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

Uma Arquitetura de Referência para o Desenvolvimento de Sistemas Interativos Multiplataformas

FABÍOLA MARIZ DA FONSECA

Natal – RN

Agosto de 2008

FABÍOLA MARIZ DA FONSECA

UMA ARQUITETURA DE REFERÊNCIA PARA O DESENVOLVIMENTO DE SISTEMAS INTERATIVOS MULTIPLATAFORMAS

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 obtenção do título de Mestre em Sistemas e Computação

Orientador: Professor Dr. Jair Cavalcanti Leite, D.Sc. – UFRN

Área de concentração: Engenharia de Software

Natal – RN
Agosto de 2008

Catalogação da Publicação na Fonte. UFRN / SISBI / Biblioteca Setorial Especializada do Centro de Ciências Exatas e da Terra – CCET.

Fonseca, Fabíola Mariz da.
Uma arquitetura de referência para o desenvolvimento de sistemas interativos multiplataformas / Fabíola Mariz da Fonseca. – Natal, 2008.
82 f. : il.

Orientador: Prof. Dr. Jair Cavalcanti 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. Engenharia de software - Dissertação. 2. Sistemas interativos multiplataformas - Dissertação. 3. IMML – Dissertação. 4. Solução arquitetural – Dissertação. I. Leite, Jair Cavalcanti. II. Título

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, D. Sc – UFRN
Orientador

_____
Profa. Dra. Thais Vasconcelos Batista, D.Sc – UFRN
Coordenadora do Programa

Banca Examinadora

_____
Prof. Dr. Jair Cavalcanti Leite, D.Sc. – UFRN
Presidente

_____
Prof. Dr. Paulo de Figueiredo Pires, D.Sc. – UFRN

_____
Profa. Dra. Maria Elizabeth Sucupira Furtado, D.Sc. – UNIFOR

Natal – RN
Agosto de 2008

AGRADECIMENTOS

Agradeço a toda a minha família pelo apoio, paciência, força e coragem, de forma especial, a minha mãe, Francisca do Socorro Mariz, pelo incentivo em todos os aspectos.

Ao meu namorado, Rodrigo dos Santos Diniz, pelos momentos de incentivo e apoio incondicional, sem os quais não teria conseguido concluir este trabalho.

Ao meu orientador, o professor Jair Cavalcanti Leite, pela paciência e orientação durante todo o processo de construção deste trabalho, tirando dúvidas e transmitindo conhecimentos que serão válidos durante toda a minha vida profissional.

Ao colega Jackson, que me ajudou no desenvolvimento desse trabalho, implementando as soluções propostas.

Aos meus amigos da pós-graduação e do grupo de trabalho do projeto GEDIG pelo grande apoio a realização desse trabalho, por todos os momentos de brincadeiras e descontração que também foram necessários durante todo esse período.

Em fim, por todos aqueles que contribuíram direta ou indiretamente na realização desse trabalho. Muito obrigada!

RESUMO

É cada vez mais comum o uso de um mesmo sistema computacional utilizando diferentes dispositivos – computadores pessoais, telefones celulares e outros – e plataformas de software – sistemas de interfaces de usuário gráficas, sistemas WEB e outros. Dependendo das tecnologias envolvidas, arquiteturas de software distintas podem ser empregadas. Por exemplo, em sistemas WEB, utiliza-se uma arquitetura cliente-servidor – normalmente estendida em três camadas. Em sistemas com interfaces gráficas, é mais comum arquiteturas com o estilo MVC. A utilização de arquiteturas com diferentes estilos dificulta a interoperabilidade de sistemas com múltiplas plataformas. Um outro agravante é que muitas vezes, as interfaces de usuário em cada um dos dispositivos possuem estrutura, aparência e comportamentos diferentes em cada dispositivo, o que leva a uma baixa usabilidade. Por fim, as interfaces de usuário específicas para cada um dos dispositivos envolvidos, com características e tecnologias distintas é um trabalho que precisa ser realizado individualmente e não permite escalabilidade. Esse trabalho procura resolver alguns destes problemas apresentando uma arquitetura de referência independente de plataforma e que possibilita que a interface de usuário possa ser construída a partir de uma especificação abstrata descrita na linguagem de especificação de interface de usuário, a IMML. Esta solução visa oferecer uma maior interoperabilidade entre as diferentes plataformas, uma maior consistência entre as interfaces de usuário e maior flexibilidade e escalabilidade para a incorporação de novos dispositivos.

Área de Concentração: Engenharia de Software

Palavras-chaves: Sistemas interativos multiplataformas, IMML, solução arquitetural.

ABSTRACT

It is increasingly common use of a single computer system using different devices - personal computers, telephones cellular and others - and software platforms - systems graphical user interfaces, Web and other systems. Depending on the technologies involved, different software architectures may be employed. For example, in Web systems, it utilizes architecture client-server - usually extended in three layers. In systems with graphical interfaces, it is common architecture with the style MVC. The use of architectures with different styles hinders the interoperability of systems with multiple platforms. Another aggravating is that often the user interface in each of the devices have structure, appearance and behaviour different on each device, which leads to a low usability. Finally, the user interfaces specific to each of the devices involved, with distinct features and technologies is a job that needs to be done individually and not allow scalability. This study sought to address some of these problems by presenting a reference architecture platform-independent and that allows the user interface can be built from an abstract specification described in the language in the specification of the user interface, the MML. This solution is designed to offer greater interoperability between different platforms, greater consistency between the user interfaces and greater flexibility and scalability for the incorporation of new devices.

***Area of Concentration**: Software Engineering.*
***Key words**: Systems interactive multi-platform, IMML and architectural solution.*

SUMÁRIO

LISTA DE FIGURAS

LISTA DE SIGLAS

AIO	*Abstract Interaction Objects*
API	*Application Programming Interface*
AUIML	*Abstract User Interface Markup Language*
AUIT	*Adaptative User Interface Technology*
BRIDGE	*Interface Design Generator Environment*
CIO	*Concrete Interaction Objects*
CORBA	*Common Object Request Broker Architecture*
DBC	Desenvolvimento Baseado em Componentes
DIUBM	Desenvolvimento de Interface de Usuário Baseado em Modelos
DHTML	*Dinamic HTML*
GOMS	*Goals, Operators, Methods, and Selection rules*
GUI	*Graphics User Interface*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
HUI	*Handheld User Interface*
IHC	Interação Humano - Computador
IMML	*Interactive Message Modeling Language*
IU	Interface de Usuário
JDBC	*Java Database Connectivity*
LDIU	Linguagem de Descrição de Interface de Usuário
JSP	*JavaServer Page*
LEMD	Linguagem de Especificação da Mensagem de *Designer*
MVC	*Model-view-controller*
PAC	*Presentation-Abstraction-Control*
PDAs	*Personal Digital Assistant*
RMI	*Remote Method Invocation*
SBB	Sistema Baseado em Browser
SQL	*Structured Query Language*
UIML	User Interface Markup Language
UML	*Unified Modeling Language*

URLs	*Uniform Resource Locator*
WAP	*Wireless Application Protocol*
WML	*Wireless Markup Language*
WUI	*Web-based User Interface*
W3C	*World Wide Web Consortium*
XICL	*eXtensinble User Interface Components Language*
XIML	*Extensible Interface Markup Language*
XML	eXtensible Markup Language
XUL	*XML User Interface Language*

CAPÍTULO 1

1. Introdução

Este capítulo introdutório mostra uma visão geral do trabalho, começando pelos fatores que motivaram a realização do mesmo, e seguindo com os problemas relevantes dentro do escopo de sistemas interativos multiplataformas que propomos solucionar. E, em seguida, descrevemos os objetivos para os quais este trabalho foi realizado.

1.1. Motivação

A variedade dos dispositivos tecnológicos onde sistemas podem ser utilizados, como em celulares ou *palmtops*, oferece vantagens de praticidade e comodidade aos usuários que utilizarem os serviços que lhes sejam úteis. Digamos, por exemplo, que um cliente de um banco deseje efetuar um pagamento, mas no momento o único dispositivo que tenha acesso é o seu celular, e que este possua um navegador da WEB. Então, o problema do cliente seria resolvido se o sistema bancário permitisse efetuar a operação de pagamento pelo celular. Em outro momento, poderia ser que o cliente dispondo de um computador convencional (*desktop*) desejasse verificar o seu extrato. Nesse caso, a consulta poderia ser realizada se o sistema fosse desenvolvido para aplicações WEB, onde um navegador WEB para *desktop* pudesse ser utilizado para acessar o sistema. Ou então, se uma aplicação utilizando alguma API - Interface de Programação de Aplicação – fosse desenvolvida para acessar o sistema bancário.

Um cenário que ilustra possibilidades interessantes é poder acessar e editar um mesmo arquivo de trabalho em casa, utilizando um computador de mesa, no carro através de um PDA (*Personal Digital Assistant*), ou numa reunião no escritório, utilizando uma tela de grande escala, com interação por controle remoto.

Com o desejo dos usuários em usarem aplicativos em qualquer lugar e a partir de diferentes tipos de dispositivos computacionais, surge a necessidade de que aplicações sejam desenvolvidas para todos eles, independente de características de dispositivos de entrada/saída, de sistema operacional utilizado, de tecnologias para desenvolvimento das

aplicações, ou de forma de acesso à aplicação (seja acessando a aplicação via navegador da WEB, ou utilizando alguma API - Interface de Programação de Aplicação- de interface de usuário gráfica GUI).

1.2. Definição do Problema

Diversos desafios precisam ser resolvidos para o desenvolvimento de sistemas que possam ser utilizados em vários tipos de dispositivos (por exemplo, celulares ou *desktops*) e plataformas operacionais (por exemplo, o sistema operacional utilizado, tecnologias utilizadas para desenvolvimento das aplicações, aplicativo utilizado para acessar o sistema ou linguagem de programação utilizada). Eles vão desde problemas de infraestrutura de comunicação até o desenvolvimento de interfaces de usuário (IU) para cada um dos diferentes tipos de dispositivos. As aplicações e os seus dados devem permitir acesso remoto e visualização por diferentes tipos de IU. Isto requer uma infraestrutura de rede com pontos de acesso os mais amplos possíveis. É necessária ainda uma arquitetura de sistema baseada em cliente-servidor de forma que a aplicação e os dados possam ser acessados a partir dos diversos clientes distribuídos geograficamente.

O foco desse trabalho está nos desafios associados ao desenvolvimento de sistemas interativos, que possam ser utilizados em vários tipos de dispositivos e plataformas operacionais. Um dos principais desafios no desenvolvimento de sistemas interativos multiplataformas é a separação dos aspectos do sistema que são específicos de dispositivos e plataformas (por exemplo, tipos de elementos de interface, quantidade de telas apresentadas para a realização de uma função) daqueles que são comuns (funcionalidades das aplicações). Essa separação é importante para que clientes, em diferentes plataformas (utilizando vários dispositivos tecnológicos, com diferentes características de entrada e saída de dados, com diferentes sistemas operacionais e usando meios distintos de acesso à aplicação), possam utilizar, de maneira transparente, as mesmas funções do sistema, garantindo assim a característica de interoperabilidade.

Outro desafio é a preservação de aspectos de usabilidade do sistema nas multiplataformas. A necessidade de que os sistemas atendam as expectativas dos usuários em relação à satisfação de uso, permitindo que eles possam realizar suas tarefas de forma satisfatória em qualquer dispositivo para a qual a aplicação seja desenvolvida nos vários estilos de interfaces de usuários, faz-se necessário que as interfaces de usuário para as diferentes plataformas considerem os aspectos de usabilidade.

Não faz sentido, por exemplo, que a interface para um sistema bancário tenha boa usabilidade em um sistema num computador tipo *desktop,* com uma interface gráfica *(GUI)* e uma péssima usabilidade em um celular, com acesso via WAP, com navegador para este protocolo.

A principal característica de usabilidade que deve ser considerada para o desenvolvimento de interfaces de usuários múltiplas é a consistência, que se refere às interfaces que possuem diferentes *look-and-feel* manterem o mesmo comportamento nas diferentes plataformas [Seffah and Javahery, 2004]. Segundo Nielsen [1990], consistência é a garantia de que um mesmo comando ou ação tenha sempre a mesma estrutura e o mesmo efeito de forma que as interfaces não possuam convenções ambíguas. Ressaltando também que a forma de interação do usuário com o sistema deve ser a mesma para qualquer plataforma que execute o sistema.

Um exemplo simples dessa consistência seria um sistema bancário que deve disponibilizar em todas as plataformas não apenas as mesmas funções – como, calcular saldo, emitir extrato e efetuar pagamento – mas também uma forma padronizada de interação. Para a execução de cada uma dessas funções, os mesmos dados devem ser requisitados e, em seguida, o resultado deve ser mostrado. Por exemplo, para a consulta do saldo, o sistema deve requisitar a agência, a conta e a senha do usuário, seguindo essa ordem, e em seguida deve exibir o nome do cliente, a data da emissão do saldo e o valor deste. O que vai diferir são os elementos de interfaces utilizados por cada plataforma de acordo o seu estilo de interface, como também a quantidade de telas necessárias para visualização dos elementos interativos. Para um *desktop* o resultado da função extrato cabe em uma única tela, enquanto em um celular é necessária a exibição de várias telas. Portanto, é trabalho dos desenvolvedores gerar interfaces de usuários finais para cada dispositivo (que utilizam diferentes plataformas operacionais) de forma que as interfaces sejam portáveis para as multiplataformas que devem interagir com o servidor da aplicação. Se o sistema for desenvolvido levando em consideração a consistência do diálogo das informações e o usuário souber usá-lo de forma correta em uma plataforma específica ele não terá dificuldade de usá-lo em outra plataforma e, conseqüentemente, diminuirá a probabilidade de ocorrência de erro.

Portanto, os problemas que estamos considerando para o desenvolvimento de sistemas interativos multiplataformas resumem-se em definir uma organização do sistema de forma a separar os aspectos comuns a todos os dispositivos e plataformas (por exemplo, a funcionalidade do sistema acessado) dos aspectos específicos aos mesmos (por exemplo, a

forma de apresentação e interação com o sistema). Dessa forma, a organização do sistema deve considerar os aspectos de portabilidade, ou seja, a mesma estrutura do sistema deve ser usada nas diferentes plataformas, em que aspectos específicos de plataformas devem interagir com um mesmo servidor do sistema.

Também é importante que as interfaces de usuários geradas pelos desenvolvedores sejam portáveis para diferentes plataformas. Evitando assim desenvolver diversas versões de uma mesma IU para cada uma das diferentes plataformas, pois isso requer que o desenvolvedor conheça cada uma das plataformas alvo e as diversas variações que cada uma delas possa requerer.

1.3. Objetivos

Para o desenvolvimento de sistemas interativos que executem em múltiplos dispositivos e plataformas, muitos atributos de qualidade são envolvidos no processo e no produto. Este trabalho não tem por objetivo resolver todos os problemas relacionados ao desenvolvimento de sistemas interativos multiplataformas. A proposta é resolver os problemas descritos na seção anterior.

O trabalho propõe o estilo de arquitetura que deve ser utilizado com o objetivo de oferecer uma referência para os desenvolvedores construir sistemas interativos multiplataformas, ou seja, propomos uma arquitetura de referência [Bass et al., 2003] que é uma arquitetura generalizada de diversos sistemas que compartilham um ou mais domínios comuns, nesse caso os domínios são os sistemas interativos que possam ser executados em multiplataformas. Essa arquitetura deve permitir que o sistema tenha características de portabilidade, interoperabilidade e usabilidade. Para garantir a portabilidade, a arquitetura deve ser organizada de maneira que permita que os sistemas possam ser utilizados em qualquer plataforma. Para satisfazer à interoperabilidade, a arquitetura deve possibilitar que os vários clientes, nas diferentes plataformas, utilizem o mesmo núcleo funcional do sistema, de forma transparente. E a característica de consistência das interações do usuário com o sistema (importante medida de usabilidade) pode ser obtida usando os modelos de especificação da IMML [Leite, 2007] na arquitetura proposta. A IMML é linguagem abstrata para especificação de interfaces de usuários. A utilização dessa linguagem aliada à arquitetura proposta possibilita organizar os sistemas, especificando as interações do usuário com o sistema de maneira independente de dispositivo e plataforma, como também descrever a organização das interfaces para cada plataforma de maneira abstrata seguindo um único

modelo de interação, ou seja, um único conjunto de interações é definido na IMML e deve ser utilizado para gerar as interfaces de usuários para as diferentes plataformas. Isso é possível por utilizar, na arquitetura, as especificações dos modelos de interação e de comunicação da IMML descritos no capítulo 4 aliado as regras de mapeamento definidas por [Sousa, 2004] e [Silva, 2007].

O nosso interesse é o desenvolvimento de sistemas interativos multiplataformas, onde estamos considerando como dispositivos computacionais os computadores convencionais (*desktops*) e celulares e como estilos de interfaces o GUI e WUI. Ou seja, as aplicações do sistema poderão ser acessadas através de computadores convencionais usando como plataforma de execução um navegador da WEB ou uma aplicação de interface gráfica GUI. Ou podem ser acessadas através de um celular usando como plataforma de execução um navegador da WEB para celular.

A solução arquitetural proposta deve ser utilizada em sistemas interativos que possam ser acessados por clientes em vários dispositivos com diferentes tamanhos de telas, independente de sistemas operacionais, e que acessem a aplicação (núcleo funcional) do sistema através de um navegador WEB (para *desktops* ou celulares) ou usando aplicações de interfaces gráficas GUI.

Este documento está organizado em 6 capítulos incluindo a introdução. O capítulo 2 descreve alguns conceitos básicos de forma a contextualizar o trabalho. No capítulo 3 constam os trabalhos relacionados encontrado nessa pesquisa. O quarto capítulo apresenta a linguagem IMML, incluindo as alterações feitas nessa linguagem. O capítulo 5 introduz a arquitetura de referência proposta. O último capítulo faz as considerações finais.

CAPÍTULO 2

2. Conceitos Básicos

Este trabalho tem por objetivo definir uma arquitetura de referência, que é um padrão arquitetural específico de um domínio, que possa ser utilizada para o desenvolvimento de sistemas interativos multiplataformas capaz de tratar os problemas nos quais o desenvolvimento destes sistemas estão inseridos, conforme descrito na introdução. Para isso vamos esclarecer alguns conceitos necessários para entender a solução que será apresentada adiante.

Inicialmente consideramos o *design* da arquitetura, englobando os estilos arquiteturais para sistemas interativos. Em seguida, descrevemos o que são sistemas interativos multiplataformas. Adiante, descrevemos as propostas de desenvolvimento baseadas em modelos. E, por fim vamos discutir as abordagens que propõem soluções baseadas em linguagens de descrição de interfaces de usuário (LDIU).

2.1 Arquitetura de Software

Um aspecto necessário e complementar ao desenvolvimento de sistemas interativos multiplataformas é o modelo arquitetural empregado. Durante o desenvolvimento de sistemas, os projetistas fazem uso de uma arquitetura de *software* para estruturar o sistema em termos de seus componentes (entidades computacionais com funcionalidade específica).

Tamanha importância dada à arquitetura é devida a alguns fatores, entre eles: a capacidade de diminuir os custos devido à característica de reusabilidade e características que facilitam o processo de manutenção e evolução dos sistemas [Garlan; Shaw, 1994]; [Garlan, 2000].

A principal contribuição em elaborar o *design* da arquitetura, em termos gerais, é diminuir o "*gap"* entre a especificação dos requisitos e a implementação do sistema em código fonte [Garlan, 2000]. A figura 2-1 mostra o momento em que esta se encontra.

A solução arquitetural proposta nesse trabalho deve ser utilizada com o objetivo de oferecer uma referência para os desenvolvedores construir sistemas interativos multiplataformas. No contexto aqui abordado a elaboração de uma arquitetura permite especificar como os sistemas interativos multiplataformas podem ser organizados de forma a incluir características de portabilidade, interoperabilidade e garantir as características presentes na especificação em IMML.

