Universidade Federal do Rio Grande do Norte
Centro de Ciências Exatas e da Terra
Departamento de Informática e Matemática Aplicada
Mestrado em Sistemas e Computação

GERAÇÃO DE INTERFACES DE USUÁRIO DE SISTEMAS WEB PARA MÚLTIPLOS DISPOSITIVOS COM O USO DE COMPONENTES DE IU

Lirisnei Gomes de Sousa

Natal – RN
Fevereiro de 2007

Lirisnei Gomes de Sousa

GERAÇÃO DE INTERFACES DE USUÁRIO DE SISTEMAS WEB PARA MÚLTIPLOS DISPOSITIVOS COM O USO DE COMPONENTES DE IU

Dissertação submetida ao Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte como parte dos requisitos para a obtenção do título de Mestre em Sistemas e Computação.

Orientador: Prof. Dr. Jair Cavalcanti Leite - UFRN

Natal – RN
Fevereiro de 2007

Divisão de Serviços Técnicos
Catalogação da Publicação na Fonte. UFRN / Biblioteca Central Zila Mamede

Sousa, Lirisnei Gomes de.
Geração de interfaces de usuário de sistemas Web para múltiplos dispositivos com o uso de componentes de IU / Lirisnei Gomes de Sousa. – Natal [RN], 2007.
108 f.

Orientador: Jair Cavalcante Leite.

Dissertação (Mestrado) – Universidade Federal do Rio Grande do Norte. Centro de Ciências Exatas e da Terra. Departamento de Informática e Matemática Aplicada. Programa de Pós-Graduação em Sistemas e Computação.

1. interface do usuário - Dissertação. 2. Design de interface do usuário - Dissertação. 3. Linguagem XICL - Dissertação. I. Leite, Jair Cavalcante. II. Universidade Federal do Rio Grande do Norte. III. Título.

RN/UF/BCZM CDU 004.5(043.3)

Lirisnei Gomes de Sousa

GERAÇÃO DE INTERFACES DE USUÁRIO DE SISTEMAS WEB PARA MÚLTIPLOS DISPOSITIVOS COM O USO DE COMPONENTES DE IU

Esta Dissertação foi julgada adequada para a obtenção do título de mestre em Sistemas e Computação e aprovado em sua forma final pelo Programa de Pós-Graduação em Sistemas e Computação do Departamento de Informática e Matemática Aplicada da Universidade Federal do Rio Grande do Norte.

_____

Prof. Dr. Jair Cavalcanti Leite - UFRN
Orientador

_____

Profa. Dra. Thais Vasconcelos Batista – UFRN
Coordenador do Programa

Banca Examinadora

_____

Prof. Dr. Jair Cavalcanti Leite - UFRN
Presidente

_____

Profa. Dra. Anamaria Martins Moreira.– UFRN

_____

Prof. Dr. Celso Alberto Saibel Santos – UNIFACS

Natal – RN

Abril de 2007

AGRADECIMENTOS

Agradeço primeiramente a Deus, pela sua presença em todos os momentos e pela saúde e disposição para a elaboração deste trabalho.

Agradeço a toda a minha família, de forma especial, aos meus pais e meus irmãos, pelo incentivo em todos os aspectos.

A todos da minha família de Natal, que até aqui, foram capazes de me suportar por sete anos. É muita paciência!.

Ao orientador, Professor Jair, pela confiança e paciência, tirando dúvidas e transmitindo um conhecimento que será importante durante toda a minha vida profissional.

Aos meus colegas da pós-graduação e do grupo de estudos em IHC da UFRN.

À CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior), pela concessão da bolsa de estudos.

E por fim a todos que direta ou indiretamente contribuíram no desenvolvimento deste trabalho.

RESUMO

Este trabalho apresenta um processo de geração de protótipos de Interfaces de Usuário (IU) para software que tem como plataforma operacional um navegador Web. Este processo faz o uso de componentes de interface de usuário mais complexos que os elementos HTML. Para descrever estes componentes mais complexos este trabalho sugere o uso da linguagem XICL (eXtensible User Interface Components Language). A XICL é uma linguagem, baseada em XML, para a descrição de componentes de IU e de IUs. A XICL permite reusabilidade e extensibilidade no desenvolvimento de interfaces de usuários. Foram desenvolvidos dois compiladores, um que gera código XICL a partir de código IMML (Interactive Message Modeling Language) e outro que a partir de XICL gera DHTML.

Área de Concentração: Engenharia de Software.

Palavras-chave: Design de Interface do Usuário, Linguagem de descrição de Interface de usuário, Desenvolvimento baseado em componentes e IMML.

ABSTRACT

This work presents an User Interface (UI) prototypes generation process to the softwares that has a Web browser as a plataform. This process uses UI components more complex than HTML elements. To described this components more complex this work suggest to use the XICL (eXtensinble User Interface Components Language). XICL is a language, based on XML syntax, to describe UI Components and IUs. XICL promotes extensibility and reusability in the User Interface development process. We have developed two compiler. The first one compiles IMML (Interactive Message Modeling Language) code and generates XICL code. The second one compiles XICL code and generates DHTML code.

***Area of Concentration**: Software Engineering.*

***Key words**: User Interface Design, User Interface Description Language, Component Based Development and IMML.*

Sumário

Lista de Figuras

Lista de Acrônimos

DIUBM	Desenvolvimento de Interface de Usuário Baseado em Modelos
DHTML	Dinamic HTML
GUI	Graphics User Interface
HUI	Handheld User Interface
IHC	Interação Humano - Computador
IMML	*Interactive Message Modeling Language*
IU	Interface de Usuário
LDIU	Linguagem de Descrição de Interface de Usuário
OWL	Web Ontology Language
UML	*Unified Modeling Language*
WUI	Web-based User Interface
W3C	*World Wide Web Consortium*
XICL	*eXtensinble User Interface Components Language*
XML	*eXtensible Markup Language*

1 Introdução

Com o surgimento de diversos tipos de dispositivos computacionais, tais como computadores de mão (palmtops) e telefones celulares com telas com maiores resolução, também surge a necessidade da criação de software para estes dispositivos. Além disso, os usuários querem usar, nestes novos dispositivos, as aplicações que eles já utilizam em seus computadores convencionais (desktops), como por exemplo, consultar e-mails ou realizar transações bancárias pelo celular. Sistemas que podem ser usados a partir de diversos dispositivos são chamados de sistemas multi-plataforma.

Um *contexto de uso* de um sistema multi-plataforma é definido por (i) um usuário, (ii) uma plataforma computacional (hardware/software) e (iii) um ambiente de uso (Calvary et al., 2003). Algumas Interfaces de Usuário (IUs) são desenvolvidas de forma a serem *adaptadas* diante das mudanças de contexto (Thevenin 1999). Para isto, é necessário que haja mecanismos que façam com que, quando a interface estiver sendo utilizada em um contexto c_i, a mesma interface possa ser ajustada para ser usada em um contexto c_j. Por exemplo, se um Sistema de Informações Geográficas pode ser utilizado por geólogos e por turistas, a interface de usuário do sistema pode ser adaptada ao perfil do usuário que está interagindo com a mesma. Neste caso, a adaptação consiste em adequar as informações aos interesses e cultura do usuário, pois um turista, normalmente, não tem interesses em informações sobre a estrutura geológica do local, por exemplo. Portanto, informações deste tipo não precisam ser apresentadas ao turista. Estes mecanismos de adaptação podem estar embutidos dentro da própria interface ou em uma ferramenta. Desta forma uma mesma IU pode ser usada em diferentes contextos.

Com a variação dos dispositivos computacionais surgiram estilos de interfaces de usuário. Os estilos mais comuns a uma interface são (Seffah and Javahery, 2004):

- Graphics User Interface (GUI): este estilo é o mais conhecido pela maioria dos usuários, normalmente usado nos desktops. Também é conhecido como WIMP (Windows, Icons, Menus and Pointers);
- Web-based User Interface (WUI): este estilo é o utilizado nas aplicações que têm como plataforma de execução um navegador Web, este por sua vez é um software que segue o estilo GUI.
- Handheld User Interface (HUI): este estilo está presente nos telefones móveis, nos assistentes digitais pessoais (PDAs) e em alguns terminais eletrônicos. O usuário interage através de gestos, usando caneta especial e/ou toque na tela.

Alguns sistemas já estão sendo desenvolvidos para serem utilizados através de diversos dispositivos. Como exemplos destes sistemas podemos citar (Bittencourt 2004), que prevê o uso de jogos computacionais através de diversos dispositivos. O serviço de mensagens eletrônicas Gmail™ da empresa Google[1] também permite que o usuário acesse o serviço através de celulares.

O foco deste trabalho está no desenvolvimento de protótipos de WUI para múltiplos dispositivos, tais como: computadores de mesa, computadores de mão e telefones celulares.

1.1 DESENVOLVIMENTO DE IU PARA MÚLTIPLAS PLATAFORMAS

Uma interface de usuário pode ser implementada diretamente em uma linguagem de programação. Entretanto, alguns os processos de desenvolvimento de interface de usuário mais rigorosos (Palanque 1994, Paterno 1997) sugerem que, antes de implementar a interface em uma linguagem de programação, é importante que seja feita uma especificação da interface. No desenvolvimento de software, através de uma especificação é possível descrever: os requisitos de software; a interface de usuário do software; a arquitetura do software; e até mesmo os casos de teste do software. Uma especificação pode ser descrita: em linguagem natural, o que possibilita o

[1] Informações podem ser obtidas em http://www.google.com.

entendimento por não especialistas da área de software; em notação gráfica, como a UML; ou em uma notação baseada em um formalismo matemático.

A metodologia proposta em (de Souza et al., 1999) sugere a especificação da interface de usuário no processo de desenvolvimento da mesma, e o processo desenvolvimento deve ser iniciado com a "análise de usuários e tarefas (que pode ser considerada parte da análise de requisitos) e deve ser conduzido num processo cíclico ou iterativo no qual cada ciclo apresenta melhorias em relação ao clico anterior. Cada ciclo compreende 3 passos: a *especificação* da funcionalidade e do modelo de interação; a *prototipação* de interfaces; e a sua *avaliação* junto aos usuários". Para aplicar esta metodologia no desenvolvimento de IUs para sistemas multi-plataforma, é necessário repetir o processo para cada dispositivo, ou seja, para desenvolver uma IU para o usuário interagir com o sistema através de um desktop é necessário realizar o ciclo as vezes que se façam necessárias. E para desenvolver uma interface para o usuário interagir, com o mesmo, sistema através de um celular é necessário repetir o ciclo outras vezes. A figura 1.1 mostra uma visão geral da metodologia.

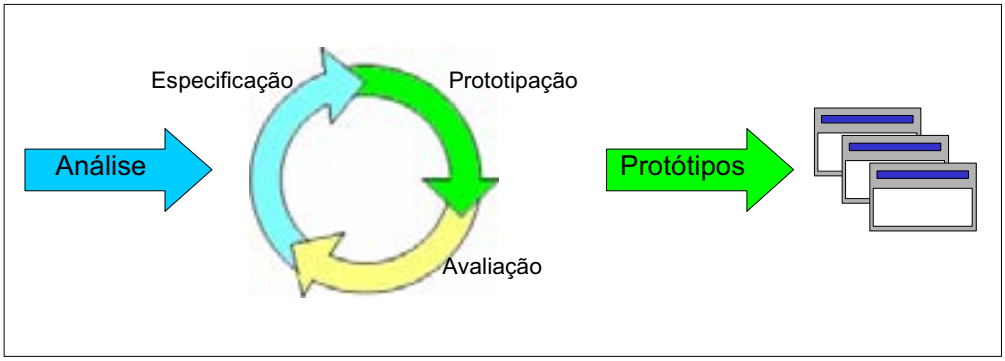


Figura 1.1: Processo de design de interfaces (de Souza et al., 1999)

Outra maneira que o *designer* pode escolher para desenvolver as diferentes IUs, para que os usuários possam interagir com um sistema multi-plataforma, em diferentes contextos, é fazer uma especificação de uma interface de usuário multi-plataforma (IUMP). Uma especificação de uma interface de usuário multi-plataforma é uma especificação que pode ser usada para gerar, com o auxílio de ferramentas, diversas IUs para um sistema, uma para cada contexto.

1.2 ESPECIFICAÇÕES DE INTERFACES DE USUÁRIO

A especificação de uma interface de usuário deve descrever como o usuário pode interagir com o sistema usando elementos de software e de hardware. Existem diferentes linguagens de especificação de interfaces e cada uma é mais apropriada para descrever interfaces em um estilo específico (GUI, WUI, ou HUI). Cada linguagem de especificação possibilita a descrição de um conjunto de aspectos relacionados à interface. Estes aspectos são, por exemplo: diálogo entre o usuário e o sistema; a estrutura visual da interface; as tarefas que podem ser realizadas pelo usuário; a funcionalidade do sistema; e aspectos relacionados à aparência da interface como cores e fontes. Estes aspectos são descritos através de *descrições abstratas*. Uma descrição abstrata é dependente da linguagem de especificação utilizada. Por exemplo, uma linguagem de especificação pode definir que o *designer* deve especificar a ação do usuário "inserir dado" com o uso da frase *enter information*. A descrição abstrata desta mesma ação "inserir dado" pode ser feita, em outra linguagem de especificação, através do uso da frase *insert information,* por exemplo.

Entre as linguagens que podem ser utilizadas para especificar IUs estão: GOMS *(Goals, Operators, Methods, and Selection Rules*) (Card et al. 1983), que permite a descrição de forma textual das tarefas que o usuário possivelmente realizará; A UML (*Unified Modeling Language*) (Booch et al. 1999), considerada uma linguagem padrão de modelagem na Engenharia de Software; e as Linguagens de Descrição de IU (LDIU) baseadas em XML (Puerta and Eisenstein 2001, Phanouriou 2002, Limbourg and Vanderdonckt 2004, Leite 2003).

As LDIUs baseadas em XML, é a técnica mais empregada para especificar interfaces de usuário multi-plataforma. Exemplos de LDIUs baseadas em XML que podem ser utilizadas no desenvolvimento multi-plataforma são: UIML (Phanouriou 2002), XIML (Puerta and Eisenstein, 2001), UsiXML (Limbourg and Vanderdonckt, 2004) e IMML (Leite 2003). As linguagens XIML, UsiXML e IMML seguem o paradigma do desenvolvimento de IUs baseado em modelos (Puerta and Szkeley, 1994), onde uma IU é composta por um conjunto de modelos e cada um deles agrupa um conjunto de aspectos relacionados a IU.

A especificação de uma IU feita com a linguagem UIML deve ser descrita levando-se em consideração as limitações dos dispositivos (entrada/saída) que serão

utilizados pelo usuário, para interagir com a IU. Com isto, uma especificação UIML que descreve uma IU para ser utilizada em um desktop, por exemplo, pode ser utilizada para: gerar uma IU para rodar em um navegador Web, instalado em um desktop; ou para gerar uma IU codificada em Java, também para ser utilizada em um desktop. Entretanto, esta mesma especificação UIML não pode ser utilizada para gerar uma IU para ser executada em celular, isto porque o tamanho da tela do celular, provavelmente, não permitiria o usuário interagir com a IU, já que a mesma foi desenvolvida para ser utilizada em um desktop.

As descrições feitas nas linguagens XIML e UsiXML são independentes de dispositivo, ou seja, a partir de uma mesma descrição é possível gerar IUs para diferentes dispositivos. A desvantagem dessa independência de dispositivo está no fato de que as interfaces geradas, a partir de descrições deste tipo, podem necessitar de vários ajustes.

A IMML apresenta uma abordagem intermediária, entre os dois casos citados acima. Assim, uma parte da descrição é independente de dispositivo e a outra parte da descrição considera aspectos relacionados às características do dispositivo pelo qual o usuário interage com a IU. Esta linguagem apresenta uma proposta que segue uma perspectiva, diferente das anteriores, baseada na teoria da Engenharia Semiótica (de Souza, 1993). A IMML está sendo desenvolvida pelo grupo de estudos em IHC da Universidade Federal Rio Grande do Norte.

1.3 MAPEANDO ESPECIFICAÇÕES EM COMPONENTES

A geração de uma IU executável a partir de uma especificação é feita através do mapeamento de *descrições abstratas* em objetos de interface (*widgets*), disponíveis na plataforma alvo, que possibilitem a interação com o usuário. *Widgets* são elementos visuais (gráficos), com aparência de controles reais (botões), com os quais o usuário pode interagir através de um dispositivo de entrada, como o mouse.

Se em uma especificação de uma IU, uma descrição abstrata relacionada ao aspecto visual da IU descreve um *frame*, por exemplo, ao se gerar uma IU em HTML este *frame* pode ser mapeado para o elemento **frame** da linguagem HTML. Se esta mesma IU for compilada para Java este *frame* pode ser mapeado para um JFrame, ou para um JPanel, ou para outro componente que tenha a função de agrupar elementos

de interação. As ferramentas fazem este mapeamento baseando-se em *regras de mapeamento*.

Como as descrições abstratas são mapeadas em componentes de IU da plataforma alvo, o uso de bibliotecas de componentes de IU facilita a definição de regras de mapeamento. Como exemplos de bibliotecas de componentes de IU, podemos citar: a biblioteca Swing para a linguagem Java e a biblioteca GTK para a linguagem C/C++.

A falta de uma biblioteca de componentes de IU na plataforma alvo dificulta o processo de geração de interfaces de usuário. Esta dificuldade é encontrada na definição de regras de mapeamento para gerar IUs do estilo WUI, desenvolvidas com tecnologias que sigam as recomendações do W3C, ou seja, desenvolvidas com DHTML (HTML Dinâmica) (Goodman 1998). Não há recomendação de linguagens e modelos para o desenvolvimento de componentes de IU, utilizando as tecnologias padrões do W3C. Isto dificulta a definição de regras de mapeamento para a geração automática de IUs em DHTML.

A definição de regras de mapeamento para a linguagem Java, por exemplo, sem o uso de uma biblioteca de componentes de IU, é um processo complicado. O uso de uma biblioteca, como a Swing, facilita o processo. Pois esta biblioteca além de possuir uma grande variedade de componentes, ainda permite que novos componentes sejam desenvolvidos a partir dos já existentes na biblioteca.

A linguagem HTML já possui elementos que são interpretados pelos navegadores Web, e estes apresentam objetos visuais que provêem interação entre o usuário e a interface. Entretanto, se uma especificação de uma IU contém a descrição de um **menu**, por exemplo, como não existe um elemento HTML equivalente a um menu, esta descrição deverá ser mapeada em um conjunto de elementos HTML, que devem ser agrupados para formar um menu em HTML. Outro fator de dificulta é a inexistência de uma maneira padronizada para desenvolver, estender e reutilizar componentes fazendo o uso somente das tecnologias padrões. A Figura 1.2 representa a dificuldade de mapeamento na geração de uma janela a partir de uma especificação: em C/C++, usando a biblioteca GTK só é preciso fazer uso de um componente que representa a janela; em Java, usando a biblioteca Swing também só é preciso fazer uso

de um componente; e para HTML é necessário agrupar vários elementos HTML para montar uma janela.

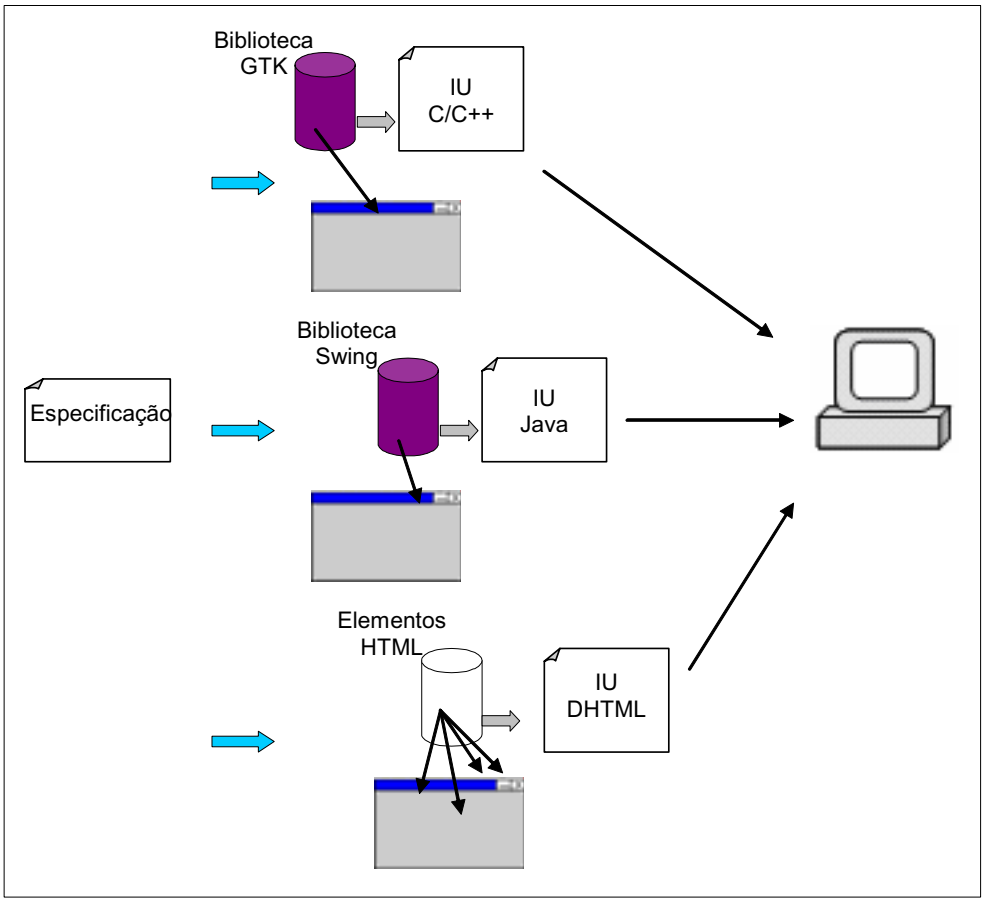


Figura 1.2: Geração de várias IUs a partir de uma especificação abstrata

1.4 OBJETIVOS

Os objetivos deste trabalho são: (1) elaborar um processo de geração de protótipos de IUs, no estilo WUI, a partir de especificações de interfaces de usuário multi-plataforma; (2) utilizar, neste processo, componentes de IU; e (3) desenvolver uma ferramenta para implementar esse processo.

O trabalho apresenta um processo de geração de protótipos de IUs a partir de especificações feitas na linguagem de descrição de interface usuário IMML. Neste processo, as descrições IMML são mapeadas em componentes de XICL.

Os componentes envolvidos no projeto visam permitir abstração, reuso e extensão de elementos HTML. Estes componentes podem ser: barras de menu; barras de ferramentas; e etc. Para descrever os componentes, o trabalho sugere o uso da XICL (eXtensible User Interface Components Language). Os componentes XICL são descritos a partir de componentes HTML e/ou de componentes XICL já descritos. Esta linguagem oferece uma forma padronizada para o desenvolvimento de componentes que viabilize a reusabilidade, a extensibilidade e a portabilidade dos mesmos. Com o uso destes componentes a definição das regras de mapeamento é facilitada.

Para apoiar o processo de geração, foi desenvolvida uma ferramenta com dois compiladores. O compilador IMML-XICL gera código XICL a partir de especificações de IUs feitas em IMML. Como o código XICL não é interpretado pelos navegadores Web, foi desenvolvido o compilador XICL-DHTML que gera código em DHTML a partir de código XICL. Com isso é possível gerar código DHTML a partir de especificações IMML, fazendo uso de componentes.

Esta dissertação esta organizada em 7 capítulos, incluindo a introdução. O capítulo 2 apresenta um levantamento de algumas técnicas propostas para o desenvolvimento de IUs multi-plataforma. O capítulo 3 apresenta os principais conceitos e elementos da linguagem IMML. O capítulo 4 apresenta a XICL com exemplos. O capítulo 5 apresenta o processo de geração de IUs para diversos dispositivos a partir de descrições IMML. O capítulo 6 apresenta os compiladores para IMML e XICL. O capítulo 7 faz as considerações finais.

2 Fundamentação Teórica

Um software e/ou uma interface de usuário (IU) podem ser utilizados em diferentes lugares, por diferentes pessoas. Essa variação de pessoas e lugares levou à criação da expressão *contexto de uso*, que é composta por três conceitos (Calvary et al., 2003):

- Usuário: que representa o indivíduo que faz o uso do sistema, levando em consideração o conhecimento do mesmo sobre o domínio da aplicação e sobre o software.
- Plataforma: é o conjunto de combinações hardware/software através dos quais os usuários podem interagir com o sistema.
- Ambiente: é o ambiente em que os usuários podem estar inseridos ao interagir com o sistema, levando em consideração fatores como o ruído e a luminosidade.

Existem várias possibilidades de variação no contexto de uso de um sistema. Os softwares que são utilizados em vários contextos recebem o nome de *multi-plataforma*. A expressão *multi-plataforma* pode levar ao entendimento de que só são previstas variações na plataforma (software/hardware), mas a variação pode ser tanto de plataforma quanto de usuário e de contexto. O foco deste trabalho está restrito às variações que podem ocorrer na plataforma (software/hardware). Entretanto, consideramos somente as variações relativas ao hardware, uma vez que estamos interessados em sistemas em que as IUs estão no estilo WUI.

