens disponíveis nas linguagens XML (por exemplo: XHTML, MathML e SVG) também estão disponíveis em XUL. Com essa linguagem pode-se criar facilmente aplicações para Web ricas e sofisticadas.

As especificações definidas com XUL podem, com pouco esforço, ser traduzidas para outras linguagens. Pode-se ainda facilmente implementar uma interface e modificá-la rapidamente. Alguns dos elementos que podem ser criados em XUL são:

- Controles de entrada como *checkboxes* e *textboxes*;
- Barra de ferramentas com botões ou outros tipos de conteúdos;
- Barras de menus ou menus de pop up;
- Árvores hierárquicas ou tabelas de informação;
- Atalhos para o teclado.

Similarmente ao HTML, em XUL é possível criar uma interface que usa uma linguagem de marcação, podendo usar o CSS *Style Sheet* para definir a aparência e *Java Script* para definir o comportamento do site. XUL pode, também, ser usada com inúmeros padrões existentes incluindo o XSLT, o XPath e o DOM como funções para manipular uma interface do usuário. Na realidade, o XUL

permite a especificação de interfaces mais ricas do que aquelas disponibilizadas estritamente através da WWW, como a interface do *browser* FireFox, que pode ser visualizada na Figura 4, implementada utilizando a linguagem XUL.

Figura 4 - Exemplo da tela do *browser* FireFox.

Descritas em linguagem XUL, as interfaces requerem um interpretador para que sejam geradas de forma automática para sua visualização no *browser*. Desta maneira a sua principal desvantagem é a baixa portabilidade, pois não é possível utilizar XUL em todos os *browsers*. Na Figura 5 apresenta-se um exemplo que descreve quatro botões em linguagem XUL, e a sua visualização concreta (os botões prontos para serem visualizados pelo usuário final) é ilustrada na Figura 6.

```
<button label="Left" image="happy.png"/>
<button label="Right" image="happy.png" dir="rtl"/>
<button label="Above" image="happy.png" orient="vertical"/>
<button label="Below" image="happy.png" orient="vertical" dir="rtl"/>
```

Figura 5 - Descrição de botões em linguagem XUL.

Figura 6 - Visualização dos botões descritos na Figura 5.

Como pode ser visto na Figura 5, a descrição dos elementos de interfaces é feita em um nível não muito abstrato, pois propriedades específicas da interface concreta são definidas. Dessa maneira, a descrição dos elementos em XUL está

mais próxima da ontologia de *widgets* concretos do que de *widgets* abstrato, sendo mesmo possível realizar o mapeamento da ontologia abstrata para as especificações em XUL.

3.2.2. Sistema Laszlo

Similar ao XUL, o sistema Laszlo utiliza a linguagem XML para descrever aplicações Web, permitindo a utilização das linguagens como *Java Script* e XPath para enriquecer essas aplicações. Possibilita ainda a construção rápida e on-line de aplicações ricas para a Internet, através de um "gerador". Este gerador possui um sistema que interpreta o XML e gera um elemento em *flash*, para apresentar a interface final para o usuário através de um *plugin* de *Flash*, acessado a partir de qualquer *browser*.

As aplicações Web derivadas do sistema Laszlo rodam dentro de todos os *browsers* e plataformas mais conhecidas. Esse sistema permite uma nova classe de aplicações ricas, interativas e sofisticadas para a Internet, mais fáceis de desenvolver e utilizar. A Figura 7 exemplifica um código desse sistema, que gera quatro botões. Na Figura 8 pode-se visualizar os botões concretos referentes a descrição da Figura 7.

```
<canvas>
   <button x="100" y="40" width="60">Left</button>
   <button x="100" y="60" width="60">Right</button>
   <button x="100" y="80" width="60">Above</button>
   <button x="100" y="100" width="60">Below</button>
</canvas>
```

Figura 7 - Descrição de quatro botões utilizando o sistema Laszlo

Figura 8 - Visualização dos botões descritos na Figura 7.

Como pode ser visualizado na Figura 7, a descrição dos elementos de interface é semelhante a forma como é realizada na linguagem XUL, pois são definidas informações específicas dos elementos concretos. A descrição realizada por esse sistema não é muito abstrata e, assim, pode-se utilizar essa linguagem para gerar os elementos concretos da proposta dessa dissertação, ou seja, substituir a ontologia de *widgets* concretos pelos elementos gerados pelo sistema Laszlo.