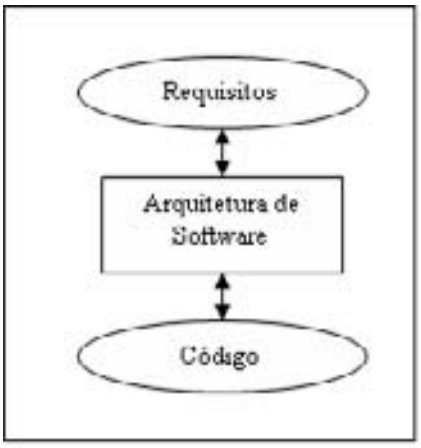

Figura 2-1:Arquitetura como uma ponte (figura extraída de [Garlan, 2000])

Dentre as várias definições de arquitetura de *software* que existem na literatura, podemos dizer que elaborar arquitetura do sistema significa organizá-lo de maneira abstrata em componentes e conectores e definir a topologia, ou seja, mostrar como esses elementos estão relacionados uns com os outros [Garlan, 2000].

Garlan [2000] define componentes como blocos de alto nível que descrevem as partes da arquitetura que realizam algum tipo de computação e que armazenam algum dado do sistema. Cada componente possui uma funcionalidade bem definida, sendo assim capaz de oferecer modularidade e separação dos conceitos. Eles também publicam uma ou mais interfaces que constitui(em) sua identificação. As interfaces publicam as propriedades visíveis externamente, que possibilitam fazer a conexão entre os componentes.

Neste trabalho, os componentes arquiteturais são definidos de forma conceitual, especificados de forma abstrata, servindo para auxiliar a construção dos sistemas interativos multiplataformas.

Os componentes conhecem uns aos outros através das suas interfaces publicadas. Não constitui parte do componente a exibição da implementação, ou seja, eles são blocos abstratos que escondem completamente a implementação.

Os conectores descrevem as estruturas de comunicação entre os componentes estabelecendo restrições de como as mensagens fluem entre os componentes. Ex: chamadas de métodos ou procedimentos, protocolos de comunicação cliente-servidor.

Na elaboração de uma arquitetura de *software* comumente são utilizados diferentes terminologias ou conceitos. Conceitos importantes são o de padrão arquitetural e estilo arquitetural. Eles são usados com o mesmo significado e servem para definir alguns aspectos como o mecanismo e o relacionamento entre os componentes, caracterizando uma família de sistemas que são relacionados pelo compartilhamento de propriedades estruturais e comportamentais (semântica) [Hofmeister et al., 2000]. Eles definem um vocabulário para os elementos da arquitetura: componentes e conectores. Além disso, apresentam regras e restrições sobre a combinação dos componentes arquiteturais, indicando como organizar os elementos de um sistema para solucionar problemas específicos em determinado contexto [Garlan; Shaw, 1994]. Exemplos de padrões arquiteturais são: camadas, pipes e filtros e o padrão *model-view -controller* (MVC).

Outros termos comumente usados são: arquitetura de referência e arquitetura de *software* de domínio específico. Eles são utilizados com o mesmo significado. Da mesma forma que os padrões arquiteturais eles definem os tipos dos elementos arquiteturais e como eles podem ser relacionados, mas nesse caso eles são aplicados em um domínio de sistema particular definindo como as funcionalidades de um domínio de sistema são representadas em elementos arquiteturais, ou seja, em componentes e conectores [Hofmeister et al., 2000]. Uma arquitetura de referência ou arquitetura de *software* de domínio específico deve usar um conjunto de padrões arquiteturais na sua estrutura. Uma definição dada por [Bass et al., 2003] é que uma arquitetura de referência é uma arquitetura generalizada de diversos sistemas que compartilham um ou mais domínios comuns. Ela pode ser instanciada para criar uma arquitetura de *software* específico.

Nesse trabalho a arquitetura de referência é a solução proposta para se construir sistemas interativos que possam ser utilizados em multiplataformas. Essa solução foi descrita utilizando as visões conceitual e de módulo [Hofmeister et al., 2000], que diferente do que propõe [Kruchten, 2000] não necessariamente deve ser totalmente instanciada, testada e descrita utilizando as visões 4+1. Ela deve auxiliar a construção de sistemas interativos multiplataformas indicando qual funcionalidade do sistema cada elemento arquitetural representa.

2.2 Estilos arquiteturais para sistemas interativos

No contexto do desenvolvimento de sistemas interativos encontramos alguns padrões arquiteturais onde os mais conhecidos são: o MVC (*Model-View-Controller*) [Goldberg, 1984] e PAC (*Presentation-Abstraction-Control*) [Coutaz, 1987]. Essas soluções arquiteturais são para aplicações seguindo o estilo de interface de usuário GUI, não suportando o desenvolvimento de múltiplas interfaces de usuários. Diante do problema tratado, que é o desenvolvimento de sistemas interativos para multiplataformas, estamos usando o propósito básico desses estilos que é a separação da parte da apresentação da parte que contém os dados da aplicação, adicionando os aspectos de portabilidade e usabilidade.

2.2.1 MVC - *Model-View-Controller*

O MVC divide as aplicações em três módulos: o *model* (responsável pela funcionalidade da aplicação), o *view* (que recebe, recupera e exibe dados) e o *controller* (que gerencia os eventos de entrada dos usuários, coordenando o *model* e o *view*). Desse modo ele desacopla os módulos do sistema, cada um com a sua responsabilidade. A estrutura desse padrão arquitetural permite que um único *model* possa se ligar a vários pares de *controller* – *view*, como mostra a figura 2-2.

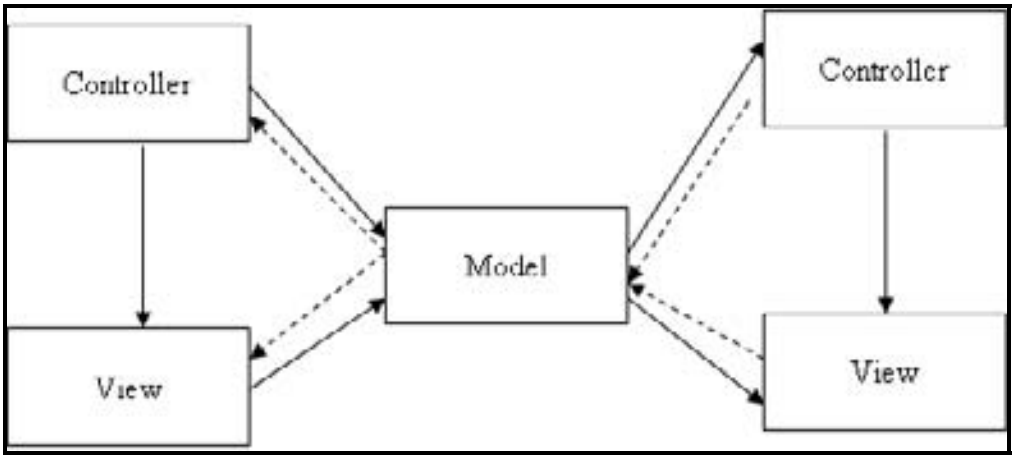

Figura 2-2: Estrutura do MVC

O seu controle do fluxo ocorre da seguinte maneira:

1. O usuário interage com a interface de alguma forma (por exemplo, o usuário aperta um botão).

2. O *Controller* manipula o evento da interface do usuário através de uma rotina pré-escrita.
3. O *Controller* acessa o *Model*, possivelmente atualizando-o de maneira apropriada, baseado na interação do usuário (por exemplo, atualizando os dados de cadastro do usuário).
4. Algumas implementações de *View* utilizam o *Model* para gerar uma interface apropriada (por exemplo, mostrando na tela os dados que foram alterados juntamente com uma confirmação). O *View* obtém seus próprios dados do *Model*. O *Model* não toma conhecimento direto do *View*.
5. A interface do usuário espera por próximas interações, que iniciarão o ciclo novamente.

2.2.2 PAC - *Presentation-Abstraction-Control*

O PAC é um estilo muito similar ao MVC. Ele é usado como uma estrutura hierárquica de agentes, cada um constituído de três partes: apresentação, abstração e controle. Essas partes se comunicam uma com a outra somente através da parte de controle de cada trio, conforme mostra a figura 2-3. Cada agente do PAC é formada por:

- Apresentação – é exatamente como o *view* do MCV. Ele mostra as informações da abstração.
- Abstração – contém os dados da aplicação e é responsável pela funcionalidade da aplicação, como o *model* do MVC, mas não tem o papel de notificar mudanças.
- Controle – é similar ao *controller* do MVC. Ele é o responsável pelo controle do diálogo e por gerenciar o relacionamento entre os diferentes trios. Ele processa eventos externos e atualiza o modelo. Ele também atualiza diretamente a parte da apresentação. Ele é diferente do *controller* do MVC porque um componente PAC pode sofrer mudanças de outro componente PAC pai.

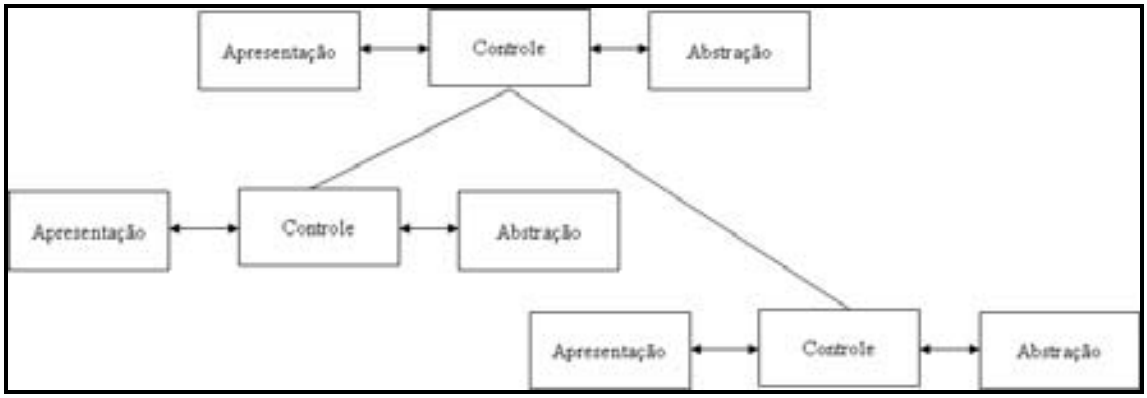

Figura 2-3: Estrutura do PAC

A diferença entre o PAC e o MVC é que o PAC não tem o *model* como seu componente central, mas uma estrutura hierárquica de componentes PAC.

A solução arquitetural proposta neste trabalho utiliza a proposta dos estilos descritos acima, que é a separação da parte da apresentação (Componente *Cliente*) da parte que contém os dados da aplicação (Componente *Servidor Funcional*), inserindo partes intermediárias (Componentes *Controle* e *Repositório*).

Diante do problema que estamos tratando, que é o desenvolvimento de sistemas interativos para multiplataformas, estamos incluindo também a característica de portabilidade, considerando os vários tipos de dispositivos (Celular, *Desktop*, ou outros dispositivos móveis) e estilos de interfaces (GUI e WUI), onde o sistema poderá ser executado.

2.3 Sistemas Interativos Multiplataforma

Com o surgimento cada vez maior de vários dispositivos computacionais, como por exemplo, telefones celulares e computadores de mão (*palmtops*), com diferentes características de tamanho da tela e formato do teclado, diversos termos surgiram para classificá-los.

Os sistemas que podem ser usados em diversos dispositivos computacionais (por exemplo, *desktops* ou celulares), considerando os vários estilos de interfaces de usuários (por exemplo, GUI e WUI), são chamados de sistemas interativos multiplataformas.

Seffah &Javahery [2004] apresentam uma proposta em três categorias:

- *Graphics User Interface* (GUI): este estilo é o mais conhecido pela maioria dos usuários, normalmente usado nos *desktops*. Também é conhecido com WIMP (*Windows, Icons, Menus and Pointers*);

- *Web-based User Interface* (WUI): 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): estilo 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.

Para o desenvolvimento de sistemas interativos que possam ser acessados por clientes em vários dispositivos com diferentes tamanhos de telas, independente de sistemas operacionais, e que acessem a aplicação (núcleo funcional) do sistema através de um navegador WEB (para *desktops* ou celulares) ou usando aplicações de interfaces gráficas GUI, diversas tecnologias devem ser empregadas no desenvolvimento desses sistemas. Não é o foco deste trabalho o detalhamento dessas tecnologias, porém citamo-las como forma de inseri-las no contexto dos sistemas interativos multiplataformas.

Para computadores *desktop,* aplicações que apresentem o sistema aos usuários seguindo o estilo de interface GUI, podem ser implementadas, por exemplo, usando a API do Java Swing, que possui classes para a construção de interfaces de janelas de forma totalmente independente de sistema operacional ou ambiente de janelas. Os seus componentes utilizados para construir objetos de interfaces só dependem da máquina virtual Java. A comunicação entre as aplicações dos clientes e os servidores funcionais pode ser feita, digamos, com a comunicação *sockets*, que é uma espécie de interface lógica utilizada pelo cliente para se conectar ao servidor.

Para dispositivo *desktop* com o estilo de interface WUI, um *browser* baseado em HTML deve ser usado para apresentar o sistema aos usuários e acessar as funções do sistema através de algum servidor WEB usando o protocolo de comunicação HTTP. Para os dispositivos celulares no estilo de interface WUI, o sistema deve ser apresentado aos usuários através de *browser* baseado em WML que acessa as funções do sistema usando um servidor WEB através do protocolo WAP. As páginas geradas pelos servidores WEB podem ser implementadas usando a tecnologia JSP para gerar as páginas HTML ou WML, conforme o dispositivo. As funções do sistema podem ser implementadas usando a linguagem Java, por possuir a característica de portabilidade. E, dessa forma, todas as aplicações usando o Java Swing ou o JSP podem acessar as funções do sistema.

Esses são exemplos de possíveis tecnologias que podem ser empregadas para o desenvolvimento de sistemas interativos multiplataformas.

O foco desta dissertação é desenvolver uma solução arquitetural de maneira conceitual que possibilite o uso dessas tecnologias nas plataformas citadas: *desktop* e celular nos estilos de interfaces GUI e WUI. É importante que a solução arquitetural utilize de especificações de interfaces de usuários, para que as interfaces finais de usuários sejam baseadas em especificações de interfaces abstratas e que possam ser construídas de forma dinâmica e independente de plataforma.

Não faz parte desse trabalho a construção de interfaces multimodais [Coutaz, 1991], ou seja, interfaces capazes de processar vários tipos de entrada do usuário de uma maneira combinada permitindo que os usuários escolham o modo de interação. Também não estamos considerando a adaptação do conteúdo das interfaces de acordo com o contexto de uso, por exemplo, o conteúdo das interfaces variando conforme as características dos usuários. Nossa proposta é que os usuários possam acessar todas as funções do sistema em qualquer dispositivo ou plataforma, o que deve variar são os elementos de interfaces de acordo com o estilo de interação utilizado e a quantidade de telas necessárias para apresentar a seqüência das interações.

2.4 Desenvolvimento de Interfaces de Usuários Baseada em Modelos

Uma das abordagens na qual a IMML esta baseada é o desenvolvimento de interface de usuário baseado em modelos (DIUBM). Este 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 & Szkeley, 1994].

O trabalho de Puerta & Eisenstein [2001] define os modelos que compõem uma interface de usuário. Esses modelos são os seguintes: modelo de tarefa, modelo de domínio, modelo de usuário, modelo de apresentação e modelo de diálogo. Cada modelo apresenta aspectos específicos de uma interface, e cada um deles é classificado segundo a categoria de abstrato, que são todos aqueles independentes de plataformas e que são encontrados nos modelos de tarefas, domínio e usuário; e de concretos, que são todos aqueles que possuem características específicas de plataformas e que são encontrados nos modelos de apresentação e diálogo.

O DIUBM propõe o mapeamento entre os modelos abstratos, independente de dispositivos, e concretos, específicos de dispositivos, e a partir deste mapeamento são derivadas as IUs executáveis. O mapeamento pode ocorrer de forma automática, semi-automática ou manual. Exemplos de mapeamentos entre estes modelos são: tarefa-diálogo, tarefa-apresentação, domínio-apresentação, tarefa-usuário, tarefa-domínio, apresentação-diálogo.

Outra abordagem para o DIUBM é o *Framework de Referência Camaleon* (Calvary et al., 2003), que propõe que o desenvolvimento de IU multiplataforma 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 descritas 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.

A solução aqui apresentada segue a idéia de desenvolvimento baseado em modelos, uma vez que as descrições das interfaces, descritas em IMML, estão armazenadas em repositórios. Esses modelos são utilizados para gerar a interface de usuário final dinamicamente.

Como veremos em mais detalhes no capítulo 4, na IMML, as IUs são descritas em três modelos: de domínio, de interação e de comunicação. Em relação aos níveis de abstração, o modelo de domínio especifica os conceitos da aplicação, o modelo de interação especifica as interfaces abstratas e o modelo de comunicação especifica as interfaces concretas. 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 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.

Em relação aos sistemas baseados em modelos, usa-se representações genéricas abstratas das interfaces em linguagens de descrição de interfaces de usuário (LDIU), descritas na próxima seção, para tentar produzir automaticamente modelos de interfaces concretas através de um mapeamento dos elementos abstratos (AIO) em elementos concretos da interface (CIO).

2.5 Linguagens de descrição de interfaces de usuário.

Para o desenvolvimento de interfaces de usuários considerando os vários estilos de interfaces, ou seja, de interfaces de usuários multiplataformas, comumente são utilizadas as linguagens de descrição de interfaces de usuários (LDIU). Neste trabalho a LDIU utilizada para especificar os aspectos funcionais, interativos e comunicativos de interfaces de usuário para múltiplataformas é a IMML, descrita no capítulo 4.

Uma LDIU (Linguagem de Descrição de Interfaces de Usuários) é uma das formas mais utilizadas para descrever aspectos interativos de interfaces de usuários. Elas 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. A partir destas descrições, podem ser geradas interfaces de usuário finais, através do mapeamento entre modelos.

Segundo Trewin et al. [2003], inúmeras propostas de LDIU surgiram nos últimos anos, principalmente pela necessidade de desenvolver interfaces de usuários para múltiplas plataformas e dispositivos. Usando uma LDIU é possível descrever diferentes aspectos de uma interface do usuário de forma abstrata e independente de plataforma ou dispositivo. Ela é composta por uma sintaxe, que define os termos e a gramática da linguagem, e uma semântica, que define o significado e as relações entre os termos [Souchon e Vamderconckt, 2003].

A maioria das LDIUs são linguagens de marcação baseadas em XML (*eXtensible Markup Language*) [Bray et al., 1998], que é um padrão recomendado pelo W3C, bem estabelecido e extensível, que pode trabalhar com novas plataformas sem muitas mudanças. Essas linguagens são especialmente adequadas para a especificação de interfaces independentes de dispositivos, plataformas e contexto de uso. Segundo Trewin el al. [2003], existe atualmente uma grande variedade de linguagens com vários propósitos e que descrevem a interface em diferentes níveis de abstração. São exemplos de LDIUs: UIML

[Abrams; Phanouriou, 1999], XIML [Puerta; Eisenstein, 2001], Tesesa XML [Mori, 2003], AUIL [Siebelink, 2000] e XUL [XUL, 2001].

CAPÍTULO 3

3. Trabalhos Correlatos

A heterogeneidade dos dispositivos e plataformas operacionais impõe desafios para o desenvolvimento de aplicações, dos quais podemos citar os seguintes: a descrição, independente de dispositivo e plataforma, da interface com usuário; e o desenvolvimento de aplicações multiplataformas.

Diversos trabalhos foram propostos para solucionar cada um desses desafios. Contudo, não foi encontrada na literatura nenhuma abordagem arquitetural que auxiliasse o desenvolvimento de sistemas que pudessem ser utilizados em plataformas seguindo o estilo de interface WUI e GUI e que permitisse a geração automática de interfaces de usuários para as multiplataformas utilizando uma descrição da interface e dos seus dados, de forma que essa descrição seja independente de um dispositivo específico ou de uma plataforma de programação.

Neste capítulo são citados alguns trabalhos que estão relacionados com o desenvolvimento de sistemas interativos para multiplataformas. Todos eles propõem soluções arquiteturais para múltiplos dispositivos, contudo somente para aplicações desenvolvidas para WEB. E nenhum deles, inclui como parte da solução a vantagem da utilização de uma linguagem de especificação de interface de usuário juntamente com a arquitetura.