Ao desenvolver diferentes versões da IU, uma para cada contexto de uso, o *designer* pode encontrar alguns problemas, como os destacados em (Paterno and Tschelig, 2003):

- requer esforços extras no desenvolvimento e manutenção, aumentando os custos e complicando o gerenciamento de configuração, isto devido ao fato de que a manutenção deve ser feita em cada interface individualmente;
- a proliferação de diferentes versões dilui os recursos disponíveis para engenharia de usabilidade e requer uma cara manutenção da consistência da interface entre as várias plataformas. A cada mudança é necessário comparar a usabilidade das IUs geradas para as diferentes plataformas.

Outra estratégia, para prover interação em diferentes contextos, é desenvolver interfaces de usuário multi-plataforma (IUMP). Uma IUMP pode ser: (i) uma IU que pode ser *adaptada* de um contexto para outro (Thevenin 1999); (ii) ou uma especificação abstrata, a partir da qual seja possível gerar IU executáveis em diferentes plataformas. O uso de IUMP pode gerar alguns problemas, como os destacados em (Myers et al., 2000), que são:

- a imprevisibilidade do comportamento de cada IU gerada;
- a dificuldade na definição de regras para mapeamento as descrições abstratas em componentes de software da plataforma alvo;
- os programadores precisam aprender uma nova linguagem para especificar os modelos, elevando o esforço necessário para empregar a abordagem.

A seguir, são apresentados alguns conceitos relacionados ao desenvolvimento de interfaces de usuário multi-plataforma.

2.1 DESENVOLVIMENTO DE INTERFACE DE USUÁRIO BASEADO EM MODELOS

O desenvolvimento de interface de usuário baseado em modelos (DIUBM) é um paradigma de desenvolvimento que usa um repositório central para armazenar uma descrição de todos os aspectos do projeto de uma interface. Este repositório central é chamado de *modelo*, que tipicamente contém informações sobre os objetos do domínio da aplicação, as características dos usuários, as tarefas que os usuários esperam realizar utilizando a interface, bem como a estrutura e o comportamento da própria interface (Puerta and Szkeley, 1994).

Seguindo o mesmo paradigma, outro trabalho que merece ser citado está relacionado a criação do *framework computacional genérico para sistemas de desenvolvimento de interface baseado em modelo* (Puerta and Eisenstein, 1999). Este trabalho define os modelos que compõem o *modelo de interface*. Estes modelos podem ser descritos em uma linguagem que suporte tal descrição. Os modelos são os seguintes:

- modelo de tarefa: descreve a organização dinâmica e estática das tarefas que podem ser realizadas pelos usuários de uma aplicação através de uma interface;
- modelo de domínio: descreve os objetos do domínio, especificando seus atributos, métodos e relacionamentos. Estes são os objetos disponíveis para o usuário ver, acessar e manipular através de uma interface;
- modelo de usuário: este modelo é usado para caracterizar os diferentes tipos de usuários de uma aplicação e especificar suas percepções acerca das tarefas e organização do trabalho, seus privilégios de acesso aos objetos de domínio e suas preferências de modalidades de interação;
- modelo de apresentação: descreve os elementos visuais, tácteis e auditivos que uma interface oferece a seus usuários. Este modelo define as características estáticas da interface.
- modelo de diálogo: este modelo descreve como será o processo de interação entre a interface e o usuário, especificando como o usuário deve agir através dos elementos de interação e como estes devem comunicar a reação da aplicação.

Cada modelo apresenta aspectos específicos de uma interface, e os elementos de cada modelo são classificados, quanto à natureza, em duas categorias (Puerta and Eisenstein, 1999): abstratos, que são todos aqueles que os usuários só podem acessar indiretamente e que são encontrados nos modelos de tarefas, domínio e usuário; e concretos, que são todos aqueles que os usuários podem acessar diretamente e que são encontrados nos modelos de apresentação e diálogo.

O DIUBM propõe o mapeamento entre os modelos abstratos e concretos, e a partir deste mapeamento são derivadas as IUs executáveis. O mapeamento pode

ocorrer de forma automática, semi-automática ou manual. Normalmente, o resultado gerado pela parte automática do processo necessita de ajustes manuais. Exemplos de mapeamentos entre estes modelos são: tarefa-diálogo, tarefa-apresentação, domínio-apresentação, tarefa-usuário, tarefa-domínio, apresentação-diálogo. No caso do mapeamento tarefa-apresentação "a decomposição de tarefas em sub-tarefas pode ser mapeada em hierarquias de componentes e sub-componentes no modelo de apresentação, bem como o agrupamento de sub-tarefas pode ser mapeado em agrupamentos de sub-componentes" (Puerta and Eisenstein, 1999).

Outra abordagem para o DIUBM é o *Framework de Referência Camaleon* (Calvary et al., 2003), que propõe que o desenvolvimento de IU multi-plataforma seja feito em quatro etapas. Em cada etapa são tratados modelos em um nível de abstração diferente. Os níveis de abstração são:

- Tarefas e conceitos: neste nível são descritos as tarefas a serem executadas e os conceitos relacionados ao domínio da aplicação.
- IU abstrata: define o agrupamento de tarefas, e o relacionamento semântico entre as mesmas. Neste nível são usados objetos de interação abstratos (AIO), os quais são independentes de mecanismos de interação, ou seja, não importa se o usuário vai fornecer um dado através de um teclado ou de um comando de voz.
- IU concreta: neste nível os objetos de interação concreta (CIO) são dependentes do contexto de uso. Uma IU concreta descreve se o usuário vai inserir um dado através de uma caixa de seleção ou através de uma caixa de texto.
- IU final: é uma IU que pode ser executada em uma plataforma computacional, por exemplo, uma IU que pode ser executada em um navegador Web.

O mapeamento pode ocorrer entre os modelos e entre diferentes contextos de uso. A figura 2.1 mostra dois contextos de uso em que o usuário pode interagir com o sistema. As setas com sentido para baixo representam transformações no sentido de chegar a objetos mais próximos da interface utilizável pelo usuário. As setas para cima representam o processo de abstração, onde as especificações são levadas para

um nível mais abstrato. As setas na horizontal representam o mapeamento de uma interface de um contexto de uso para outro contexto.

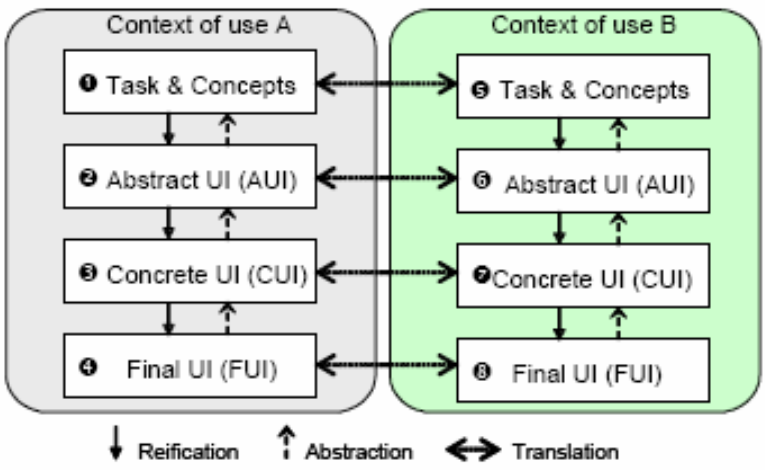


Figura 2.1: Framework Camaleon (Calvary et al., 2003)

O trabalho de (Costa and Leite, 2005) apresenta uma proposta para o DIUBM que segue uma nova perspectiva, baseada na teoria da Engenharia Semiótica.

A Engenharia Semiótica (de Souza, 1993), considera que o sistema computacional é um artefato de meta-comunicação, que conduz uma mensagem unidirecional do designer ao usuário. Esta mensagem comunica a solução que o *designer* concebeu para as possíveis necessidades do usuário. O *designer* e o usuário são interlocutores deste processo comunicativo. Nesta perspectiva, o *designer* concebe o sistema com o objetivo de que este realize o processo comunicativo quando o usuário esta interagindo através da interface de usuário..

Seguindo a perspectiva teórica da engenharia semiótica, (Leite 2006) considera que esta mensagem deve descrever os aspectos funcionais, interativos e comunicativos da interface. Assim, uma IU deve ser descrita com três modelos: de domínio, de interação e de comunicação.

Os modelos de domínio e de interação são independentes de plataforma, já o modelo de comunicação possibilita a descrição dos aspectos com que a IU deve comunicar a funcionalidade ao usuário, de uma forma dependente da plataforma em que a IU será utilizada. O modelo de comunicação não é utilizado para descrever aspectos de implementação na plataforma alvo, mas com este modelo o designer pode definir melhor os aspectos de interatividade entre os usuários e a IU na plataforma

alvo. Com isto, o problema da imprevisiblidade do comportamento, citado anteriormente, pode ser amenizado. Esta abordagem propõe o uso da linguagem IMML para realizar as descrições dos modelos. O capítulo 3 apresentará os detalhes desta abordagem.

2.2 LINGUAGENS DE DESCRIÇÃO DE INTERFACE DE USUÁRIO

As Linguagens de Descrição de Interfaces de Usuários (LDIUs) oferecem um suporte para a descrição de interfaces de usuário segundo o DIUBM. É através destas linguagens que o designer descreve os modelos que compõem uma IU. E, a partir destas descrições, são geradas interfaces de usuário finais (executáveis), através do mapeamento entre modelos. O estudo mostrado em (Souchon and Vanderdockt, 2003) compara algumas LDIUs baseadas em XML. A seguir, são apresentadas quatro LDIUs.

A XIML (*eXtensible Interface Markup Language*) é uma linguagem que dá suporte a todo o ciclo de vida de uma interface de usuário (Puerta and Eisenstein, 2001). Esta linguagem aborda todos os modelos considerados em (Puerta and Eisenstein, 1999). Ela oferece um mecanismo padrão para o intercâmbio de dados entre ferramentas e aplicações desde o design até a operação e manutenção da interface.

A UIML (*User Interface Markup Language*) é uma linguagem que foi projetada em conformidade com a XML e com uma aparência de HTML (Phanouriou 2002). A UIML permite especificação de interfaces para diferentes plataformas numa linguagem única a partir da qual se pode gerar automaticamente o código da interface de usuário final em diferentes linguagens de programação. A especificação de uma interface em UIML descreve a aparência (modelo de apresentação), a interação com o usuário (modelo de diálogo) e como ela deve ser conectada com o núcleo funcional (lógica da aplicação).

A UsiXML (*User Interface Description Language for Context-Sensitive User Interfaces*) (Limbourg and Vanderdonckt, 2004) foi desenvolvida para suportar o desenvolvimento de IU segundo o *Framework Camaleon*. Esta linguagem possui uma coleção de modelos de IU, tais como: tarefa, domínio, modelo AUI (IU abstratas), modelo CUI (IU concretas), contexto, transformação e mapeamento. Todos

os modelos são vistos como modelos que estendem um modelo básico chamado de *UiModel* (User interface Model). O modelo AUI define o espaço de interação e o esquema de navegação entre o espaço de interação e a seleção de objetos abstratos, que são independentes de qualquer maneira de interação ou contexto de uso, por exemplo, voz, realidade virtual e vídeo. O modelo CUI é uma concretização do modelo AUI para um dado contexto de uso, este modelo é composto de objetos concretos de interação como os *widgets* e objetos que permitem a navegação na interface. O trabalho (Limbourg and Vanderdonckt, 2004) apresenta uma abordagem de mapeamento entre os modelos descritos em UsiXML usando transformações de grafos.

A IMML (*Interactive Message Modeling Language*) (Leite 2003) propõe um modelo conceitual para a IU que é dividido em: *modelo de domínio, modelo de interação e modelo de comunicação*. Com estes modelos é possível descrever as interações, os comandos, e as tarefas que o usuário deve desempenhar. Em sua versão original esta linguagem não suportava a descrição de interfaces multi-plataforma, mas o trabalho de (Costa Neto 2005) propôs uma extensão para tornar isto possível.

Existem iniciativas para a descrição de interfaces seguindo a UML. A CUP (*Context-Sensitive User Interface Profile*) é uma extensão da UML 2.0 que dá suporte a modelagem de aplicações interativas sensíveis ao contexto (Van den Bergh 2005). Os modelos considerados nesta linguagem são: o de apresentação; o de tarefa-diálogo; o de contexto; o de apresentação abstrata; e o de apresentação concreta. Os modelos de apresentação são representados pela simbologia de pacotes em UML. As atividades são modeladas com o diagrama de atividades. A Visual IMML é uma versão diagramática da IMML proposta por (Machado 2006), o capítulo 3 (seção 3.5) possui uma descrição desta linguagem.

2.3 DESENVOLVIMENTO DE INTERFACES DE USUÁRIO BASEADO EM COMPONENTES

Na Engenharia de Software existem várias definições para componente. Segundo (D'Souza and Wills, 1999) um componente é uma unidade de software bem definida e que pode ser utilizada em conjunto com outros componentes para formar grandes unidades de software. No desenvolvimento de um componente é importante

se preocupar com o reuso do mesmo, pois um dos maiores problemas na reutilização de software é, justamente, a criação de componentes que possam ser reutilizados em outras aplicações (Pietro-Diaz 1987).

Neste trabalho, consideramos componentes de interfaces de usuário como sendo unidades de software, que seguindo a definição de (D'Souza and Wills, 1999) também, promovam interação entre os usuários e o sistema. A interação pode ocorrer de várias formas tais como: entrada de dados; seleção de opções em menus; listas de checagem; perguntas ao usuário; etc. Nas interfaces no estilo GUI e WUI, exemplos de componentes são: caixa de seleção, botão, caixa de texto, etc. Estes componentes também são chamados de *widgets*.

O objetivo dos componentes de IU é proporcionar um desenvolvimento mais fácil, reduzindo o tempo de trabalho do *designer*. No caso em que é desenvolvida uma IU diferente para cada contexto, o *designer* desenvolve cada versão juntando e ajustando pequenas partes (componentes de IU). Neste caso, se o *designer* fizer uso de uma biblioteca de componentes, a IU pode ser desenvolvida mais rapidamente. Componentes *mais complexos* tendem a facilitar a representação dos aspectos de interação na IU. Isto não é regra geral, pois um componente que ofereça muita interação também pode permitir que o usuário manipule a interface de uma maneira não desejada pelo *designer*. Portanto, é necessário ter cuidados com a granularidade dos componentes (Hopkins 2000). Estamos considerando que um componente de IU *C1* é *mais complexo* que outro componente de IU *C2*, se *C1* oferecer mais recursos de interação do que *C2*. Por exemplo, uma janela que permite ao usuário minimizá-la é um componente é *mais complexo* que outra janela que não possui tal mecanismo.

No caso em que as IUs são geradas, com o uso de ferramentas, a partir de uma especificação abstrata, as descrições realizadas em uma LDIU são mapeadas em componentes de IUs. A ferramenta que processa o código da LDIU faz agrupamentos e ajustes em instâncias de componentes de forma a gerar uma IU final. Neste caso, o uso de componentes mais abstratos tende a facilitar a definição das regras de mapeamento, isto porque o número de ajustes e agrupamentos de instâncias de componentes tende a ser menor. Neste caso, também é necessário tomar cuidados com a granularidade dos componentes.

A utilização de linguagens e modelos padronizados visa possibilitar maior reuso, portabilidade e interoperabilidade de software (Freeman 1980). A falta de recomendação de linguagens e modelos para o desenvolvimento de componentes de IU utilizando as tecnologias DHTML dificulta o uso das mesmas no processo de geração automática de interfaces de usuário. Isto devido às dificuldades encontradas na definição da regras de mapeamento. Se as regras de mapeamento forem criadas usando somente os elementos HTML a granularidade fica muito baixa, o que dificulta o agrupamento dos mesmos no processo de geração automática. Portanto, a criação de linguagens e/ou metodologias padrões para desenvolver componentes de IU baseados nas tecnologias padrão recomendadas pelo W3C facilita o uso destas tecnologias na geração automática de IUs.

Este trabalho propõe o uso de componentes XICL, que são componentes mais *complexos* que os elementos HTML, no processo de geração automática de IUs descritas com as tecnologias DHTML. Isto facilita a definição de regras de mapeamento. Os componentes XICL foram utilizados para o processo de geração de interfaces de usuários, no estilo WUI, a partir de especificações feitas na linguagem IMML.

Este capítulo descreveu os principais conceitos associados ao desenvolvimento de interfaces de usuário para múltiplas plataformas e dispositivos. Descreveu ainda as vantagens de reutilização de componentes nesse processo. O Capítulo 3 apresenta a linguagem IMML.

3 A IMML (Interactive Message Modeling Language)

A IMML (Leite 2003) é uma linguagem de descrição de interface de usuário que possibilita o desenvolvimento de interfaces de usuários multi-plataforma e tem seus fundamentos no desenvolvimento baseado em modelos e na Engenharia Semiótica (de Souza 1993). A IMML suporta a especificação da interface do usuário em um nível abstrato que orienta o design, integrando os aspectos funcionais, interativos e comunicativos da interface e possibilitando a especificação da interface através de três modelos: modelo de domínio, modelo de interação e modelo de comunicação (Leite 2006). Estas características diferenciam a IMML de grande parte das linguagens de descrição de IU existentes que tem seu foco nos aspectos de interação (ou diálogo) ou tarefas.

No capítulo 5, mostraremos como a IMML será usada em conjunto com a XICL para o desenvolvimento de IU para múltiplas plataformas.

A seguir serão descritos, brevemente, cada um dos modelos abordados pela IMML. Os trechos de código IMML utilizados nos exemplos a seguir fazem parte da descrição de uma aplicação bancária. O código completo está no anexo I.

3.1 MODELO DE DOMÍNIO

O modelo de domínio descreve a funcionalidade de um sistema, ou seja, através desse modelo podem ser determinadas quais as funções são oferecidas ao usuário. O modelo de domínio é composto de objetos do domínio (*domain-object*) e funções do domínio (*domain-function*). Os objetos de domínio se referem a registros do banco de dados, arquivos, mensagens eletrônicas, e vários outros objetos que os usuários possam conhecer em seu domínio. Eles são representados em um sistema computacional como estrutura de dados no nível de programação e como texto, ícones

ou *widgets* no nível de interface. Para especificar cada objeto do domínio, o designer deve determinar o nome do objeto e o tipo de representação. Alguns objetos são formados a partir de outros, a figura 3.1 exibe um objeto de domínio que define, para o sistema bancário, um objeto composto chamado "Extrato". Este objeto "Extrato" é fornado por dois outros objetos de domínio: transação e saldo. Cada item utilizado para formar objeto "Extrato" deve ser declarado com o elemento <item>.

```
1. <domain-object name="Extrato" type="composed">
2.     <item domain-object="Transação" />
3.     <item domain-object="Saldo" />
4. </domain-object>
```

Figura 3.1: Objeto de Domínio "Extrato".

Uma função de domínio refere-se a um processo completo executado pelo computador e pode mudar o estado da aplicação, isto é, dos objetos do domínio e seus relacionamentos. Ela pode ser vista como um serviço computacional que realiza um caso de uso. Para especificar uma função de domínio (<*domain-function*>) em IMML o designer deve determinar: o nome da função; os operandos de entrada (<*input-operands*>) e saída (<*output-operands*>), que são objetos de domínio; as pré (<*pre-conditions*>) e pós-condições (<*post-conditions*>); os estados (<*states*>) do sistema e; os controles (<*controls*>) que permitem ao usuário mudanças de estado.

A figrura 3.2 exibe a função de domínio "EmitirExtrato". Os operandos de entrada (linhas 3 a 7) da função são: "Conta", "Agencia", "Senha", "DataInicio" e "DataFim". O operando de saída (linha 10) é o objeto "Extrato". Os estados (linhas 17 a 19) são: "inicial", "consultanto" e "final". A função tem dois controles (linhas 13 e 14), o primeiro controle é o "consultar", e faz uma transição do estado "inicial" para o estado "consultando". O segundo controle faz uma transição do estado "consultando" para o estado "final". Neste ultimo controle a transição é feita de forma automática, esta transição é feita após o processamento da função que faz a consulta do extrato.

A descrição de uma pré-condição <*pre-condition*> é feita com um texto em linguagem natural, pois ainda não foi definida uma sintaxe formal para descrever uma pré-condição em IMML. Esse texto deve ser lido pelo desenvolvedor para auxiliá-lo no processo de implementação da função.

```
1. <domain-function name="EmitirExtrato">
2.     <input-operands>
3.          <item domain-object="Conta" />
4.          <item domain-object="Agencia" />
5.          <item domain-object="Senha" />
6.          <item domain-object="DataInicio" />
7.          <item domain-object="DataFim" />
8.       </input-operands>
9.       <output-operands>
10.         <item domain-object="Extrato" />
11.      </output-operands>
12.      <controls>
13.         <control name="consultar" from-state="inicial" to-
    state="consultando" />
14.         <control automatic="yes" from-state="consultando" to-
    state="final" />
15.      </controls>
16.      <states>
17.         <state name="inicial" />
18.          <state name="consultando" />
19.          <state name="final" />
20.      </states>
21.    </domain-function>
```

Figura 3.2: Função de domínio "EmitirExtrato".

3.2 MODELO DE INTERAÇÃO

O modelo de interação representa o processo de interação entre o usuário e o sistema, através da interface, definindo as ações do usuário para comandar as funções de domínio. Os elementos básicos deste modelo são: comando de função, resultado de função, tarefas, interação básica e estruturas de interação.

Uma interação básica se refere a uma determinada interação do usuário com a interface, através de uma ação, como clicar num botão, selecionar um elemento numa lista, digitar um texto ou número e visualizar um resultado, por exemplo. A IMML possui as seguintes interações básicas: entrar com informação (*enter-information*), selecionar informação (*select-information*), ativar controles (*activate*) e visualizar informação (*perceive-information*).

As estruturas de interação são as responsáveis por organizar os comandos e resultados de função dentro de uma tarefa, e as interações básicas dentro dos comandos e resultados de função. As estruturas podem estar aninhadas com outras estruturas.

A IMML provê as seguintes estruturas de interação: a estrutura *sequence* especifica que o usuário deve realizar cada interação de uma forma ordenada. A

estrutura *repeat* é usada quando o usuário deve realizar a interação mais de uma vez. A estrutura *select* permite ao usuário escolher uma de diversas interações. A estrutura *combine* é usada quando duas ou mais interações têm um tipo de dependência entre si. Por último, a estrutura *join* é usada para agrupar interações que têm algum relacionamento, mas não requerem uma ordem específica para serem realizadas.

Um comando de função é usado para inserir informação e para controlar a execução de funções do domínio. Ele permite ao usuário, o controle da execução da função. É importante informar que cada comando de função deve estar associado com uma função de domínio. Uma função de domínio pode estar associada a vários comandos de função. Um comando de função é uma composição de um conjunto de interações básicas de uma forma estruturada, onde uma interação básica se refere a uma ação do usuário ao interagir com um *widget* de interface, por exemplo. A figura 3.3 exibe o comando de função para a função de domínio "EmitirExtrato". Os elementos de interação básica estão estruturados com *select, sequence,* e *join*. Existe um elemento <select> (linhas 2 a 22) que engloba todos os outros elementos, este elemento define que o usuário pode executar uma seqüência (*sequence* definida da linha 3 a 20) ou não executar a função (*go* definido na linha 21). Se o usuário escolher por executar a seqüência, então, primeiro o usuário deve entrar com três informações: "Conta", "Agencia", "Saldo". Depois o usuário informa o período do extrato, isto pode ser informado de duas formas, o usuário escolhe (*select*) a maneira como quer informar o período. A primeira opção é informar o mês do extrato (*join* definido da linha 9 a 12), e a segunda é informar a data de início e a data de fim do extrato (*join* definido da 13 a 17).

```
1. <function-command name="Extrato" domain-
   function="EmitirExtrato">
2.   <select>
3.     <sequence>
4.       <enter-information domain-object="Conta" />
5.       <enter-information domain-object="Agencia" />
6.       <enter-information domain-object="Senha" />
7.
8.       <select>
9.         <join>
10.          <perceive-information>Escolha o mes que deseja
               consultar extrato
11.          </perceive-information>
12.          <enter-information domain-object="Mes" />
13.        </join>
14.        <join>
15.          <perceive-information>Entre com a data de inicio e
   final, ou somente a data de inicio
16.          </perceive-information>
17.          <enter-information domain-object="DataInicio" />
18.          <enter-information domain-object="DataFim" />
19.        </join>
20.      </select>
21.      <activate control="Consultar" />
22.    </sequence>
23.    <go direction="away" />
24.  </select>
25.</function-command>
```

Figura 3.3: Comando de função para a função de domínio "EmitirExtrato".

O Resultado de função é a saída de funções de domínio, mensagens de erro ou alertas. Responsável por provê uma resposta para o usuário sobre o processo de interação ocorrida. A figura 3.4 exibe o resultado de função "SaidaExtrato" para a função de domínio "EmitirExtrato".

```
1. <function-result name="SaidaExtrato" domain-
   function="EmitirExtrato">
2.          <perceive-information domain-object="Extrato" />
3. </function-result>
```

Figura 3.4: Resultado de função "SaidaExtrato".