3.2.3. XAML

A XAML é uma linguagem de marcação extensível para definição de interfaces para aplicações. Ela está sendo desenvolvida pela Microsoft e permite definir interfaces para aplicativos a partir de uma sintaxe XML. Essa linguagem trata a lógica separada da aparência, ou seja, permite a utilização de *JavaScript* para definir a lógica e CSS para a aparência. Na Figura 9 apresenta-se um exemplo que representa um formulário de *Login*, descrito em XAML.

```
<Canvas xmlns="http://schemas.microsoft.com/2003/XAML"  ID="LoginPage">

  <Label ID="lblUserId" FontFamily="verdana"
    FontSize="8" Canvas.Top="10" Canvas.Left="10">
    User ID:</Label>

  <TextBox ID="txtUserId" FontSize="8" Canvas.Top="8"
   Canvas.Left="75"></TextBox>

  <Label ID="lblPassword" FontFamily="verdana"
    FontSize="8" Canvas.Top="35"
    Canvas.Left="10">Password:</Label>

  <TextBox ID="txtPassword" FontSize="8"
    Canvas.Top="33" Canvas.Left="75"></TextBox>

  <Button ID="cmdOK" Width="50" Canvas.Top="60"
   Canvas.Left="155">OK</Button>

</Canvas>
```

Figura 9 - Definição de um formulário de *Login* em linguagem XAML.

Neste exemplo pode-se observar a idéia de XAML, onde cada interface é definida por um arquivo XAML. Cada interface possui um elemento principal (*tag Canvas*) onde os elementos gráficos são definidos. Essa definição é feita em um nível não muito abstrato da mesma maneira como em XUL e no sistema Laszlo.

A principal desvantagem dessa linguagem é que ela constrói objetos *Avalon*, que são objetos nativos do novo sistema operacional *Longhorn*, uma nova versão do Microsoft Windows que irá substituir o Windows XP em 2005 ou 2006. Tal fato torna essa tecnologia dependente da plataforma Windows.

3.2.4. UIML

UIML também é similar às propostas citadas acima, pois ela é uma linguagem declarativa baseada em XML, que descreve interfaces para o usuário. A UIML tem como objetivo criar interfaces para múltiplas plataformas de

softwares e para dispositivos diferentes de diversas aplicações. Essa linguagem permite a implementação de interfaces para qualquer dispositivo, sem requerer o aprendizado de linguagens e aplicações de programação de interfaces (APIs) específicas dos dispositivos. Ela mantém uma separação entre o código de interface e o código lógico da aplicação.

Na linguagem UIML cada elemento pode ser organizado de vários modos, para diferentes categorias de usuário finais e diferentes dispositivos. Um elemento pode ser composto de um conteúdo específico (por exemplo: texto, sons e imagens) e de uma classe, que descreve o estilo de apresentação (por exemplo: fonte, cor, tamanho, entre outros) do mesmo. Na Figura 10 é possível visualizar uma interface abstrata, descrita em linguagem UIML, enquanto a interface concreta correspondente ao código dessa figura é apresentada pela Figura 10.

```
1  <UIML>
2    <HEAD>
3        <AUTHOR>Stephen Williams</AUTHOR>
4         <DATE>December 3, 1997</DATE>
5         <VERSION>1.0</VERSION>
6     </HEAD>

7     <APP CLASS="App" NAME="DialogApp">
8        <GROUP CLASS="Dialog" NAME="PrintFinishedDialog">
9              <ELEM CLASS="DialogMessage" NAME="PrintFinishedMsg"/>
10              <ELEM CLASS="DialogButton" NAME="OKButton"/>
11       </GROUP>
12    </APP>

13    <DEFINE NAME="OKButton">
14       <PROPERTIES>
15           <ACTION
16               VALUE="DialogApp.EXISTS=false"
17               TRIGGER="Selected"
18             />
19       </PROPERTIES>
20    </DEFINE>
21 </UIML>
```

Figura 10 - Definição de uma interface em linguagem UIML.

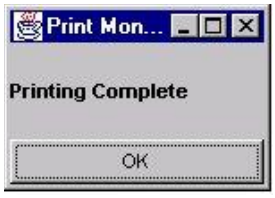


Figura 11 - Interface concreta correspondente ao código da Figura 10.