3.1 Arquitetura para Sistemas Interativos Multiplataformas

Grundy & Zou, [2004] apresentam uma arquitetura de sistema composta por quatro camadas. Essa arquitetura é para aplicações WEB considerando os diferentes tipos de usuários e tarefas específicas.

As camadas da arquitetura são:

1. Dispositivos dos clientes: acessam o sistema através de dispositivos móveis, utilizando *browsers* baseados em WML, e/ou através de *laptops* ou *desktops* executando *browsers* em HTML. Esses dispositivos usam serviços dos servidores WEB, através de protocolos HTTP ou WAP.

2. Servidores WEB oferecem os serviços utilizados pela WEB. Eles requisitam serviços, usando o protocolo Java RMI dos Servidores de aplicação.
3. Servidores de aplicação executam o interpretador Java *beans*. Este encapsula a lógica do negócio e realiza o processamento dos dados, acesssando as funções da aplicação, através da interface CORBA e protocolos XML, e acessando os serviços do servidor de banco de dados, através de protocolos SQL ou JDBC.
4. Funções da aplicação oferecem as funcionalidades da aplicação.
5. Servidor de banco de dados oferece os dados necessários para a realização das funções.

Esse trabalho propõe o uso da tecnologia AUIT – *Adaptative User Interface Technology* – para representar as páginas geradas pelos servidores WEB. As páginas em AUIT são implementadas em JSPs contendo uma linguagem de marcação independente de dispositivos, que geram páginas HTML e WML, com descrições de elementos da tela organizados em *layouts*. Essas descrições são representadas usando *tags* da XML, organizando os elementos de telas em *layouts* de acordo com os papéis dos usuários e tarefas dos usuários.

As descrições das interfaces AUIT contêm aspectos comuns às linguagens de especificação baseada em modelos por especificar elementos da tela, o *layout*, o usuário e suas tarefas.

Os elementos arquiteturais definidos nessa arquitetura se assemelham aos elementos arquiteturais propostos nesse trabalho. As principais diferenças apresentadas são: Ela não possui em sua estrutura um elemento responsável em controlar as interações do usuário com o sistema, não é aplicada para o desenvolvimento de sistemas seguindo o estilo de interface GUI, e embora utilize a tecnologia AUIT para representar aspectos de interfaces de usuários, ela não usa as vantagens de uma linguagem de descrição de interface de usuários.

3.2 Sistema de interface de usuário unificada baseada em XML

Outro trabalho, descrito em [CHI-HSING; CHIEN-HSUAN; LEE, 2000], apresenta uma arquitetura para o desenvolvimento de aplicações para internet. A arquitetura é projetada para prover diferentes visualizações de dados e serviços contidos em um servidor WEB, permitindo o acesso a esses serviços por diferentes clientes (e.g., PCs, PDAs). A arquitetura, intitulada de *XML-Based Unified User Interface System*, é dividida em três camadas: a cliente, que possui um navegador com suporte a linguagens de marcação WEB (e.g., XHTML); a *middlelayer*, que contém a lógica de programação; e a camada de dados na qual estão as bases de dados legadas ou os bancos de dados. A arquitetura é proposta para ser desenvolvida utilizando J2EE [J2EE, 2008], tecnologia Java para servidores.

A arquitetura sugere que a interface das aplicações seja descrita também em XML e que conversores devem ser escritos para mapear essa descrição em uma linguagem de marcação aceita pelos navegadores dos computadores ou dos PDAs (e.g., XHTML, WML). Esse trabalho apresenta como trabalhos futuros o desenvolvimento de uma linguagem para descrever as aplicações e a construção de conversores para linguagens de marcação (e.g., HTML, WML).

Esse trabalho não utiliza uma linguagem de especificação de interfaces como parte da solução, embora sugira a descrição das aplicações em XML. A sua idéia também envolve conversores que converta documentos XML para interfaces finais de usuários, mas somente para o estilo de interface WUI.

3.3. Arquiteturas de Adaptação de Conteúdo

Alguns trabalhos como em [HAGIMONT; LAYAÏDA, 2002], [CASTRO; LOUREIRO, 2004] e [LEMLOUMA; LAYAIDA, 2004], são descritas arquiteturas de sistemas considerando as vantagens de utilizar adaptação de conteúdo. Esses trabalhos apresentam propostas de soluções arquiteturais para os casos de sistemas poderem ser acessados por dispositivos móveis e computadores *desktops*, usando navegadores da WEB. Eles têm como foco a adaptação do conteúdo de acordo com os dispositivos. Para eles, o conteúdo deve ser cuidadosamente selecionado e adaptado às condições restritivas do equipamento de destino (e.g. dimensões do *display*, quantidade de cores, memória disponível). Esses trabalhos enfatizam que o mesmo conteúdo direcionado para um computador fixo (i.e. *desktop*) não deve ser enviado a um celular, por exemplo. Nesse caso,

tanto deve haver redução das informações, como modificação da forma de apresentação dos dados. A adaptação de conteúdo não, necessariamente, baseia-se somente nas características dos dispositivos que acessam as informações. As preferências e o contexto do usuário que utiliza a aplicação também podem ser relevantes nessa adaptação.

A nossa solução arquitetural proposta não esta considerando a adaptação do conteúdo para cada plataforma. Aqui, as mesmas informações (funcionalidades e interações) devem ser apresentadas em todos os dispositivos em todas as plataformas, o que deve mudar são os elementos de interfaces utilizados e a quantidade de telas para apresentar as funções do sistema.

CAPÍTULO 4

4. A Aplicação da IMML em Sistemas Interativos Multiplaformas

As diferentes partes de um *software* podem ter propósitos distintos. Uma clássica separação, já discutida na seção anterior. A solução que propomos segue a proposta de separação de partes de um *software* em seus aspectos funcionais, interativos e comunicativos [Leite, 2007]. Nessa visão, a especificação do *software* deve separar esses aspectos em diferentes modelos.

O modelo funcional especifica os requisitos funcionais, representado às tarefas que o *desinger* considera que o usuário quer realizar com a aplicação. Um exemplo de requisito funcional é consultar um extrato de conta corrente num sistema bancário ou fazer a reserva de um quarto de hotel.

O modelo de interação descreve as regras e protocolos pelos quais os usuários interagem com o sistema. Ela deve estar adequada às capacidades físicas e cognitivas dos usuários e aos recursos tecnológicos dos sistemas, permitindo a estes serem atraentes, proporcionando uma boa interação com o usuário. Por exemplo, para a funcionalidade de consultar o extrato bancário, o especialista em IHC deve especificar de que forma o usuário vai acessar a consulta ao seu extrato, como vai fornecer os dados de entrada e de saída. As decisões envolvem se o usuário vai digitar comandos, escolher opções de um menu ou preencher dados de um formulário. A interatividade refere-se à forma como o *designer* pretende que o usuário utilize a aplicação para realizar as tarefas.

O modelo de comunicação descreve os aspectos de comunicabilidade. A comunicabilidade refere-se à maneira como o *designer* comunica os aspectos de funcionalidade e interatividade [Leite, 2005]. A comunicabilidade do sistema bancário será satisfatória, por exemplo, se os elementos utilizados na interface conseguem informar ao usuário que existe a possibilidade de fazer consulta ao extrato e quais são as ações para ele

escolher essa função e fornecer os dados necessários. A comunicabilidade é um conceito baseado na teoria da engenharia semiótica de Souza [2005], que considera o sistema computacional um artefato de meta-comunicação, que conduz uma mensagem unidirecional do *designer* ao usuário.

A separação dos conceitos de funcionalidade, interatividade e comunicabilidade traz alguns benefícios para o desenvolvimento de *software*. Alguns destes benefícios já foram discutidos por Leite [2007]. No presente trabalho, tratamos dos benefícios que esta separação de conceitos traz para o desenvolvimento de sistemas que precisam executar em múltiplataformas.

Para descrever os modelos, precisamos de uma linguagem de propósito específico. A IMML [Leite, 2005] é uma Linguagem de Descrição de Interface de Usuário (LDIU) abstrata baseada na abordagem de Desenvolvimento de IU Baseado em Modelos (DIUBM) [Puerta; Szkeley, 1994] e na teoria Engenharia Semiótica [De Souza, 1993]. No capítulo de conceitos básicos já explicamos a abordagem do DIUBM. A teoria da Engenharia Semiótica é uma abordagem na qual os sistemas computacionais são vistos como artefatos de meta-comunicação. Nessa abordagem, a interface de um sistema é vista como sendo uma mensagem enviada pelo *designer* ao usuário [De Souza, 1993]. A Engenharia Semiótica considera que a mensagem do *designer* para o usuário descreve os aspectos de funcionalidade, interatividade e comunicabilidade.

A IMML permite representar a funcionalidade, interatividade e comunicabilidade dos sistemas interativos em seus modelos constituintes. Esses modelos foram criados para fornecer um maior controle em níveis diferentes de abstração, possibilitando o mapeamento entre os mesmos. No modelo de domínio é descrita a funcionalidade da interface, no modelo de interação é feita a descrição do processo de interação entre usuário e o sistema. No modelo de comunicação é descrito de que forma o *designer* quer comunicar para o usuário sobre o que foi definido nos modelos anteriores, ou seja, como serão apresentados as funcionalidades, os comandos e os resultados.

O modelo de comunicação, desenvolvido por Costa Neto [2005], descreve o que foi especificado nos demais modelos – de domínio e de interação – mapeando as soluções específicas para cada plataforma de aplicação. Dessa forma, pode-se gerar diferentes interfaces de usuários concretas a partir das descrições de cada modelo de interação para plataformas específicas. Ou seja, um único conjunto de interações é definido e as interfaces concretas para as multiplataformas (especificadas no modelo de comunicação) são baseadas nesse conjunto de interações.

A proposta de Costa Neto [2005] foi implementada em Sousa [2004]. Nesta, um compilador faz a tradução das descrições da interface em IMML e gera código em DHTML para os dispositivos celular e *desktop*.

Embora a IMML permita a especificação levando em consideração esta separação de conceitos, ela não indica como o sistema deve ser desenvolvido e como os diferentes aspectos podem ser preservados durante a implementação, isto é, se o produto final está completamente de acordo com a especificação IMML.

A separação dos conceitos da IMML já é um caminho importante em busca da solução para este problema. De uma maneira geral, ela indica como o sistema precisa preservar esta separação conceitual em seus componentes constituintes.

A incorporação da linguagem de especificação de interface de usuário, a IMML, na solução arquitetural proposta, permite que as interfaces de usuários sejam descritas em vários níveis de abstração mantendo o mesmo processo de interação usuário-sistema nas multiplataformas. Dessa forma, o que se espera é que as interfaces finais de usuários, construídas com base nas interfaces abstratas, possuam consistência comportamental nas multiplataformas.

A solução hora apresentada tem como base a separação dos conceitos representados pela IMML. A arquitetura proposta utiliza especificações em IMML, especificamente os modelos de interação e de comunicação, que ficam armazenadas em repositórios, para que as interfaces de usuários finais sejam construídas baseadas nas interfaces abstratas especificadas em IMML. Dessa forma, o elemento arquitetural *Controlador*, descrito no capítulo 5, utiliza as interações (definidas no modelo de interação) e interfaces concretas (definidas no modelo de comunicação) para gerenciar todo o fluxo de entrada e saída dos dados apresentados nas telas para os usuários.

Em Costa Neto [2005], foi apresentado o uso da IMML de maneira satisfatória para o desenvolvimento de IU multiplataformas. Neste foi apresentado o modelo de comunicação da linguagem que define interfaces concretas específicas de dispositivos utilizando os elementos interativos especificados no modelo de interação.

Embora a IMML permita especificar características importantes de sistemas interativos e para múltiplas plataformas, ela não é suficiente para implementação dos sistemas, e só consegue elaborar protótipos de interfaces de usuários sem funcionalidades reais, como demonstra os trabalhos de Silva [2007] e de Sousa [2004]. É necessário incluir o *design* da arquitetura no processo de desenvolvimento de sistemas possibilitando descrever

características mais específicas que tornem a modelagem mais próxima da implementação do sistema.

Esse capítulo apresenta um resumo da linguagem IMML, já descrita em trabalhos anteriores [Costa Neto, 2005], [Sousa, 2004], [Machado, 2006], [Lira, 2006] e [Silva, 2007] mostrando os seus modelos constituintes e como eles podem ser utilizados no desenvolvimento de sistemas interativos multiplataformas. Algumas alterações na IMML, feita nesse trabalho, também são descritas aqui, justificando-as.

4.1 A IMML – Interactive Message Modeling Language

A IMML é uma linguagem baseada na XML - *Extensible Markup Language* [Bray et al., 1998]. Ela originou-se de um aprimoramento da LEMD (Linguagem de Especificação da Mensagem de *Designer*) [Leite, 1998] que passou por uma série de transformações em sua sintaxe com o aperfeiçoamento, adaptação e surgimento de novos termos e conceitos. A gramática da IMML foi escrita em XML Schema para facilitar o processamento por ferramentas de desenvolvimento de IU [Costa Neto, 2005].

A solução arquitetural proposta nesse trabalho, tem como base a separação dos aspectos definidos na IMML. A principal vantagem em usar os modelos da IMML na arquitetura, além de permitir a separação dos conceitos, é poder construir interfaces de usuários finais, com funcionalidades reais, baseada nas interfaces abstratas e concretas definidas pelos modelos de interação e de comunicação.

No presente trabalho, algumas alterações na IMML foram realizadas com a finalidade de incluir dois aspectos importantes: o tratamento de exceções que possam ocorrer durante a execução dos sistemas e a composição das interfaces, de forma que para cada ação realizada pelo usuário o sistema forneça uma interface formada por um conjunto de outras tarefas, ou seja, para cada interação do usuário com o sistema, este mostre um conjunto de outras interações que possam ser realizadas. Essas alterações resultaram em mudanças nos três modelos da IMML. A seguir, está descrito, brevemente, cada um desses modelos, indicando a inclusão dos novos elementos. Mais detalhes de cada modelo podem ser encontrados em outros trabalhos como, por exemplo, em [Costa Neto, 2005].

Os exemplos descritos a seguir foram extraídos de um estudo de caso um sistema bancário fictício, chamado “Banco Rico”. Este sistema possui quatro funcionalidades básicas: consultar saldo, emitir extrato, efetuar pagamento simples e efetuar pagamento com consulta de saldo. Esse sistema já foi descrito usando cenários descritivos e especificado em IMML em trabalhos anteriores – Fonseca [2005], Lira [2006] e Costa Neto [2005] –, utilizado como

exemplo exploratório no próprio desenvolvimento da IMML. Os cenários descritivos são mostrados no anexo I deste trabalho. A especificação completa em IMML feita com base nos cenários descritivos foi originalmente descrita por Costa Neto [2005], e está reproduzida em [http://www.ppgsc.ufrn.br/~fabiola/BancoRicoIMML.xml].

4.1.1. Modelo de domínio (*domain model*)

O modelo de domínio (*domain model*) determina a funcionalidade do sistema, ou seja, aquilo que um sistema permite fazer através dos objetos e funções de domínio. Os objetos do domínio (*domain-objects*) se referem a registros de 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 de domínio, o *designer* deve determinar o nome do objeto e o tipo de representação. Alguns objetos são formados a partir de outros. A figura 4-1 exibe um objeto de domínio (*domain-object*) que define para o sistema bancário um objeto composto chamado "Extrato". Este objeto "Extrato" é formado por dois outros objetos de domínio: transação e saldo. Cada item utilizado para formar objeto "Extrato" deve ser declarado como elemento <*item*>. As figuras apresentadas a seguir não descrevem a sintaxe da linguagem. Elas mostram trechos de um exemplo, um sistema bancário fictício, que já foi descrito em outros trabalhos.

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

Figura 4-1: Objeto de Domínio Extrato

As funções do domínio (*domain-function*) referem-se aos processos executados pelo computador que mudam o estado de um objeto do domínio. Do ponto de vista do usuário, é um serviço computacional que realiza um caso de uso. Uma função do domínio deve definir os operandos (*input-operands* e *output-operands*) (objetos do domínio), exceções (*exceptions*), pré-condições (*pre-conditions*), pós-condições (*pos-conditions*), controle de execução (*controls*) e estado da execução (*states*). A figura 4-2 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". A exceção (linha 13) indica que um dado de entrada incorreto foi fornecido pelo usuário. Os estados (linhas 20 a 22) são: "inicial", "consultando" e "final". A função tem dois controles (linhas 16 e 17): 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 último a transição é feita de forma automática, após o processamento da função que faz a consulta do extrato.

A descrição de uma pré-condição <*pré-condition*> é feita como um texto em linguagem natural. Este texto deve ser lido pelo desenvolvedor para auxiliá-lo no processo de implementação da função.

Na solução arquitetural proposta, foi definido um elemento arquitetural, *Servidor Funcional*, descrito no próximo capítulo, com a mesma função do modelo de domínio. Ele deve possibilitar a execução das funções definidas no modelo de domínio, incluindo os objetos de domínio especificados nesse esse modelo.

```
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.    <exceptions>
13.          <exception name="input-operand-exception">Se o operando não satisfaz a pré-condição ou o dado fornecido não existe</exception>
14.    </exceptions>
15.    <controls>
16.        <control name="Consultar" from-state="Inicial" to- state="Consultando"/>
17.        <control automatic="yes" from-state="Consultando" to-state="Final"/>
18.    </controls>
19.    <states>
20.        <state name="Inicial" />
21.        <state name="Consultando" />
22.        <state name="Final" />
23.    </states>
24. </domain-function>
```

Figura 4-2: Função de domínio "EmitirExtrato"

4.1.2. Modelo de Interação (*interaction model*)

O modelo de interação representa o processo de interação entre o usuário e a aplicação. Define as ações executadas pelo usuário para comandar funções do domínio. Os elementos básicos deste modelo são: tarefas (*tasks*), comandos de função (*function-commands*), resultados de função (*function-results*), exceções de funções (*function-exceptions*), interações básicas e estruturas de interação. Uma interação básica refere-se a

determinada interação do usuário com a interface através de uma ação, como clicar num botão, selecionar um elemento em uma lista, digitar um texto ou número e visualizar um resultado. 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. 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. Já um comando de função é uma composição de um conjunto de interações básicas de forma estruturada, onde uma interação básica se refere a uma ação do usuário ao interagir com um *widget* de interface.

A figura 4-3 exibe o comando de função (*function-command*) para a função de domínio (*domain-function*) "EmitirExtrato". Os elementos de interação básica estão estruturados como *select, sequence,* e *join.* Existe um elemento <*select*> (linhas 2 a 20) que engloba todos os outros elementos, este elemento define que o usuário pode executar uma seqüência (*sequence* definida da linha 3 a 18) ou não executar a função (*go* definido na linha 19). Se o usuário escolher executar a seqüência, então, primeiro o usuário deve entrar com três informações: "Conta", "Agencia" e "Saldo". Depois o usuário informa o período do extrato, o que pode ser informado de duas formas. O usuário escolhe (*select* das linhas 7 a 20) a maneira como quer informar o período. A primeira opção é informar o mês do extrato (*join* definido da linha 8 e 11), e a segunda é informar a data de início e a data de fim do extrato (*join* definido da linha 12 a 16).

```
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.      <select>
8.        <join>
9.          <perceive-information> Escolha o mês que deseja consultar o extrato </perceive-information>
10.           <enter-information domain-object="Mes"/>
11.       </join>
12.       <join>
13.         <perceive-information> Entre com a data de inicio e final ou somente a data de inicio</perceive-information>
14.         <enter-information domain-object="DataInicio"/>
15.         <enter-information domain-object="DataFim"/>
16.       </join>
17.     <activate control="Consultar"/>
18.   </sequence>
19.  <go direction="away"/>
20. </select>
21.</function-command>
```

Figura 4-3: Comando de Função para a função de domínio "EmitirExtrato"

O resultado de função (*function-result*) é a saída de funções de domínio. Ele é o responsável por prover uma resposta para o usuário com o resultado sobre o processo de interação ocorrida. A figura 4-4 exibe o resultado de função "SaidaExtrato" para a função de domínio "EmitirExtrato".