Uma tarefa (*task*) é uma composição estruturada de comandos de função, resultados de função e/ou outras tarefas. Esta composição também é organizada através das estruturas de interação. O papel da tarefa é organizar de forma estruturada, através de estruturas de controle, um conjunto de comandos necessários para organizar as atividades de usuário para atingir uma meta. A meta, embora não especificada de forma explícita na IMML, pode ser inferida de acordo com os estados inicial e final das funções associadas aos comandos que compõem a tarefa. A Figura 3.5 exibe a tarefa "EmitirExtrato", que possui a estruturas de interação *sequence*, o comando de função "Extrato" e o resultado de função "SaidaExtrato".

```
1. <task name="EmitirExtrato">
2.          <sequence>
3.               <do function-command="Extrato" />
4.               <do function-result="SaidaExtrato" />
5.          </sequence>
6. </task>
```

Figura 3.5: Tarefa "EmitirExtrato".

3.3 MODELO DE COMUNICAÇÃO

O modelo de comunicação foi adicionado a IMML pelo trabalho de Costa Neto (2005) e se refere à mensagem global que o *designer* constrói para comunicar o modelo de domínio e de interação para o usuário, considerando aspectos sobre a plataforma alvo. Ele deverá determinar a comunicabilidade do sistema. A comunicabilidade "é a qualidade que determina se o sistema comunica eficientemente e efetivamente aos usuários as intenções pretendidas pelo designer" (Prates 2000). Este aspecto é de fundamental importância para um sistema interativo, seguindo a visão da Engenharia Semiótica.

Os elementos do modelo de comunicação contêm três elementos principais: o meio de comunicação através do qual ocorre o processo metacomunicativo (em inglês, foi originalmente denominado *display medium*); os Signos de Interfaces que são os elementos utilizados no processo comunicativo, compondo a mensagem interativa enviada ao usuário; e as ferramentas de interação, através do qual o usuário interage com os signos de interface veiculados no meio de comunicação. Exemplos de signos de interfaces são os widgets. Usando as ferramentas de interação, o usuário pode interagir com estes signos. Assim, além de conduzirem mensagens ao usuário, os signos de interface também permitem ao usuário a interação com a mesma.

Os Signos de Interface são categorizados em signos de interação básica e signos de organização ou composição. Os signos de organização são: quadro; painel de comando; área de exibição e; ambiente de tarefa.

O quadro (*frame*) é um signo de interface que serve para organizar a apresentação de outros signos. Os signos de interação que compõem o quadro irão permitir ao usuário a interação com a interface, realizando os comandos de função e obtendo os seus resultados.

A forma e a ordem de apresentação dos signos contidos no quadro deverão ser determinadas através das transições, que descrevem também quando cada signo deverá ser apresentado ou ocultado. A Figura 3.6 exibe o *frame* "grupoSuperior", que possui quatro botões e a interação de cada botão está relacionada a um *frame*.

```
1. <frame name="grupoSuperior" title="Banco Rico"
   orientation="horizontal"    align="left">
2.     <push-button label="Saldo" show="frameSaldo" />
3.     <push-button label="Extrato" show="frameExtrato" />
4.     <push-button label="Pagamento" show="framePagamento" />
5.     <push-button label="Pagamento Com Saldo
   Antes"show="framePagamentoComSaldo" />
6. </frame>
```

Figura 3.6: Frame "grupoSuperior".

Um painel de comando (*command-panel*) organiza e apresenta os signos interativos que serão utilizados para realizar um comando de função. A sua organização é realizada através das estruturas de apresentação e de interação. Desta forma, o painel de comando deve estar associado a um comando de função especificado no modelo de interação. A figura 3.7 exibe um painel de comando para a função de comando "Saldo", que possui três caixas para a edição de texto (linhas 2, 3 e 4) e dois botões (linhas 6 e 7) . O primeiro botão está associado a um comando de transição (transition) para uma área de exibição (display-area). O segundo é utilizado para cancelar a execução da função "Saldo".

```
1. <command-panel function-command="Saldo" name="frameSaldo"
   title="Banco Rico" orientation="vertical" align="left">
2.     <edit-box label="Agencia" domain-object="Agencia" />
       <edit-box label="Conta" domain-object="Conta" />
3.     <edit-box label="Senha" domain-object="Senha" />
4.     <group orientation="horizontal" align="center">
5.          <push-button label="Consultar"
   control="Consultar"transition="MostraSaldo"
   target="grupoInferior" />
6.          <push-button label="Cancelar" hide="this" />
7.     </group>
8. </command-panel>
```

Figura 3.7: Painel de Comando para a função de comando "Saldo".

A área de exibição (*display-area*) é o local onde os signos de interface devem ser inseridos para comunicar ao usuário os resultados de uma função de domínio. A área de exibição também pode ser parte de um painel de comando de forma a permitir a união de signos que representam resultados de função com signos de acionamento de controle de comando. Em outras palavras, ao visualizar um resultado de uma função o usuário pode interagir com um botão para solicitar a realização de outra

função de domínio. A figura 3.8 ilustra a área de exibição para o resultado de função "SaidaSaldo". Foram usados quatro elementos <text> para exibir objetos de domínio e um elemento <push-button> que declara que a interface deve mostrar um botão.

```
1. <display-area function-result="SaidaSaldo" name="MostraSaldo"
   orientation="vertical" align="left">
2.     <text label="Data" domain-object="Data" />
3.     <text label="Agencia" domain-object="Agencia" />
4.     <text label="Conta" domain-object="Conta" />
5.     <text label="Saldo" domain-object="Saldo" />
6.     <push-button label="Sair" hide="this" />
7. </display-area>
```

Figura 3.8: Área de exibição do resultado de função "SaídaSaldo".

O ambiente de tarefa (*task-environment*) também é um signo de composição, ou seja, é utilizado para agregar signos, mas diferente dos outros, ele é usado apenas para especificar o conjunto de quadros necessários para realização de uma tarefa em uma plataforma. Desta forma, ele tem que estar associado à sua respectiva tarefa especificada no modelo de interação e a uma plataforma. O comportamento do ambiente de tarefa deve ser especificado através das transições entre os quadros que ele agrega.

A figura 3.9 exibe o ambiente de tarefa "ExtratoNoTerminal", contendo *frames*, painéis de comando e áreas de exibição.

```
1. <task-environment name="ExtratoNoTerminal" task="Transacao
   Bancaria" platform="Terminal">
2.   <frame>
3.     <frame name="grupoSuperior" title="Banco Rico"
   orientation="horizontal" align="left">
4.              ...
5.     </frame>
6.     <frame name="grupoInferior" />
7.   </frame>
8.
9.   <command-panel function-command="Saldo" name="frameSaldo"
   title="Banco Rico" orientation="vertical" align="left">
10.             ...
11.    </command-panel>
12.
13.    <display-area function-result="SaidaSaldo"
   name="MostraSaldo"   orientation="vertical" align="left">
14.       ...
15.    </display-area>
16.
17.    <command-panel function-command="Extrato"
   name="frameExtrato"  orientation="vertical" align="left">
18.       ...
19.    </command-panel>
20.
21.    <display-area function-result="SaidaExtrato"
   name="MostraExtrato"  orientation="vertical" align="left">
22.       ...
23.    </display-area>
24.
25.    <command-panel function-command="Pagamento"
   name="framePagamento" orientation="vertical" align="left">
26.       ...
27.    </command-panel>
28.
29.    <display-area function-result="SaidaPagamento"
   name="MostraPagamento" orientation="vertical" align="left">
30.       ...
31.    </display-area>
32.
33.     <command-panel function-command="PagamentoComSaldoAntes">
34.       ...
35.     </command-panel>
36.
37.     <display-area function-result="SaidaPagamentoComSaldo"
   name="MostraPagamentoComSaldo" orientation="vertical"
   align="left">
38.        ...
39.     </display-area>
40.
41.  </task-environment>
```

Figura 3.9: Ambiente de tarefa “ExtratoNoTerminal”

3.4 A IMML NO DESENVOLVIMENTO DE IU PARA MULTI-PLATAFORMAS

Através do modelo de domínio é possível descrever aspectos do domínio da aplicação de forma independente da plataforma em que a interface será utilizada. Uma função de domínio pode ser especificada de forma independente da plataforma que será executada. Um dado do tipo data, por exemplo, pode ser editado como data tanto através de um celular como através de um desktop. Na geração da IU final (executável na plataforma alvo) as informações contidas no modelo de domínio auxiliam na formatação dos dados mostrados e coletados do usuário. Este modelo define se um dado chamado *dia*, por exemplo, é uma cadeia de caracteres, um número inteiro ou mesmo um dado do tipo data. Da mesma forma, se a interface tem uma caixa de texto para que o usuário informe um valor monetário, a interface deve sugerir o formato do dado a ser inserido e evitar que o usuário não insira uma data, por exemplo.

O modelo de interação permite especificar a maneira como o usuário deve interagir para executar uma tarefa, sem considerar o dispositivo usado para a interação. Os elementos básicos de interação como *enter-information* e *perceive-information* representam ações do usuário que podem ser implementados em diversos dispositivos. A maneira com que o modelo permite especificar uma seqüência de passos para realizar uma tarefa também depende do dispositivo de interação. Na geração de IUs finais, este modelo fornece informações para implementar mecanismos que controlam os passos que o usuário deve executar para realizar a tarefa. As estruturas de interação *repeat*, *select*, *combine*, *join* e *sequence* definem a seqüência a ser seguida pelo usuário.

Diferentemente dos modelos de interação e do modelo de domínio, o modelo de comunicação é dependente da plataforma que o usuário usa para interagir com a interface, pois considera aspectos como largura de tela e quantidade de telas necessárias para realizar uma tarefa. O modelo de comunicação define aspectos como a maneira que a IU deve permitir que o usuário insira os dados necessários, e como a IU vai apresentar as informações ao usuário. Com o modelo de comunicação, a ferramenta que gera a IU final, monta a estrutura visual usando componentes de IU. Estes componentes de IU são escolhidos a partir de regras de mapeamento, que definem como os elementos do modelo de comunicação devem ser implementados na

IU final. Uma regra de mapeamento pode definir que o elemento <edit-box> deve ser implementado em uma IU Java com o componente JTextField, da biblioteca Swing, por exemplo.

Para gerar IUs finais a partir de especificações IMML, é necessário definir um conjunto de regras de mapeamento para cada plataforma. As regras de mapeamento podem ser descritas em (i) um documento específico para este propósito, ou (ii) podem ser definidas dentro da própria ferramenta que gera IUs finais. O primeiro caso é mais flexível, mas é preciso definir uma linguagem que possibilite a descrição das regras de mapeamento. A figura 3.10 representa o processo de geração de IU a partir de descrições IMML, com o uso de regras de mapeamento. Para um modelo de interação e um modelo de dados foram descritos dois modelos de comunicação: um para o desktop; e outro para celular. Para cada plataforma foi definido um conjunto de regras de mapeamento diferente.

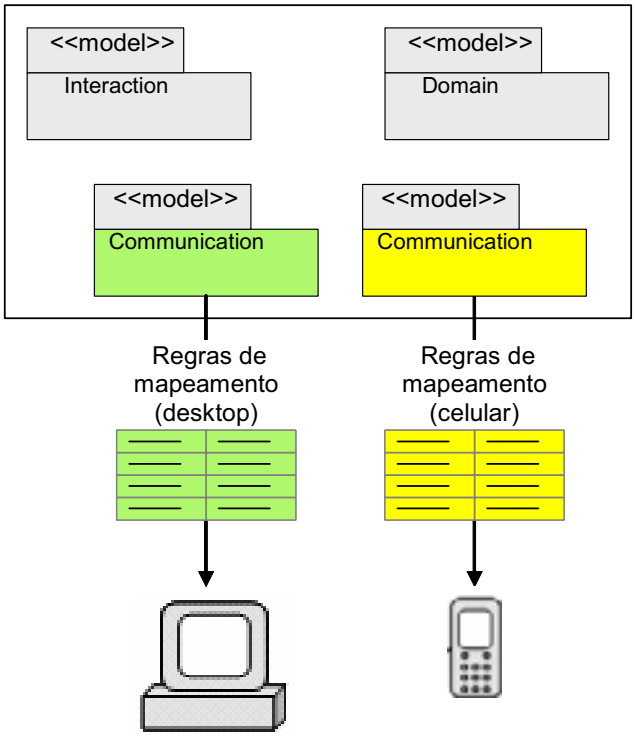


Figura 3.10: Geração de IU a partir de especificações IMML

3.5 TRABALHOS RELACIONADOS À IMML

Além deste trabalho, outros trabalhos relacionados à IMML, foram e, estão sendo desenvolvidos. À medida que os trabalhos vão sendo desenvolvidos são sugeridos alguns ajustes na IMML.

Foram realizados experimentos (Fonsêca 2005) para comparar a IMML com as técnicas de especificação e modelagem centradas no usuário GOMS (Card et al. 1983) e Storyboarding (Landay e Myers 1996). Estes experimentos identificaram algumas vantagens e desvantagens da IMML frente às técnicas GOMS e Storyboarding o que possibilitou melhorias na IMML.

A VISUAL IMML (Machado 2006) é uma extensão da UML baseada da IMML para a modelagem visual (diagramática). Portanto, os modelos considerados nesta linguagem são os mesmos da IMML. A Visual IMML propõe um conjunto de novos elementos de modelagem (estereótipos) elaborados para a especificação e documentação de interfaces de usuário, considerando os aspectos de comunicação, interação e de funcionalidade de forma integrada. Encontra-se em desenvolvimento uma ferramenta para a modelagem de IU em VISUAL IMML. O objetivo é permitir que o *designer* modele a interface em VISUAL IMML e a ferramenta gere código em IMML.

Foi desenvolvida uma ontologia com a descrição semântica da IMML (Lira 2006). A ontologia encontra-se escrita em OWL. Esta ontologia pode ser utilizada para a validação semântica de interfaces descritas em IMML. Isto pode melhorar o processo de geração de interfaces de usuário a partir de especificações IMML.

O BRIDGE (Interface Design Generator Environment) é uma ferramenta que desenvolvida para gerar IU em Java a partir de descrições feitas em IMML (Silva 2007). Esta ferramenta permite a interação do *designer* no processo geração da IU final.

Este capítulo apresentou a linguagem IMML que é utiliza para a especificação de interfaces para múltiplas plataformas. O capitulo 4 apresenta a linguagem XICL, uma linguagem para descrever componentes, que foram usados no processo de geração de IUs a partir de especificações IMML.

4 A Linguagem XICL

A XICL é uma linguagem de marcação para descrição de componentes de interfaces de usuário no estilo WUI (Sousa 2004). O código XICL pode conter elementos próprios da linguagem XICL e código DHTML, que é composto por: HTML, JavaScript e CSS (Cascade Style Sheets). No código JavaScript pode ser feito uso da API DOM (Document Object Model) (Hégaret 2004). Com a XICL também é possível descrever interfaces de usuário, fazendo o uso dos componentes XICL.

A descrição de um componente XICL é feita pelo agrupamento de elementos HTML e/ou instâncias de outros componentes XICL. Isto permite a extensão dos componentes.

Uma interface de usuário XICL é composta pelo agrupamento de elementos HTML e componentes XICL. Os componentes utilizados na descrição de uma IU XICL estão descritos dentro do mesmo documento, ou armazenados em uma biblioteca, que também é um documento XICL.

Um documento XICL é um arquivo XML que contém a descrição de (zero ou) uma interface de usuário XICL e a descrição de (zero ou) vários componentes de interface de usuário XICL.

A figura 4.1 mostra o relacionamento entre estes conceitos da linguagem XICL. Os relacionamentos representados por seta representam o uso de algo já definido. Os outros relacionamentos, representados pelo símbolo de agrupamento da UML, representam a definição de algo novo. Por exemplo, um novo componente deve ser definido dentro de um documento XICL. Um documento XICL pode fazer referência a um segundo documento, este último funciona como biblioteca para o primeiro.

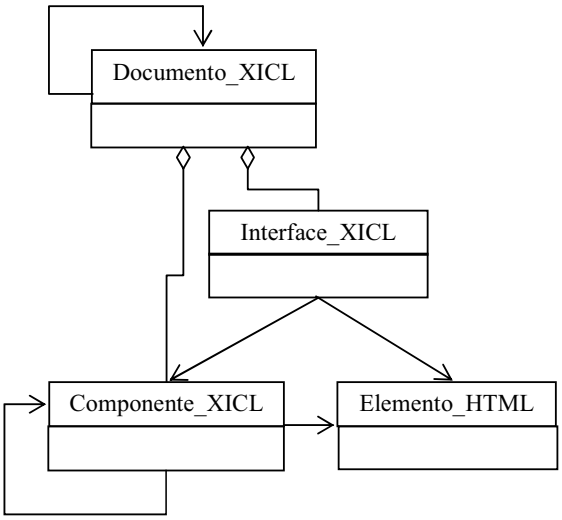


Figura 4.1: Relação entre os principais conceitos da linguagem XICL.

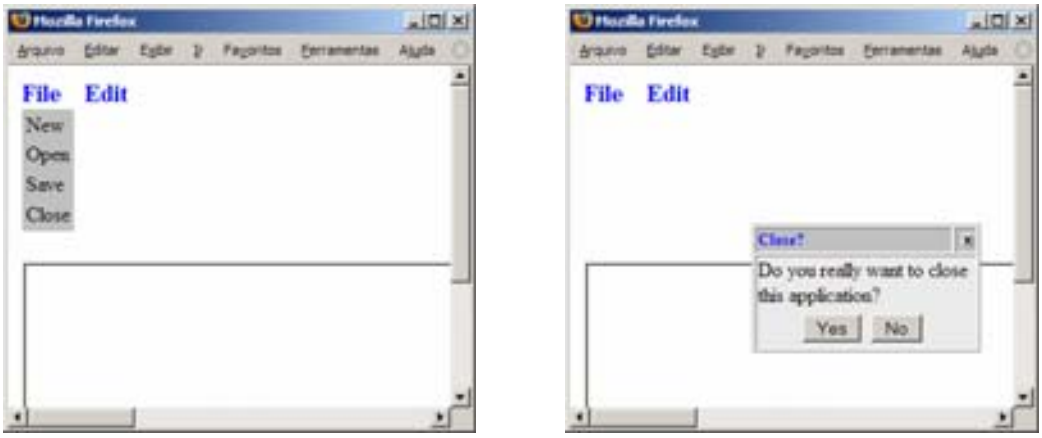
A seguir, será apresentado como a descrição de uma IU XICL deve ser realizada, fazendo o uso de componentes XICL. Depois será apresentado o processo de descrição de componentes XICL.

4.1 DESCRIÇÃO DE INTERFACE DE USUÁRIO EM XICL

A figura 4.2 mostra uma IU em DHTML de um editor de texto simples, executada em um navegador Web, em um desktop. Esta IU foi gerada a partir de uma descrição IU XICL. A figura 4.2a mostra a interface com seus elementos em um primeiro momento, composta por uma barra de menus com dois menus (**File** e **Edit**) e uma área para editar texto. Cada menu tem um conjunto de opções. A figura 4.2b mostra a interface após o usuário escolher a opção **close** do menu **File**, onde é mostrada uma caixa de diálogo para o usuário confirmar (ou cancelar) a saída da aplicação.

A figura 4.3 mostra o código XICL[2] resumido com a descrição da interface apresentada na Figura 4.2. O código completo está no anexo II.

2 Para auxiliar o desenvolvimento de componentes e IUs em XICL um documento com a especificação da linguagem e exemplos está disponível no endereço Web www.dimap.ufrn.br/~jair/XICL.



(a) (b)

Figura 4.2: Interface em DHTML gerada de uma descrição XICL

```
1. <?xml version="1.0" encoding="ISO-8859-1"?>
2. <XICL version="1.0" >
3.
4.  <IMPORT url="lib1.xml"/>
5.
6.  <INTERFACE >
7.
8.    <menuBar>
9.       <Menu menuCaption="File" id="File">
10.        <menuOption caption="New"  onchoose="new();" menuCaption="File"/>
11.        <menuOption caption="Open" onchoose="open();" menuCaption="File" />
12.        <menuOption caption="Save" onchoose="save();"  menuCaption="File"/>
13.        <menuOption caption="Close" onchoose="mes1.show();"
menuCaption="File"/>
14.      </Menu>
15.      <Menu menuCaption="Edit" id="Edit">
16.        <menuOption caption="Copy" onchoose="copy();" menuCaption="Edit" />
17.        <menuOption caption="Paste" onchoose="paste();" menuCaption="Edit"
/>
18.     </Menu>
19.   </menuBar>
20.
21.   <ConfirmBox id="mes1" title="Close?" onConfirm="close()" > Do you really
want to close this application? </ConfirmBox>
22.
23.   <textarea rows="12" cols="60">
24.
25.   </textarea>
26.
27.   <SCRIPT>
28.    ....
29.   </SCRIPT>
30.  </INTERFACE>
31. </XICL>
```

Figura 4.3: Código da interface descrita em XICL

4.1.1 Elementos da XICL para a descrição de IUs XICL

Os elementos específicos da linguagem XICL utilizados para descrever uma interface são:

4.1.1.1 Elemento XICL

O elemento **XICL** é o elemento raiz do documento. Todos os elementos devem estar dentro deste elemento. Ele delimita a descrição da interface e dos componentes.

4.1.1.2 Elemento IMPORT

O elemento **IMPORT** é utilizado para importar uma biblioteca de componentes, que está em outro documento XICL. O atributo **url** é utilizado para definir o caminho do arquivo que contém a biblioteca. Na descrição de uma interface podem ser importadas várias bibliotecas de componentes XICL.

4.1.1.3 Elemento INTERFACE

O elemento **INTERFACE** é utilizado para definir o escopo da descrição de uma IU. Um documento XICL só pode ter, no máximo, um elemento **INTERFACE,** como citado anteriormente.

4.1.1.4 Elemento SCRIPT

Este elemento é utilizado para definir funções que possam controlar o comportamento da interface e processar as requisições dos usuários. Este elemento também pode ser usado para definir funções para um componente.

4.1.2 Um exemplo de descrição de uma IU XICL

Todos os elementos apresentados acima foram utilizados na descrição do código apresentado na figura 4.3. Na linha 4 ocorre a importação da biblioteca de componentes que está no arquivo **lib1.xml**. O conteúdo desta biblioteca está descrito no anexo III.

Para descrever o menu da interface foram utilizados três componentes XICL: **Menu**, **MenuBar** e **MenuOption** (figura 4.3 linhas 8, 9 e 10 respectivamente). Para a caixa de diálogo foi utilizado o **ConfirmBox** (linha 21).

A parte da interface onde o usuário escreve o texto foi descrita com o elemento HTML **textarea**. Os elementos HTML e os eventos definidos em HTML são usados de acordo com a sintaxe da linguagem HTML. Considerando que se trata de um arquivo XML, é preciso respeitar as regras de um documento XML. Portanto, a descrição dos

elementos HTML deve ser feita como em XHTML (eXtensible HTML). Em XHTML todos os elementos HTML devem ser declarados com letras minúsculas. Assim, nas descrições XICL deve ser inserido código XHMTL, mas este texto sempre cita esta linguagem como HTML, já que XHTML se trata de uma adequação de HTML às regras XML.

Na definição da caixa de diálogo, utilizando o componente **ConfirmBox** (Figura 4.3, linha 21) está declarado o atributo **id**, que define um identificador para distinguir esta *instância* das outras instâncias. Isto é necessário, por exemplo, para invocar os métodos para uma instância específica, como no caso do método `show()` da instância identificada por **mes1**. O atributo **title** define um título para a caixa de diálogo.

O comportamento da interface é definido associando-se funções aos eventos. Neste exemplo, ao evento **onChoose** de cada componente **MenuOption** (linhas 10 a 13) está associada uma função que é executada quando o usuário escolhe uma opção. Estas funções processam as requisições que os usuários fazem ao interagir com a interface e são específicas desta interface, ou do domínio para o qual a interface está desenvolvida. As definições destas funções (`new()`, `open()`, `save()`, etc.) estão feitas dentro do escopo do elemento SCRIPT (linha 27). O atributo **onConfirm** está sendo utilizado para declarar uma função que será executada quando o evento onConfirm ocorrer, ou seja, quando o usuário clicar no botão **YES** da janela do componente. Neste caso foi declarada a função `close()`, descrita no elemento SCRIPT, que deve fechar a aplicação. Em relação aos elementos HTML é possível fazer uso de todos os eventos que o elemento possui, definidos na linguagem HTML.

4.2 DESENVOLVIMENTO DE COMPONENTES EM XICL

Um componente XICL é composto de quatro partes: (1) *propriedades*; (2) *estrutura*; (3) *métodos* (funções) e; (4) *eventos*. Os elementos XICL utilizados para codificar estas partes estão representados na figura 3.5. O elemento PROPERTY representa uma propriedade; o elemento STRUCTURE representa a estrutura visual do componente; o elemento SCRIPT é usado para definir o escopo das definições dos métodos de um componente, o elemento METHOD é utilizado para definir os métodos públicos do componente; o elemento EVENT representa um evento do componente.