Nas linhas 9 e 10 (Figura 10) estão declaradas as classes que irão tratar do estilo de apresentação de cada elemento. Primeiramente é descrito o nome da classe e, depois, o nome do elemento. A definição das propriedades funcionais, ou seja, que não tratam da apresentação visual do elemento, é descrita entre as linhas 13 e 20 (Figura 10). Neste caso foi especificada apenas uma propriedade para o

elemento "OKButton", que indica qual a ação que ocorre quando esse elemento for ativado.

Como pôde ser notado, para descrever uma interface em UIML deve-se especificar as propriedades próprias dos elementos concretos. Desta forma, percebe-se que essa proposta não descreve interfaces de uma forma tão abstrata quando comparada com a proposta dessa dissertação.

Todas as quatro abordagens apresentadas acima estão mais voltadas para a ontologia de *widgets* concretos do que para a de *widgets* abstratos, pois todas elas descrevem detalhes específicos dos elementos concretos. A ontologia de *widgets* abstratos proposta nessa dissertação se preocupa em descrever apenas os aspectos que são relevantes para representar as trocas de informação entre o usuário e interface. Conseqüentemente, esta ontologia pode ser mapeada para qualquer uma dessas abordagens discutidas acima, gerando os elementos concretos de interface. que podem estar descritos tanto em linguagem XUL, Laszlo, XAML ou UIML.

3.2.5. Projeto PIMA

Esse projeto é semelhante à proposta dessa dissertação. PIMA (*Platform-Independent Model for Applications*) é um projeto da IBM que visa especificar aplicações independentes de plataforma. Para se desenvolver uma aplicação hipermídia com esse projeto, a aplicação irá conter um ciclo de vida que está dividido em três partes: *design-time*, *load-time* e *run-time.*

Design-time é quando o projetista cria a aplicação. É nessa fase que ele identifica os elementos abstratos de interação, realiza a modelagem e o desenvolvimento da aplicação. A fase de *load-time* é a que compõe o sistema, adaptando e carregando os componentes que fazem parte da aplicação em dispositivos. A fase de *run-time* é quando o usuário executa a aplicação através de um dispositivo e começa a usar as funcionalidades da aplicação. O sistema fornece um ambiente, no qual a aplicação pode rodar para que sejam realizados vários testes. Essa proposta ainda está em aprimoramento. Na Figura 12 ilustra-se o resumo da proposta do projeto PIMA.

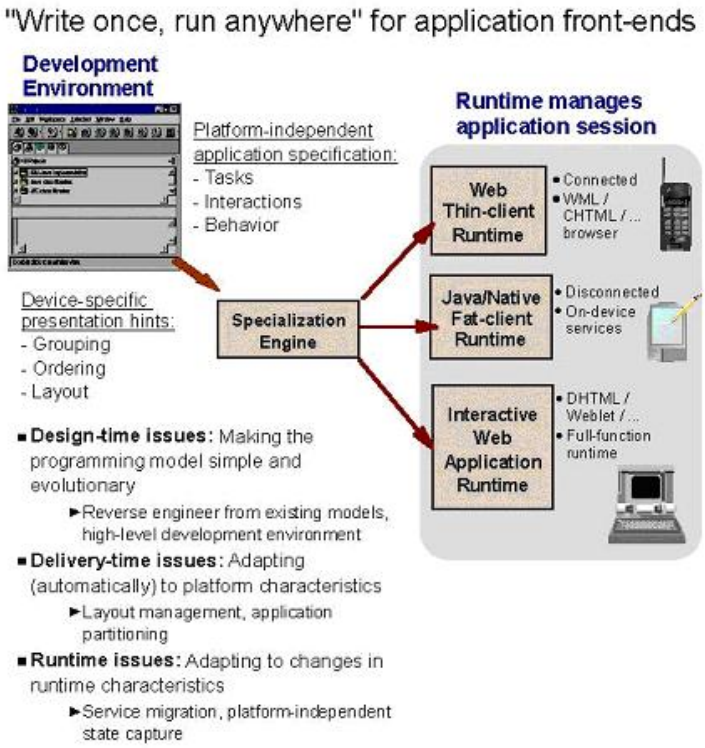


Figura 12 – Proposta do projeto PIMA [22].