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

Figura 4-4: Resultado de função "SaidaExtrato"

Para as mensagens de erro ou alertas criamos o elemento exceção de função. Ele é o responsável por prover uma resposta ao usuário caso este informe os dados incorretos para o sistema. A figura 4-5 exibe a exceção (*exception*) para a função de domínio "EmitirExtrato".

```
1.<exception name="DadosExtratoIncorretos" domain-function='EmitirExtrato'>
2.    <join>
3.       <perceive-information>Os dados fornecidos estão incorretos. Forneça- os novamente.</perceive-information>
4.       <do function-command="EmitirExtrato" />
5.    </join>
6.</exception>
```

Figura 4-5: Exceção da função 'EmitirExtrato'

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. A figura 4-6 exibe a tarefa "EmitirExtrato", que possui as estruturas de interação *sequence,* o comando de função (*function-command*) "Extrato" e o resultado de função (*function-result*) "SaídaExtrato". Para o resultado de função fica implícito que este pode ser um resultado correto (*function-result='SaidaExtrato'*) ou uma exceção (*function exception='DadosExtratoIncorretos'*).

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

Figura 4-6: Tarefa "EmitirExtrato"

4.1.3. Modelo de Comunicação (*communication model*)

O modelo de comunicação refere-se à mensagem global que o *designer* constrói para comunicar ao usuário a mensagem do domínio e da interação, considerando aspectos de plataforma. Seus principais elementos são utilizados para organizar a apresentação dos elementos funcionais e de interação, apresentados nas seções anteriores. Os elementos do modelo de comunicação são: painel de comandos (*commands-panel*), área de exibição (*display-area*), menu de tarefas (*tasks-menu*), unidade de interfaces (*UI-units*) e ambiente de tarefas (*task-environment*).

Um painel de comando (*command-painel)* organiza e apresenta os elementos interativos que serão utilizados para realizar um comando de função (*function-command*). 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 4-7 exibe um painel de comando (*command-panel*) para a função de comando "Saldo", que possui três caixas para edição de texto (*edit-box*) (linhas 2, 3 e 4) e dois botões (*push-button*) (linhas 6 e 7).

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

Figura 4-7: Painel de comando para a função de comando "Saldo"

A área de exibição (*display-area*) é o local onde os elementos interativos devem ser inseridos para comunicar ao usuário os resultados de uma função de domínio. Os resultados podem ser válidos, como a exibição do saldo de um cliente, ou inválidos, caso os dados de entrada fornecidos pelos usuários forem incorretos. As figuras 4-8 e 4-9 ilustram as áreas de exibição (*display-area*) para os resultados válidos e inválidos, respectivamente, da função Saldo. Para os resultados válidos foram usados quatro elementos <*text*> para exibir objetos de domínio e um elemento <*push-button*> que declara que a interface deve mostrar um botão. No resultado inválido foi usado um elemento <*text*> para exibir uma mensagem e um <*push-button*>.

```
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 4-8:Área de exibição do resultado de função "SaidaSaldo"

```
1.<display-area function-exception="DadosSaldoIncorretos" name="ErroSaldo" orientation="vertical" align="left">
2.      <text>Os dados fornecidos estão incorretos. Forneça-os novamente.</text>
3.      <push-button label="Sair" hide="this" />
4.</display-area>
```

Figura 4-9: Área de exibição para os dados inválidos

O elemento menu de tarefa (*task-menu*) foi criado para representar um menu que se repete em vários momentos durante a execução das tarefas. Por exemplo, para o sistema bancário fictício, o menu de tarefas é formado por quatro botões que indicam as funções do sistema, conforme mostra a figura 4-10.

```
1.<task-menu name="BancoRico" orientation="vertical" align="left">
2.    <push-button label="Saldo" Task="ConsultarSaldo" />
3.    <push-button label="Extrato" Task="ConsultarExtrato" />
4.    <push-button label="Pagamento" Task="RealizarPagamento" />
5.    <push-button label="Pagamento Com Saldo Antes"  Task="RealizarPagamentoCSaldo"/>
6.</task-menu>
```

Figura 4-10: Menu de tarefas do sistema 'BancoRico'

Outro elemento criado foi a unidade de interface de usuário (*UI-unit*). Este permite a união de signos que representam resultados de função com signos de acionamento de controle de comando. Em outras palavras, ao visualizar o resultado de uma interação, o usuário pode interagir com outros elementos solicitando outras tarefas. A figura 4-11 mostra as unidades de interfaces para a tarefa 'saldo' nas várias formas de interação. Por exemplo, no momento que a interface deve mostrar os elementos de entrada necessários para a execução da função, ela também fornece outros elementos interativos, no caso um menu que permite a escolha de novas tarefas. Assim, a *UI-unit* associada ao *function-command=''Saldo''* (linhas 1 a 4) é formada por um menu de tarefas (*task-menu*)e por um painel de comandos (*command-panel*). A unidade de IU associada a interação *function-result=''SaidaSaldo''* da tarefa saldo é formada por um menu de tarefas (*task-menu*) e por uma área de exibição (*display-area*) (linhas 5 a 8). E a unidade de IU associada a interação *function*

exception=”DadosSaldoIncorretos” da tarefa saldo é formada por um menu de tarefas e por uma área de exibição dos dados incorretos (linhas 9 a 12).

```
1.<UI-unit name="Saldo" task="ConsultarSaldo" function-command="Saldo">
2.      <show task-menu="BancoRico"/>
3.       <show command-panel="Saldo" />
4.</UI-unit>
5.<UI-unit name="ResultadoSaldoCorreto" task="ConsultarSaldo" function-result="SaidaSaldo">
6.          <show task-menu="BancoRico"/>
7.          <show display-area="SaidaSaldo" />
8.</UI-unit>
9.<UI-unit name="ResultadoSaldoIncorreto" task="ConsultarSaldo" function exception="DadosSaldoIncorretos">
10.    <show task-menu="BancoRico"/>
11.    <show display-area="DadosSaldoIncorreto" />
12.</UI-unit>
```

Figura 4-11: Unidades de interfaces para a tarefa ‘ConsultarSaldo’

O ambiente de tarefa (*task-environment*) também é um elemento de composição, utilizado para agregar outros elementos, mas diferentemente dos outros, ele é usado apenas para especificar o conjunto de elementos necessários para a realização das tarefas de uma aplicação uma plataforma específica. A figura 4-12 exibe o ambiente de tarefa (*task-environment*) para a plataforma *desktop-gui* da aplicação “BancoRico”, contendo unidades de IU (*UI-units*), menu de tarefas (*task-menu*), painéis de comando (*command-panel*) e áreas de exibição (*display-area*).

Deve ser especificado um ambiente de tarefa para cada plataforma que possa executar a aplicação do sistema. Para diferentes plataformas, os elementos terão diferentes formas de organização e apresentação, mas sempre seguindo a mesma ordem de interação especificada no modelo de interação. Por exemplo, o painel de comandos da função saldo pode apresentar todos os elementos interativos (“agencia”, “conta” e “senha”) em um único *frame* em um computador convencional, enquanto que em um celular, são necessários vários *frames* dispostos de maneiras distintas, cada *frame* apresentando um elemento interativo por vez. Cada plataforma também pode utilizar diferentes objetos de interfaces concretas para representar o mesmo elemento de interface abstrata definido no modelo de interação. Essas decisões dependem, por exemplo, se um dado vai ser inserido através de uma caixa de seleção ou através de uma caixa de texto.

Essa flexibilidade de apresentação dos elementos permite especificar, em um único modelo, interfaces concretas para cada plataforma que se queira executar o sistema.

```
<task-environment name="BancoRico" plataform="Desktop-gui" >
    <UI-units>
     <UI-unit name="Saldo" task="ConsultarSaldo" function-command="Saldo">
     </UI-unit>
     <UI-unit name="ResultadoSaldoCorreto" task="ConsultarSaldo" function-result="SaidaSaldo">
     </UI-unit>