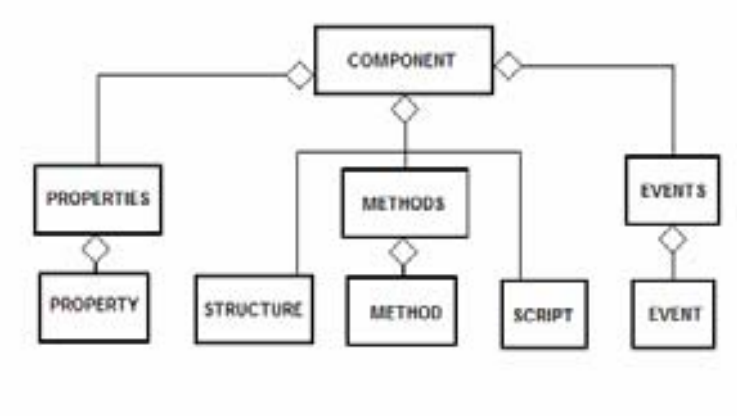


Figura 4.4: Elementos de um componente XICL

4.2.1 Elementos para a descrição de componente XICL

Os elementos usados na descrição de componentes XICL são os seguintes:

4.2.1.1 Elemento COMPONENT

Este elemento é utilizado para definir um novo componente. O atributo **name** é utilizado para *registrar* um *nome* para o componente. Uma boa prática, para melhorar a visualização do código XICL, é compor os nomes dos componentes XICL com letras maiúsculas e minúsculas. Assim teremos: os elementos específicos da XICL sempre em letras maiúsculas; os elementos HTML sempre com letras minúsculas; e os componentes criados em XICL com letras nos dois formatos. O atributo **extends** é utilizado para declarar que um componente XICL estende outro.

4.2.1.2 Elemento PROPERTY

Este elemento é utilizado para declarar uma propriedade (atributo) do componente que está sendo descrito. O atributo **name** é usado para definir um nome para a propriedade. O atributo **defaultValue** define um valor padrão para a propriedade. O atributo **dataType** define um tipo de dado para a propriedade. O atributo **ref** é utilizado para reaproveitar a definição de uma propriedade já definida para outro componente. Por exemplo, o componente **ConfirmBox** possui uma propriedade chamada **title**, esta propriedade pode ser usada na definição de outro componente somente declarando o código <PROPERTY ref="title">.

4.2.1.3 Elemento PROPERTIES

O elemento **PROPERTIES** é utilizado para agrupar os elementos PROPERTY de um componente.

4.2.1.4 Elemento STRUCTURE

O elemento **STRUCTURE** é utilizado para fazer a descrição da estrutura visual do componente, a partir de elementos da linguagem HTML e/ou de componentes XICL.

4.2.1.5 Elemento COMPOSITION

Em alguns casos, para que a estrutura do componente fique com uma boa aparência, é necessário que toda vez que um componente for instanciado sejam colocados outros componentes e/ou elementos HTML em volta desta instância. Para definir esta estrutura que se repete é utilizado o elemento COMPOSITION. O código da figura 4.5, mostra um exemplo com este elemento e, descreve que toda instância do componente **menuOption** será colocada dentro de um elemento HTML **td**, e este estará dentro de um elemento **tr**. Portanto, o elemento COMPOSITION define o contexto que deve se repetir a cada ocorrência do elemento HTML, e/ou componente XICL, que estiver definido no atributo **ref** do elemento COMPONENT.

```
1.       . . .
2.    <STRUCTURE>
3.             . . .
4.       <COMPOSITION>
5.            <tr>
6.                  <td>
7.                        <COMPONENT ref="menuOption"/>
8.                  </td>
9.             </tr>
10.    </COMPOSITION>
11.       . . .
12.     </STRUCTURE >
13.       . . .
```

Figura 4.5: Uso do elemento COMPOSITION

4.2.1.6 Elemento EVENT

Os elementos HTML possuem alguns eventos pré-definidos na linguagem. Por exemplo, muitos elementos possuem os eventos onClick, onMouseOver. Estes eventos são interpretados pelo próprio navegador Web. A linguagem XICL permite a definição de novos eventos que podem ser associados aos componentes XICL. Estes eventos, que não estão definidos em HTML, são descritos utilizando o elemento **EVENT**. O atributo

name é utilizado para definir o nome do evento. O atributo **trigged** define quando o evento ocorre. Por exemplo, o código <EVENT name="onConfirm" trigged="bOk.onClick; $onConfirm;" /> define o evento nomeado onConfirm, que ocorre depois da ocorrência do evento onClick de uma instância nomeada de bOk.

4.2.1.7 Elemento EVENTS

O elemento EVENTS é utilizado para agrupar os elementos EVENT de um componente.

4.2.1.8 Elemento SCRIPT

Este elemento também é utilizado para descrever as funções de um componente.

4.2.1.9 Elemento METHOD

Um componente pode ter várias funções, algumas são criadas para uso interno do componente (como métodos privados) e outras para serem invocadas externamente (como os métodos públicos). No entanto, em algumas linguagens de script, como em JavaScript, não existe encapsulamento. Nesse caso, o elemento METHOD é utilizado apenas para declarar os métodos públicos do componente. O atributo **name** é usado para declarar o nome do método que pode ser invocado externamente.

4.2.1.10 Elemento METHODS

O elemento METHODS é utilizado para agrupar os elementos METHOD de um componente.

4.2.2 Exemplo de descrição de um componente XICL

Para exemplificarmos a descrição de um componente, vamos utilizar como exemplo a descrição do componente **ConfirmBox**, que foi utilizado para descrever a interface mostrada na figura 4.2. A declaração do uso do componente foi feita no código que descreve a interface, mostrado na figura 4.3 (linha 21). A figura 4.6 mostra o código resumido da descrição deste componente. O código completo do componente está no anexo III.

O componente **ConfirmBox** é uma extensão do componente **Window**. Portanto, o componente **ConfirmBox** herda as características do componente **Window**, e estende a estrutura e os eventos do componente **Window**. O uso de um componente XICL é declarado da mesma forma que um elemento HTML, simplesmente colocando o nome (*name*) entre as tags de marcação, como **<**ConfirmBox ...**>**. Para evitar conflitos de nomes é importante definir o nome (*name*) de um novo componente diferente dos nomes dos elementos HTML, ou então, fazer o uso de *namespaces* (Bray et. al., 2006). Neste caso, o uso de um componente XICL é declarado como <xicl:NomeDoComponente>.

O código da figura 4.6 (linha 7) mostra a definição da propriedade **title**. O valor da propriedade é definido quando o componente é utilizado em uma interface (<ConfirmBox title="Close">). O uso do valor da propriedade deve ser definido na descrição estrutural do componente, com o uso do caractere **$** seguido pelo nome da propriedade. Na linha 27 (e 28) está sendo feito o uso da propriedade **id**, que é uma propriedade herdada do componente **Window**.

Para permitir que o componente seja estendido, ou para que o componente agrupe outros componentes XICL e/ou elementos HTML, é preciso definir um elemento, dentro da estrutura, para agrupar os componentes e/ou elementos. No exemplo usado, o elemento que recebe a extensão é o elemento **td** (figura 4.6, linha 13). Isto é definido fazendo uso do elemento **COMPONENT** com o uso do atributo **ref**. O valor deste atributo define os elementos HTML e/ou componentes XICL que podem ser colocados na estrutura do componente que está sendo estendido. Na linha 14 (figura 4.6) a declaração <COMPONENT ref="ANY"> expressa que qualquer (ANY) componente XICL e/ou elemento HTML pode ser inserido na estrutura deste componente. No exemplo da figura 4.3 (linha 21) o componente ConfirmBox recebe um texto, então na compilação este texto vai ser inserido como filho do elemento **td** declarado na linha 13.

```
1. <XICL>
2.   <IMPORT href="lib2.xml">
3.
4.   <COMPONENT name="ConfirmBox"  extends="window">
5.
6.   <PROPERTIES>
7.        <PROPERTY name="title" dataType="string" />
8.   </PROPERTIES>
9.
10.  <STRUCTURE>
11.       <table width="100%" border="0">
12.         <tr>
13.         <td width="150"  >
14.              <COMPONENT ref="any" >
15.             </td>
16.         </tr>
17.         <tr>
18.            <td align="center">
19.            <input type="button" name="bOk" value="Yes"  />
20.                 <input type="button" name="bCanc" value="No" />
21.         </td>
22.         </tr>
23.    </table>
24.  </STRUCTURE>
25.
26.<EVENTS>
27.  <EVENT name="onConfirm" trigged="bOk.onclick; $id.hide(); $onConfirm;" />
28.  <EVENT name="onCancel" trigged="bCanc.onclick; $id.hide(); $onCancel;" />
29.</EVENTS>
30.
31.</COMPONENT>
32.</XICL >
```

Figura 4.6: Descrição de um componente de interface de usuário

Foram definidos dois eventos para o componente ConfirmBox. O evento **onConfirm** (linha 27) ocorre depois da ocorrência do onClick do elemento que recebeu o nome **bOk** (linha 19), a função `hide()` deve ser executada antes do tratamento do evento, e depois é executado o que for declarado no evento onConfirm, no uso do componente. Na figura 4.3, linha 21, está declarado que no evento **onConfirm** será executado a função **close**.

4.3 GERAÇÃO DE IU PARA DIVERSOS DISPOSITIVOS

Os componentes descritos até este ponto foram desenvolvidos para IUs para desktop. A linguagem XICL também pode ser utilizada para desenvolver interfaces de usuário para diversos dispositivos, desde que estas interfaces tenham por plataforma computacional navegadores Web.

Atualmente já é possível interagir com interfaces de usuário no estilo WUI através de aparelhos celulares. Isto é possível pelo fato de já existir navegador Web para

celulares. O Opera Mini (Opera 2006) é um exemplo de um navegador que foi desenvolvido para rodar em aparelhos celulares. Este navegador interpreta código HTML, e algumas funções básicas de JavaScript. Para desenvolver uma IU equivalente à mostrada na figura 4.2 para um celular, por exemplo, primeiro é necessário verificar as limitações da plataforma do celular. A figura 4.7 mostra uma IU, equivalente à mostrada na figura 4.2, sendo executada no simulador do Opera Mini.

(a) (b)

Figura 4.7: Interface executada em um simulador de um navegador Web

A figura 4.7a mostra a IU em um primeiro momento, esperando alguma ação do usuário. Quando o usuário deseja inserir um texto, o usuário navega com auxílio (dos botões) das setas direcionais, e depois pressiona o botão que fica entre as setas direcionais. Então o navegador abre uma nova tela para o usuário digitar o texto, como mostra a figura 4.7b. A transição, da tela mostrada na figura 4.7a para a da figura 4.7b, é controlada pelo próprio navegador, ou seja, para o usuário inserir um texto o navegador sempre abre uma nova tela. Após o texto ser digitado o usuário deve

pressionar o botão OK (canto direito inferior). Então o navegador retorna a tela anterior, da figura 4.7a.

A figura 4.8a mostra a IU quando o usuário escolhe o menu **file**. O navegador abre uma nova tela para o usuário escolher uma das opções. Este tipo de transição entre telas deve ser pré-definido pelo *designer* da interface.

(a) (b)

Figura 4.8: Opções do menu file mostrado na interface executada no simulador

Esta IU executada no navegador para celular não foi gerada a partir do código mostrado na figura 4.3, foi necessário desenvolver novas descrições em XICL. A tela mostrada na figura 4.7 é um arquivo diferente do mostrado na figura 4.8. Foi necessário descrever uma tela (interface) só para o menu, isto devido ao fato do componente **menu**, desenvolvido para desktop, não funcionar no navegador para celular. Já o componente **confirmBox**, desenvolvido para o desktop, funciona no celular, como mostrado na figura 4.8b. A mensagem é mostrada quando o usuário escolhe a opção **close** do menu

file, semelhante à figura 4.2b. Isto mostra que alguns componentes desenvolvidos para desktop podem ser reaproveitados para celular.

Este capítulo descreveu a XICL e como ela pode ser utilziada para desenvolvimento de UI para múltiplas plataformas. O capítulo 5 apresenta um processo para uso de componentes XICL na geração de IUs para múltiplas plataformas.

5 Geração de IUs a partir de especificações IMML

Este capítulo apresenta um processo de geração de protótipos de interfaces de usuário no estilo WUI para diferentes dispositivos, a partir de especificações feitas em IMML, com o uso de componentes XICL. Este processo define um conjunto de passos, que devem ser executados por ferramentas, para gerar código DHTML, a partir de especificações IMML, fazendo o uso de componentes XICL. Os passos estão apresentados de forma a sugerir uma seqüência que pode ser seguida por ferramentas para realizá-los de forma automática.

Este processo é composto por duas etapas. A primeira etapa consiste no mapeamento das especificações de interfaces em IMML para componentes XICL. A segunda etapa consiste em gerar código DHTML a partir do código XICL. A figura 5.1 mostra o processo de geração em duas etapas. A seta 1 representa a geração de código XICL de um protótipo de IU que será utilizada em um navegador Web de um desktop. A seta 2 representa a geração do código DHTML, a partir do código XICL gerado. As setas 3 e 4 representam a geração, a partir da mesma especificação IMML, um protótipo de IU para ser utilizada em um navegador Web de um celular. O capítulo 6 apresentará dois compiladores, um para cada etapa deste processo.

Antes, será apresentado o processo de mapeamento de descrições IMML em componentes XICL. Em seguida, será apresentada a geração de código DHTML a partir do código XICL.

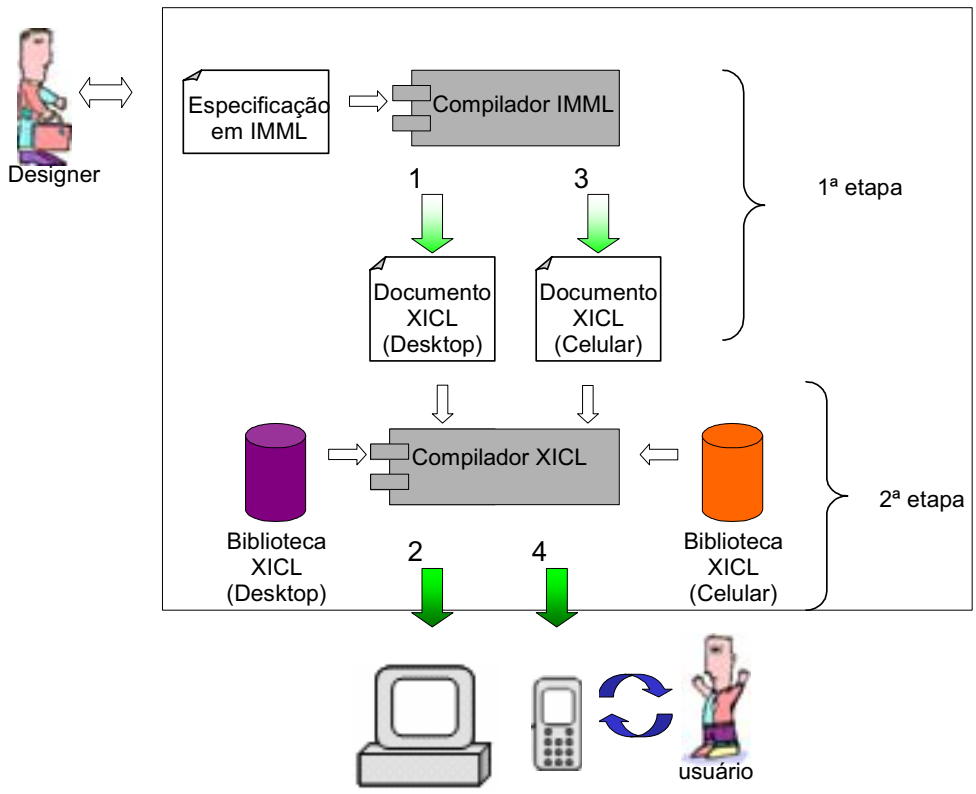


Figura 5.1: Desenvolvimento de protótipos de interfaces de usuário com o uso de componente XICL

5.1 GERAÇÃO DE CÓDIGO XICL A PARTIR DE ESPECIFICAÇÕES IMML

Esta seção apresenta um conjunto de passos que compõem a primeira etapa do processo representado na figura 5.1.

Como visto no capítulo 3, o modelo de interação pode conter a descrição de várias tarefas, e uma tarefa pode ser formada pelo agrupamento de outras (sub)tarefas. No código mostrado no anexo I, o arquivo IMML tem a descrição das tarefas bancárias: **ConsultarSaldo**, **EmitirExtrato**, **EfetuarPagamentoComSaldoAntes** e **EfetuarPagamento**. Estas tarefas foram agrupadas em uma tarefa maior chamada **TransacaoBancaria**.

Cada vez que um código IMML é processado, o objetivo deve ser a geração de um protótipo de IU que possibilite o usuário interagir com o sistema para a realização de uma tarefa em uma plataforma. Uma interface de usuário pode ser formada por uma ou

mais telas. Como as tarefas são descritas no modelo de interação, então este modelo deve guiar o processo de geração do protótipo da IU. Para gerar a(s) tela(s) que compõe(m) uma IU para um determinado dispositivo (plataforma) é necessário ter um ambiente de tarefa (task-environment) descrito, no modelo de comunicação, para a tarefa. Portanto, para gerar a interface para a tarefa **Transacão Bancária** para a plataforma **terminal**, por exemplo, é necessária a descrição de um ambiente de tarefa para o dispositivo terminal. Assim, o processamento deve iniciar pela localização do elemento <task>, no modelo de interação, com a descrição da tarefa **Transacão Bancária**. Depois deve ser localizado, no modelo de comunicação, o elemento <task-enrironment > que descreve um ambiente de tarefa para a tarefa **Transacão Bancária** ser executada, pelo usuário, na plataforma **terminal**. O ambiente de tarefa oferece informações para a escolha dos componentes XICL e/ou elementos HTML que compõem a IU.

A figura 5.2 mostra os passos a serem seguidos no processamento da tarefa **Transacão Bancária** para o **terminal**. No passo 1, é encontrado o elemento <task> que descreve a tarefa **Transacão Bancária**. No passo 2, é encontrado o elemento <task-environment> que descreve um ambiente de tarefa para que a tarefa **Transacão Bancária** seja realizada em um terminal bancário. Dentro do ambiente de tarefa foi usado o elemento IMML <edit-box>, que define que a interface permita que o usuário possa editar um texto. No passo 3, é encontrado o componente XICL **Text** que implementa a interação sugerida pelo elemento <edit-box>. Cada elemento <edit-box> faz referência a um objeto de domínio, que especifica o tipo de dado a ser editado. No passo 4, é feita uma busca pelo objeto de domínio **Agencia**, que é um objeto do tipo número (number), o que especifica que o usuário deve inserir um dado do tipo número. Neste exemplo, a interface também tem outro elemento <edit-box>, que faz referência ao objeto de domínio chamado **Conta**. O passo 4 é repetido para todos os elementos que fazem referência a um objeto de domínio. No passo 5, o modelo de interação informa os passos que o usuário deve seguir para realizar a tarefa. O modelo de interação pode especificar, por exemplo, que o usuário deve informar a agência e depois a conta.

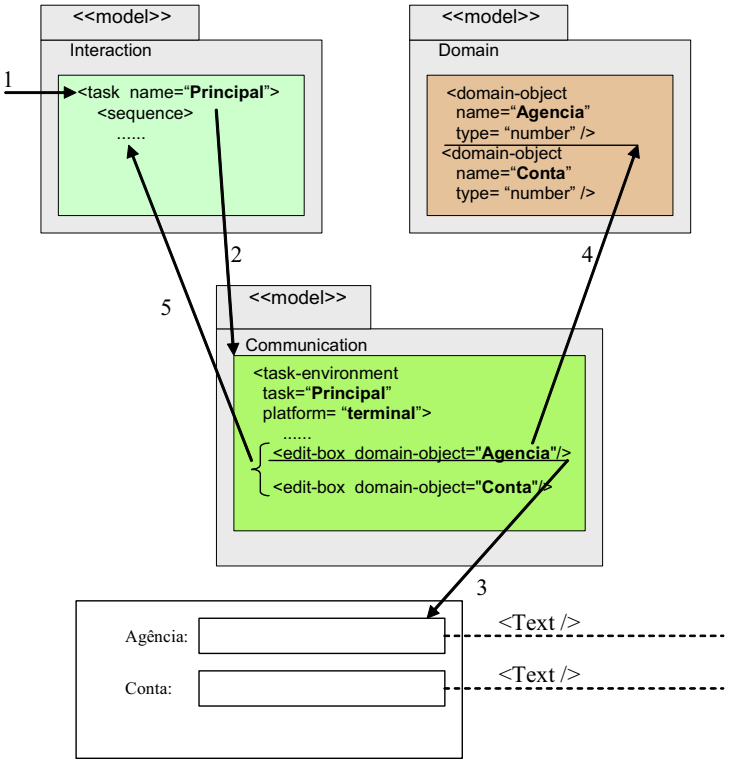


Figura 5.2: Passos para a geração de um protótipo de uma IU XICL

5.1.1 Uso de regras de mapeamento

A aplicação das regras de mapeamento ocorre basicamente no passo 3, da Figura 5.2. Neste exemplo, o elemento IMML <edit-box> foi mapeado para o componente XICL **Text**, que define uma caixa de texto. O elemento <edit-box> também poderia ter sido mapeado em outro componente XICL, ou mesmo em elementos HTML.

Os elementos IMML usados para descrever um modelo de comunicação de uma tarefa para um celular são os mesmos usados para descrever um modelo de comunicação de uma tarefa para um desktop. A diferença entre estes dois modelos de comunicação está na disposição dos elementos, pois a interface para um desktop, normalmente, tem uma estrutura visual diferente de uma interface para um celular. As regras de mapeamento usadas para gerar um protótipo de IU em XICL para um desktop podem (ou mesmo devem) ser diferentes da regras de mapeamento para gerar uma protótipo de IU em XICL para um celular. Por exemplo, para gerar uma interface para ser usada em um desktop, o elemento IMML <display-area> pode ser mapeado para um componente XICL **Window**. Contudo, para gerar uma interface para ser usada em um

celular, se o componente **Window** não tiver sido implementado para ser usado em um celular, então este elemento IMML deve ser mapeado para um elemento HTML, ou para outro componente XICL que possa ser executado em um navegador que esteja funcionando em um celular. Portanto, as regras de mapeamento dependem da plataforma para qual a protótipo de interface está sendo gerado e da biblioteca de componentes XICL disponível para a plataforma.

A figura 5.3 representa o mapeamento do elemento <display-area> para a plataforma terminal e para a plataforma celular. A seta pontilhada representa a regra de mapeamento para a plataforma celular, que mapeia o elemento <display-area> para um componente XICL chamado CB_celular. Um componente XICL desenvolvido para desktop também pode ser utilizado em uma IU para celular, por exemplo, desde que este componente respeite as limitações do celular.

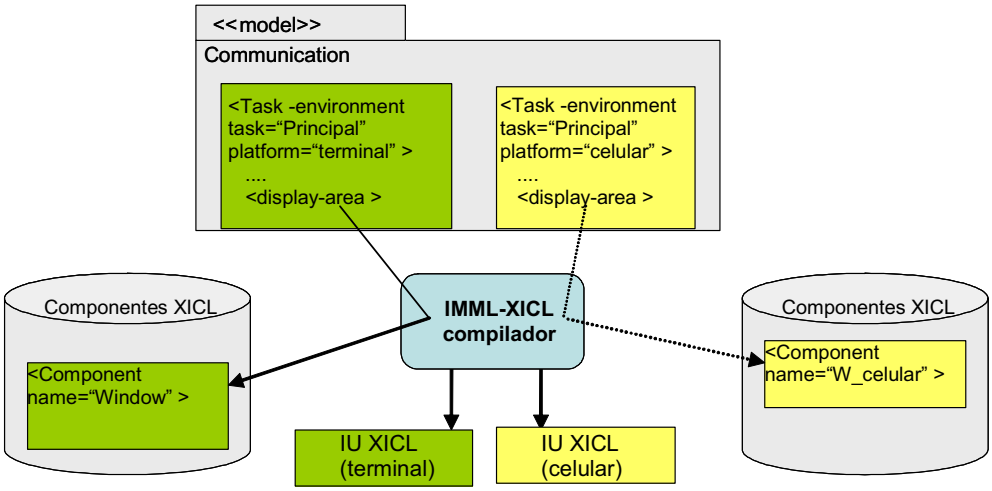


Figura 5.3: Regras para mapear elementos IMML em componentes XICL

5.1.2 Definição do comportamento da interface

O comportamento da interface de usuário é definido a partir de um conjunto de ações que a interface pode realizar quando o usuário interage com a mesma. Os três modelos da IMML contribuem para a definição do comportamento.

Quando o modelo de comunicação especifica que o usuário deve inserir alguma informação que está relacionada a um objeto de domínio (domain-object), o modelo de domínio é consultado para informar qual o tipo de dado a ser inserido pelo usuário. Se o

usuário inserir um dado em um formato não esperado alguma mensagem de erro pode ser exibida ao usuário, informando que o dado não está no formato esperado. Também pode ser exibido ao usuário um exemplo de como o dado estaria correto. Estas mensagens que podem ser exibidas ao usuário durante a interação fazem parte do comportamento da interface. No caso do componente **Text**, mostrado na figura 5.2, o dado inserido pelo usuário pode ser analisado por uma função associada ao evento onExit do componente (<Text onExit="analisa(this.value)" />). Este evento ocorre quando a caixa de texto perde o foco, ou seja, quando o usuário deixa de interagir com esta caixa de texto e passa a interagir com outra caixa de texto, por exemplo. Definir o comportamento da interface diante de cada erro cometido pelo usuário na entrada de dados torna a especificação muito extensa. Portanto, um comportamento como este pode ser definido no momento da geração da IU, pela ferramenta que faz a geração de código.

Nos casos em que é necessário que o usuário siga uma seqüência para realizar uma tarefa, a interface deve guiar o usuário para a realização da tarefa, e não permitir que a seqüência seja violada. O modelo de interação, auxilia na definição de aspectos como este, pois o mesmo define como deve ocorrer a interação do usuário com a interface para a realização da tarefa. Assim, após definir a estrutura visual da interface, é necessário definir o comportamento da mesma, processando o modelo de interação.

As transições entre as telas que compõem uma interface de usuário também fazem parte do comportamento da interface. As transições são definidas pelo modelo de comunicação. Uma tela em IMML é descrita com o elemento frame, declarado no modelo de comunicação. Portanto, o *designer* pode especificar, por exemplo, que o usuário, após clicar em um botão, pode passar a interagir com outra tela. Isto deve ser feito declarando o nome do segundo frame como valor do atributo **transition** de um botão que esta na primeira tela. Por exemplo, o código IMML <push-button label="Next" transition="segundaTela" > declara que após clicar no botão "Next" o usuário passa a interagir com a tela nomeada de **segundaTela**.

5.2 GERAÇÃO DE CÓDIGO DHTML A PARTIR DE ESPECIFICAÇÕES XICL

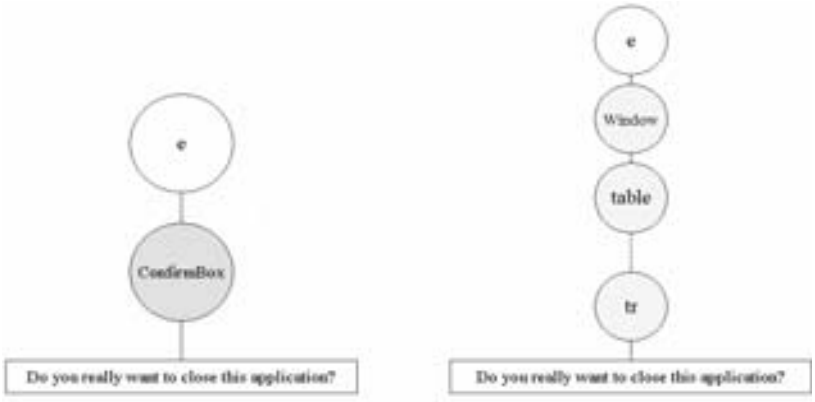
Esta seção apresenta um conjunto de passos que compõem a segunda etapa do processo representado na figura 5.1.

Para melhor entendimento do processo de geração de código, podemos considerar o código de uma IU XICL como sendo uma árvore, onde cada nó pode ser: um elemento da linguagem HTML; ou um elemento específico da linguagem XICL; ou um elemento referente a um componente XICL como <ConfirmBox>, por exemplo. Ao longo do processamento da IU XICL, todos os nós da árvore devem ser analisados e os que representam componentes XICL devem ser substituídos por outros nós que são elementos HTML ou mesmo outros componentes XICL.

Ao encontrar o seguinte código (linha 21, figura 4.3)

```
20. ...
21.   <ConfirmBox id="mes1" onConfirm="close()" title="Close?"> Do
you really want to close this application? </ConfirmBox>
22. ...
```

como o componente **ConfirmBox** é uma extensão do componente **Window**, então a ferramenta substitui o elemento <ConfirmBox> pelo elemento <Window> na árvore da IU. A figura 5.4a mostra, de forma simplificada, a árvore que representa o código antes do componente ConfirmBox ser processado e a figura 5.4b mostra a árvore depois que o componente é processado. Abaixo do componente <window> é colocado um elemento HTML <table>, isto é feito porque foi definido (figura 4.6) o uso deste elemento (table) na estrutura do componente **ConfirmBox**.



(a) (b)

Figura 5.4: Substituição de elementos no processo de compilação.

Para saber se um nó deve ser substituído ou não, para cada nó da árvore, deve ser procurado, na biblioteca de componentes, se existe um componente referente ao nó que está sendo processado. Se o componente não existir, então o nó não é alterado (substituído). A figura 5.5 representa o processo de geração de IU em DHTML a partir de descrições XICL, as bibliotecas e os arquivos XICL mostrados nessa figura são os mesmos representados na figura 5.3.

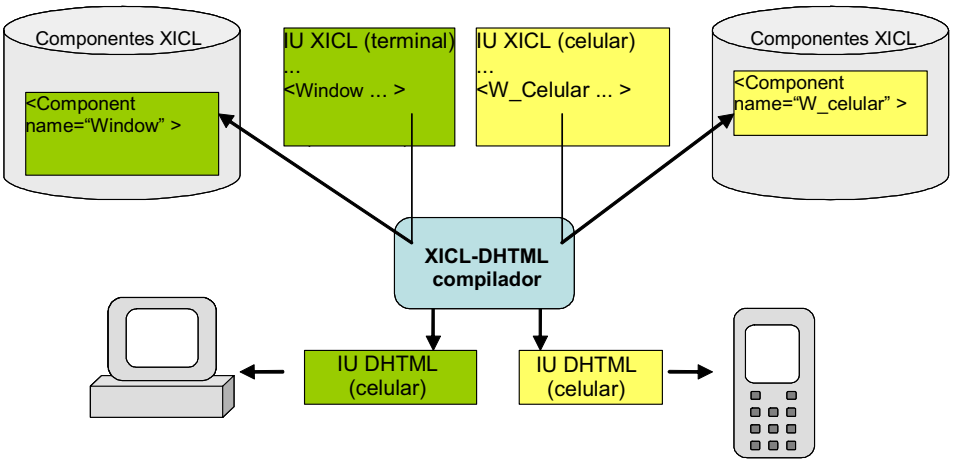


Figura 5.5: Geração de IU DHTML a partir de código XICL.

A figura 5.6 mostra um protótipo de interface gerado para ser executada em um desktop, para o usuário realizar a tarefa de consulta de saldo. O usuário deve inserir a agência, depois a conta e a senha. Por fim o usuário solicita a consulta, pressionando o botão "Consulta".

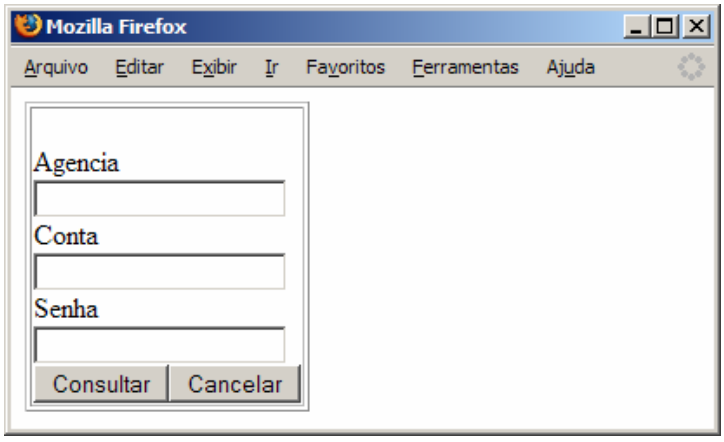


Figura 5.6: IU para a tarefa saldo em um desktop.

A figura 5.7 mostra duas telas, no simulador do opera mini, que foram geradas para serem usadas pelo usuário, para inserir dados, na realização da tarefa de consulta de saldo. A tela 5.7a mostra a tela que é exibida para o usuário inserir a agência. A tela 5.7b mostra a tela exibida para o usuário inserir a conta do cliente.

(a) (b)

Figura 5.7: IUs para a tarefa saldo em um celular.

Este capítulo descreveu as duas etapas do processo de geração de uma WUI a partir de uma especificação IMML. A primeira etapa descreveu a geração de código XICL com reuso de componentes a partir de uma especificação em IMML. A segunda etapa descreveu a geração do código DHTML de uma WUI partir da descrição da interface e dos componentes em XICL.

O capítulo 6 apresenta as ferramentas que foram desenvolvidas para executar o processo de geração de protótipos de interfaces de usuário apresentado neste capítulo.

6 Ferramenta de apoio ao Processo de Geração de IUs

Este capítulo apresenta ferramentas que foram desenvolvidas para apoiar o processo de geração de protótipos de IUs a partir de especificações IMML apresentado no capítulo 5. Foram desenvolvidos dois compiladores: um para gerar IUs XICL a partir de especificações IMML, e outro que gera código DHTML a partir de descrições XICL. Estes compiladores foram implementados na linguagem Java. Com estes dois compiladores é possível gerar protótipos executáveis em navegadores Web a partir de descrições IMML. As próximas seções mostram os compiladores IMML, XICL e, em seguida, a integração dos mesmos.

6.1 COMPILADOR IMML-XICL

Este compilador realiza a primeira etapa do processo apresentado no capítulo 5, isto é, gerar código XICL a partir de código IMML. Esta seção apresenta a arquitetura da ferramenta e alguns detalhes do processamento realizado pela mesma para realizar os passos apresentados na seção 5.1.

Este compilador foi desenvolvido com o objetivo principal de gerar código XICL, mas foi modelado de forma a ser facilmente utilizado para gerar código para outras linguagens.

A figura 6.1 mostra um diagrama de classes da arquitetura do compilador. Cada módulo é implementado em uma classe. As análises léxica e sintática são feitas por um *parser* implementado em Java do pacote *javax.xml.parsers*, que é utilizado dentro do módulo IMMLCompiler. O compilador usa a API DOM para fazer a manipulação do arquivo IMML. O DOM permite que o documento seja percorrido em forma de árvore, onde cada elemento é um nó da árvore. Após a análise léxica e sintática, são feitas, paralelamente, a análise semântica e a geração de código. Portanto, à medida que a

árvore do arquivo IMML vai sendo percorrida, o código para a plataforma alvo vai sendo gerado.

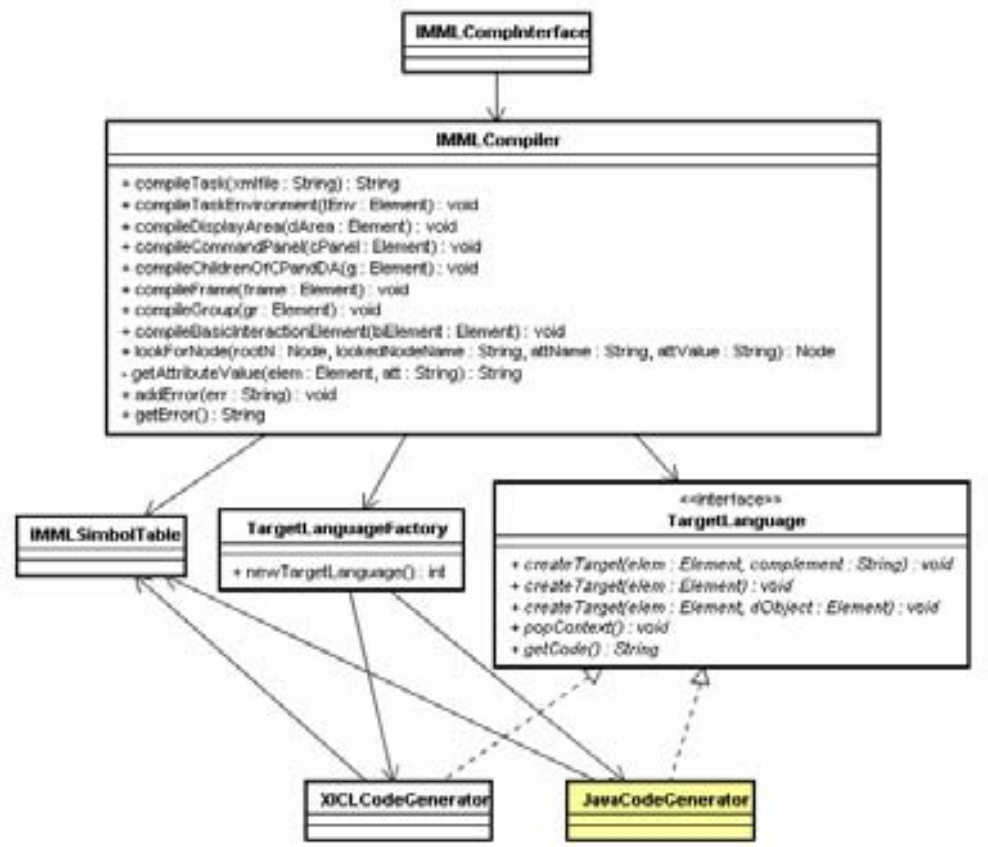


Figura 6.1: Arquitetura do compilador IMML-XICL

Para que seja possível reutilizar esta ferramenta para gerar código para outras linguagens, a partir de especificações IMML, o processo de geração de código foi deixado independente dos demais. Foi utilizado o padrão de projeto *factory*, cujo objetivo é "fornecer uma interface para a criação de família de objetos relacionados ou dependentes sem especificar suas classes concretas" (Gamma 2000). As classes de geração de código devem implementar a interface **TargetLanguage**. Portanto, antes de iniciar a análise semântica, o método newTargetLanguage() de um objeto do tipo **TargetLanguageFactory** é invocado. Este objeto, por sua vez, deve identificar o contexto e retornar uma classe que gera código para a determinada plataforma. À medida que a análise semântica vai ocorrendo, o gerador de código vai recebendo informações para gerar código. Ao final deste processo, o gerador de código é consultado para informar o código gerado durante o processo de compilação. A classe **XICLCodeGenerator** implementa a interface **TargetLanguage** e gera código XICL. A

classe **JavaCodeGenerator** foi colocada como exemplo na figura, pois a mesma ainda não foi implementada.

A figura 6.2 mostra um diagrama de seqüência que representa o processo de compilação, onde o gerador de código está gerando código XICL. No início, o compilador recebe o arquivo IMML, o nome da tarefa a ser compilada e o nome da plataforma alvo, para a qual a IU deve ser gerada. A partir de então, o módulo **IMMLCompInterface** inicia o processo de compilação. No diagrama de seqüência, cada mensagem representa a invocação de um método. Na mensagem 1 ocorre a instanciação de um objeto do tipo IMMLCompiler. Na mensagem 2 é passado o caminho do arquivo IMML a ser compilado. A mensagem 2.1 é utilizada para localizar um gerador de código que gere código para a plataforma desejada. A mensagem 2.2 informa o início da compilação de um ambiente de tarefa.

O módulo **IMMLCompiler** faz a navegação pela árvore que representa arquivo IMML e sempre tem o controle de onde está, dentro do arquivo que está sendo compilado. Por outro lado, o gerador de código não tem noção nem do tamanho do arquivo que está sendo compilado. O gerador de código é informado sobre qual o momento em que o compilador inicia a compilação de um elemento como, por exemplo, um painel de comando, pelo fato do método **createTarget()** ser invocado (mensagem 2.2.2.1), passando o elemento que representa o painel de comando como parâmetro. Desta maneira o gerador de código é informado que o compilador vai iniciar a compilação de um painel de comando. Também é necessário que o gerador de código seja informado sobre o momento no qual a compilação do painel de comando termina. Para isso é utilizado o método **popContext()**. Com isso é possível evitar conflito de contextos no processo de compilação.

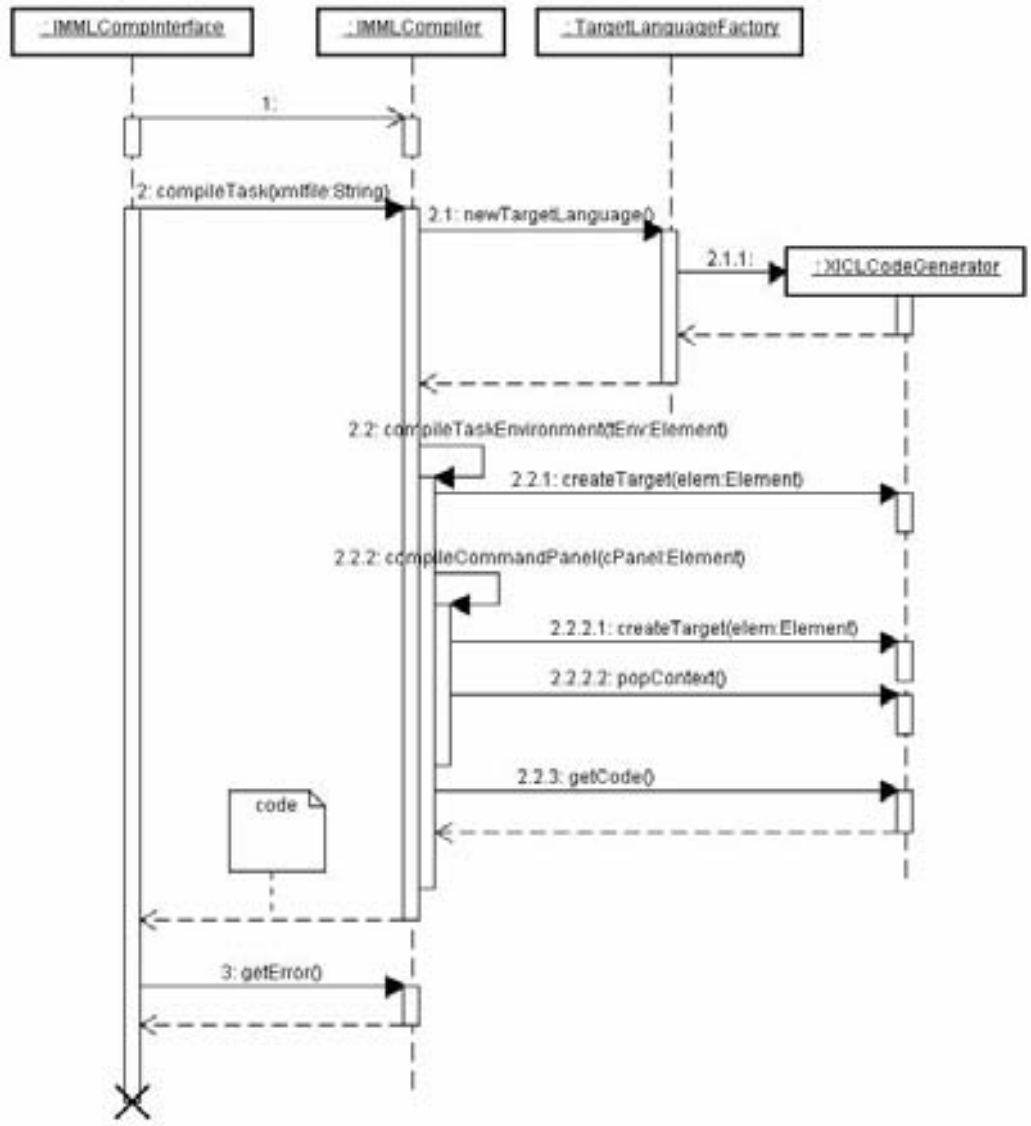


Figura 6.2: Processo de compilação

A figura 6.3 mostra um trecho de um código IMML, que tem a descrição de um elemento **group** (g1). Esse elemento, por sua vez tem dois elementos **group** (g2 e g3) e um elemento **edit-box**. O elemento g1 possui uma orientação vertical e g2 e g3 orientação horizontal. Portanto, durante o processo de compilação, se o gerador de código não for informado, no momento oportuno, que já terminou a compilação do elemento g2, e que o gerador deve abandonar este contexto, então o elemento edit-box (linha 8) será inserido dentro do elemento g2. Isto fará com que a interface tenha um aspecto visual diferente do que foi especificado pelo designer.

No final da compilação, é solicitado o código gerado (mensagem 2.2.3) e depois retornado ao módulo IMMLCompInterface. A mensagem 3 (Figura 6.2) é utilizada para receber os erros encontrados durante a compilação.

```
 1. <group name=”g1” orientation=”vertical”>
 2.
 3.    <group name=”g2” orientation=”horizontal”>
 4.       <edit-box label="Agencia" domain-object="Agencia" />
 5.       <edit-box label="Conta" domain-object="Conta" />
 6.    </group>
 7.
 8.    <edit-box label="Senha" domain-object="Senha" />
 9.
10.    <group name=”g3” orientation="horizontal" align="center">
11.         <push-button label="Consultar" transition="MostraSaldo" />
12.         <push-button label="Cancelar" hide="this" />
13.    </group>
14.
15. </group>
```

Figura 6.3: Trecho de um código IMML

A figura 6.4 mostra a interface gráfica do compilador IMML. Como, mostra a figura 6.4, primeiro é escolhido o arquivo *BancoRico.xml*. Neste momento a ferramenta localiza todas as tarefas e coloca no menu de tarefas (task). Quando uma das tarefas é escolhida, a ferramenta mostra um segundo menu (platform) com as possíveis plataformas alvo. O próximo passo é escolher um arquivo onde o resultado da compilação deve ser guardado, e depois solicitar a compilação (botão “Compile”).

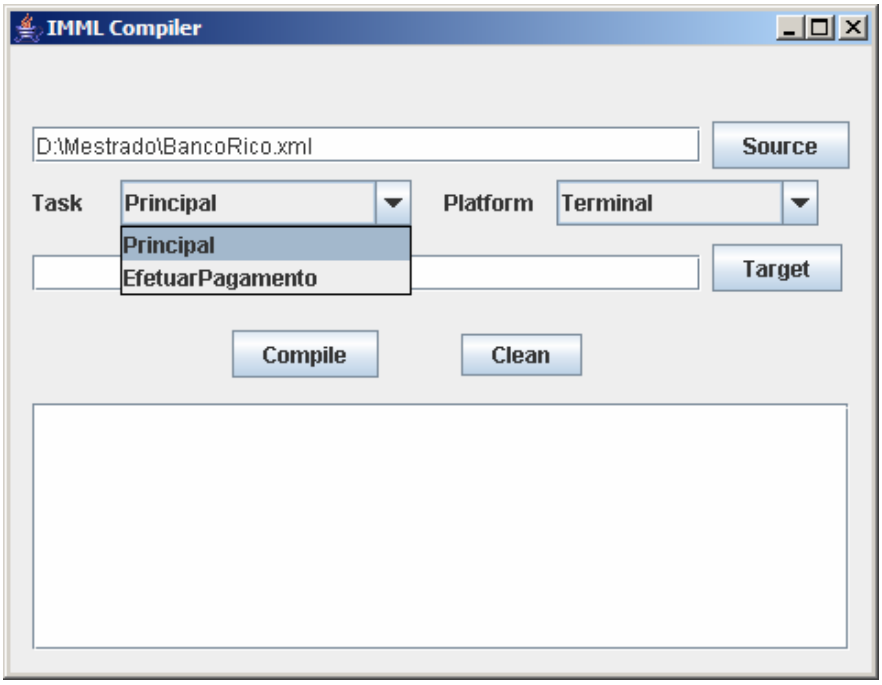


Figura 6.4: Interface gráfica da ferramenta que compila especificações IMML

6.2 COMPILADOR XICL-DHTML

Este compilador foi desenvolvido para realizar a segunda etapa do processo apresentado no capítulo 5. Esta ferramenta executa os passos apresentados na seção 5.2, que gera código DHTML a partir de descrições XICL. Este compilador pode ser utilizado para processar (i) as interfaces de usuário XICL geradas por uma ferramenta, como no caso dos protótipos gerados pelo compilador IMML-XICL, ou (ii) as IUs desenvolvidas de forma manual, por um *designer* que faz a descrição da interface utilizando uma ferramenta que permita a edição de arquivos XML.

A figura 6.5 mostra um diagrama de classes UML representando os principais módulos deste compilador. Para simplificar não foi colocada a classe que implementa a interface gráfica do compilador.

O módulo **XICLCompiler** faz a análise semântica do documento XICL. A geração de código é feita paralelamente à análise semântica. Para isto, este compilador também usa a API DOM. No início da compilação, a árvore é uma representação da IU XICL. Ao longo da compilação, os nós desta árvore que representam componentes XICL são substituídos por elementos HTML. Ao final, a árvore, composta de elementos HTML, é transformada em código.

O módulo **ComponentManager** é responsável pelo gerenciamento dos componentes XICL. O método *hasComponent* é utilizado para verificar se o componente foi implementado em XICL. O componente pode estar implementado no mesmo arquivo XICL da interface ou em outro arquivo, que sirva de biblioteca.

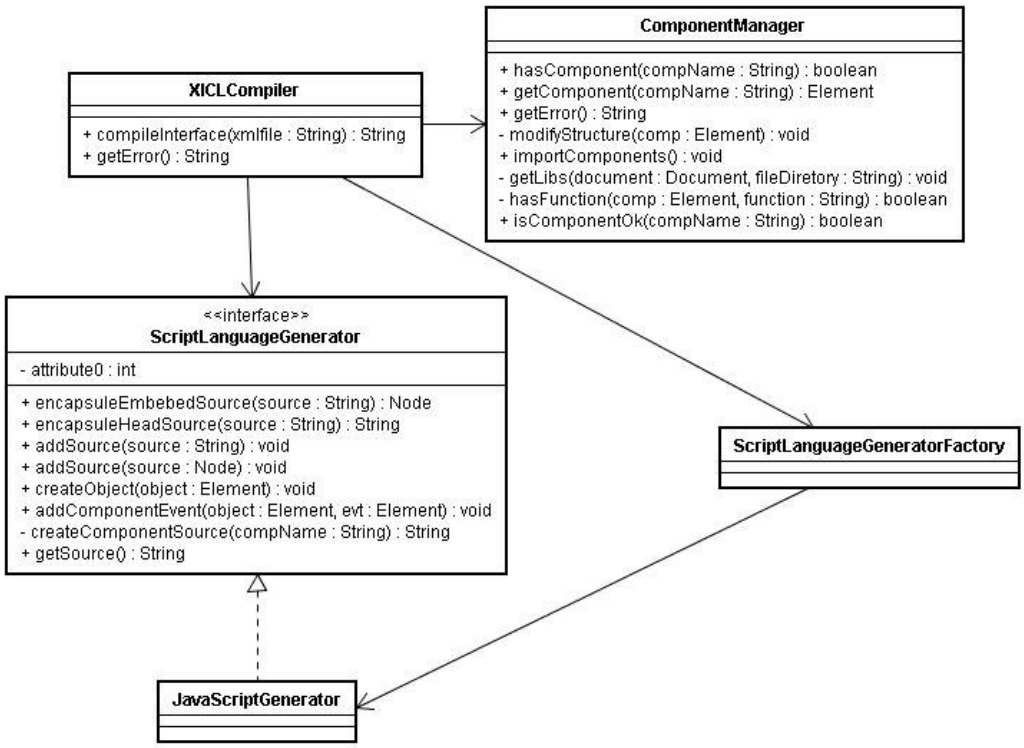


Figura 6.5: Arquitetura do compilador para XICL-DHTML

Os scripts são responsáveis pelo gerenciamento das interações do usuário com a IU. Neste compilador a geração de código script é feita por uma classe separada. Como no compilador IMML, aqui também foi utilizado o padrão *Factory*. A classe **ScriptLanguageGeneratorFactory** deve encontrar a classe mais apropriada para gerar o código do script. As classes para a geração de script devem implementar a interface **ScriptLanguageGenerator**. Até este momento só foi implementado o gerador de código para a linguagem JavaScript, a classe JavaScriptGenerator.

Para cada componente utilizado na interface, é criada uma classe em JavaScript que representa o componente. Da mesma forma, para cada instância do componente declarada na interface é criada uma instância da classe. A linguagem JavaScript, de certa forma, *simula* uma orientação a objetos, pois uma classe é declarada como uma função e depois instanciada. Todos os métodos e variáveis são públicos. Os métodos de uma classe são declarados como funções dentro do escopo da declaração da classe. Para compilar o código mostrado na figura 4.3, a ferramenta cria a classe **Window**, que possui os métodos `show()` e `hide()`, e depois é criada a classe **ConfirmBox**, que estende a classe **Window**. A figura 6.6 mostra uma parte do código criado.

```
 1. function Window(id){
 2.   this.id=id;
 3.
 4.                function hide()
 5.                {
 6.                     ...
 7.                }
 8.
 9.                function show()
10.                {
11.                     ...
12.                }
13. }
14.
15. function ConfirmBox(id){
16.   this.id=id;
17.
18.                function hide(){
19.               ...
20.               try { this.onConfirm(); }catch(e){}
21.               try { this.onCancel();  }catch(e){}
22.                 }
23. }
24. ConfirmBox.prototype = new Window();
```

Figura 6.6: Classes geradas em JavaScript

O evento de um componente pode estar associado a: (1) ocorrência de outro evento ou; a (2) execução de um método. O componente ConfirmBox possui os eventos onConfirm e onCancel. O primeiro evento está associado ao evento onClick do botão **bOk**. O evento onCancel está associado ao evento onClick do botão **bCanc**. Também foi definido que antes do tratamento de cada evento deve ser executado o método `hide()`. Neste caso, o compilador associa a execução do método `hide()` ao evento onClick do botão bOk e o tratamento do evento é colocado dentro do escopo do método `hide()`, depois da execução do método propriamente dito. Isto é feito inserindo a chamada para o tratamento do evento como sendo a última instrução do método `hide()`. Como o método `hide()` já estava definido no escopo do componente Window, então o mesmo é redefinido, como mostra a figura 6.6.

6.3 INTEGRAÇÃO DOS COMPILADORES

O compilador XICL foi colocado junto ao compilador IMML. Quando o compilador IMML termina de gerar código XICL, este código é passado para o compilador XICL, que por sua vez gera DHTML. A interface utilizada foi a do

compilador IMML, mostrada na figura 6.4. A figura 6.7 mostra um diagrama de seqüência UML com o comportamento dos objetos envolvidos na integração dos dois compiladores.

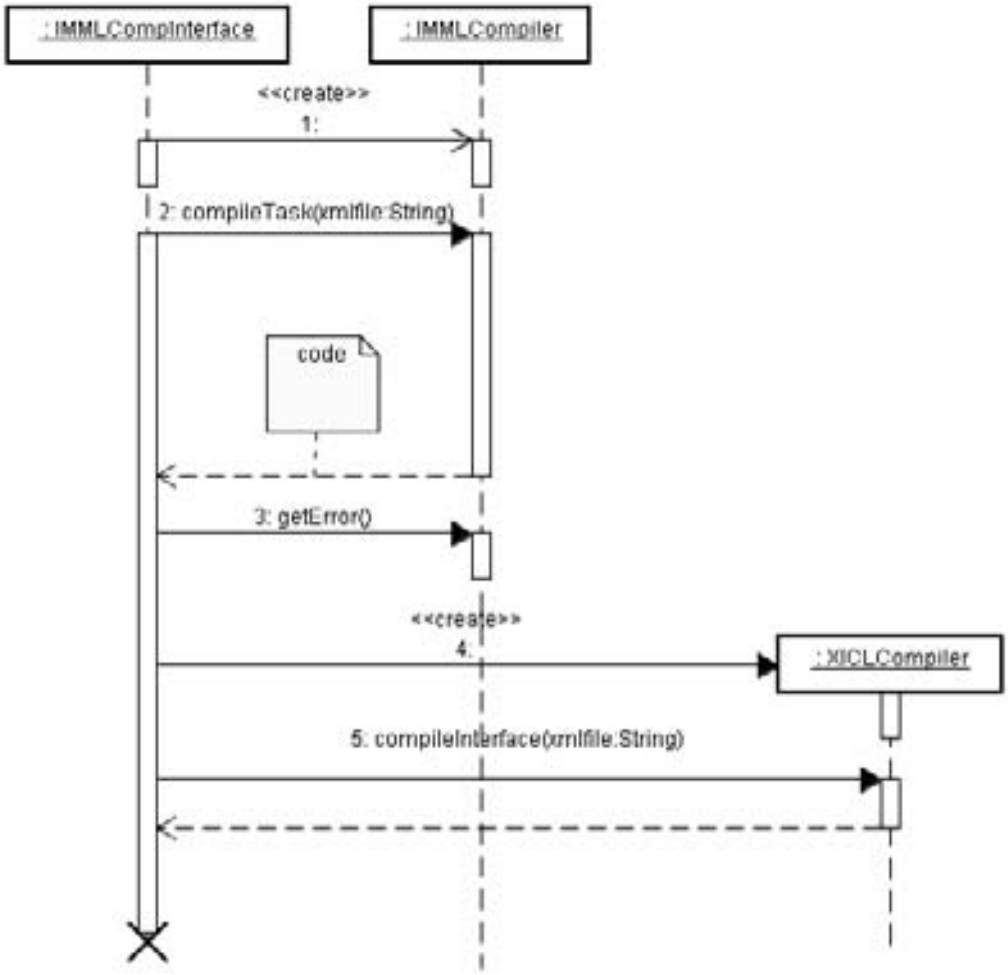


Figura 6.7: Ferramenta integrada para compilar de IMML para DHTML

O código em IMML da aplicação bancária (mostrado no anexo I) foi utilizado para testar a integração dos dois compiladores. O código XICL gerado pela primeira compilação está no anexo IV. Este código por sua vez é passado pelo compilador XICL que gera uma IU em DHTML. A figura 6.8 mostra a tela inicial do protótipo gerado. Esta tela é composta por um conjunto de botões, onde o usuário pode escolher uma tarefa para executar.

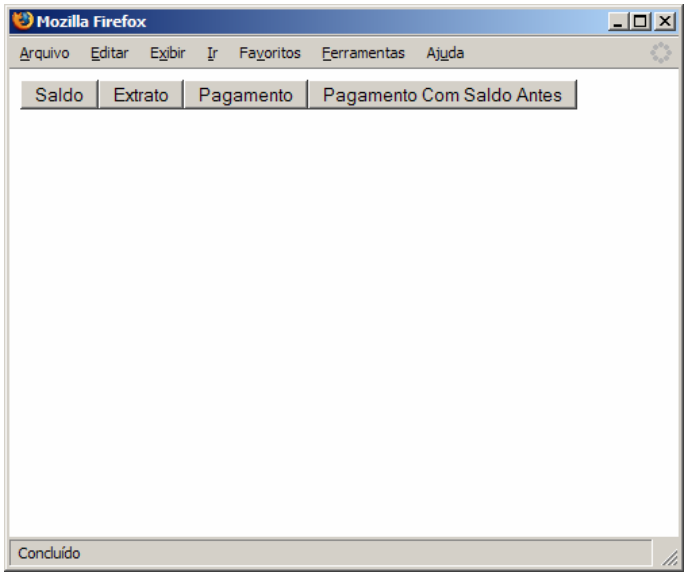


Figura 6.8: Tela inicial d protótipo gerado pelos compiladores

Ao interagir com a interface gráfica da figura 6.8, se o usuário solicitar uma consulta de saldo, através do botão "Saldo", será apresentada uma área para que o usuário realize a tarefa, conforme mostra a figura 6.9. O usuário insere o código da agência bancaria, o código da conta bancária do cliente e a senha. O botão "Consultar" solicita que o saldo seja exibido ao usuário. O botão "Cancelar" fecha a área que está sendo exibida para o usuário realizar a consulta, e a tela fica como mostrada na figura 6.8.

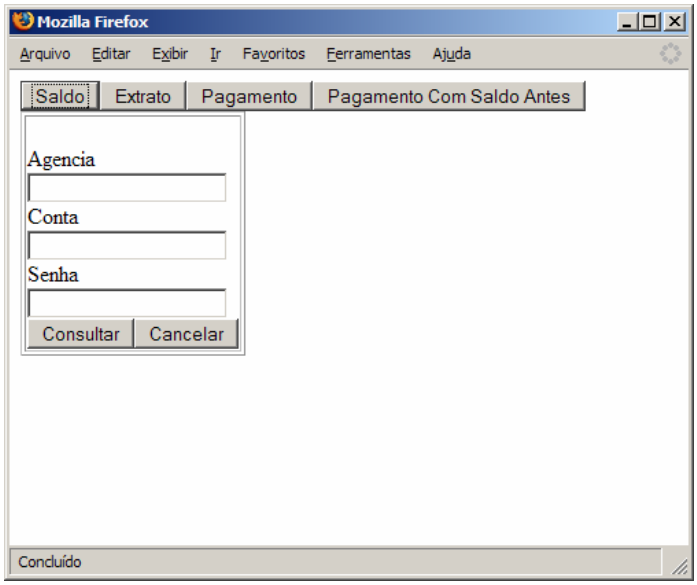


Figura 6.9: Tela da consulta de "saldo"

6.4 RECUPERAÇÃO DE COMPONENTES XICL

As regras de mapeamento implementadas no compilador IMML sempre relacionam um elemento IMML a um mesmo componente XICL. Por exemplo, o compilador sempre mapeia o elemento IMML <display-area> no componente XICL <Window>. Com isto, as regras de mapeamento sempre fazem uso de um mesmo conjunto de componentes XICL. Mas novos componentes XICL podem ser desenvolvidos e disponibilizados para serem usados pelas regras de mapeamento.

Um desenvolvedor pode criar novos componentes XICL e disponibilizá-los em repositórios de componentes, como bibliotecas, para serem utilizados por outros desenvolvedores na definição de novas regas de mapeamento. Mas um dos grandes problemas para reutilizar componentes desenvolvidos por terceiros é encontrá-los, ou seja, a dificuldade está na busca e recuperação dos componentes mais apropriados para as necessidades que surgem durante o desenvolvimento (Pietro-Diaz 1987).

As primeiras técnicas propostas para busca e recuperação de componentes fazem o uso do casamento de padrões (Pietro-Diaz 1987, Milli 1994), onde os componentes são recuperados fazendo comparações entre o nome dos mesmos, ou de seus métodos, e um verbo ou substantivo que representa a finalidade para a qual se quer fazer uso do componente. Por exemplo, quando for necessário um componente que realize o cálculo matemático para verificar se um número é primo, é feita uma busca por componentes que tenham o nome ou um método com o nome *primo*. As técnicas mais recentes sugerem o uso de um processamento de significados através do uso de ontologia (Sugumaram 2003). Uma ontologia é "a descrição de uma certa realidade com um vocabulário específico, usando um conjunto de premissas de acordo com o sentido intencional das palavras do vocabulário" (Fonseca and Engenhofer, 2000). A ontologia é formada por (i) um conjunto de conceitos que se relacionam e (ii) por um conjunto de instâncias dos conceitos. Para encontrar um componente que calcule se um número é primo, por exemplo, é feita uma busca por componentes que tenham o nome ou um método com nome que seja sinônimo da palavra *primo.* Para isto são utilizados dicionários de sinônimos. Outra possibilidade é traduzir o termo para outros idiomas para procurar por componentes. Por exemplo, procurar componentes nomeados de *prime* (primo em inglês). Estas técnicas precisam de ajustes para serem utilizadas na

recuperação de componentes de IU, pois neste caso, é necessário tratar de aspectos relacionados à interatividade, na recuperação dos componentes.

Na tentativa de recuperar componentes XICL, foi desenvolvida uma ontologia, que serve de base semântica para a busca de componentes (Sousa 2004b). Esta ontologia possui um modelo semântico baseado nos conceitos de interação propostos pelo modelo de interação da IMML. A ontologia é dividida em duas partes, a primeira é composta por conceitos de interação. A segunda parte é composta por conceitos relacionados à estrutura de um componente, conforme proposto em (Sugumaram 2003).

A ontologia foi descrita em OWL (Web Ontology Language) (Smith 2004). A figura 6.10 representa graficamente a primeira parte da ontologia, cada elipse representa um conceito e as setas representam os relacionamentos entre os conceitos. A elipse de onde a seta parte é o *sujeito*, a seta representa o *predicado* e a elipse de chegada da seta é o *objeto*. Os conceitos iniciam no usuário (**User**) que interage com (**interactWith**) a interface (**Interface**). Neste caso a interface se refere à interface de manipulação do componente pelo usuário, ou seja, a representação gráfica do componente de IU que permite que o usuário realize a interação. A interface de um componente é composta de um ou mais comandos. Quando um componente é composto de outros componentes, a interface deste componente mais complexo possui vários comandos. Um comando é composto por interações básicas ou estruturadas. Uma interação básica (**Interaction**) pode ser: **enter**, **select** e **activate** (**Button** ou **Link**). As interações básicas podem ser agrupadas por elementos de estruturação: **sequence**; **select**; **repeat**; **combine**; **join**.

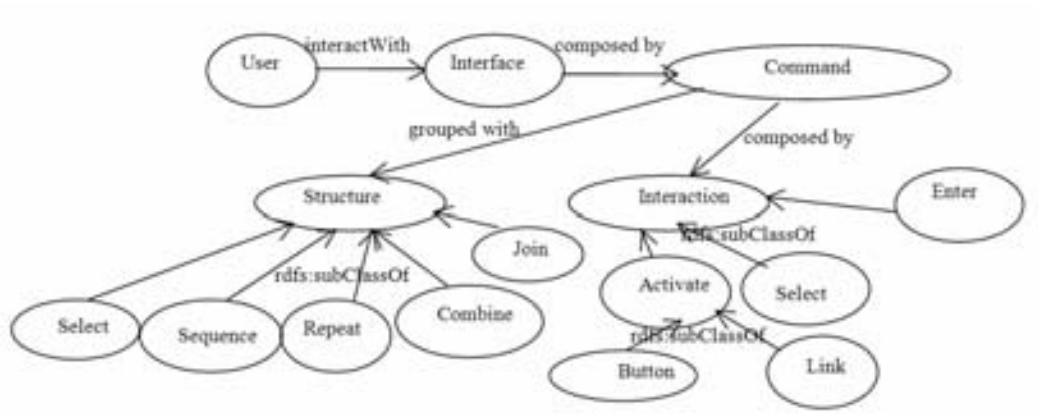


Figura 6.10: Parte da ontologia que representa a interação entre o usuário e os componentes de IU

A figura 6.11 mostra a segunda parte da ontologia. O conceito pacote (**package**), representa um conjunto de componentes, e possui uma propriedade chamada **hasComponent** cujo valor é o nome de um componente. O componente, por sua vez, pode ter duas propriedades chamadas de **hasInterface** e **hasSignature**, onde estas têm como valor uma interface e a assinatura do componente respectivamente. Definimos que a assinatura de um componente possui *métodos, eventos e atributos*. A ontologia também descreve sinônimos para os conceitos. Por exemplo, o conceito **package** possui dois sinônimos que são **directory** e **library**. Esta parte da ontologia dá suporte às buscas que são baseadas em aspectos estruturais. Por exemplo, quando o desenvolvedor de IU está procurando um componente que resolve verifica se um número é primo, a busca pode ser feita procurando um componente com o nome de *primo*.

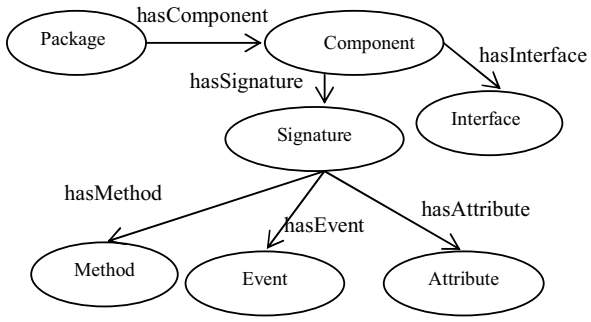


Figura 6.11: Parte da ontologia que representa estrutura do componente

O desenvolvimento de ontologia requer alguns cuidados para evitar ambigüidade. Uma ontologia representa conceitos, e cada conceito é identificado por palavra(s). As palavras podem ser ambíguas, mas os conceitos não. No desenvolvimento da nossa ontologia foram necessários ajustes para evitar ambigüidade, na representação de alguns conceitos. Um exemplo é o fato de que o conjunto dos nomes dos métodos e das propriedades de um componente recebe alguns nomes tais como: *assinatura*[3], *assinatura da interface* e *interface*. A imagem gráfica de um componente que é chamada de *interface gráfica* também pode ser chamada só de *interface* do componente. Com isso, "a interface de um componente de IU" possui dois sentidos, um referente aos métodos e outro referente à imagem gráfica, com a qual o usuário interage. Neste caso foi decidido que na segunda parte, relacionada aos aspectos estruturais, o conjunto de

[3] A ontologia foi descrita em inglês (como mostram as figuras 6.10 e 6.11), os nomes em português são ilustrativos.

métodos será referenciado das seguintes formas: *assinatura*, *assinatura dos métodos*. Na parte relacionada à interação, a interface gráfica do componente será referenciada das seguintes formas: *interface gráfica*, *interface*, *interface do usuário, interface do componente*.

Esta ontologia serve de base para desenvolver uma descrição semântica para os componentes XICL. Portanto, para cada componente deve ser desenvolvida uma descrição semântica. As descrições relacionadas a um componente são instâncias dos conceitos da ontologia. Com estas descrições é possível fazer buscas levando em consideração a interação oferecida pelo componente. Por exemplo, é possível procurar componentes que tenham um campo de texto para inserir dados, seguida de uma caixa de seleção e de um botão.

Ainda é preciso desenvolver uma ferramenta que faça o processamento desta ontologia, de forma a recuperar os componentes a partir de uma análise semântica da interatividade que os mesmos ofereçam. O uso de uma ferramenta que faça a busca automática de componentes, levando em consideração aspectos de interação, pode melhorar ainda mais o processo de descrição de regas de mapeamento, pois ajuda o *designer* a encontrar componentes disponíveis em bibliotecas, mas que são desconhecidos para ele até então.

7 Considerações Finais

O desenvolvimento de sistemas multi-plataforma tem sido um grande desafio para a Engenharia de Software. Entre as dificuldades encontradas, está o desenvolvimento de diversas interfaces de usuário para um mesmo sistema, onde cada interface deve prover interação em um contexto diferente.

A principal contribuição deste trabalho foi apresentar um processo de desenvolvimento de protótipos de interfaces de usuário, no estilo WUI, que faz o uso de componentes de interfaces de usuário. O processo considera a variação dos dispositivos que podem ser usados para a manipulação da interface de usuário. O trabalho sugere o uso de componentes XICL.

A geração de protótipos justifica-se porque consideramos que o processo de design de IU é um processo iterativo (ver seção 1.1), no qual protótipos são gerados a partir de especificações para serem avaliados quanto a sua usabilidade e outros fatores de qualidade. Este processo, muitas vezes, requer a geração de inúmeros protótipos até que o *designer* considere que a avaliação está satisfatória. Nosso trabalho propôs uma melhoria na qualidade desse processo de forma a torná-lo mais eficiente, uma vez que o processo de geração do protótipo é automatizado pela ferramenta desenvolvida. Além disso, como a linguagem de especificação utilizada está fundamentada na teoria da engenharia semiótica, espera-se que as interfaces resultantes possuam melhor usabilidade e comunicabilidade (ver capítulo 3). A ferramenta desenvolvida nesse trabalho será fundamental para trabalhos futuros da avaliação das interfaces produzidas.

Nossa solução seguiu as propostas baseadas no uso das Linguagens de Descrição de Interfaces de Usuário (LDIU) baseadas em XML. Estas linguagens permitem ao *designer* fazer descrições (especificações) abstratas de IUs. A partir destas especificações são geradas IU executáveis para diversos contextos. Cada IU é gerada

com o mapeamento das descrições que compõem a especificação em componentes de software da plataforma alvo. Vale lembrar ainda que a proposta apresentada limita-se à geração protótipos de interfaces de usuário no estilo WUI, para diversos dispositivos, a partir de especificações feitas na linguagem IMML, fazendo o uso dos componentes XICL.

A XICL (eXtensinble User Interface Components Language) é uma linguagem de marcação para a descrição de IUs e de componentes de IU no estilo WUI, que sigam as recomendações do W3C. Os componentes XICL são descritos com o agrupamento de elementos HTML e/ou outros componentes XICL. Estes componentes são utilizados para facilitar a geração de interfaces de usuário no estilo WUI, para dispositivos que tenham um navegador Web. Além do uso da XICL no processo de geração de IUs executáveis em navegadores Web, a partir de descrições abstratas, a linguagem também pode ser utilizada pelo *designer* para descrever IUs. Nestas descrições, a reusabilidade e extensibilidade dos componentes também podem ser exploradas.

A XICL possui 13 elementos específicos da linguagem. O código XICL pode ser composto pelos elementos específicos da linguagem, pelos elementos que representam componentes desenvolvidos na própria linguagem e por código DHTML. A XICL não possui um propósito tão abrangente quanto outras LDIUs, mas possui poucos elementos e conceitos específicos.

O processo utilizado para a geração de protótipos interfaces está dividido em duas etapas. Na primeira etapa, é gerado código XICL a partir de uma especificação IMML. Nesta etapa as descrições feitas em um documento IMML são mapeadas em componentes XICL. Na segunda etapa é gerado código DHTML a partir do código XICL.

O uso de componentes facilita a definição de regras de mapeamento, para gerar interfaces de usuário no estilo WUI, pois cada componente agrupa um conjunto de elementos HTML e/ou componentes XICL. Isto facilita a manipulação desses componentes na geração de interfaces a partir de especificações feitas em uma LDIU. O processo apresentado no capítulo 5 propõe uma seqüência de passos para gerar protótipos de IU em XICL a partir de descrições IMML, mas estes componentes também podem ser utilizados no processo de geração de interfaces a partir de especificações feitas em outras LDIUs. Para gerar código XICL a partir de outras LDIUs, é necessário ajustar estes passos para cada LDIU, e depois desenvolver

ferramentas que executem os passos, gerando código XICL. Com a IMML, a estrutura visual da IU é gerada com base nas descrições do modelo de comunicação. O modelo de interação e de domínio auxiliam na definição do comportamento da interface. No caso de outra LDIU, é preciso definir quais conceitos da LDIU definirão as estrutura da IU XICL e quais definirão comportamento da mesma.

No trabalho, também foram desenvolvidos dois compiladores: um para gerar código XICL a partir de descrições IMML e outro que gera DHTML a partir de XICL. Com estes dois compiladores integrados, é possível gerar protótipos de IUs em DHTML a partir de descrições feitas em IMML. Isto pode acelerar o processo de desenvolvimento de interfaces de usuário para sistemas multi-plataforma.

É importante ressaltar que, normalmente, as IUs geradas automaticamente necessitam de ajustes. Como exemplo de ajustes, podemos citar aqueles voltadas à disposição espacial dos componentes que formam a interface, pois os componentes podem ficar dispostos de forma a prejudicar a interação do usuário. Para fazer os ajustes, o *designer* precisa fazer modificações diretamente no código DHTML gerado. Como o código a ser ajustado foi desenvolvido de forma automática, o *designer* pode gastar muito tempo para realizar os ajustes, o que atrasa o processo de desenvolvimento de interfaces.

Para analisar os protótipos de interfaces gerados pelas ferramentas foram realizados alguns testes. O teste com a maior quantidade de código IMML foi o realizado com a interface que está descrita no Anexo I, que gerou código para a plataforma terminal e para a plataforma celular. Estes testes também ajudaram a melhorar alguns aspectos da relacionados à sintaxe da linguagem IMML. Outros testes ainda precisam ser realizados na geração de protótipos de interfaces para outros dispositivos móveis (por exemplo, palmtop) e para outras aplicações (sistemas) multi-plataforma. Para isto é necessário desenvolver outras especificações IMML, com a descrição de vários modelos de comunicação, um para cada dispositivo a ser utilizado. Para realização destes testes é importante o desenvolvimento de mais componentes XICL. A elaboração de mais testes com protótipos gerados a partir de especificações IMML, tornará possível um estudo sobre a usabilidade e outras vantagens, e desvantagens, encontradas com a especificação de interfaces de usuário através da IMML.

Outros experimentos ainda podem ser realizados com a XICL na geração dinâmica de conteúdo para a Web. Por exemplo, os sites que geram conteúdo para a Web a partir de informações armazenadas em banco de dados, normalmente geram as IU, para exibir os dados, de forma automática. Para isto existe um módulo responsável por processar as consultas feitas no banco de dados e gerar código DHTML. Este módulo responsável por gerar código DHTML pode ser ajustado para gerar código XICL, ao invés do código DHTML. E, a partir do código XICL, é gerado código DHTML. Isto introduz um passo a mais na geração de código DHTML, mas por outro lado, o uso da XICL pode facilitar a descrição de interfaces para apresentar os dados, uma vez que normalmente um componente XICL agrupa vários elementos HTML.

O processo apresentado neste trabalho também pode ser estendido para gerar protótipos de IU, para diversos dispositivos, processando informações sobre as características do dispositivo em que a interface será utilizada. As características de cada dispositivo podem ser descritas de acordo com um perfil CC/PP (Klyne et al., 2004), que é conjunto de atributos na forma dos tradicionais pares nome/valor, que descrevem as características físicas do dispositivo.

8 Referências

Bittencourt, J. R., Giraffa, L. M. M. (2004) Desenvolvendo Jogos Computadorizados Multiplataforma com Amphibian. In: Anais do V Workshop sobre Software Livre. Porto Alegre : Sociedade Brasileira de Computação. p. 119-122.

Booch, G.; Rumbaugh , J. & Jacobson , I.(1999) The Unified Modeling Language User Guide, Reding, MA: AddisonWesley.

Bray, T., Hollander, D., Layman, A., Tobin, R. (2006) Namespaces in XML 1.0 (Second Edition) http://www.w3.org/TR/REC-xml-names.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., Yergeau, F., Cowan, J., (2004) Extensible Markup Language (XML) 1.1 http://www.w3.org/TR/2004/REC-xml11-20040204

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. (2003). A unifying reference framework for multi-target user interfaces in: *Interacting with Computers*, 15(1):289.308.

Card, S; Moran, T; Newell, A. (1983) The psychology of Human-Computer Interaction. Lawrence Erlbaum Associates, Hillsdate, N.J.

Costa Neto, M. A.; Leite, J. C. (2005) Uma extensão da IMML para apoiar o Design de Interfaces de Usuário Multi-Plataforma. Dissertação (Mestrado) – Programa de Pós-Graduação em Sistema e Computação da Universidade Federal do Rio Grande do Norte, Natal.

de Souza, C. S. (1993). The semiotic engineering of user interface languages. *International Journal of Man-machine Studies*, 39(5):753.773.

de Souza, C. S.; Leite, J. C.; Prates, R.O. & Barbosa, S.D.J. (1999) Projeto de Interfaces de Usuário: Perspectivas Cognitiva e Semiótica. Anais da *Jornada de Atualização em Informática*, XIX Congresso da Sociedade Brasileira de Computação, Rio de Janeiro.

D'Souza, D. and Wills, A.C. (1999) Objects, Components and Frameworks with UML: The Catalysis Approach. Addison Wesley, Reading, MA.

Fonseca, F., Engenhofer M. "Ontologia e interoperabilidade semântica entre SIGs", Workshop Brasileiro de GeoInformática, São Paulo, 2000.

Fonsêca, F. M. (2005) Um Estudo Comparativo de Técnicas de Especificação e Modelagem de Interfaces: Medindo o Potencial da IMML. Monografia (Trabalho de Conclusão de Curso) – Departamento de Informática e Matemática Aplicada.Universidade Federal do Rio Grande do Norte, Natal.

Freeman, P. (1980) Reusable Software Engeneering: A statement of long-range research objectives, Technical Report, TR-159, Univ. Califórnia.

Gamma, E., Helm, R., Jhonson , R., Vlissides, J., (2000) Padrões de Projeto e Soluções Reutilizáveis de Software Orientado a Objetos, Bookman.

Gmail (2007) www.gmail.com, acessado em 01 de Fevereiro de 2007.

Goodman, D. (1998) Dynamic HTML – The Definitive Reference. O'Reilly,.

Hopkin, J. (2000) Component Primer, Communications of the ACM, Vol. 43, No. 10, pp. 27-30.

Klyne, G., Reynolds, F., Woodrow, C., Ohto, H., Hjelm, J., Butler, M. H., and Tran, L. (2004). Composite capability/preference profiles (cc/pp): Structure and vocabularies 1.0. W3C Recommendation 15 January 2004.

Landay, J., MYERS, B. (1996) Sketching Storyboards to Illustrate Interface Behaviors. In CHI'96 Conference Companion:Human Factors in Computing Systems, Vancouver, Canada,. http://www.w3.org/TR/2004/RECCCPP-struct-vocab-20040115.

Leite, J. C. (2003). Specifying the user interface as an interactive message. In Proceedings of *Human-Computer Interaction International* (HCII'2003), volume 2, pages 716. 720, Heraklion, Creta. Lawrence Erlbaum Associates.

Leite, J. C. (2006). A Semantic Model to Interactive System. In *Proceedings of the International Conference on Organizational Semiotics*. v. 1. p. 81-89.

Limbourg, Q., Vanderdonckt, J. (2004). Usixml: A User Interface Description Language Supporting Multiple Levels Of Independence. In Proceedings of *First International Workshop on Device Independent Web Engineering*, pag 325-338. Munich.

Lira,H.B., Leite, J. C., (2006) Uma ontologia para a descrição da semântica IMML Dissertação (Mestrado) – Programa de Pós-Graduação em Sistema e Computação da Universidade Federal do Rio Grande do Norte, Natal.

Machado, T. L., Leite, J. C., (2006) VISUAL IMML: Um Perfil UML para a Modelagem de Interfaces de Usuário Dissertação (Mestrado) – Programa de Pós-Graduação em Sistema e Computação da Universidade Federal do Rio Grande do Norte, Natal.

Mili, A., Mili, R., Mittermeir, R. (1994) Storing and Retrieving Software Components: A Refinement-Based System. In: *IEEE Transactions on Software Enginnering*. Vol. 23, No 7, pp. 445-460.

Myers, B., Hudson, S. E., and Pausch, R. (2000). Past, present, and future of user interface software tools. *ACM Trans. Comput.-Hum. Interact.*, 7(1):3.28.

Opera (2006) Opera Mini http://www.operamini.com/demo/ - acessado em 12 de Dezembro de 2006

Paterno, F. and Tschelig, M. (2003). Design of usable multi-platform interactive systems. In CHI'03 extended abstracts on *Human factors in computing systems*, pages 872. 873, Ft. Lauderdale, Florida, USA. ACM Press.

Palanque, P., Bastide, R. (1994) Formal specification of HCI for increasing software's ergonomics. ERGONOMICS'94, Warwick, England.

Paterno, F., Mancini, C., Meniconi, S. (1997) ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. Proceedings Interact'97, pp.362-369, Sydney, Chapman&Hall.

Phanouriou, Constantinos (2002) UIML: A Device-Independent User Interface Markup Language Phd Thesis, Virginia Polytechnic Institute, Blackburg, Virginia.

Pietro-Diaz, R., Freeman, P. (1987) Classifying Software for Reuse, *IEEE Software*, Vol. 4 No. 1, pp. 6-16.

Prates, R.O., de Souza, C.S., Barbosa, S.D.J. (2000) A Method for Evaluating the Communicability of User Interfaces. In: *ACM interactions*, Jan-Feb 2000. pp 31-38.

Puerta, A. R. and Eisenstein, J. (1999) Towards a general computational framework for model-based interface development systems In: Proceedings of the 4th *international conference on Intelligent user interfaces*, pages 171.178. ACM Press.

Puerta, A. R. and Eisenstein, J. (2001) XIML: A Common Representation for Interaction Data. In: Proceedings of the 7 th *international conference on Intelligent user interfaces*, pp. 214-215, 2002.

Puerta, A. R. and Szkeley, P. (1994) Model-based interface development: In *Conference companion on Human factors in computing systems*, pp 389-390, Boston, Massachusetts, United States. ACM Press.

Seffah, A. and Javahery, H., editors (2004) Multiple User Interfaces: Cross-Platform Applications and Context-Aware Interface. John Wiley & Sons, Ltd.

Silva, S. F. de S., Leite, J.C. (2007) BRIDGE - Uma Ferramenta para Design de Interfaces de Usuário baseada em Especificações IMML Dissertação (Mestrado) – Programa de Pós-Graduação em Sistema e Computação da Universidade Federal do Rio Grande do Norte, Natal.

Smith , M.K., Welty, C., McGuinness, D.L. (2004) OWL – Web Ontology Language. W3C Recommendation 10 de Fevereiro de 2004.

Souchon, N. and Vanderdonckt, J. (2003) A review of xml-compliant user interface description languages In: Proceedings of the *10th International Conference on Design*, Specification, and Verification of Interactive Systems, Madeira.

Sousa, L. G and Leite, J. C. (2004) XICL An extensible Markup Language for Developing User Interface and Components. In: *Computer-Aided Design of User Interfaces IV*. (ed.R. Jacob, Q. Limbourg and J. Vanderdonckt) Kluwer Academic Publishers.

Sousa, L. G and Leite, J. C. (2004b) Utilizando ontologia para a descrição da interação em componentes de interface de usuário. In: VI Simpósio sobre Fatores Humanos em Sistemas Computacionais, Curitiba.

Sugumaram, V., Sotrey, V.C. A (2003) A Semantic-Based Approach to Component Retrieval. The DATA BASE for Advances in Information Systems – Summer 2003, Vol. 34, No.3.

Thevenin D. and Coutaz J. (1999) Plasticity of User Interfaces: Framework and Research Agenda, in Proceedings of *Interact'99*, vol. 1, Edinburgh: IFIP, IOS Press, pp. 110-117.

Van den Bergh, J., Coninx, K. (2005) Using UML 2.0 and Profiles for Modelling Context-Sensitive User Interfaces. In: Proceedings of the *MoDELS'05 Workshop on Model Driven Development of Advanced User Interfaces*, Montego Bay, Jamaica.

ANEXOS

Anexo I

Este anexo contem o código em IMML da descrição de um IU de uma aplicação bancária.

```
1. <?xml version="1.0" encoding="ISO-8859-1"?>
2. <imml application="BancoRico" xmlns="http://www.dimap.ufrn.br/imml"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
3.     <domain-model>
4.       <domain-objects>
5.          <domain-object name="Conta" type="number"/>
6.          <domain-object name="Agencia" type="number"/>
7.          <domain-object name="Senha" type="number"/>
8.          <domain-object name="DataInicio" type="date"/>
9.          <domain-object name="Data" type="date"/>
10.         <domain-object name="DataFim" type="date"/>
11.         <domain-object name="DataTransacao" type="date"/>
12.         <domain-object name="NomeTransacao" type="text"/>
13.         <domain-object name="ValorTransacao" type="number"/>
14.         <domain-object name="TipoTransacao" type="text"/>
15.         <domain-object name="Saldo" type="number"/>
16.         <domain-object name="CodigoPagamento" type="number"/>
17.         <domain-object name="DataPagamento" type="date"/>
18.         <domain-object name="ValorPagamento" type="number"/>
19.         <domain-object name="NomeDoMes" type="text"/>
20.
21.         <domain-object name="EstatusPagamento" type="finit-set">
22.            <item>Pagamento Efetuado com Sucesso!</item>
23.            <item>Pagamento Não Efetuado!</item>
24.         </domain-object>
25.
26.         <domain-object name="Extrato" type="composed-object">
27.           <item domain-object="Transação"/>
28.           <item domain-object="Saldo"/>
29.         </domain-object>
30.
31.         <domain-object name="Transação" type="table">
32.           <item domain-object="DataTransacao"/>
33.           <item domain-object="NomeTransação"/>
34.           <item domain-object="ValorTransacao"/>
35.           <item domain-object="TipoTransação"/>
36.         </domain-object>
37.
38.         <domain-object name="Mes" type="table">
39.           <item domain-object="NomeDoMes"/>
40.         </domain-object>
41.     </domain-objects>
42.
43.     <domain-functions >
44.
45.       <domain-function name="EmitirExtrato">
46.         <input-operands>
47.           <item domain-object="Conta"/>
```

```
48.            <item domain-object="Agencia"/>
49.             <item domain-object="Senha"/>
50.             <item domain-object="DataInicio"/>
51.             <item domain-object="DataFim"/>
52.            </input-operands>
53.
54.            <output-operands>
55.             <item domain-object="Extrato"/>
56.            </output-operands>
57.
58.           <controls>
59.             <control name="Consultar" from-state="initial" to-
   state="Consultando"/>
60.             <control automatic="yes" from-state="Consultando" to-
   state="final"/>
61.            </controls>
62.            <states>
63.              <state name="initial"/>
64.              <state name="Consultando"/>
65.              <state name="final"/>
66.            </states>
67.
68.          </domain-function>
69.
70. <!-- CONSULTAR SALDO -->
71.          <domain-function name="ConsultarSaldo">
72.             <input-operands>
73.               <item domain-object="Conta"/>
74.               <item domain-object="Agencia"/>
75.               <item domain-object="Senha"/>
76.             </input-operands>
77.
78.             <output-operands>
79.               <item domain-object="Saldo"/>
80.             </output-operands>
81.
82.             <controls>
83.               <control name="Consultar" from-state="Disponível" to-
   state="Consultando"/>
84.               <control automatic="yes" from-state="Consultando" to-
   state="ExibindoSaldo"/>
85.             </controls>
86.
87.             <states>
88.               <state name="Disponivel"/>
89.               <state name="Consultando"/>
90.               <state name="Exibindo"/>
91.             </states>
92.            </domain-function>
93.
94.             <!-- EFETUAR PAGAMENTO -->
95.            <domain-function name="EfetuarPagamento">
96.               <input-operands>
97.                 <item domain-object="Conta"/>
98.                 <item domain-object="Agencia"/>
99.                 <item domain-object="Senha"/>
100.                     <item domain-object="CodigoPagamento"/>
101.                <item domain-object="DataPagamento"/>
102.                <item domain-object="ValorPagamento"/>
103.             </input-operands>
104.
105.             <output-operands>
106.               <item domain-objects="Saldo"/>
107.             </output-operands>
108.
109.             <controls>
110.               <control name="Pagar" from-state="Disponível" to-
   state="EfetuandoPagamento"/>
```

```
111.                <control automatic="yes" from-state="EfetuandoPagamento" to-
    state="ExibindoResultadoDoPagamento"/>
112.              </controls>
113.
114.              <states>
115.                <state name="Disponivel"/>
116.                <state name="EfetuandoPagamento"/>
117.                <state name="ExibindoResultadoDoPagamento"/>
118.              </states>
119.
120.            </domain-function>
121.
122.  <!-- EFETUAR PAGAMENTO COM SALDO -->
123.            <domain-function name="EfetuarPagamentoComSaldo"
    extends="EfetuarPagamento">
124.
125.              <controls>
126.                <control name="Consultar" from-state="Disponivel" to-
    state="ConsultandoSaldo"/>
127.                <control automatic="yes" from-state="ConsultandoSaldo" to-
    state="ExibindoSaldo"/>
128.                <control name="Pagar" from-state="ExibindoSaldo" to-
    state="EfetuandoPagamento"/>
129.              </controls>
130.
131.              <states>
132.                <state name="ConsultandoSaldo"/>
133.                <state name="ExibindoSaldo"/>
134.              </states>
135.          </domain-function>
136.
137.      </domain-functions>
138.    </domain-model>
139.
140.  <!-- INTERACTION MODEL -->
141.    <interaction-model>
142.       <tasks>
143.          <task name="TransacaoBancaria">
144.             <select>
145.                 <do task="EmitirExtrato"/>
146.                 <do task="ConsultarSaldo"/>
147.                 <do task="EfetuarPagamentoSimples"/>
148.                 <do task="EfetuarPagamentoComSaldoAntes"/>
149.             </select>
150.          </task>
151.
152.          <task name="EmitirExtrato">
153.            <sequence>
154.              <do function-command="Extrato"/>
155.              <do function-result="SaidaExtrato"/>
156.            </sequence>
157.          </task>
158.
159.          <task name="ConsultarSaldo">
160.            <sequence>
161.              <do function-command="Saldo"/>
162.              <do function-result="SaidaSaldo"/>
163.            </sequence>
164.          </task>
165.
166.          <task name="EfetuarPagamentoSimples">
167.             <sequence>
168.                 <do function-command="PagamentoSimples"/>
169.                 <do function-result="SaidaPagamento"/>
170.             </sequence>
171.          </task>
172.
173.          <task name="EfetuarPagamentoComSaldoAntes">
```

```
174.            <sequence>
175.                <do function-command="Saldo"/>
176.                <do function-command="PagamentoSimples"/>
177.                <do function-result="SaidaPagamento"/>
178.            </sequence>
179.         </task>
180.     </tasks>
181.
182.     <function-commands>
183.
184.       <function-command name="Extrato" domain-function="EmitirExtrato">
185.        <select>
186.         <sequence>
187.           <enter-information domain-object="Conta"/>
188.           <enter-information domain-object="Agencia"/>
189.           <enter-information domain-object="Senha"/>
190.
191.           <select>
192.             <join>
193.               <perceive-information>Escolha o mes que deseja consultar
    extrato</perceive-information>
194.               <enter-information domain-object="Mes"/>
195.             </join>
196.             <join>
197.               <perceive-information>Digite a data de inicio e final, ou
    somente a data de inicio</perceive-information>
198.                <enter-information domain-object="DataInicio"/>
199.                <enter-information domain-object="DataFim"/>
200.             </join>
201.
202.           </select>
203.           <activate control="Consultar"/>
204.         </sequence>
205.
206.         <go direction="away"/>
207.       </select>
208.     </function-command>
209.
210.     <function-command name="Saldo" domain-function="ConsultarSaldo">
211.        <select>
212.          <sequence>
213.            <enter-information domain-object="Conta"/>
214.            <enter-information domain-object="Agencia"/>
215.            <enter-information domain-object="Senha"/>
216.            <activate control="Consultar"/>
217.          </sequence>
218.          <go direction="away"/>
219.        </select>
220.      </function-command>
221.
222.     <function-command name="PagamentoSimples" domain
    function="EfetuarPagamento">
223.        <select>
224.          <sequence>
225.            <enter-information domain-object="Conta"/>
226.            <enter-information domain-object="Agencia"/>
227.            <enter-information domain-object="Senha"/>
228.            <enter-information domain-object="CodigoPagamento"/>
229.            <enter-information domain-object="DataPagamento"/>
230.            <enter-information domain-object="ValorPagamento"/>
231.            <activate control="Pagar"/>
232.          </sequence>
233.          <go direction="away"/>
234.        </select>
235.      </function-command>
236.
237.      <function-command name="PagamentoComSaldoAntes" domain-
    function="EfetuarPagamentoComSaldoAntes">
```

```
238.         <select>
239.            <sequence>
240.               <join>
241.                 <enter-information domain-object="Conta"/>
242.                 <enter-information domain-object="Agencia"/>
243.                  <enter-information domain-object="Senha"/>
244.               </join>
245.               <activate control="Consultar"/>
246.               <join>
247.                 <perceive-information>O seu saldo é:</perceive-information>
248.                 <perceive-information domain-object="Saldo"/>
249.                 <perceive-information>Deseja continuar o pagamento?</perceive-
    information>
250.                 <select>
251.                    <go direction="away"/>
252.                    <join>
253.                       <go direction="ahead"/>
254.                       <sequence>
255.                          <enter-information domain-object="CodigoPagamento"/>
256.                          <enter-information domain-object="DataPagamento"/>
257.                          <enter-information domain-object="ValorPagamento"/>
258.                       </sequence>
259.                       <activate control=" Pagar"/>
260.        <!-- Do estado Consultando Saldo para efetuando pagamento<navigate
    direction="go"/-->
261.
262.                     </join>
263.                   </select>
264.                 </join>
265.               </sequence>
266.               <go direction="away"/>
267.             </select>
268.          </function-command>
269.
270.       </function-commands>
271.
272.       <function-results>
273.
274.          <function-result name="SaidaExtrato" domain-function="EmitirExtrato">
275.            <perceive-informaiton domain-object="Extrato"/>
276.          </function-result>
277.
278.          <function-result name="SaidaSaldo" domain-function="ConsultarSaldo">
279.             <sequence>
280.               <perceive-information> O seu saldo é: </perceive-information>
281.               <perceive-informaiton domain-object="Saldo"/>
282.             </sequence>
283.           </function-result>
284.
285.           <function-result name="SaidaPagamento" domain-
    function="EfetuarPagamento">
286.             <sequence>
287.               <perceive-information>Pagamento efetuado com Sucesso!</perceive-
    information>
288.               <perceive-informatio>O seu saldo é:</perceive-informatio>
289.               <perceive-informaiton domain-object="Saldo"/>
290.             </sequence>
291.           </function-result>
292.
293.           <function-result name="SaidaPagamentoComSaldo" domain-
    function="EfetuarPagamentoComSaldoAntes">
294.             <sequence>
295.               <perceive-information>Pagamento efetuado com Sucesso!</perceive-
    information>
296.               <perceive-information>O seu saldo é:</perceive-information>
297.               <perceive-informaiton domain-object="Saldo"/>
298.             </sequence>
299.           </function-result>
```

```
300.       </function-results>
301.     </interaction-model>
302.
303.     <communication-model>
304.
305.       <task-environment task="TransacaoBancaria" platform="Terminal">
306.         <frame>
307.           <frame name="grupoSuperior" title="Banco Rico"
   orientation="horizontal" align="left">
308.             <push-button label="Saldo" show="frameSaldo"
   target="grupoInferior"/>
309.             <push-button label="Extrato" show="frameExtrato"
   target="grupoInferior"/>
310.             <push-button label="Pagamento" show="framePagamento"
   target="grupoInferior"/>
311.             <push-button label="Pagamento Com Saldo Antes"
   show="framePagamentoComSaldo" target="grupoInferior"/>
312.           </frame>
313.
314.           <frame name="grupoInferior">
315.             <command-panel function-command="Saldo" name="frameSaldo"
   title="Banco Rico" orientation="vertical" align="left">
316.               <edit-box label="Agencia" domain-object="Agencia"/>
317.               <edit-box label="Conta" domain-object="Conta"/>
318.               <edit-box label="Senha" domain-object="Senha"/>
319.               <group orientation="horizontal" align="center">
320.                 <push-button label="Consultar" control="Consultar"
   transition="MostraSaldo" target="grupoInferior"/>
321.                 <push-button label="Cancelar" hide="this"/>
322.               </group>
323.             </command-panel>
324.
325.             <display-area function-result="SaidaSaldo" name="MostraSaldo"
   orientation="vertical" align="left">
326.               <text label="Data" domain-object="Data"/>
327.               <text label="Agencia" domain-object="Agencia"/>
328.               <text label="Conta" domain-object="Conta"/>
329.               <text label="Saldo" domain-object="Saldo"/>
330.               <push-button label="Sair" hide="this"/>
331.             </display-area>
332.
333.             <command-panel function-command="Extrato" name="frameExtrato"
   orientation="vertical" align="left">
334.               <group orientation="horizontal">
335.                 <edit-box label="Agencia" domain-object="Agencia"/>
336.                 <edit-box label="Conta" domain-object="Conta"/>
337.               </group>
338.
339.               <edit-box label="Senha" domain-object="Senha"/>
340.               <group orientation="vertical" align="center">
341.                 <text>Escolha o mes que deseja ver Extrato</text>
342.                 <combo-box name="Meses" label="NomeMeses" label-
   position="Center" domain-object="Mes"/>
343.               </group>
344.
345.               <group>
346.                 <text>Ou digite no campo uma data inicial e final, ou
   somente uma final</text>
347.                 <edit-box label="DataInicial" domain-object="DataInicio"/>
348.                 <edit-box label="DataFinal" domain-object="DataFim"/>
349.               </group>
350.
351.               <group orientation="horizontal" align="center">
352.                 <push-button label="Consultar" control="Consultar"
   transition="MostraExtrato" target="grupoInferior"/>
353.                 <push-button label="Cancelar" hide="this"/>
354.               </group>
355.             </command-panel>
```

```
356.
357.              <display-area function-result="SaidaExtrato"
      name="MostraExtrato" orientation="vertical" align="left">
358.                <table domain-object="Extrato" head-position="top">
359.                  <column label="NomeTransacao" domain-
      object="NomeTransacao"/>
360.                  <column label="DataTransacao" domain-
      object="DataTransacao"/>
361.                  <column label="ValorTransação" domain-
      object="ValorTransacao"/>
362.                  <column label="TipoTransacao" domain-
      object="TipoTransacao"/>
363.                </table>
364.
365.                <text label="Saldo" domain-object="Saldo"/>
366.                <push-button label="Sair" hide="this"/>
367.
368.              </display-area>
369.
370.            <command-panel function-command="PagamentoSimples"
      name="framePagamento" orientation="vertical" align="left">
371.
372.              <group orientation="horizontal">
373.                <edit-box name="Agencia" domain-object="Agencia"/>
374.                <edit-box name="Conta" domain-object="Conta"/>
375.              </group>
376.
377.              <edit-box label="DataPagamento" domain-object="DataPagamento"/>
378.              <edit-box label="Valor do Pagemento" domain-
      object="ValorPagamento"/>
379.              <edit-box label="CodigoPagamento" domain-
      object="CodigoPagamento"/>
380.               <edit-box name="Senha" domain-object="Senha"/>
381.               <group orientation="horizontal" align="center">
382.                 <push-button label="Pagar" control="Pagar"
   transition="MostraPagamento" target="grupoInferior"/>
383.                 <push-button label="Cancelar" hide="this"/>
384.               </group>
385.
386.            </command-panel>
387.
388.            <display-area function-result="SaidaPagamento"
   name="MostraPagamento" orientation="vertical" align="left">
389.               <text label="Resultado do Pagamento" domain-
   object="EstatusPagamento"/>
390.               <text label="Saldo" domain-object="Saldo"/>
391.               <push-button label="Sair" hide="this"/>
392.            </display-area>
393.
394.            <command-panel name="cpPagamentoComSaldo" function-
   command="PagamentoComSaldoAntes">
395.
396.               <frame name="framePagamentoComSaldo" orientation="vertical"
   align="left">
397.                 <group orientation="horizontal">
398.                   <edit-box label="Agencia" domain-object="Agencia"/>
399.                   <edit-box label="Conta" domain-object="Conta"/>
400.                 </group>
401.                 <edit-box label="Senha" domain-object="Senha"/>
402.
403.                 <group orientation="horizontal" align="center">
404.                   <push-button label="ConsultarSaldo" control="Consultar"
   transition="MostraSaldo" target="grupoInferior"/>
405.                   <push-button label="Cancelar" hide="this"/>
406.                 </group>
407.
408.                 <text label="Saldo" domain-object="Saldo"/>
409.                 <text>Deseja Continuar?/</text>
```

```
410.                <push-button label="Sim" direction="go"
   transition="frameContinuarPagamento"/>
411.                <push-button label="Nao" hide="this"/>
412.              </frame>
413.
414.              <frame name="frameContinuarPagamento" orientation="vertical"
   align="left">
415.                <edit-box label="DataPagamento" domain-
   object="DataPagamento"/>
416.                <edit-box label="ValorPagemento" domain-
   object="ValorPagamento"/>
417.                <edit-box label="CodigoPagamento" domain-
   object="CodigoPagamento"/>
418.
419.                <group orienthisign="center">
420.                  <push-button label="Pagar" control="Pagar"
   transition="MostraPagamentoComSaldo" target="grupoInferior"/>
421.                  <push-button label="Cancelar" hide="this"/>
422.                </group>
423.              </frame>
424.            </command-panel>
425.
426.            <display-area function-result="SaidaPagamentoComSaldo"
   name="MostraPagamentoComSaldo" orientation="vertical" align="left">
427.              <text label="Resultado do Pagamento" domain-
   object="EstatusPagamento"/>
428.              <text label="Saldo" domain-object="Saldo"/>
429.              <push-button label="Sair" hide="this"/>
430.            </display-area>
431.          </frame>
432.        </frame>
433.     </task-environment>
434.
435.     <!-- Celular -->
436.
437.     <task-environment name="PagamentoSimples" task="EfetuarPagamentoSimples"
   platform="celular">
438.        <command-panel function-command="PagamentoSimples">
439.          <frame name="primeiro" orientation="vertical" align="left">
440.            <edit-box label="Agencia" domain-object="Agencia"/>
441.            <push-button label="Proximo" transition="segundo"/>
442.          </frame>
443.          <frame name="segundo" orientation="vertical" align="left">
444.            <edit-box label="Conta" domain-object="Conta"/>
445.            <push-button label="Proximo" transition="terceiro"/>
446.          </frame>
447.          <frame name="terceiro" orientation="vertical" align="left">
448.            <edit-box label="Senha" domain-object="Senha"/>
449.            <push-button label="Proximo" transition="quarto"/>
450.          </frame>
451.          <frame name="quarto" orientation="vertical" align="left">
452.            <edit-box label="Codigo" domain-object="CodigoPagamento"/>
453.            <push-button label="Proximo" transition="quinto"/>
454.          </frame>
455.          <frame name="quinto" orientation="vertical" align="left">
456.            <edit-box label="Valor" domain-object="ValorPagamento"/>
457.            <push-button label="Proximo" transition="sexto"/>
458.          </frame>
459.          <frame name="sexto" orientation="vertical" align="left">
460.            <edit-box label="Data" domain-object="DataPagamento"/>
461.            <push-button label="Pagar" control="Pagar" transition="setimo"/>
462.          </frame>
463.        </command-panel>
464.
465.        <display-area function-result="SaidaPagamento">
466.          <frame name="setimo" orientation="vertical" align="left">
467.            <text label="Resultado do Pagamento" domain-
   object="EstatusPagamento"/>
```

```
468.            <text label="Saldo" domain-object="Saldo"/>
469.            <push-button label="Sair" hide="this"/>
470.          </frame>
471.        </display-area>
472.      </task-environment>
473.    </communication-model>
474.
475. </imml>
```

Anexo II

A seguir se encontra o código em XICL da interface de usuário tratada no capítulo 3. Este código é o código completo, que foi mostrado em parte na figura 4.3.

```
1. <?xml version="1.0" encoding="ISO-8859-1"?>
2. <XICL version="1.0" >
3.
4.   <IMPORT url="lib1.xml"/>
5.
6.   <INTERFACE >
7.    <menuBar>
8.     <Menu menuCaption="File" id="File">
9.      <MenuOption caption="New"  onchoose="new();" menuCaption="File"/>
10.     <MenuOption caption="Open" onchoose="open();" menuCaption="File" />
11.     <MenuOption caption="Save" onchoose="save();"  menuCaption="File"/>
12.     <MenuOption caption="Close" onchoose="mes1.show();"
   menuCaption="File"/>
13.    </Menu>
14.    <Menu menuCaption="Edit" id="Edit">
15.     <MenuOption caption="Copy"  onchoose="copy();" menuCaption="Edit"
   />
16.     <MenuOption caption="Paste"  onchoose="paste();" menuCaption="Edit"
   />
17.    </Menu>
18.   </MenuBar>
19.  <ConfirmBox id="mes1" title="Close?" onConfirm="close()" > Do you
      really want to close   this application? </ConfirmBox>
20.  <textarea rows="12" cols="60">
21.
22.     <textarea>
23.
24.   <SCRIPT>
25.    <!--
26.       function new(){
27.              alert('this function create a new file')
28.       }
29.       function open(){
30.              alert('this function open a new file')
31.       }
32.       function save(){
33.              alert('this function save the file')
34.       }
```

```
35.
36.       function copy(){
37.              alert('this function copy the text')
38.       }
39.
40.       function paste(){
41.              alert('this function paste the text')
42.
43.       }
44.       function close(){
45.              alert('this function close the application')
46.       }
47.  -->
48.   </SCRIPT>
49.  </INTERFACE>
50.</XICL>
```

Anexo III

A seguir está o código da biblioteca utilizada na descrição apresentada no anexo II. Os componentes descritos nesta biblioteca são: SimpleWindow, Window, ConfirmBox, MessageBox, ToolBar, Button, Menu, MenuOption e MenuBar.

```
1. <?xml version="1.0" encoding="ISO-8859-1"?>
2. <XICL author="Lirisnei " >
3.
4.     <COMPONENT name="SimpleWindow" >
5.       <STRUCTURE>
6.         <div id="$id">
7.           <div style="visibility:hidden" id="ch$id">
8.              <div style="position:absolute; top:$top; left:$left; ">
9.                 <table border="1" style=" width:$width; height:$height;
   background-color:white">
10.
11.                    <tr >
12.                       <td colspan="2">
13.                         <COMPONENT ref="ANY"/>
14.                        </td>
15.                    </tr>
16.                </table>
17.             </div>
18.           </div>
19.         </div>
20.       </STRUCTURE>
21.
22.       <EVENTS>
23.          <EVENT name="beforeClose"   trigged="$beforeClose;
   closeWindow();  " />
24.          <EVENT name="afterClose" trigged=" closeWindow();
   $afterClose"  />
25.       </EVENTS>
26.
27.
28.       <SCRIPT><!--
29.          function close()
30.           {
31.              objId=this.id
```

```
32.
33.               var div=document.getElementById(objId)
34.               var divch=document.getElementById("ch"+objId)
35.               if ( (div.nodeName!="div") && (div.nodeName!="DIV"))
36.                    return
37.              div.innerHTML="<div style=\"visibility:hidden\"
                 id=\"ch"+objId+"\">"+divch.innerHTML+"</div>"
38.               }
39.
40.          function show()
41.           {
42.              objId=this.id
43.               var div=document.getElementById(objId)
44.               var divch=document.getElementById("ch"+objId)
45.
46.               if ( (div.nodeName!="div") && (div.nodeName!="DIV"))
47.               return
48.               div.innerHTML="<div style=\"visibility:visible\"
   id=\"ch"+objId+"\">"+divch.innerHTML+"</div>"
49.                    }
50.        -->
51.      </SCRIPT>
52.

53.
54.      <METHODS>
55.        <METHOD name="close"  />
56.        <METHOD name="show"  />
57.      </METHODS>
58.
59.
60.    </COMPONENT>
61.

62.    <COMPONENT name="Window" >
63.      <STRUCTURE>
64.        <div id="$id">
65.          <div style="visibility:hidden" id="ch$id">
66.             <div style="position:absolute; top:$top; left:$left; ">
67.                <table border="1" style=" width:$width; height:$height;
   background-color:white">
68.                  <tr height="12">
69.                    <td align="left" border="0">
70.                      <a  style="color:blue; font-size:12 background-
   color:blue">$title</a>
71.                    </td>
72.                    <td width="2" align="right" border="0">
73.                      <a  onclick="$id.close();" name="figclose">
74.                        <img src="close.gif"/>
75.                      </a>
76.                    </td>
77.                  </tr>
78.                  <tr >
79.                    <td colspan="2">
80.                      <COMPONENT ref="ANY"/>
81.                    </td>
82.                  </tr>
83.                </table>
84.              </div>
85.            </div>
86.          </div>
87.        </STRUCTURE>
88.
```

```
89.
90.       <EVENTS>
91.          <EVENT name="beforeClose"   trigged="$beforeClose;
   closeWindow();  " />
92.          <EVENT name="afterClose" trigged=" closeWindow();
   $afterClose"  />
93.       </EVENTS>
94.
95.       <SCRIPT><!--
96.          function close()
97.           {
98.              objId=this.id
99.              var div=document.getElementById(objId)
100.                  var divch=document.getElementById("ch"+objId)
101.                  if ( (div.nodeName!="div") && (div.nodeName!="DIV"))
102.                       return
103.             div.innerHTML="<div style=\"visibility:hidden\"
                 id=\"ch"+objId+"\">"+divch.innerHTML+"</div>"
104.                  }
105.
106.               function show()
107.                {
108.                  objId=this.id
109.                  var div=document.getElementById(objId)
110.                  var divch=document.getElementById("ch"+objId)
111.
112.                  if ( (div.nodeName!="div") && (div.nodeName!="DIV"))
113.                  return
114.                  div.innerHTML="<div style=\"visibility:visible\"
   id=\"ch"+objId+"\">"+divch.innerHTML+"</div>"
115.                       }
116.          -->
117.       </SCRIPT>
118.

119.
120.       <METHODS>
121.         <METHOD name="close"  />
122.         <METHOD name="show"  />
123.       </METHODS>
124.
125.
126.
127.   </COMPONENT>
128.
129.    <COMPONENT name=“ConfirmBox”  extends=“window”>
130.
131.      <PROPERTIES>
132.        <PROPERTY name=“title” dataType=“string” />
133.      </PROPERTIES>
```

```
134.
135.      <STRUCTURE>
136.        <table width="100%" border="0">
137.          <tr>
138.            <td width="150"  >
139.               <COMPONENT ref=“any” >
140.            </td>
141.          </tr>
142.          <tr>
143.            <td align="center">
144.               <input type="button" name="bOk" value="Yes"  />
145.               <input type="button" name="bCanc" value="No" />
146.            </td>
147.          </tr>
148.        </table>
149.      </STRUCTURE>
150.      <EVENTS>
151.        <EVENT name=“onConfirm” trigged=“bOk.onclick; $id.hide();
   $onConfirm;” />
152.        <EVENT name=“onCancel” trigged=“bCanc.onclick; $id.hide();
   $onCancel;” />
153.      </EVENTS>
154.
155.    </COMPONENT>

156.    <COMPONENT name="MessageBox" extends="Window">
157.
158.        <STRUCTURE>
159.          <table width="100%" border="0" >
160.            <form>
161.              <tr>
162.                <COMPONENT ref="ANY"/>
163.              </tr>
164.              <tr>
165.                <td align="right">
166.                  <input type="button" name="bOk" value=" Yes"
   onclick="$id.close();" />
167.                </td>
168.                <td align="left" >
169.                  <input type="button" name="bCanc" value=" No"
   onclick="$id.close();" />
170.                </td>
171.              </tr>
172.            </form>
173.          </table>
174.        </STRUCTURE>
175.
176.         <EVENTS>
177.           <EVENT  name="onConfirm" trigged="bOk.onclick;
   $onConfirm;"/>
178.           <EVENT  name="onCancel"  trigged=" bCanc.onclick;
   $onCancel;"/>
179.         </EVENTS>
180.
181.       </COMPONENT>
182.
```

```
183.
184.     <!-- TOOLBAR DESCRIPTION -->
185.
186.        <COMPONENT name="toolBar" >
187.
188.           <STRUCTURE>
189.             <table style="z-index:0;">
190.               <tr>
191.                 <COMPONENT ref="ANY"/>
192.               </tr>
193.             </table>
194.           </STRUCTURE>
195.        </COMPONENT>
196.
197.        <COMPONENT name="Button" >
198.          <STRUCTURE>
199.            <td width="$width" height="$height">
200.              <image  width="10" height="10" name="img1"  src="$src"/>
201.            </td>
202.          </STRUCTURE>
203.
204.          <EVENTS>
205.            <EVENT name="onchose" when="after"
   objectEvent="img1.onclick" />
206.          </EVENTS>
207.
208.        </COMPONENT>
209.        <!-- Menu -->
210.
211.        <COMPONENT name="Menu" >
212.
213.          <STRUCTURE>
214.            <td >
215.              <a style="color:blue; font-size:20;"
   onClick="$id.show();">
216.                  $menuCaption
217.               </a>
218.             <div>
219.               <div style="visibility:hidden;"  id="$id" >
220.                 <table style="background-color:bbbbdd" >
221.
222.                   <COMPOSITION>
223.                     <tr>
224.                       <td >
225.                         <COMPONENT ref="menuOption"/>
226.                       </td>
227.                     </tr>
228.                   </COMPOSITION>
229.
230.                 </table>
231.               </div>
232.             </div>
233.           </td>
234.         </STRUCTURE>
235.
236.         <METHODS>
237.           <METHOD name="show"  />
238.           <METHOD name="hide"  />
239.         </METHODS>
240.
```

```
241.
242.          <SCRIPT>
243.          <!--
244.             var visible=false;
245.             function show(){
246.                  if (visible)
247.                  visible.hide()
248.                  visible=this
249.
250.                  div =document.getElementById(visible.id)
251.                  div.parentNode.innerHTML="<div
                   style=\"visibility:visible;\" id=\""+visible.id+"\"
                   >"+div.innerHTML+"</div>"
252.                         }
253.
254.             function hide(){
255.
256.              div =document.getElementById(visible.id)
257.              div.parentNode.innerHTML="<div style=visibility:hidden;
                   id=\""+visible.id+"\"  >"+div.innerHTML+"</div>"
258.                          visible=false
259.             }
260.          -->
261.          </SCRIPT>
262.        </COMPONENT>
263.
264.        <COMPONENT name="MenuOption" >
265.
266.          <STRUCTURE>
267.            <a style="color:red;" onMouseDown="$menuCaption.hide();"
            onMouseUp="      $onClick;" name="text" >
268.                $caption
269.             </a>
270.          </STRUCTURE>
271.
272.
273.          <EVENTS>
274.            <EVENT name="onchoose" trigged=" $onchoose;
   text.onMouseDown;  "/>
275.          </EVENTS>
276.
277.        </COMPONENT>
278.
279.        <COMPONENT name="MenuBar" >
280.           <STRUCTURE>
281.             <table style="z-index:100;">
282.               <tr>
283.                 <COMPOSITION>
284.                   <td >
285.                     <COMPONENT ref="menu"/>
286.                   </td>
287.                 </COMPOSITION>
288.               </tr>
289.             </table>
290.           </STRUCTURE>
291.        </COMPONENT>
292.
293.   </XICL>
```

Anexo IV

A seguir se encontra o código em XICL gerado, pelo compilador IMML, na compilação da IU da aplicação bancária descrita em IMML, cujo código está no anexo I.

```
1.   <?xml version="1.0" encoding="ISO-8859-1" ?>
2.   <XICL>
3.     <IMPORT url="lib1.xml" />
4.     <INTERFACE id="0">
5.        <frameset id="1">
6.          <frame align="left" id="grupoSuperior">
7.             <input id="2" onMouseUp="javascript:frameSaldo.show();"
   type="button" value="Saldo" />
8.             <input id="3" onMouseUp="javascript:frameExtrato.show();"
   type="button" value="Extrato" />
9.             <input id="4" onMouseUp="javascript:framePagamento.show();"
   type="button" value="Pagamento" />
10.            <input id="5"
   onMouseUp="javascript:framePagamentoComSaldo.show();" type="button"
   value="Pagamento Com Saldo Antes" />
11.         </frame>
12.         <frame id="grupoInferior" />
13.       </frameset>
14.
15.       <simpleWindow align="left" id="frameSaldo">
16.         <br />
17.         <span>Agencia</span>
18.         <br />
19.         <input id="6" type="text" />
20.         <br />
21.         <span>Conta</span>
22.         <br />
23.         <input id="7" type="text" />
24.         <br />
25.         <span>Senha</span>
26.         <br />
27.         <input id="8" type="text" />
28.         <br />
29.         <div align="center" id="9">
30.           <input id="10" onMouseUp="javascript:frameSaldo.close();
   MostraSaldo.show();" type="button" value="Consultar" />
31.           <input id="11" onMouseUp="javascript:frameSaldo.close();"
   type="button" value="Cancelar" />
32.         </div>
33.       </simpleWindow>
34.
35.       <simpleWindow align="left" id="MostraSaldo">
36.         <br />
```

```
37.          <span>Data</span>
38.          <br />
39.          <span>Agencia</span>
40.          <br />
41.          <span>Conta</span>
42.          <br />
43.          <br />
44.          <span>Saldo</span>
45.          <br />
46.          <input id="12" onMouseUp="javascript:MostraSaldo.close();"
   type="button" value="Sair" />
47.        </simpleWindow>
48.
49.        <simpleWindow align="left" id="frameExtrato">
50.          <br />
51.          <div id="13">
52.            <span>Agencia</span>
53.            <input id="14" type="text" />
54.            <span>Conta</span>
55.            <input id="15" type="text" />
56.          </div>
57.          <span>Senha</span>
58.          <input id="16" type="text" />
59.          <div align="center" id="17">
60.            <br />
61.            <span>Escolha o mes que deseja ver Extrato</span>
62.            <br />
63.            <span>NomeMeses</span>
64.            <br />
65.            <select id="Meses" />
66.          </div>
67.          <br />
68.          <div id="18">
69.            <span>Ou digite no campo uma data inicial e final, ou
   somente uma final</span>
70.            <span>DataInicial</span>
71.            <input id="19" type="text" />
72.            <span>DataFinal</span>
73.            <input id="20" type="text" />
74.          </div>
75.          <div align="center" id="21">
76.            <input id="22" onMouseUp="javascript:frameExtrato.close();
   MostraExtrato.show();" type="button" value="Consultar" />
77.            <input id="23" onMouseUp="javascript:frameExtrato.close();"
   type="button" value="Cancelar" />
78.          </div>
79.        </simpleWindow>

80.        <simpleWindow align="left" id="MostraExtrato">
81.          <table id="24">
82.            <br />
83.            <span>NomeTransacao</span>
84.            <br />
85.            <td id="25" />
86.            <br />
87.            <span>DataTransacao</span>
88.            <br />
89.            <td id="26" />
90.            <br />
91.            <span>ValorTransação</span>
92.            <br />
93.            <td id="27" />
94.            <br />
95.            <span>TipoTransacao</span>
96.            <br />
97.            <td id="28" />
98.          </table>
99.           <br />
```

```
100.         <span>Saldo</span>
101.         <br />
102.         <input id="29" onMouseUp="javascript:MostraExtrato.close();"
   type="button" value="Sair" />
103.       </simpleWindow>
104.
105.       <simpleWindow align="left" id="framePagamento">
106.         <div id="30">
107.           <input id="Agencia" type="text" />
108.           <input id="Conta" type="text" />
109.         </div>
110.
111.         <span>DataPagamento</span>
112.         <input id="31" type="text" />
113.         <span>Valor do Pagemento</span>
114.         <input id="32" type="text" />
115.         <span>CodigoPagamento</span>
116.         <input id="33" type="text" />
117.         <input id="Senha" type="text" />
118.         <div align="center" id="34">
119.           <input id="35" onMouseUp="javascript:framePagamento.close();
   MostraPagamento.show();" type="button" value="Pagar" />
120.           <input id="36"
   onMouseUp="javascript:framePagamento.close();" type="button"
   value="Cancelar" />
121.         </div>
122.       </simpleWindow>
123.
124.       <simpleWindow align="left" id="MostraPagamento">
125.         <br />
126.         <span>Resultado do Pagamento</span>
127.         <br />
128.         <span>Saldo</span>
129.         <br />
130.         <input id="37" onMouseUp="javascript:MostraPagamento.close();"
   type="button" value="Sair" />
131.       </simpleWindow>
132.
133.       <simpleWindow id="framePagamentoComSaldo">

134.         <frame align="left" id="frameSaldoComSaldo">
135.         <br />
136.         <div id="38">
137.           <span>Agencia</span>
138.           <input id="39" type="text" />
139.           <span>Conta</span>
140.           <input id="40" type="text" />
141.         </div>
142.         <span>Senha</span>
143.         <input id="41" type="text" />
144.
145.         <div align="center" id="42">
146.           <input id="43"
   onMouseUp="javascript:frameSaldoComSaldo.close(); MostraSaldo.show();"
   type="button" value="ConsultarSaldo" />
147.           <input id="44"
   onMouseUp="javascript:frameSaldoComSaldo.close();" type="button"
   value="Cancelar" />
148.         </div>
149.
150.         <span>Saldo</span>
151.         <span>Deseja Continuar?/</span>
152.         <input id="45"
   onMouseUp="javascript:frameSaldoComSaldo.close();
   frameContinuarPagamento.show();" type="button" value="Sim" />
153.         <input id="46"
   onMouseUp="javascript:frameSaldoComSaldo.close();" type="button"
   value="Nao" />
```

```
154.      </frame>
155.
156.      <frame align="left" id="frameContinuarPagamento">
157.        <br />
158.        <span>DataPagamento</span>
159.        <br />
160.        <input id="47" type="text" />
161.        <br />
162.        <span>ValorPagemento</span>
163.        <br />
164.        <input id="48" type="text" />
165.        <br />
166.        <span>CodigoPagamento</span>
167.        <br />
168.        <input id="49" type="text" />
169.        <br />
170.        <div align="center" id="50">
171.          <input id="51"
    onMouseUp="javascript:frameContinuarPagamento.close();
    MostraPagamentoComSaldo.show();" type="button" value="Pagar" />
172.          <input id="52"
    onMouseUp="javascript:frameContinuarPagamento.close();" type="button"
    value="Cancelar" />
173.        </div>
174.
175.      </frame>
176.    </simpleWindow>

177.    <simpleWindow align="left" id="MostraPagamentoComSaldo">
178.      <br />
179.      <span>Resultado do Pagamento</span>
180.      <br />
181.      <br />
182.      <span>Saldo</span>
183.      <br />
184.      <br />
185.      <input id="53"
    onMouseUp="javascript:MostraPagamentoComSaldo.close();" type="button"
    value="Sair" />
186.    </simpleWindow>
187.
188.   </INTERFACE>
189.
190.   </XICL>
```