     <UI-unit name="ResultadoSaldoIncorreto" task="ConsultarSaldo" function exception="DadosSaldoIncorretos">
     </UI-unit>
     ...
    </UI-units>
    <task-menu name="BancoRico" orientation="vertical" align="left">
        <push-button label="Saldo" Task="ConsultarSaldo" />
        <push-button label="Extrato" Task="ConsultarExtrato" />
        <push-button label="Pagamento" Task="RealizarPagamento" />
        <push-button label="Pagamento Com Saldo Antes"  Task="RealizarPagamentoCSaldo"/>
    </task-menu>
    <commands-panel>
      <command-panel function-command="Saldo" name="Saldo" title="BancoRico" orientation="vertical" align="left">
      </command-panel>
      <command-panel function-command="Extrato" name="Extrato" title="BancoRico" orientation="vertical" align="left">
      </command-panel>
      <command panel function command="Pagamento" name="Pagamento" title="BancoRico" orientation="vertical" align="left">
      </command-panel>
      <command-panel function-command="PagamentoCSaldo" name="PagamentoCSaldo" title="BancoRico" orientation="vertical" align="left">
      </command-panel>
    </commands-panel>
    </displays-area>
      <display-area function-result="SaidaSaldo" name="MostraSaldo" orientation="vertical" align="left">
      </display-area>
      <display-area function-result="SaidaSaldo" function-exception="DadosSaldoIncorretos" name="ErroSaldo"
    orientation="vertical"  align="left">
      </display-area>
      <display-area function-result="SaidaExtrato" name="MostraExtrato" orientation="vertical" align="left">
      </display-area>
      <display-area function-result="SaidaExtrato" function-exception="DadosSaldoIncorretos name="ErroExtrato"
    orientation="vertical" align="left">
      </display-area>
      <display-area function-result="SaidaPagamento" name="MostraPagamento" orientation="vertical" align="left">
      </display-area>
      <display-area function-result="SaidaPagemento" function-exception="DadosSaldoIncorretos
    name="ErroPagemento" orientation="vertical"    align="left">
      </display-area>
    </displays-area>
</task-environment>
```

Figura 4-12: Ambiente de tarefa para a plataforma "Desktop-gui"

4.2. Modificações na IMML

Como descrito anteriormente, este trabalho resultou em alterações na IMML, visando melhorar alguns aspectos também já descritos na seção anterior. Essa seção mostra um resumo de quais foram os elementos incluídos na linguagem.

Visando melhorar a composição das interfaces, de forma que para cada ação realizada pelo usuário o sistema forneça uma interface formada por um conjunto de outras tarefas, ou seja, para cada interação do usuário com o sistema, o sistema mostre um conjunto de outras interações que possam ser realizadas, foram criadas duas estruturas no modelo de comunicação: o menu de tarefas <*task-menu*>, e as unidades de interfaces de usuário <*UI-unit*>.

O menu de tarefas <*task-menu*> é formado por um conjunto de tarefas principais do sistema, que deverão ser exibidas na maioria das interações do usuário com o sistema, de forma que as tarefas formadas por esse menu estejam disponíveis a qualquer momento.

As unidades de IU <*UI-unit*> são as estruturas que permitem compor cada interação das tarefas do sistema a partir de outras estruturas de interações.

O tratamento de exceções que possam ocorrer durante a execução dos sistemas resultou em mudanças descritas a seguir.

A inclusão do elemento *exception* no modelo de domínio. Este deve estar inserido na especificação de cada função de domínio, com a finalidade de especificar a possibilidade de exceções nos casos em que os dados de entrada sejam informados incorretamente. Dessa forma, essa estrutura serve para especificar quando a aplicação do sistema não conseguir executar determinada função de domínio devido os dados de entrada estarem incorretos.

Na estrutura do modelo de interação foi incluído o elemento <*function-exception*> que deve estar associado a cada função de domínio. Este elemento é composto por estruturas de interação que informem que aconteceu um erro durante a execução da função.

No modelo de comunicação, é possível especificar as exceções nas unidades de IU que compõem o resultado de tarefas e nas áreas para exibir as exceções de cada tarefa. Existe uma *UI-unit* associada a cada tarefa para compor a interface que mostra a exceção. Um dos elementos formados pela *UI-unit* associada a interação *function-exception* é o *display-area* que contém os objetos de interfaces concretas para apresentar a exceção.

Essas mudanças na IMML foram mais uma contribuição para melhorar a forma de especificar as interfaces de usuários, incluindo alguns aspectos considerados importantes que a linguagem não permitia especificar. Vários trabalhos envolvendo a IMML foram realizados em outros trabalhos visando o aprimoramento, validação, extensões e aplicações desta, com o objetivo de acompanhar as evoluções no *design* e na especificação de IU. Esses trabalhos são descritos na próxima seção.

O uso da IMML, na arquitetura proposta caracteriza-se como parte da solução por permitir a separação dos conceitos funcionais, interativos e comunicativos, como também por permitir especificar interfaces de usuários para multiplataformas.

4.3 Trabalhos relacionados à IMML

A aplicação da IMML no desenvolvimento de IU multiplataformas foi realizada e serviu de base para a construção de novas aplicações [Costa Neto, 2005]. A IMML mostrou-se adequada para esse tipo de aplicação devido ao nível de abstração que ela possibilita e a característica de separar os aspectos funcionais, interativos e de comunicação de uma

aplicação interativa. O mesmo trabalho apresentou a descrição da sintaxe da IMML usando o XML Schema.

Para facilitar o processo de modelagem utilizando a IMML foi criada uma versão diagramática da mesma. A Visual IMML [Machado, 2006] é um perfil UML baseado na IMML que tem por objetivo permitir uma melhor visualização dos modelos que descrevem uma interface de usuário. A VISUAL IMML foi construída de forma a manter o alto nível de abstração da IMML e utiliza um conjunto de novos estereótipos que permitem construir diagramas que descrevem os modelos de domínio, de interação e de comunicação, mostrando ainda como eles estão relacionados.

Experimentos de avaliação da IMML foram conduzidos. Experimentos de *design* utilizando a IMML e outras técnicas de modelagem e especificação centradas-no-usuário, como GOMS e Prototipação foram conduzidos para avaliar as vantagens e desvantagens da IMML em comparação com estas técnicas. Esta avaliação foi utilizada para melhorar a IMML [Fonseca, 2005].

A ferramenta BRIDGE (*Interface Design Generator Environment*), mapeia uma especificação abstrata da IMML em uma interface concreta de usuário em Java que pode ser aproveitada diretamente em um sistema computacional ou somente para a criação de protótipos. Ela auxilia o *designer* a projetar e construir interfaces de usuário, diminuindo a distância que existe entre especificações existentes e as interfaces que eles visam descrever, através de um processo automático provido pelo BRIDGE (Silva, 2007).

O trabalho de Sousa [2004] apresenta a XICL (*eXtensinble User Interface Components Language*), uma linguagem baseada em XML para descrição de interfaces de usuário e de componentes de IU para sistemas baseado em *browser* (SBB). A XICL visa oferecer uma forma padronizada para o desenvolvimento de interfaces que viabilize a reusabilidade, a extensibilidade e a portabilidade dos componentes, com isso facilitando a geração de IUs para os SBB a partir de IUs mais abstratas. Foram implementados, em Java, dois compiladores, um que gera código XICL a partir de especificações de IUs feitas em IMML, e outro 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.

Os principais trabalhos desenvolvidos com IMML que estão relacionados com a solução arquitetural proposta são: [Costa Neto, 2005] que estendeu a linguagem desenvolvendo o modelo de comunicação que permite descrever interfaces de usuários para múltiplas plataformas, [Sousa, 2004] que implementou regras de mapeamento possibilitando gerar interfaces finais a partir de especificações em IMML para aplicações no estilo de

interface WUI, e o trabalho de [Silva, 2007] que também implementou regras que mapeiam especificações em IMML para interfaces finais, nesse caso para o estilo de interface GUI. A solução arquitetural proposta utiliza os modelos de interação e de comunicação para gerenciar e controlar as tarefas dos usuários e as interfaces apresentadas para cada plataforma, e utiliza as regras de mapeamento como uma possibilidade para gerar interfaces de usuários de forma automática.

CAPÍTULO 5

5. A Arquitetura de Referência

A solução aqui proposta foi baseada na separação dos conceitos propostos pela IMML permitindo que os componentes incorporassem aspectos funcionais, interativos e comunicativos dos sistemas interativos de forma independente, onde os aspectos funcionais estariam totalmente independentes da interação e da comunicação, aspectos de interação estariam dependentes da parte funcional e os aspectos da comunicação dependentes da interação e da funcionalidade.

Esta solução arquitetural descrita a seguir engloba a modelagem do sistema usando a visão conceitual, em termos de componentes, conectores e configurações, e a visão de módulo [Hofmeister et al., 2000], mostrando também o comportamento da arquitetura aplicado em um estudo de caso, usando o diagrama de colaboração da UML.

5.1 Visão Conceitual

A solução arquitetural é proposta para um ambiente cliente-servidor. O princípio básico usado para a elaboração da visão conceitual foi a separação da parte funcional da aplicação da parte da apresentação do sistema ao usuário [Coutaz, 1993]. Essa separação é importante para permitir que um único servidor de aplicação seja desenvolvido e possa ser acessado por vários clientes em diversos dispositivos e plataformas, evitando que para cada plataforma que utilize o sistema diferentes aplicações sejam desenvolvidas. Na estrutura da arquitetura foram incorporadas partes intermediárias correspondentes aos aspectos interativos e comunicativos. Essas partes são responsáveis em controlar a interação do usuário com o sistema e organizar as interfaces concretas de usuário de acordo com as plataformas que requisitam os serviços do sistema. Esse controle e gerenciamento de interação e interfaces

concretas são realizados de maneira transparente a parte da apresentação, garantindo a interoperabilidade da comunicação, ou seja, o cliente que acessa o sistema, independente de que dispositivo e plataforma ele use, interagirá com o mesmo seguindo uma ordem de interação e comunicação definida pelas especificações em IMML, sem perceber que o processo de realização das atividades está sendo controlado.

A visão conceitual foi definida utilizando como base algumas fundamentações teóricas [Garlan & Shaw, 1994] para auxiliar a construção da arquitetura em termos de componentes e conectores. Para representar estes componentes, vamos usar a notação da UML 2.0 [OMG, 2005] para descrever essas estruturas.

Os componentes e conectores foram definidos considerando a independência funcional entre eles, onde cada um tem uma função bem definida dentro do contexto de sistemas interativos multiplataformas. A organização dos elementos é mostrada na figura 5-1.

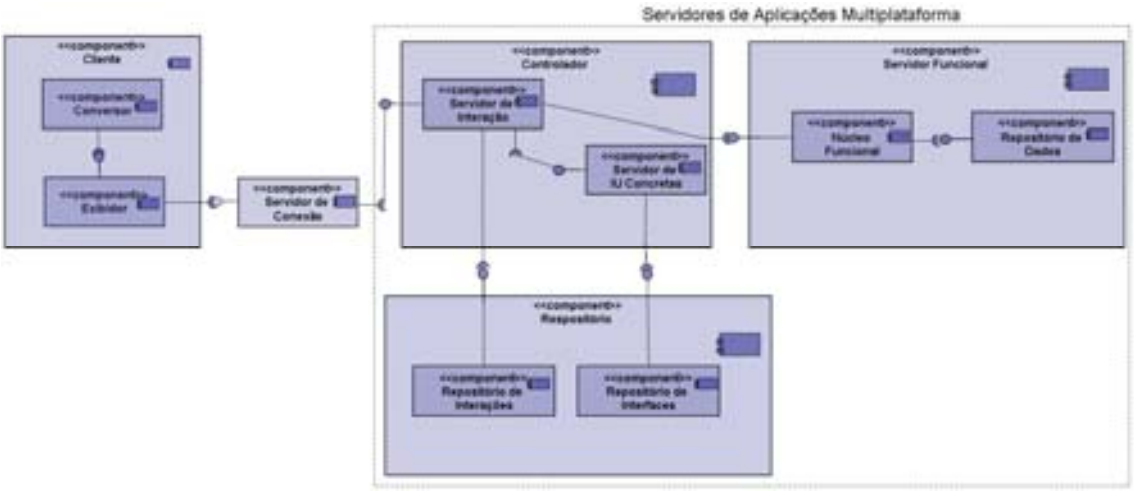

Figura 5-1: Visão Conceitual para Sistemas Interativos Multiplataformas

A idéia principal desta arquitetura é dividir responsabilidades entre diferentes componentes. Essa divisão segue o modelo conceitual da IMML, separando os aspectos funcionais, interativos e de comunicação. Dessa forma, a arquitetura prevê um componente que oferece serviços para cada um desses aspectos.

Foi definido um conjunto de componentes arquiteturais que oferecem serviços aos clientes que acessam o sistema nas multiplataformas. Esses servidores, chamados 'Servidores de Aplicações Multiplataformas' provêm os mesmos serviços para os diferentes clientes que acessam o sistema utilizando diferentes dispositivos e plataformas.

O componente *Servidor funcional* tem por objetivo prover as funcionalidades de uma aplicação. As funcionalidades oferecidas por este componente seguem a especificação do modelo de domínio da IMML.

O componente *Cliente* é responsável por exibir as IU para os usuários. O *Cliente* é implementado em uma plataforma específica e precisa converter a IU concreta numa IU final. Ou seja, uma IU concreta descrita em IMML precisa ser convertida para a forma final que o dispositivo é capaz de exibir. Por exemplo, se a interface for para a plataforma WUI em um computador de mesa (*desktop*), um conversor deve transformar a descrição na tela em IMML em linguagem HTML que pode ser exibida em um *browser*. Se a interface for para uma GUI, também em um computador de mesa (*desktop*), o conversor deve transformar de IMML para Java, utilizando uma API como Swing.

O componente *Controlador* é elemento central desta arquitetura. Ele recebe solicitações de clientes (a execução de uma tarefa) e toma as decisões para que essa tarefa seja executada. O *Controlador* é formado por dois componentes: *Servidor de interação* e *Servidor de IU concretas*. O componente *Servidor de interfaces de usuário (IU) concretas* tem por objetivo fornecer os componentes de IU específicas para as interações de cada tarefa. A IU concreta deve estar de acordo com a plataforma do cliente solicitante (o tipo de dispositivo).

O *Servidor de interação* é o responsável por intermediar esta solicitação. Ou seja, ele recebe uma solicitação de tarefa de um cliente em uma plataforma específica, solicita ao *Repositório de interações* a seqüência de interação para a tarefa e solicita ao *Servidor de IU concretas* as telas correspondentes à tarefa, interação e plataforma. A IU concreta é descrita na linguagem IMML e corresponde aos elementos do modelo de comunicação. Mais adiante, explicaremos como a IMML descreve as IU concretas.

Os componentes *Repositório* e *Servidor funcional* auxiliam o *Controlador* a gerenciar a execução das funções do sistema. O *Repositório* disponibiliza os elementos de interfaces de usuário (telas para entradas e saídas, por exemplo), e o *Servidor funcional* executa a funcionalidade do sistema. O conhecimento que o *Controlador* tem sobre o processo de interação, isto é, qual funcionalidade está ligada a cada tarefa e qual IU deve ser utilizada para ela, é especificada no modelo de interação e de comunicação da IMML, que deve estar armazenada nos componentes *Repositório de interações* e *Respositório de interfaces,* respectivamente.

O *Controlador* deve se conectar aos clientes através de um *Servidor de conexão*. Este *Servidor de conexão* deve abstrair as diferentes formas de conexões possíveis entre os *Clientes* e o *Controlador*. Por exemplo, na plataforma WEB, essa conexão seria em HTTP

com um servidor *Apache*. Para uma plataforma GUI, essa conexão poderia ser com *sockets*, para os quais teríamos um servidor para estabelecer conexões com o cliente.

Em resumo, esta arquitetura permite que o *Controlador* possa atender solicitações de clientes de diversas plataformas e oferecer uma forma de IU concreta descrita em IMML para cada um deles. Além disso, a arquitetura permite que um único grupo de componentes, Servidor de Aplicação Multiplataformas (formado pelos componentes *Controlador*, *Servidor Funcional* e *Repositório*), seja utilizado para atender às diferentes solicitações.

A configuração da arquitetura está representada pelas ligações entre os componentes. O sentido de requisições de serviços acontece da esquerda para a direita, ou seja, o componente *Cliente* requisita serviço ao componente *Servidor de conexão*, que requisita serviço ao componente *Controlador*, que usa os serviços dos componentes *Servidor funcional* e *Repositório*.

A seguir, daremos alguns detalhes sobre esses componentes, dessa vez, descrevendo o fluxo de execução a partir do cliente.

5.1.1 Componente Cliente

O componente *Cliente* tem como responsabilidade apresentar a IU aos usuários em uma plataforma específica, possibilitando a interação do usuário com o mesmo. Também é responsabilidade desse componente converter arquivos IMML em código da plataforma de IU final, conforme o dispositivo usado. Por exemplo, para um sistema bancário, o componente *Cliente* disponibiliza o acesso às funções consultar saldo, emitir extrato e efetuar pagamento em uma tela de *desktop* ou celular usando algum *browser*, caso o estilo seja WUI, ou apresenta essas funções através de algum dispositivo usando uma API (Interface de Programação de Aplicação, na sigla em inglês) de interface de usuário gráfica (GUI). A implementação desse componente deve variar conforme a plataforma usada para acessar o sistema.

Quando um usuário solicita a execução de qualquer função do sistema, o componente *Cliente* deve passar essa solicitação ao componente *Servidor de conexão,*

5.1.2 Componente Servidor de Conexão

O componente servidor de conexão é o responsável em estabelecer a conexão com o *Cliente* usando algum tipo de comunicação cliente/servidor. Por exemplo, para o *Cliente* usando a plataforma no estilo WUI, o *Servidor de conexão* pode ser implementado pelo

Apache, mas para *Cliente* usando plataforma no estilo GUI o *Servidor de conexão* pode ser implementado pelos protocolos do servidor *socket*. Para as plataformas no estilo de interação WUI o conector que liga o componente *Cliente* ao componente *Servidor de conexão* deve implementar algum protocolo cliente/sevidor, como o protocolo de comunicação HTTP ou WAP, dependendo se o dispositivo usado for *desktop* ou celular. Para as plataformas no estilo de interação GUI o conector deve ser implementado por uma chamada de função.

O componente *Servidor de conexão* é o responsável em reconhecer solicitações de *Cliente* em qualquer plataforma. Esse componente transmite a solicitação ao componente *Servidor de interação*.

5.1.3 Componentes Controlador e Repositório

O componente *Servidor de interação* é o componente responsável em gerenciar as interações do usuário com o sistema. Ele foi elaborado com base no modelo de interação da IMML que contém a descrição da seqüência de execução das tarefas do usuário. Esse componente é o responsável em controlar os dados de entrada e saída de informações para cada função específica do sistema. Exemplo: para a função "saldo" do sistema bancário, esse componente sabe que os dados de entrada necessários para a execução dessa função são: agência, conta e senha, e que em seguida o resultado da função é representado através da data e o saldo. Essas informações são obtidas do *Repositório de interações* que contém as especificações do modelo de interação da IMML.

Para gerar as interfaces que serão enviadas aos clientes, o *Servidor de interação* utiliza a função do componente *Servidor de IU concreta*. O *Servidor de IU concreta* monta as interfaces de usuários, usando a especificação do modelo de comunicação da IMML. Ou seja, o *Servidor de UI concreta* organiza em descrições IMML o conteúdo das páginas que devem ser exibidas para cada plataforma específica. O *Servidor de interação* informa quais os dados que devem ser apresentados na tela e para qual plataforma ele quer a interface, e o *Servidor de UI concreta* organiza esses dados de acordo com a plataforma específica. Essa organização é feita usando as especificações do modelo de comunicação contida no *Repositório de interfaces*. O *Servidor de interação* também deve solicitar serviço ao componente *Núcleo funcional* quando for necessária a execução de determinada função.

5.1.4 Componente Servidor Funcional

O *Núcleo funcional* é o responsável pela execução da funcionalidade do sistema. Exemplificando: para um sistema bancário, ele executa o processamento das funções saldo, extrato e pagamento fornecendo os seus resultados para cada cliente específico. Para isso ele pode usar dados do repositório de dados que contém, por exemplo, dados sobre os clientes, contas e movimentações. Por exemplo, para o cálculo do extrato, o *Servidor de interação*, dispondo dos dados de entrada para executar essa função, requisita o extrato ao componente *Núcleo funcional.* Este componente fornece o resultado do extrato para o servidor de interação. O *Núcleo funcional* deve ser implementado baseado no modelo de domínio da IMML. Portanto, ele deve representar as mesmas funções e objetos de domínio especificados na IMML, bem como obedecer as pré-condições e pós-condições especificados no modelo de domínio.

Um exemplo à guisa de síntese: o componente *Núcleo funcional* fornece os valores para o extrato requisitado para um determinado cliente. O componente *Servidor de interação* recebe o resultado do núcleo funcional. Em seguida, solicita ao *Servidor de IU concretas* a IU concreta para a plataforma específica. Para isto, o *Servidor de IU concreta* verifica no repositório qual é a IU de resultados daquela tarefa para a plataforma específica e retorna para o *Servidor de Interação* seguindo a ordem de interação obtida do *Repositório de interação*. As IU concretas são, portanto, um esqueleto de tela (*template*) no qual existe um local onde os resultados específicos da tarefa fornecidos pelo núcleo funcional podem ser inseridos.

Em seguida, o *Servidor de interações* deve retornar a IU concreta em IMML e os resultados para o componente *Cliente*, por intermédio do *Servidor de conexão.* No *Cliente* as descrições IMML devem ser transformados em interfaces da GUI ou em páginas HTML, conforme os estilos de interação, seja GUI ou WUI, respectivamente. Para essas transformações são usadas as propostas de Silva [2007], que transforma elementos da IMML em elementos do Java Swing, e de Sousa [2007], que faz um mapeamento de elementos da IMML para HTML.

Como as IU concretas especificadas em IMML para as multiplataformas seguem o mesmo modelo de interação, em outras palavras, as IU concretas seguem as mesmas especificações de organização e estrutura (*layout* de telas) e de protocolos de interação (comportamento), possibilita que elas sejam consistentes entre si. Portanto, como as interfaces de usuários finais são baseadas nas interfaces concretas, se espera que as interfaces finais possuam comportamentos consistentes em todas as plataformas.

5.2 Visão de Módulo

A solução proposta é formada por camadas baseada no ambiente cliente-servidor. A camada de nível superior utiliza serviços da camada inferior. A figura 5-2 ilustra a organização em camadas. Para acessar as funções do sistema a camada Cliente usa os serviços da camada Conexão que por sua vez requisita os serviços da camada Controlador, que acessa os serviços da camada Servidor Funcional. A incorporação das camadas Conexão e Controlador auxiliam controlar as atividades dos usuários e organizar as interfaces apresentadas aos clientes. Cada camada é formada por módulos que encapsulam os dados e as operações que fornecem os serviços das camadas.

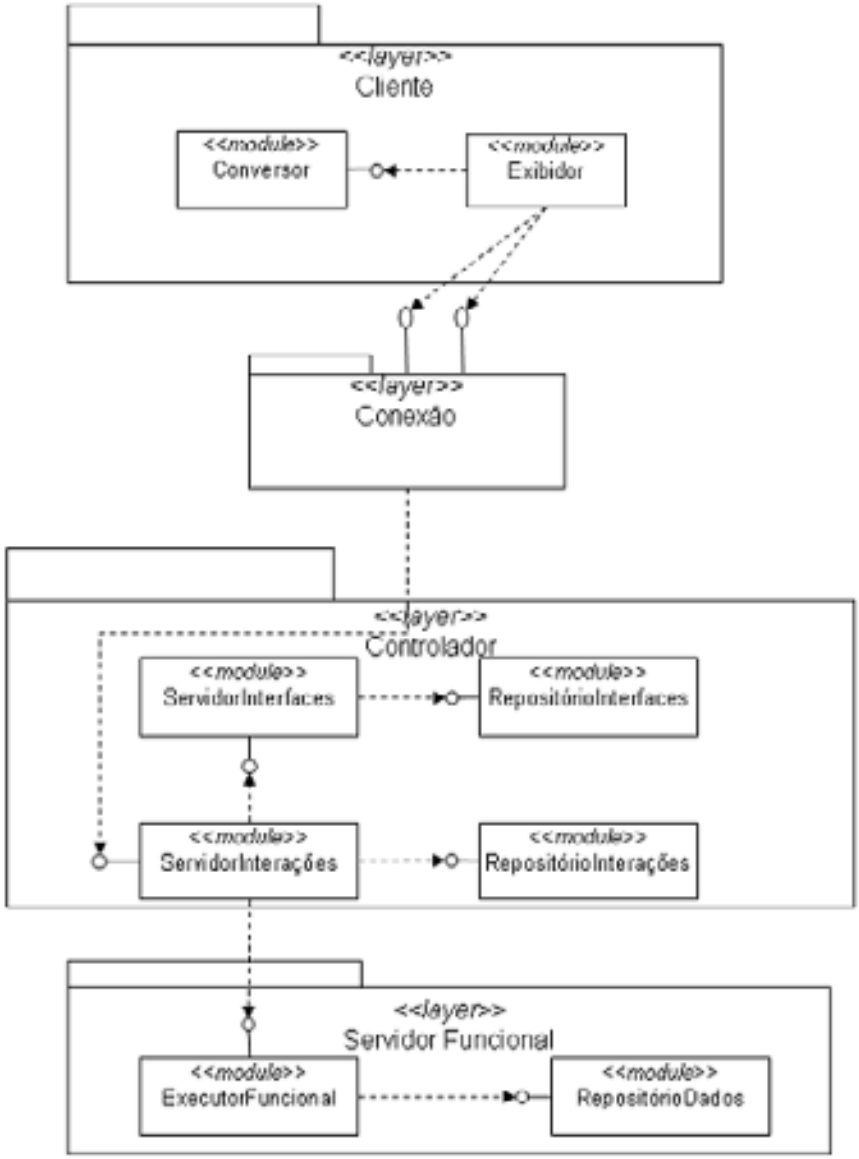

Figura 5-2: Estrutura em camadas

Essa seção apresenta uma possível implementação dos módulos. Serão descritos os principais métodos que devem ser implementados pelas classes, ou seja, serão descritas as interfaces de cada módulo juntamente com a descrição do comportamento do sistema fornecendo um melhor entendimento dos módulos e suas interfaces.

As explicações a seguir são dividas em duas seções: Uma que detalha a arquitetura para as plataformas seguindo o estilo GUI e a outra que detalha a arquitetura para as plataformas no estilo WUI. Essa separação é necessária uma vez que a forma de implementação dos componentes *Cliente* e *Servidor de conexão* e a forma como acontece a comunicação entre esses componentes diferem nessas duas categorias de plataformas.

5.2.1 Descrevendo a Arquitetura para as Plataformas no Estilo de Interação GUI

Para as plataformas no estilo de interação GUI todas as interfaces estão exemplificadas considerando a linguagem de programação Java, pois essa dispõe de uma API adequada para aplicações multiplataformas. São descritos a seguir os principais métodos que devem ser implementados pelas classes, aqueles que são visíveis às outras classes e que possibilitam o entendimento do comportamento da arquitetura.

A camada *Cliente* é implementada pelas interfaces: *IExibidorGUI* e *IConversor*, como mostra a figura 5-3, que correspondem as interfaces dos módulos *Exibidor* e *Conversor* da figura 5-2.

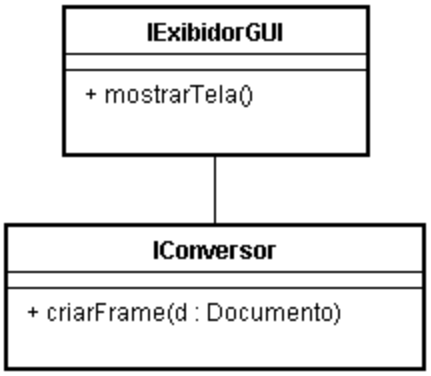

Figura 5-3: Interfaces dos módulos Conversor e Exibidor do estilo GUI

O *IExibidorGUI* tem a função de exibir a IU final e receber os dados do usuário de forma a permitir a interação usuário-sistema. A interface *IConversor* tem como função principal fazer a conversão das especificações IMML para elementos da GUI. Ou seja: a classe que implementa esse método traduz interfaces concretas, descritas em IMML, para

interfaces de usuário final. O trabalho de Silva [2007] propõe uma conversão do modelo da IMML para elementos da GUI descritos em Java.

A figura 5-4 mostra as interfaces que devem ser implementadas pela camada Conexão para as plataformas no estilo GUI.

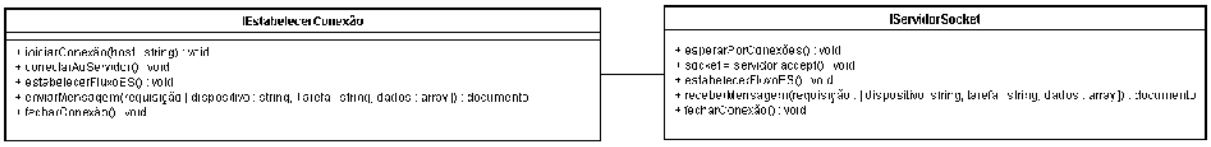

Figura 5-4: Interfaces da camada *Conexão* no estilo GUI

As interfaces *IEstabelecerConexão* e *IServidorSocket* devem implementar todos os métodos necessários para a comunicação via *socket*, desde a criação da conexão, troca de mensagens até a finalização da conexão.

A interface *IServidorSocket* aguarda conexões até que o cliente solicite um serviço e a *IEstabelecerConexão* inicie uma conexão. Após conexão estabelecida, ambos devem estabelecer um fluxo para entrada e saída dos dados para então passar a mensagem. A troca de informações é feita pelos métodos *enviarMensagem( )* e *receberMensagem( ),* que recebem como argumentos uma requisição composta por duas *strings* indicando a plataforma que requisita o serviço e o nome da tarefa solicitada e os dados, compostos por um conjunto de valores necessários à execução da função. Por exemplo, as *strings* "*desktop*" e "BancoRico" indica a requisição da tela inicial do aplicativo Banco Rico para a plataforma desktop, as *strings "*celular*"* e "saldo" indica a requisição da tela para entrada de dados para a consulta do saldo no celular. Essas funções aguardam como resposta um documento XML, especificação da interface concreta que será transformada em interface final. O método *receberMensagem( )* faz uma chamada ao método *gerenciarInterações( )* do módulo Servidor de Interações descrito mais adiante.

As interfaces descritas a seguir são equivalentes em qualquer plataforma.

Os módulos *Servidor de interação* e *Servidor de Interfaces* da camada Controlador são implementados pelas interfaces *IServidorDeInterações,* e *IServidorDeInterfaces.*

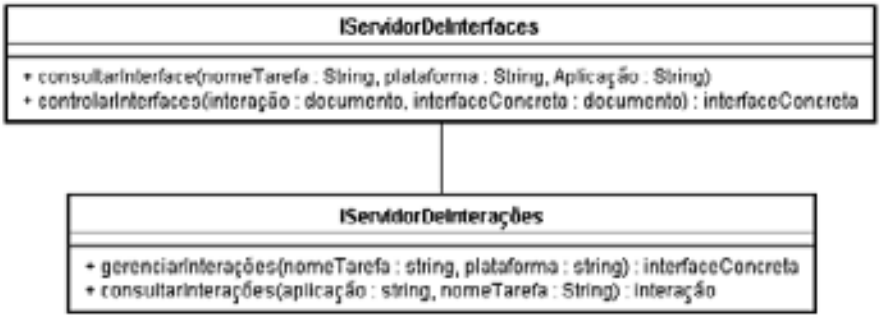

Figura 5-5: Interfaces da camada Controlador

A interface *IServidorDeInterações* é a responsável em implementar a funcionalidade do componente *Servidor de interação*. Ela gerencia as tarefas do sistema de acordo com a requisição recebida e com a plataforma. Os principais métodos responsáveis pelo gerenciamento são o *gerenciarInterações( )* e o *consultarInterações ( )*. O *gerenciarInterações( )* , após receber requisições do *Servidor de conexão*, faz uma chamada ao método *consultarInterações( ).* Esse método auxilia o *gerenciarInterações ( )* fazendo uma consulta ao *Repositório De Interações* para obter o conjunto de interações necessárias para execução da tarefa requisitada.

A *IServidorDeInterfaces* é a interface que implementa o componente *Servidor de IU concretas*. Os seus principais métodos são: o *consultarInterface( )* e o *controlarInterfaces ( )* . O *consultarInterfaces ( )* obtém do *Repositório De Tarefas* as especificações da interface concreta para uma determinada aplicação em uma plataforma específica. O *controlarInterfaces()* analisa os documentos da especificação das interações (fornecido pelo *Repositório De Interações*) e da especificação das interfaces concretas (fornecido pelo *Repositório De Interfaces)*. Nesta análise, ele precisa identificar quais as unidades de interface são necessárias para a interação definida no modelo de interação. Para tanto, os parâmetros de entrada para esse método são o documento com as interações e o documento com a interface concreta. O resultado da análise é o trecho da especificação da interface concreta contendo a descrição da unidade da interface que permite realizar a próxima interação, definida no modelo de interação.

Os módulos Repositório de interfaces e Repositório de interações são implementados pelas interfaces *IRepositórioDeInterfaces* e *IRepositórioDeInterações* respectivamente.

Figura 5-6: Interfaces dos módulos Repositórios

A interface *IRepositórioDeInterações* é responsável por armazenar as especificações do modelo de interação da IMML em arquivos separados por nome da tarefa contendo a seqüência das interações que deverão ser realizadas. Ela contém um método *recuperarInterações*( ) que deve retornar ao *Servidor De Interações* o documento em IMML com a interação correspondente a tarefa solicitada. Por exemplo, se o usuário entrou com a tarefa "saldo", o *recuperarInterações*( ) procura o documento de interações correspondente para essa tarefa. Esse documento especifica que a próxima interação a ser executada é o cálculo da função saldo e a tela a ser apresentada ao usuário deve conter os dados para entrada dos dados necessários para o cálculo da função, e em seguida deve aparecer o resultado do saldo.

A interface *IRepositórioDeInterfaces* é responsável por armazenar as especificações do modelo de comunicação da IMML, organizando em diretórios os arquivos em IMML separados por plataformas, por aplicação e por tarefas. Ou seja, ele armazena especificações IMML do modelo de comunicação, organizadas por aplicações (uma aplicação bancária, por exemplo), por plataforma (para um celular WAP, por exemplo), e por tarefas (a função extrato, por exemplo). À medida que novas plataformas sejam necessárias para uma aplicação, novas especificações para aquela plataforma serão armazenadas no repositório. A *IRepositórioDeInterfaces* retorna para a *IServidorDeInterfaces* um arquivo com a especificação IMML com o modelo de comunicação de uma tarefa de uma aplicação para uma determinada plataforma. O método *recuperarInterface*( ) da *IRepositórioDeInterfaces* é o responsável em retornar ao *Servidor De Interfaces* as interface concreta em IMML para a aplicação, tarefa e plataforma solicitada.

As módulos Executor Funcional e Repositório de Dados da camada do Servidor Funcional são implementados pelas interfaces: *IExecutorFuncional* e *IRepositórioDeDados,* respectivamente.

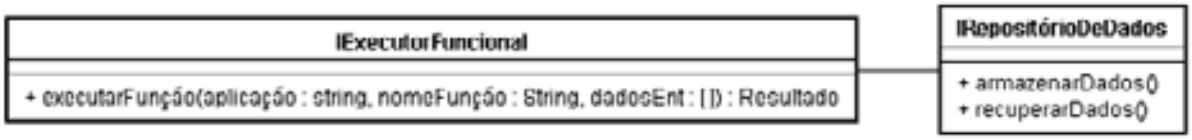

Figura 5-7: Interfaces dos módulos Executor Funcional e Repositório de Dados

Seguindo o fluxo descrito anteriormente, após obter o documento com as interações, o *gerenciarInterações( )* solicita ao módulo *Executor Funcional* a execução da função desejada, enviando os parâmetros fornecidos pelo usuário. A interface *IExecutorFuncional* contém um método que calcula as funções do sistema, como o valor do saldo de um cliente específico. Para execução das funções, o *Executor Funcional* pode efetuar uma consulta ao *Repositório De Dados*. Esse repositório contém os dados necessários para a execução das funções. No caso de um sistema bancário, ele deve conter dados dos clientes e suas movimentações. A consulta a esses dados pode ser realizada através de algum conector com bases de dados, como a API do JDBC. Após a execução da função requisitada, o *Executor Funcional* devolve o resultado da função para o *gerenciarInterações( )*. Em seguida, o *gerenciarInterações( )* requisita ao *Servidor De Interfaces* a interface concreta necessária para apresentar o resultado de função passando como parâmetros a plataforma que solicita o serviço, a aplicação e a tarefa requisitada. Para isso, ele faz uma chamada ao método *consultarInterface( )* da interface *IServidorDeInterfaces.* Esse método faz uma consulta ao *Repositório De Interfaces* para obter a especificação em IMML que indica a composição de uma interface para a tarefa e plataforma solicitadas.

Por fim, o *gerenciarInterações( )* obtém a especificação IMML contendo a interface concreta para a tarefa solicitada pelo usuário (obtido do *Servidor De Interfaces*) e o resultado da execução da função (obtido do *Executor Funcional*).

Este documento e o resultado são retornados ao *Cliente* que transformará em interface de usuário final.

5.2.2. Descrevendo a arquitetura para Plataformas no Estilo de Interação WUI

Para o estilo de interação WUI os módulos das camadas Gerenciador e Sistema são implementados da mesma forma para as plataformas no estilo de interação GUI, descrita na

seção anterior. O que difere é a forma de implementar os módulos da camada Cliente e Conexão.

A camada Cliente é implementada pelas classes responsáveis pela visualização dos elementos da interface de usuário final no *Browser*, juntamente com a classe que tem como função fazer a conversão de documentos XML para páginas HTML. O trabalho de Sousa [2004] propõe o mapeamento de elementos do modelo de comunicação da IMML para elementos HTML.

A camada Conexão pode ser implementado por classes responsáveis pelo estabelecimento da conexão no *Browser* e por algum servidor WEB, como o *Apache*. Não faz parte deste trabalho detalhar as classes implementadas pelo *Browser* e também como ocorre a conexão via WEB, uma vez que não conhecemos os métodos, as classes e protocolos, mas sabemos que existem e são usados. Portanto, abstraindo os detalhes de como ocorre a comunicação representamos por componentes as classes que devem ser implementadas no *Cliente* e o *Servidor de conexão* da WUI . A figura 5-9 mostra essa abstração.

O componente *ExibidorBrowser* tem a responsabilidade de apresentar as páginas HTML nos dispositivos; o componente *Conversor* é responsável em transformar documentos IMML em páginas HTML; o componente *ConectorBrowser* faz a conexão do *Browser* com o servidor WEB e o *ServidorWEB,* que por sua vez aceitam pedidos HTTP dos *browsers* servindo como resposta o documento HTML. Como todo o restante do sistema deve estar implementado em Java, o *ServidorWEB* pode ser o *Tomcat* executando JSPs. Os programas JSPs devem possuir métodos que recebam as solicitações dos clientes e envie a solicitação ao método *gerenciarInterações( )* da classe *ServidorDeInteração.* Essa interação é possível, pois JSP é uma tecnologia independente de plataforma e compatível com Java.

Embora implementados por classes diferentes e usando protocolos diferentes para comunicação, as responsabilidades dos componentes *Cliente* e *Servidor conexão* são mantidas em todos os estilos de interação, seguindo os mesmos conceitos dos elementos arquiteturais definidos na arquitetura de referência.

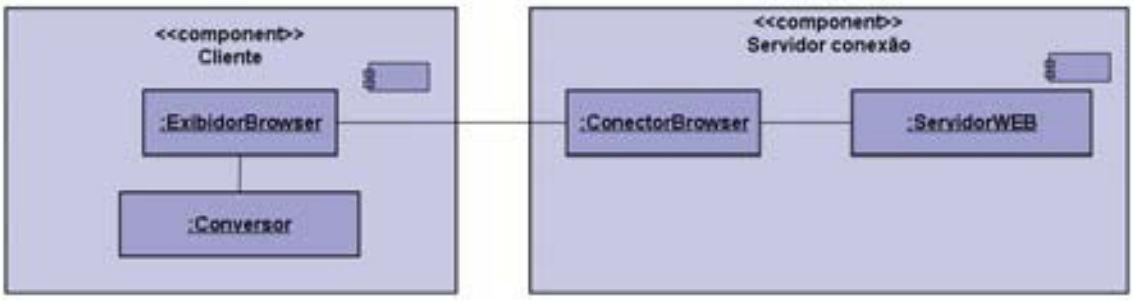

Figura 5-8: Representação dos componentes *Clientes* e *Servidor Conexão* para plataformas WUI

5.3 Aplicando a Solução Arquitetural em um Estudo de Caso

Um sistema baseado na arquitetura foi implementado por um aluno. Os códigos do sistema estão disponíveis em www.lcc.ufrn.br/~jacksondvs. Esse sistema pode ser utilizado para exemplificar a arquitetura aplicando um estudo de caso. O estudo de caso que utilizamos nesse trabalho, é o "Banco Rico". O mesmo que foi descrito em trabalhos anteriores utilizado para a especificação em IMML. O estudo de caso descreve a utilização de uma mesma aplicação de um sistema com diversas operações (consulta de saldo, consulta ao extrato, efetuar pagamento simples e efetuar pagamento com consulta de saldo) e várias formas de utilizações (clientes utilizando celular ou computadores convencionais acessando o sistema através de navegadores da WEB e aplicações baseada em API de interfaces gráficas.).

A seguir descrevemos o comportamento do sistema "Banco Rico" sendo acessado por um cliente em *desktop* utilizando uma aplicação de interface gráfica GUI.

5.3.1 Comportamento do Sistema

Para descrever o comportamento do sistema foi utilizado o diagrama de colaboração da UML. Esses diagramas mostram os principais objetos que formam o sistema e a interação entre eles durante a sua execução do sistema. Nessa seção é mostrado o comportamento do sistema bancário fictício "Banco Rico" especificamente para o caso de um usuário (usando uma aplicação de interface gráfica GUI em *desktop*) acessar o sistema e verificar o seu saldo.

A figura 5-10 mostra a seqüência das mensagens trocadas entre os objetos para abrir a tela inicial do sistema. Essas mensagens correspondem aos métodos das classes que implementam os componentes da arquitetura, descritas na seção anterior. Nos diagramas foi abstraída a troca de mensagens para estabelecer as conexões entre o cliente e os servidores de conexão.

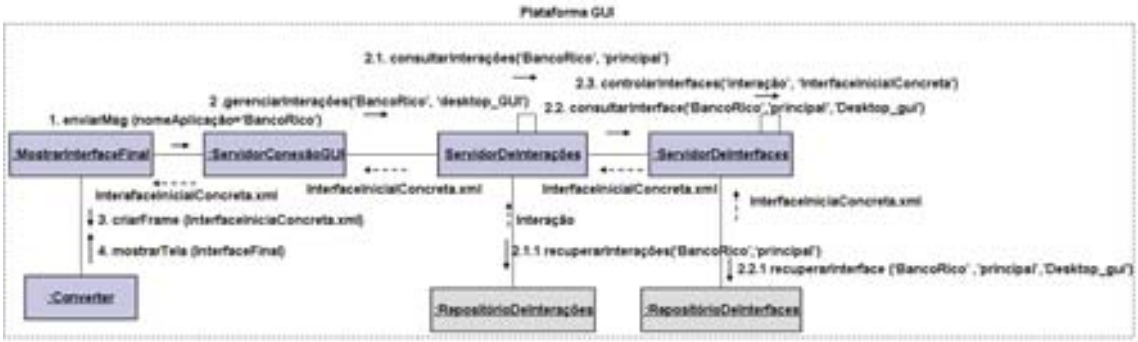

Figura 5-9: Diagrama de colaboração para a seqüência de abrir a tela inicial do sistema

Seguindo o diagrama, considerando que a conexão foi estabelecida, o cliente solicita o acesso ao sistema. A partir desse momento, o objeto *MostrarInterfaceFinal* requisita esse acesso ao objeto *ServidorConexãoGUI*, através da mensagem 1, passando como parâmetro o nome da aplicação – no caso 'Banco Rico' – solicitando a tela inicial do sistema.

mensagem 1: enviarMsg(nomeAplicacao="BancoRico")

Em seguida, o *ServidordeConexãoGUI* (representando todas as classes que estabelecem conexão com *sockets*) solicita ao *ServidorDeInterações* o gerenciamento das interações, mensagem 2, passando como parâmetros o nome da aplicação e a plataforma, nesse caso "Banco Rico" e "desktop_gui".

mensagem 2: gerenciarInteracoes("BancoRico", "desktop_GUI")

Para gerenciar as interações, o *ServidorDeInterações* requisita métodos auxiliares: o *consultarInterações ( )*, mensagem 2.1, deste mesmo objeto, o *consultarInterfaces ( )* , mensagem 2.2, do objeto S*ervidorDeInterfaces* e o *controlarInterfaces( )* do objeto *ServidorDeInterfaces.*

O *consultarInterações ( )* recebe como parâmetros o nome da aplicação e a tarefa requisitada, que nesse caso de solicitação da tela inicial do sistema, o nome da aplicação é "Banco Rico" e o nome da tarefa é "Principal". O *consultarInterações ( )* é o método que solicita ao objeto *RepositórioDeInterações* o conjunto de interação para a tarefa requisitada, mensagem 2.1.1.

mensagem 2.1: consultarInterações ("BancoRico", "Principal") mensagem 2.1.1: recuperarInterações ("BancoRico", "Principal")

O *RepositórioDeInterações* que tem armazenadas as especificações do modelo de interação da IMML organizadas em diretórios separados por nomes de aplicações e tarefas, devolve ao *ServidorDeInterações* o documento XML, contendo as interações para a tarefa "Principal" da aplicação 'Banco Rico'. A organização das especificações em diretórios separados por nomes da aplicação e tarefas facilita a busca do documento XML das interações correspondente à tarefa solicitada. Nesse momento, o documento retornado ao *ServidorDeInterações* é:

```
<task name="Principal">
 <select>
  <do task="EmitirExtrato"/>
  <do task="ConsultarSaldo"/>
  <do task="EfetuarPagamentoSimples"/>
  <do task="EfetuarPagamentoComSaldoAntes"/>
 </select>
</task>
```

Figura 5-10: Seqüência de interação para a tarefa 'Principal' da aplicação 'BancoRico'

Em seguida, o método *consultarInterfaces()* é requisitado recebendo como parâmetros o nome da aplicação ("BancoRico"), a tarefa ("Principal") e a plataforma ("Desktop_gui") . O *consultarInterfaces ( )* é o método que solicita ao objeto *RepositórioDeInterfaces* a interface concreta de acordo com a aplicação, tarefa e plataforma requisitadas, mensagem 2.2.1.

mensagem 2.2: consultarInterfaces ("BancoRico", "Principal","desktop_gui")

mensagem 2.2.1: recuperarInterfaces ("BancoRico","Principal", "desktop_gui")

O *RepositórioDeInterfaces* que tem armazenadas as especificações do modelo de comunicação da IMML organizadas em diretórios separados por nomes de plataformas, aplicações e tarefas, devolve ao *ServidorDeInterfaces* o seguinte documento XML, contendo a interface concreta para a tarefa 'Principal' da aplicação 'Banco Rico' na plataforma 'desktop_gui':

```
1.  <UI-unit name="BancoRico" task="Principal">
2.      <show task-menu="BancoRico"/>
3.  </UI-unit>
4.  <task-menu name="BancoRico" orientation="vertical" align="left">
5.    <push-button label="Saldo" Task="ConsultarSaldo" />
6.    <push-button label="Extrato" Task="ConsultarExtrato" />
7.    <push-button label="Pagamento" Task="RealizarPagamento" />
8.    <push-button label="Pagamento Com Saldo Antes"  Task="RealizarPagamentoCSaldo"/>
9.  </task-menu>
```

Figura 5-11: Interface concreta para tarefa 'Principal' da aplicação 'BancoRico'na plataforma Desktop_GUI

Essa figura mostra que a interface concreta da tarefa "Principal" da aplicação "Banco Rico" é composta por um menu de tarefas (linhas 1 a 3). A especificação do menu de tarefas também é retornada (linhas 4 a 8).

Em seguida, o método *controlarInterfaces( )* do objeto *ServidorDeInterfaces* faz uma comparação entre a seqüência das interações , figura 5-8, (obtida do *RepositórioDeInterações*) e as interfaces concretas, figura 5-9, (obtidas do

RepositórioDeInterfaces), retornando a interface concreta correspondente à próxima interação até o cliente.

mensagem 2.3: controlarInterfaces (interações, interfaceInicialConcreta)

Nesse momento, as interações para a tarefa 'Principal' é uma seleção de outras tarefas que o usuário pode escolher, conforme mostra figura 5-7. Essa interação corresponde à interface concreta da figura 5-8. Essa é a interface concreta retornada até o cliente onde o objeto *Converter* deve transformá-la em interface de usuário final, mensagem 3.

mensagem 3: criarFrame (interfaceInicialConcreta)

Em seguida, o objeto *MostrarInterfaceFinal* exibe na tela a interface final, mensagem 4.

mensagem 4: mostrarTela (interfaceFinal)

Com a tela inicial do sistema exibida no dispositivo, o usuário pode requisitar qualquer função do sistema. A figura 5-9 mostra a seqüência das mensagens para o caso de o usuário requisitar a execução da função saldo.

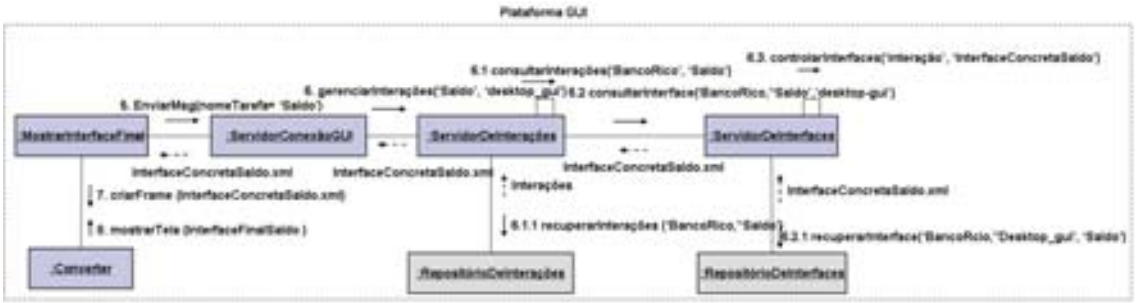

Figura 5-12: Diagrama de colaboração da seqüência para solicitação do saldo

Quando o usuário requisita a função saldo da aplicação 'BancoRico', o objeto *MostrarInterfaceFinal* solicita o método *EnviarMsg ( )* do objeto *ServidorConexãoGUI,* passando como parâmetro o nome da tarefa 'saldo', mensagem 5.

mensagem 5: enviarMsg ("Saldo")

Em seguida, o objeto *ServidorConexaoGUI* requisita que o objeto *ServidorDeInterações* gerencie as interações para a execução da função saldo na plataforma *desktop-gui*, mensagem 6.

mensagem 6: gerenciarInterações ("Saldo", "desktop_gui")

Para isso, o *ServidorDeInterações* deve requisitar o documento contendo as interações para a tarefa 'saldo' ao objeto *RepositórioDeInterações*, mensagens 6.1.

mensagem 6.1: consultarInterações ("BancoRico", "Saldo")
mensagem 6.1.1: recuperarInterações ("BancoRico", "Saldo")

Como o *RepositórioDeInterações* tem armazenado as especificações do modelo de interação da IMML organizadas em diretórios separados por nomes de aplicações e tarefas, o método *recuperarInterações( ),* faz uma busca nos diretórios por nome da aplicação e por tarefa, e retorna o documento XML armazenado.

O *RepositórioDeInterações* recupera o documento com as interações para a função saldo, mensagem 6.1.1, e retorna ao *ServidorDeInterações* o seguinte documento XML do modelo de interação:

```
1. <task name="ConsultarSaldo">
2.    <sequence>
3.         <do  function-command ="Saldo" />
4.         <do function-result ="SaidaSaldo" function exception="DadosSaldoIncorretos"/>
5.     </sequence>
6. </task>
```

Figura 5-13: Seqüência de interações para a tarefa saldo

Consultadas as interações para a tarefa 'saldo' da aplicação 'Banco Rico', o objeto *servidorDeInteracoes* solicita ao objeto *ServidorDeInterfaces* a interface concreta correspondente à tarefa 'saldo' da aplicação 'Banco Rico' na plataforma 'Desktop-gui', mensagem 6.2, passando como parâmetros essas informações. O método *consultarInterfaces ( )* obtém a especificação da interface concreta corresponde a aplicação, tarefa e plataforma solicitada usando o método *recuperarInterfaces ( )* do objeto *RepositórioDeInterfaces,*mensagem 6.2.1.

mensagem 6.2: consultarInterfaces ("BancoRico", "Saldo","desktop_gui")
mensagem 6.2.1: recuperarInterfaces ("BancoRico","Saldo", "desktop_gui")

O objeto *RepositórioDeInterfaces* que tem armazenadas as especificações do modelo de comunicação da IMML organizadas em diretórios separados por nomes de aplicações, tarefas e plataformas, devolve ao *ServidorDeInterfaces* o seguinte documento XML, contendo

a interface concreta para a tarefa ‘Saldo’ da aplicação ‘Banco Rico’ na plataforma ‘desktop_gui’:

```
1.<UI-unit name="Saldo" task="ConsultarSaldo" function-command="Saldo">
2.      <show task-menu="BancoRico"/>
3.      <show command-panel="Saldo" />
4.</UI-unit>
5.<UI-unit name="ResultadoSaldoCorreto" task="ConsultarSaldo" function-result="SaidaSaldo">
6.          <show task-menu="BancoRico"/>
7.          <show display-area="SaidaSaldo" />
8.</UI-unit>
9.<UI-unit name="ResultadoSaldoIncorreto" task="ConsultarSaldo" function exception="DadosSaldoIncorretos">
10.    <show task-menu="BancoRico"/>
11.    <show display-area="DadosSaldoIncorreto" />
12.</UI-unit>
13.<task-menu name="BancoRico" orientation="vertical" align="left">
14.      <push-button label="Saldo" Task="ConsultarSaldo" />
15.      <push-button label="Extrato" Task="ConsultarExtrato" />
16.      <push-button label="Pagamento" Task="RealizarPagamento" />
17.      <push-button label="Pagamento Com Saldo Antes"   Task="RealizarPagamentoCSaldo"/>
18. </task-menu>
19. <command-panel function-command="Saldo" name="Saldo" title="BancoRico" orientation="vertical" align="left">
20.       <edit-box label="Agencia" domain-object="Agencia" />
21.       <edit-box label="Conta" domain-object="Conta" />
22.       <edit-box label="Senha" domain-object="Senha" />
23.        <group orientation="horizontal" align="center">
24.            <push-button label="Consultar" control="Consultar"/>
25.            <push-button label="Cancelar" hide="this" />
26.        </group>
27.</command-panel>
28.<display-area function-result="SaidaSaldo" name="MostraSaldo" orientation="vertical" align="left">
29.    <text label="Data" domain-object="Data"/>
30.    <text label="Agencia" domain-object="Agencia"/>
31.    <text label="Conta" domain-object="Conta"/>
32.    <text label="Saldo" domain-object="Saldo"/>
33.    <push-button label="Sair" hide="this"/>
34.</display-area>
35.<display-area function-exception="DadosSaldoIncorretos" name="ErroSaldo" orientation="vertical" align="left">
36.       <text> Os dados fornecidos estão incorretos. Forneça-os novamente.</text>
37.       <push-button label="Sair" hide="this" />
38.</display-area>
```

Figura 5-14: Interface concreta para a tarefa ‘saldo’ da aplicação ‘BancoRico’

Esse documento contém a composição das interfaces para a tarefa “Saldo” em todas as suas formas de interações: a entrada de dados (linhas 1 a 4), saída dos dados corretos (linhas 5 a 8) e saída para dados inválidos (linhas 9 a 12). A especificação também inclui os elementos que compõe cada interação: o menu de tarefa (linhas 4 a 18), o painel de comandos para entrada de dados (linhas 19 a 27), área de exibição para saída dos dados corretos (linhas 28 a 34) e a área de exibição para os dados inválidos (linhas 35 a 38).

Em seguida, o método *controlarInterfaces( )* do objeto *ServidorDeInterfaces* faz uma comparação entre a seqüência das interações (obtida do *RepositórioDeInterações*) e as interfaces concretas (obtidas do *RepositórioDeInterfaces*), retornando a interface concreta correspondente à próxima interação até o cliente.

mensagem 6.3: controlarInterfaces (interações, interfaceConcretaSaldo)

Nesse caso, as interações para a tarefa 'Saldo' é formada por uma seqüência de ações do sistema, como mostra a figura 5-11. A primeira seqüência da interação, conforme mostra a figura 5-11, é o comando para entrada de dados (linha 3). O método *controlarInterfaces ( )*, percorre esse documento e identifica que essa é próxima interação. Então, ele analisa o documento da interface concreta, figura 5-12, e verifica que a interface concreta que deve ser retornada até o cliente é a parte da especificação da figura 5-12, formada pelas unidades de interfaces que esta associada ao comando para entrada de dados (linhas 1 a 4) como também os seus elementos constituintes, menu de tarefas (linhas 13 a 18) e (linhas 19 a 27). Resultando então na seguinte interface concreta:

```
<UI-unit name="Saldo" task="ConsultarSaldo" function-command="Saldo">
     <show task-menu="BancoRico"/>
      <show command-panel="Saldo" />
</UI-unit>
<task-menu name="BancoRico" orientation="vertical" align="left">
    <push-button label="Saldo" Task="ConsultarSaldo" />
    <push-button label="Extrato" Task="ConsultarExtrato" />
    <push-button label="Pagamento" Task="RealizarPagamento" />
    <push-button label="Pagamento Com Saldo Antes"   Task="RealizarPagamentoCSaldo"/>
</task-menu>
<command-panel function-command="Saldo" name="Saldo" title="BancoRico" orientation="vertical" align="left">
     <edit-box label="Agencia" domain-object="Agencia" />
     <edit-box label="Conta" domain-object="Conta" />
     <edit-box label="Senha" domain-object="Senha" />
      <group orientation="horizontal" align="center">
         <push-button label="Consultar" control="Consultar"/>
         <push-button label="Cancelar" hide="this" />
      </group>
</command-panel>
```

Figura 5-15 : Interface concreta para a interação *function-command* da tarefa 'saldo'

Essa interface concreta é retornada até o cliente onde o objeto *Converter* deve transformá-la em interface de usuário final, mensagem 7.

mensagem 7: criarFrame (interfaceConcretaSaldo)

Em seguida, o objeto *MostrarInterfaceFinal* exibe na tela a interface final para entrada dos dados para a função "saldo", mensagem 8.

mensagem 8: mostrarTela (interfaceFinalSaldo)

Com a tela de entrada dos dados para o cálculo do saldo exibida no dispositivo, o usuário deve interagir com a interface enviando os dados da agência, conta e senha e o sistema deve mostrar um resultado. A figura 5-17 mostra a seqüência das mensagens durante esse processo.

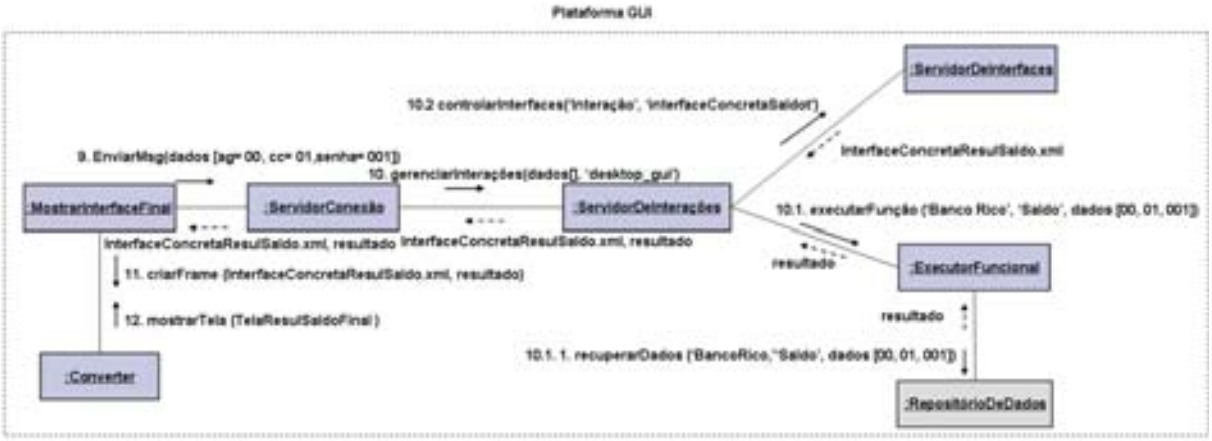

Figura 5-16: Diagrama de colaboração para a seqüência do resultado do saldo

Quando o cliente submete os dados de entrada (agencia, conta e senha), eles serão enviados para o *ServidorConexaoGUI*, mensagem 9, que recebe como parâmetros os valores dos dados necessários para a execução da função "saldo".

mensagem 9: enviarMsg (ag=00, cc=01,senha=001)

Em seguida, o objeto *ServidorDeInterações* irá gerenciar as próximas interações, mensagem 10, recebendo como parâmetros os dados de entrada para a execução da função e nome da plataforma.

mensagem 10: gerenciarInterações ([00, 01, 001], "desktop_gui")

Em seguida, o objeto *ServidorDeInterações* chama a função *executarFunção( )* do objeto *ExecutorFuncional*, passando como parâmetros o nome da aplicação, o nome da função e os dados necessários para executar o saldo, mensagem 10.1.

mensagem 10.1: executarFunção ("BancoRico", "saldo", dados[00, 01, 001])

Para executar a função, o objeto *ExecutorFuncional*, pode consultar no *RepositórioDeDados* os dados da conta e movimentações bancárias do cliente, mensagem 10.1.1. Após a execução da função saldo, a *ExecutorFuncional* retorna o resultado ao

ServidorDeInterações. Esse resultado pode ser o valor do saldo do cliente, contendo o valor do saldo e data, ou pode ser um resultado inválido, caso os dados fornecidos pelo cliente estiverem incorretos.

Com os resultados obtidos, o objeto *ServidorDeInterações* requisita ao objeto *ServidorDeInterfaces*, a interface para o resultado obtido, mensagem 10.2.

mensagem 10.2: controlarInterfaces (interações, interfaceConcretaSaldo, resultado)

O método *controlarInterfaces( ) do* objeto *ServidorDeInterfaces* faz uma comparação entre a seqüência das interações (obtida do *RepositórioDeInterações*) e as interfaces concretas (obtidas do *RepositórioDeInterfaces*), retornando a interface concreta correspondente à próxima interação até o cliente.

Nesse caso, o *controlarInterações ( )* ao percorrer o documento com as interações da função "saldo", figura 5-11, identifica que próxima seqüência de interação é o resultado da função saldo (linha 4). Então, ele analisa o documento da interface concreta, figura 5-12, e verifica que a interface concreta que deve ser retornada até o cliente ou é a parte da especificação formada pela unidade de interface que esta associada ao resultado da função para os dados corretos (linhas 5 a 8) ou a unidade de interface associada ao resultado inválido (linhas 9 a 12) .

Se o resultado obtido for o resultado válido, a seguinte interface concreta é obtida:

```
1.<UI-unit name="ResultadoSaldoCorreto" task="ConsultarSaldo" function-result="SaidaSaldo">
2.          <show task-menu="BancoRico"/>
3.          <show display-area="SaidaSaldo" />
4.</UI-unit>
5.<task-menu name="BancoRico" orientation="vertical" align="left">
6.     <push-button label="Saldo" Task="ConsultarSaldo" />
7.      <push-button label="Extrato" Task="ConsultarExtrato" />
8.      <push-button label="Pagamento" Task="RealizarPagamento" />
9.      <push-button label="Pagamento Com Saldo Antes"   Task="RealizarPagamentoCSaldo"/>
10.</task-menu>
11.<display-area function-result="SaidaSaldo" name="MostraSaldo" orientation="vertical" align="left">
12.    <text label="Data" domain-object="Data"/>
13.    <text label="Agencia" domain-object="Agencia"/>
14.    <text label="Conta" domain-object="Conta"/>
15.    <text label="Saldo" domain-object="Saldo"/>
16.    <push-button label="Sair" hide="this"/>
17.</display-area>
```

Figura 5-17: Interface concreta para a interação *function-result* da tarefa 'saldo'

Caso a aplicação reporte um resultado inválido a especificação da interface concreta seria:

```
1.<UI-unit name="ResultadoSaldoIncorreto" task="ConsultarSaldo" function exception="DadosSaldoIncorretos">
2.   <show task-menu="BancoRico"/>
3.   <show display-area="DadosSaldoIncorreto" />
4.</UI-unit>
5.<task-menu name="BancoRico" orientation="vertical" align="left">
6.    <push-button label="Saldo" Task="ConsultarSaldo" />
7.    <push-button label="Extrato" Task="ConsultarExtrato" />
8.    <push-button label="Pagamento" Task="RealizarPagamento" />
9.    <push-button label="Pagamento Com Saldo Antes"   Task="RealizarPagamentoCSaldo"/>
10.</task-menu>
11.<display-area function-exception="DadosSaldoIncorretos" name="ErroSaldo" orientation="vertical" align="left">
12.      <text> Os dados fornecidos estão incorretos. Forneça-os novamente.</text>
13.      <push-button label="Sair" hide="this" />
14.</display-area>
```

Figura 5-18: Interface concreta para a interação *function-exception* da tarefa 'saldo'

Essas interfaces são retornadas ao cliente, juntamente com resultado da função. O *Cliente* converte em interface final com o resultado e mostra na tela do dispositivo, mensagens 11 e 12. O objeto *Converter* transforma em interface de usuário final, mensagem 11, e o *MostrarIntefaceFinal* exibe na tela a interface final.

mensagem 11: criarFrame (interfaceConcretaResultadoSaldo, resultado)

mensagem 12: mostrarTela (telaResultadoSaldoFinal)

As seqüências das mensagens foram descritas para o caso de um cliente acessando a função 'saldo' da aplicação 'BancoRico' utilizando uma aplicação de interface gráfica GUI em um computador convencional (*desktop*). A forma como é estabelecida a conexão foi abstraída nesses diagramas, mas sua implementação na integra está em www.lcc.ufrn.br/~jacksondvs/BancoRicoGUI e foi feita usando o *sockets*.

Outra situação de uso poderia ser um cliente utilizando o mesmo sistema através do celular também no estilo de interface GUI, onde uma aplicação de interface gráfica deveria ser desenvolvida para o celular acessar o "Banco Rico". Para essa situação, a mesma arquitetura pode ser utilizada, seguindo o mesmo fluxo de comunicação. O que difere é a organização das interfaces finais, que estariam baseadas em interfaces concretas específicas para celulares, descritas em IMML.

Para que o mesmo sistema "Banco Rico" possa ser acessado por dispositivos *desktops* e celulares usando um navegador da WEB, a conexão entre o cliente e os Servidores de Aplicação Multiplaforma seria diferente. As requisições deveriam ser enviadas através de URLs, obedecendo ao padrão de troca de mensagens implementada pela WEB. O servidor de conexão seria outro, nesse caso um servidor da WEB. Outra mudança seria a conversão que o cliente teria que fazer, de documentos XML (interfaces concretas descritas em IMML) para páginas HTML ou WML, conforme o dispositivo fosse *desktop* ou celular. Embora a conexão

entre clientes e servidores seja diferente, a estrutura do sistema obedece a mesma arquitetura, seguindo o mesmo fluxo de comunicação.

A figura 5-20 mostra um resumo de como acontece a troca de mensagens entre os objetos que diferem para a plataforma no estilo de interação WUI. O *ExibidorBrowser* envia as solicitações ao *ServidorWEB* por meio de URL. Nessa etapa envolve todo um conjunto de protocolos utilizados para comunicação via WEB. O objeto *ServidorWEB* (abstraindo as classes que implementam, por exemplo, o *Tomcat*, e as classes do *browser* responsáveis em estabelecer a conexão com o servidor WEB) executam arquivos JSPs . Esses arquivos podem conter uma chamada a função *gerenciarInterações( )* do objeto *ServidorDeInterações*. A mesma função que também é requisita pelo *ServidorConexãoGUI.* A partir daí a troca de mensagens é igual a como acontece nas plataformas GUI.

A interface concreta que é retornada ao cliente deve ser convertida pelo objeto *ConverterDocumento* em páginas HTML ou WML, caso o dispositivo seja *desktop* ou celular.

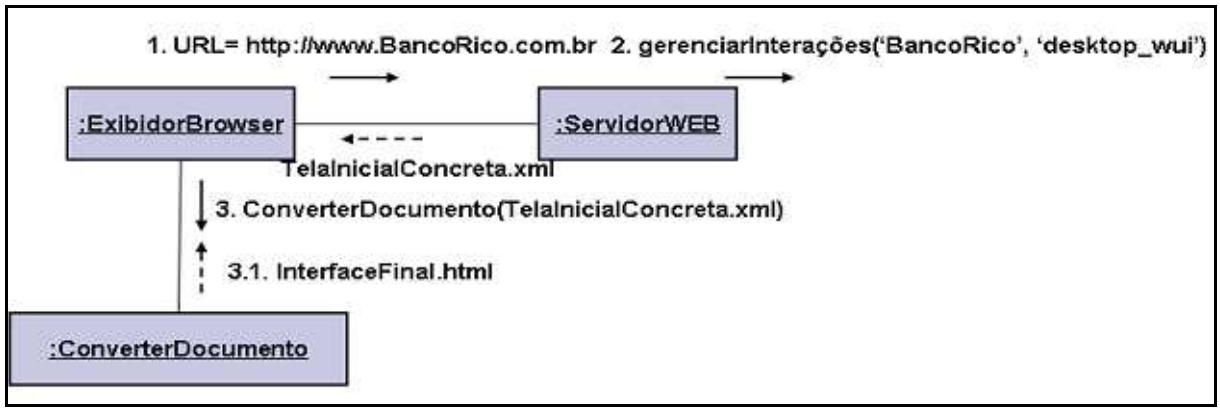

Figura 5-19: Comunicação entre *cliente* e *servidor de conexão* para as plataformas WUI.

Em todas as multiplataformas consideradas (celular e desktops nas plataformas WUI e GUI) a estrutura do sistema segue a arquitetura proposta, garantindo a portabilidade do sistema.

CAPÍTULO 6

6. Conclusão

Foi apresentada uma solução para o desenvolvimento de sistemas interativos multiplataformas. Esta solução descreve uma arquitetura de referência que orienta a construção dos sistemas permitindo a geração de interfaces para as diferentes plataformas em tempo de execução. Esta arquitetura segue o estilo camadas e cliente-servidor; onde podemos ter vários clientes, em diversos dispositivos (celulares ou *desktops*) e plataformas operacionais (sistemas operacionais, ambiente de desenvolvimento, estilo de interface gráfica e meio de acesso a aplicação) utilizando um único servidor de aplicações multiplataformas. Esse servidor é formado por um conjunto de componentes arquiteturais cujos principais serviços oferecidos são: a funcionalidade da aplicação utilizada (por exemplo, as funções saldo, extrato e pagamento de um sistema bancário) e o controle e gerenciamento do processo de interação usuário-sistema, intermediando o acesso do cliente as funções do sistema. Essa solução arquitetural esta baseada na abordagem de desenvolvimento baseado em modelos e no uso de linguagens de descrição de interfaces de usuário. Dessa forma, descrições da interface de usuário abstratas ficam armazenadas em um repositório. Durante a execução do sistema, servidores fornecem aos clientes uma interface concreta para cada plataforma específica, e os clientes convertem em interface final.

Essa solução oferece vantagens importantes. Uma única organização do sistema pode ser construída baseada na arquitetura de referência proposta e usada no desenvolvimento de sistemas interativos multiplataformas. Como vimos no capítulo que descreve o detalhamento da arquitetura, os componentes arquiteturais podem ser instanciados em classes que devem ser implementadas para desenvolver sistemas que possam ser utilizados por vários dispositivos e plataforma. Além disso, o uso de especificações em IMML na arquitetura possibilita construir interfaces de usuários que possam ser apresentadas nas multiplataformas. Costa Neto [2005] apresentou uma extensão da IMML para descrever interfaces para as multiplataformas. Sousa [2004] e Silva [2007] provaram o que foi definido em Costa Neto [2005] implementando interfaces de usuários finais para dispositivos com variações de

tamanho de telas e formas de entrada de dados (*desktops* e celulares) e de estilos de interfaces gráficas (GUI e WUI). Embora eles tenham implementado protótipos de sistemas, sem funcionalidades reais, as regras de mapeamento por eles definidas podem ser empregadas pelos *Conversores* dos *Clientes*, definidos na arquitetura para converter os documentos IMML em interfaces da GUI ou em páginas HTML, conforme o aplicativo utilizado pelo usuário para acessar o sistema.

A arquitetura de referência propõe que os sistemas possam ser construídos de forma que os diferentes clientes acessem um único núcleo funcional do sistema, que o processo de interação usuário-sistema possa ser gerenciado e as interfaces de usuários possam ser dinamicamente construídas para cada dispositivo e plataforma que acesse o sistema. A capacidade de que vários clientes nos diferentes dispositivos e plataformas acessem o mesmo núcleo de funcionalidade do sistema, incorpora a característica de interoperabilidade na arquitetura.

Na solução proposta, uma única arquitetura do sistema permite gerar interfaces de usuários dinamicamente para multiplataformas, obedecendo, portanto, ao princípio de portabilidade. O sistema implementado por um aluno utilizou as classes da instância da arquitetura de referência, descritas no capítulo 5. Esse sistema esta sendo utilizado para exemplificação do estudo de caso "Banco Rico". O seu comportamento para o caso do cliente acessando o sistema para a consulta do saldo através de um *desktop* no estilo de interação GUI foi descrito no capítulo 5. Mostramos que a mesma estrutura do sistema, seguindo a arquitetura de referência utilizada, pode ser aplicada para o dispositivo celular no estilo de interação GUI e para celulares e *desktops* usando navegadores da WEB. Para esses últimos, o tipo de conexão entre os clientes e servidores difere, como também a forma como os clientes devem converter as interfaces concretas em interfaces finais.

A geração das interfaces de usuário de forma dinâmica para as multiplataformas é possível com a utilização das interfaces concretas, que estão armazenadas em repositórios, e que são retornadas até o *Cliente* para serem convertidas em interfaces de usuários finais.

A idéia principal da arquitetura foi dividir responsabilidades entre diferentes componentes. Os principais componentes da arquitetura são: *Servidor funcional*, *Controlador, Repositório, Servidor de conexão* e *Clientes*. Os componentes, *Controlador, Servidor Funcional* e *Repositório* formam os servidores de aplicações multiplataformas. Esses servidores oferecem os mesmos serviços aos clientes nas diversas plataformas.

Durante a descrição do comportamento da arquitetura, onde as especificações em IMML são utilizadas, percebemos a necessidade de alterações na própria estrutura da

linguagem. Esses ajustes foram feitos para possibilitar que essa linguagem especificasse aspectos até então não especificados. Esses aspectos incluíram a descrição de exceções – para o caso de os usuários fornecerem dados incorretos – e a possibilidade de combinar estruturas de interações, de forma que cada interação resultante de uma solicitação de tarefa possa ser composta por outras interações que permitam a realização de novas tarefas. Essas mudanças afetaram os três modelos da IMML, descritos no capítulo 4.

A inclusão dos *exceptions* nos modelos de domínio e de interação resultou na inclusão de um *display-area function-exception* para cada tarefa no modelo de comunicação. Esse *display-area function-exception* apresenta uma área de exibição para os casos em que o usuário fornece os dados não reconhecidos pela aplicação. Para representar a composição das interfaces foi criado o elemento *UI-unit* no modelo de comunicação e o elemento *task-menu* para especificar as tarefas comuns que aparecem na maioria das interações. Como o seu nome indica, ele é formado por um menu de tarefas.

Para o estudo de caso utilizado, sistema bancário fictício 'BancoRico', elaboramos uma nova descrição em IMML incluindo as novas estruturas. As funções desse sistema e as suas interfaces finais para os dispositivos *desktops* (no estilo GUI e WUI) e celular (no estilo WUI) estão sendo implementadas por um aluno, seguindo a arquitetura proposta, e suas principais classes estão descritas no capítulo 5.

Diversas tecnologias são empregadas no desenvolvimento desses sistemas. Aplicações seguindo o estilo de interface GUI foi implementada utilizando a API do Java Swing, que possui classes para a construção de interfaces de janelas de forma totalmente independente de sistema operacional ou ambiente de janelas. Os servidores de aplicações multiplataformas (controlador e funcionalidade do sistema) foram implementados utilizando a linguagem de programação Java.

A comunicação entre a aplicação do cliente *desktop_gui* e os servidores de aplicações foi feita usando *sockets*, que é uma espécie de interface lógica utilizada pelo cliente para se conectar ao servidor. Para dispositivo *desktop* com o estilo de interface WUI, esta sendo utilizado um *browser* baseado em HTML para apresentar o sistema aos usuários que acessam os servidores da aplicação através do servidor *Apache.* Para a geração das páginas em HTML foi utilizada a tecnologia JSP.

Uma limitação desse trabalho é o fato de que a arquitetura possibilita a geração de interfaces de usuários para as plataformas previamente definidas em IMML que estão armazenadas em repositórios. Para que novos dispositivos possam acessar o sistema, e suas

interfaces possam ser geradas dinamicamente, um novo ambiente de tarefa (*task-envinoment*) deve ser especificado em IMML e armazenado no *Repositório.*

É importante ressaltar que, embora as aplicações dos clientes utilizem os mapeamentos entre elementos da IMML e objetos de interfaces GUI e WUI, propostos por Silva [2007] e Sousa [2004], para gerar as interfaces de usuários finais, normalmente, as IUs geradas automaticamente necessitam de ajustes. Como exemplo de ajustes, podemos citar aqueles voltados à disposição espacial dos elementos que formam a interface, pois estes podem ficar dispostos de forma a prejudicar a interação do usuário. Para fazer os ajustes, o *designer* precisa fazer modificações diretamente no código, e isso pode gastar muito tempo para realizar os ajustes, o que atrasa o processo de desenvolvimento.

Essas limitações abrem perspectivas para a realização de trabalhos futuros. Um deles seria a adaptação nas regras de mapeamento utilizadas pelos *Conversores* cada vez que uma interface final fosse gerada automaticamente. Dessa forma, ajustando as regras a cada interface gerada será possível gerar interfaces finais que necessitem cada vez de menos ajustes.

Como visto na introdução, para o desenvolvimento de sistemas interativos multiplataformas, faz-se necessário considerar os aspectos de usabilidade, portanto outro ponto que deve ser considerado é a avaliação das interfaces geradas. A usabilidade de um sistema não é tão simples de ser medida e envolve vários fatores que devem ser consideradas em conjunto durante a avaliação de usabilidade. A prática e a experiência dos avaliadores e projetistas de sistema somadas às padronizações já existentes teve como resultado uma série de princípios que visam melhorar a usabilidade dos sistemas. Esses princípios devem ser considerados como medidas de usabilidade. Dentre os princípios, os mais conhecidos são as Heurísticas de Nielsen [Nielsen, 1990], formadas por dez características que são desejáveis para os sistemas.

O modelo de comunicação da IMML foi desenvolvido como uma forma de possibilitar a descrição de interfaces de usuários concretas para multiplataformas. Essas interfaces são descritas considerando um único conjunto de interações definido no modelo de interação da IMML, a fim de que as interfaces concretas possuam consistência nas informações. Isso foi um dos principais motivos em utilizarmos as especificações em IMML no processo arquitetural, em que as interfaces finais seriam construídas a partir das interfaces concretas. Portanto, podemos incluir como trabalhos futuros a realização de testes de usabilidade nas interfaces geradas para verificar a característica de consistência do comportamento das interfaces.

Para isso é importante que outras aplicações (sistemas), sejam desenvolvidas e implementadas para outros dispositivos (por exemplo, *palmtops*), aplicando a solução arquitetural em uma maior quantidade de sistemas interativos multiplataformas. Portanto, outras especificações em IMML devem ser descritas e armazenadas nos repositórios e, a partir dessas especificações, interfaces sejam construídas dinamicamente.

REFERÊNCIAS BIBLIOGRÁFICAS

ABRAMS, M.; Phanouriou, C.(1999). **"UIML: An XML language for building device independent user interfaces"**. Proceedings of XML'99 (XML'99, Philadelphia), 232-247.

BASS, L.; Clements, P.; Kazman, R. (2003). **Software Architecture in Practice**. 2. ed. ISBN 0-321-15495-9, 2003.

BRAY, T.; Paoli,J.; Sperberg-McQueen, C. M. (1998) **"Extensible markutp language (XML) 1.0 W3C Recommendation."** Disponível em: http://www.w3c.org/TR/1998/REC-xmk-19980210. Acesso em : 17 Mai 2008.

BROWN, A. W (2000). **Large-Scale, Component-Based Development**. USA: Prentice Hall PTR, Upper Saddle River, NJ, 2000.

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

CASTRO, M.; LOUREIRO, A. A (2004). F. Adaptation in Mobile Computing. In: XXII Simpósio Brasileiro de Redes de Computadores, 2004, Gramado, RS. Anais do XXII Simpósio Brasileiro de Redes de Computadores, p.439 – 452, 2004.

CHI-HSING Chu; CHIEN-HSUAN Huang; LEE, M. (2000); Building an XML-based unified user interface system under J2EE architecture. Anais do International Symposium on Multimedia Software Engineering, p.208 – 214, Dezembro, 2000

COSTA NETO, M. A.; Leite, J. C. (2004) **"Uma proposta para o desenvolvimento de interfaces de usuário multi-plataformas com tecnologias Web".** VI Simpósio sobre Fatores Humanos em Sistemas Computacionais — Mediando e Transformando o Cotidiano. UFPR, CEIHC - SBC, Curitiba.

COSTA NETO, M. A. (2005) **"Uma extensão da IMML para apoiar o Design de Interfaces de Usuário Multi-Plataformas".** 122f. Dissertação (Mestrado) – Departamento de Informática e Matemática Aplicada, Universidade Federal do Rio Grande do Norte, Natal, 2005.

COUTAZ, J. (1987) **"PAC, an Object Oriented Model for Dialog Design."** In: H.-J. Bullinger, B. Shackel (eds.), Human-Computer Interaction - INTERACT'87, proc. of the 2nd IFIP conference on HCI, pp. 431-436.

COUTAZ, J. (1993) **"Software Architecture Modeling for User Interfaces",** in Encyclopedia od Software Engineering, Wiley.

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.

DE SOUZA, C.S. (2005). **The Semiotic Engineering of Human-Computer Interaction**, MIT Press, 2005.

FONSECA, 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. Universideda Federal do Rio Grande do Norte, Natal, 2005.

GAMMA, E.; Helm, R.; Johnson, R.; Vlissides, J. (1995), **Design Patterns: Elements of Reusable Object-Oriented Software,** Addison-Wesley, Reading, Massachusetts.

GARLAN, D. (2000) **"Software architecture: a roadmap".** In Proceedings of International Conference on Software Engineering (ICSE00) Limerick, Ireland.

GARLAN, D.; Shaw, M. (1994) **"An Introduction to Software Architecture".** In Advances in Software Engineering and Knowledge Engineering, pages 1-39, Singapore.

GOLDBERG, A. (1984), **Smalltalk-80: The Interactive Programming Environment**. Addison- Wesley Publishers, Menio Park.

GRUNDY, J.; Zou, W. (2004) **"AUIT: Adaptable User Interface Technology, with Extended Java Server Pages"**. In A. Seffah & H. Javahery (eds.). Multiple User Interfaces. John Wiley & Sons, New York, 2004, 149--167.

HAGIMONT, D.; LAYAÏDA, N. (2002). Adaptation d'une application multimédia par un code mobile. In Technique et Science Informatique (TSI), numéro spécial "Agents et code mobile", Volume 21, Issue 6/2002, Junho, 2002.

HOFMEISTER, C.; Nord, R. L.; Soni, D. (2000), **Applied Software Architecture**. Addison Wesley, Reading, MA.

J2EE (2008). Java 2 Plataform Enterprise Edition (J2EE). Disponível em: <http://www.java.sun.com/j2ee/index.jsp >. Acessado em Julho de 2008.

LEITE, J. C (1998). **Modelos e Formalismos para Engenharia Semiótica de Interfaces de Usuário**. Rio de Janeiro: PUC, 1998. 194p. Tese (Doutorado) – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 1998.

_____. (2003) **"Specifying a User Interface as an Interactive Message"**, In: Human-Computer Interaction International, 2003, Heraklion, Creta. Proceedings of HCI International 2003. Mahwah, NJ : Lawrence Erlbaum Associates, 2003. v. 2. p. 716-720.

_____. (2005) **"Improving usability by communicating functional and interaction models through the user interface".** In: HCI International 2005 - 11th International Conference on Human-Computer Interaction, Las Vegas. Human Computer International 2005, U.S. CD. Mahwah, NJ : Lawrence Erlbaum Associates, 2005. v. 1. p. 1-10.

_____.. (2006) **"A Semantic Model to Interactive System".** In Proceedings of the

International Conference on Organizational Semiotics, Brasil, Campinas, SP. v. 1. p. 81 89.

_____. (2007) **"A model-based approach to develop interactive system using IMML".** In: Coninx, Karin; Luyten, Kris; Schneider, Kevin A.. (Org.). Task Models and Diagrams for Users Interface Design. 1 ed. Berlin: Springer, 2007, v. 4385, p. 68-81.

LEMLOUMA, T. LAYAIDA, N (2004). Context-aware adaptation for mobile devices. This paper is from the WAM Project, INRIA, Saint Martin, France. In: Proceedings of IEEE International Conference on Mobile Data Management, p.106 - 111, 2004.

LIRA, H.B. (2006) **Uma ontologia para descrição da semântica da IMML**. Dissertação de Mestrado – Departamento de Informática e Matemática Aplicada, Universidade Federal do Rio Grande do Norte, Natal, 2006

MACHADO,T.L. (2006) **Visual IMML: Um Perfil UML para Modelagem de Interfaces de Usuário.** Dissertação de Mestrado – Departamento de Informática e Matemática Aplicada, Universidade Federal do Rio Grande do Norte, Natal, [2006].

MORI, G., PATERNÒ, F., SANTORO, C. (2003). **Tool Support for Designing Nomadic Applications**. In Proceedings of IUI 2003, Miami, Florida, January, 2003. Disponível em: <http://giove.cnuce.cnr.it/teresa/pdf/iui03.pdf> **.** Acesso em Jun 2008.

NIELSEN, J.; Molich, R. (1990). **Heuristic evaluation of user interfaces**, Proc. ACM CHI'90 Conf. Seattle, WA, 1-5 April, 249-256.

OMG (2005). **UML 2.0 Superstructure Specification**, Object Management Group. Disponível em: <http://www.omg.org/cgi-bin/apps/doc?formal/05-07-04.pdf>. Acesso em:

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

PUERTA, A. R.; 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.

PUERTA, A.; Eisenstein, J. (2001) **"XIML: A Multiple User Interface Representation Framework for Industry"**. In A. Seffah & H. Javahery (eds.). Multiple User Interfaces. John Wiley & Sons, New York, 2004, 119–148.

SEFFAH, A. ; Javahery, H. (2004). **"Multiple User Interfaces - Cross-plataform applications and context-aware interfaces"**. Wiley.

SIEBELINK, R. (2000). **Towards a profile driven service authoring and adaptation platform**. W3C Workshop on Web Device Independent Authoring. October 2000. Disponível em: <http://www.w3.org/2000/10/DIAWorkshop/siebelink.htm>. Acesso em Maio 2008.

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.

SOUCHON, N.; VANDERDONCKT, J. (2003). **A review of xml-compliant user interface description languages.** Trabalho apresentado ao 10. International Conference on Design, Specification, and Verification of Interactive Systems, Madeira, 2003.

SOUSA, L.G. ; 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.

TREWIN, S. ; Zimmermann, G.; Vanderheiden,G. (2003) **Abstract user interface representations: How well do they support universal access?.** In Proceedings of the 2nd ACM International Conference on Universal Usability, Vancouver, Canada, 2003.

XUL (2001) - **Extensible User Interface Language**. Outubro, 2002. Disponível em: <http://www.mozilla.org/projects/xul/xul.html>. Acesso em: 12 maio 2008>.

ANEXO I

Cenários descritivos utilizados em trabalhos anteriores durante o desenvolvimento da IMML e usados como exemplo para aplicar a solução arquitetural proposta. Estes cenários são para uma mesma aplicação com diversas operações e várias formas de utilizações. Ele foi descrito em trabalhos anteriores e usado durante o desenvolvimento da IMML.

O presidente do Banco Rico estava interessado em facilitar o atendimento ao cliente com o investimento em tecnologias. Assim, ele solicitou ao diretor de desenvolvimento uma pesquisa de possíveis situações nas quais os clientes necessitam fazer uso dos serviços dos bancos.

Os resultados desta análise foram os seguintes casos:

Caso 1

João queria saber se já tinha sido depositado o pagamento do seu salário. Desta forma, ele dirigiu-se a um terminal de auto-atendimento que ficava na esquina da sua casa e solicitou um extrato. No terminal, ele escolheu a opção de emitir extrato, forneceu os dados da sua conta através do cartão bancário e forneceu a senha diretamente ao sistema. O sistema perguntou se ele queria o extrato impresso ou para visualização na tela. João escolheu ver na tela. Como o número de transações era alto, foi necessário usar alguns mecanismos do sistema para ver todas as transações.

Caso 2

Maria já estava no ônibus, viajando, quando lembrou que seria o último dia para o pagamento da conta de energia. Seria tarde quando desembarcasse. Assim, através do

aparelho de telefone celular ela realizou o pagamento. Por questões de segurança, o sistema permite o acesso apenas para usuários e aparelhos já cadastrados. Para ter acesso, o usuário deve fornecer a sua conta e a senha utilizando o seu aparelho, que as transmite codificada para o banco. Em seguida, ele deve escolher a opção de pagamento, fornecendo o código de pagamento, a data e o valor. O sistema confirmou o pagamento e apresentou o saldo restante em sua conta.

Caso 3

Pedro não tem muita experiência com Internet, mas como já era tarde e estava chovendo bastante, ele optou for fazer suas tarefas bancárias através do sistema WEB do Banco Rico. Pedro precisava pagar uma conta de telefone, mas precisava saber se tinha saldo suficiente para o pagamento. Na versão WEB do sistema bancário, o usuário tem a opção de fazer pagamento com consulta de saldo. Esta opção evita que ele tenha que fornecer os dados bancários duas vezes, como seria caso ele fizesse as duas tarefas independentes. Assim, Pedro optou por esta tarefa e forneceu a conta e a senha. Em seguida, o sistema apresentou o saldo e ofereceu a opção de fazer o pagamento ou desistir. Ele escolheu o pagamento e forneceu o código de pagamento, a data e o valor. O sistema confirmou o pagamento e apresentou o saldo restante em sua conta.

Com base nestes cenários apresentados, os engenheiros foram encarregados de desenvolver um módulo do sistema que permitisse que os clientes do banco pudessem acessar os serviços de pagamento de contas e consulta de saldo e extrato através de terminais de auto-atendimento, aparelhos de telefonia celular e computadores ligados à WEB. Este módulo do sistema deveria utilizar um mesmo núcleo funcional para os serviços em qualquer uma das interfaces de usuário. Ou seja, as funções de consulta e pagamento deveriam ser as mesmas para todas as formas de acesso.

O diagrama de caso de uso mostrando as funções do Banco Rico: