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

# VISUAL IMML:
# Um Perfil UML para a Modelagem de Interfaces de Usuário

Thais Lima Machado

Natal/RN
Fevereiro de 2006

Thais Lima Machado

# VISUAL IMML:
# Um Perfil UML para a Modelagem de Interfaces de Usuário

Dissertação de Mestrado apresentada à banca examinadora do Programa de Pós-Graduação em Sistemas e Computação da Universidade Federal do Rio Grande do Norte, como requisito parcial para a obtenção do título de Mestre em Sistemas e Computação.

**Orientador:** Prof. Jair Cavalcanti Leite

Natal/RN
Fevereiro de 2006

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

Machado, Thais Lima.

Visual IMML: um perfil UML para a modelagem de interfaces de usuário/ Thais Lima Machado. – Natal, RN, 2006.

102 p. : il.

Orientador: Jair Cavalcanti Leite.

Dissertação (mestrado) - Universidade Federal do Rio Grande do Norte. Centro de Ciências Exatas e da Terra. Programa de Pós-Graduação em Sistemas e Computação.

1. Engenharia de software - Interface de usuário – Dissertação. 2. UML - Dissertação. 3. Visual IMML - Dissertação. I. Leite, Jair Cavalcanti. II. Universidade Federal do Rio Grande do Norte. III. Título.

RN/UF/BCZM CDU 004.41

Thais Lima Machado

# VISUAL IMML:
# Um Perfil UML para a Modelagem de Interfaces de Usuário

Dissertação de Mestrado apresentada à banca examinadora do Programa de Pós-Graduação em Sistemas e Computação da Universidade Federal do Rio Grande do Norte, como requisito parcial para a obtenção do título de Mestre em Sistemas e Computação.

**Orientador:** Prof. Jair Cavalcanti Leite

_____

Prof. Jair Cavalcanti Leite, D. Sc. – UFRN

_____

Prof[a]. Anamaria Martins Moreira, D. Sc. – UFRN

_____

Prof[a]. Maria Cecília Calani Baranauskas, D. Sc. – UNICAMP

Natal/RN
Fevereiro de 2006

Dedico este trabalho, e tudo o que ele representa, a Deus e à minha família, em especial aos meus pais e meu irmão, companheiros constantes e fundamentais em minha vida.

# A G R A D E C I M E N T O S

*A Deus, pela companhia fiel, sempre iluminando os meus caminhos... Ah! Senhor, obrigada também pela vida, por Teu amor e o conforto nas horas mais difíceis... Obrigada Pai por me conduzir em Teus braços quando tudo parecia perdido, reabastecendo-me com a Tua força, fazendo-me acreditar que ao amanhecer tudo seria diferente, os obstáculos e desafios seriam vencidos e as recompensas logo viriam...*

*Aos meus familiares, em especial ao meu pai e minha mãe, Harlano e Daisy, e ao meu irmão Bruno. Obrigada painho por me guiar nos caminhos corretos, pelo esforço e dedicação para que hoje eu pudesse atingir meus objetivos. Obrigada mainha pelos conselhos, participação no meu caminhar e principalmente pela amizade. Você é a minha melhor amiga! Eu sei que não é fácil amar e educar os filhos com responsabilidade, mas hoje, podem ter a certeza de que os seus esforços não foram em vão. Ao meu irmão, obrigada pelo carinho, amizade e desentendimentos (risos)... Tenham a certeza de que tudo isso foi muito importante para a minha formação. Amo vocês!*

*Ao meu futuro esposo, Erick. Que apareceu no fim desta etapa da minha vida, mas que tem uma grande importância. Obrigada por tudo, pelo amor, carinho, atenção, preocupações, ajudas... Sem seu apoio, não seria possível a conclusão deste trabalho. Amor, você é um presente de Deus em minha vida. Amo você!*

*Ao meu orientador e amigo, Prof. Dr. Jair C. Leite que acreditou e me fez acreditar do quanto seria capaz e ajudou nos momentos mais difíceis, desesperadores até, e conduziu a nossa parceria para que tudo saísse "perfeito". Agradeço-lhe principalmente pela confiança, demonstrada também na execução do meu trabalho de dissertação.*

*Ao Grupo de Estudos GIHC da UFRN, sem vocês esse trabalho não teria saído. Obrigada pelas opiniões, criticas e elogios, formamos um grupo lindo. Espero contar sempre com vocês, como vocês poderão contar sempre comigo. Obrigada Pessoal!*

*Aos verdadeiros amigos que encontrei durante o mestrado e que se tornaram minha família em Natal. Sem eles eu não teria sido ninguém! Não seria*

*justo citar nomes, mas aqueles que são realmente especiais sabem o valor que têm para mim. Podem ter a certeza de que aonde eu estiver, vocês estarão comigo.*

*Aos meus amigos de João Pessoa, que são anjos em minha vida. Existem quatro anjos "T" que são mais que especiais, são amigas para todas as horas, da fé às farras. Aprontamos muitas juntas! Obrigada Meninas!. Agradeço: ao anjo do funga-funga que agüentou todas as minhas carências enquanto morava fora de casa; ao anjo da proteção, que toda vez que vinha a Natal me tirava de alguma encrenca; ao anjo madrinha, que se transformou em uma amiga muito especial; ao anjo preto e ao anjo irmã, que sabem o que significam pra mim, sem precisar de palavras e finalmente, ao anjo gordo e ao anjo má, que são meus amigos, mesmo que o tempo e a distância tenham nos separado.Muito obrigada, adoro vocês, meus AMIGOS!*

*A Renata, que me agüentou um ano dentro de um apartamento, participando de minhas alegrias, vitórias, decepções e tristezas. Você é uma grande amiga, ou melhor, uma irmã que Deus me deu a honra de escolher. Obrigada por tudo!*

*Enfim, o meu muito obrigada a todos que, diretamente ou indiretamente, ajudaram durante e até a conclusão desse mestrado...*

*"Porque a Deus nenhuma coisa é impossível!"*

(Lucas 1, 37)

*"O valor das coisas não está no tempo em que elas duram, mas na intensidade com que acontecem. Por isso existem momentos inesquecíveis, coisas inexplicáveis e pessoas incomparáveis."*

(Fernando Pessoa)

**VISUAL IMML:**

**Um Perfil UML para a Modelagem de Interfaces de Usuário**

Autora: Thais Lima Machado

Orientador: Profº Jair Cavalcanti Leite

# R e s u m o

Os processos de desenvolvimento de software proposto pelas abordagens mais recentes da Engenharia de Software utilizam modelos e a UML foi proposta como a linguagem padrão para a modelagem. A interface de usuário é uma parte importante do software e fundamental para melhorar a usabilidade. Porém a UML padrão não oferece recursos adequados para a modelagem da interface de usuário. Várias propostas vêm surgindo para solucionar esse problema. Alguns autores têm utilizado modelos no desenvolvimento de interface (Desenvolvimento Baseado em Modelos) e várias propostas de extensões da UML têm sido elaboradas. No entanto, nenhuma delas considera a perspectiva teórica proposta pela engenharia semiótica que considera que através do sistema o designer deve comunicar para o usuário o que ele pode fazer e como ele utiliza o próprio sistema. Este trabalho apresenta um Perfil UML que enfatiza os aspectos da perspectiva da engenharia semiótica. Esse Perfil está baseado na IMML que é uma linguagem textual declarativa. A Visual IMML, como é denominada, é uma proposta que vem melhorar o processo de especificação com a IMML através de uma modelagem visual (diagramática). Ela propõe um conjunto de novos elementos de modelagem (estereótipos) elaborados para a especificação e documentação de interfaces de usuário, considerando os aspectos de comunicação, interação e funcionais de forma integrada.

**Palavras-Chaves:** Interface de Usuário, UML, IMML e Visual IMML.

**VISUAL IMML:**

**Um Perfil UML para a Modelagem de Interfaces de Usuário**

Autora: Thais Lima Machado

Orientador: Profº Jair Cavalcanti Leite

# A b s t r a c t

The software development processes proposed by the most recent approaches in Software Engineering make use old models. UML was proposed as the standard language for modeling. The user interface is an important part of the software and has a fundamental importance to improve its usability. Unfortunately the standard UML does not offer appropriate resources to model user interfaces. Some proposals have already been proposed to solve this problem: some authors have been using models in the development of interfaces (Model Based Development) and some proposals to extend UML have been elaborated. But none of them considers the theoretical perspective presented by the semiotic engineering, that considers that, through the system, the designer should be able to communicate to the user what he can do, and how to use the system itself. This work presents Visual IMML, an UML Profile that emphasizes the aspects of the semiotic engineering. This Profile is based on IMML, that is a declarative textual language. The Visual IMML is a proposal that aims to improve the specification process by using a visual modeling (using diagrams) language. It proposes a new set of modeling elements (stereotypes) specifically designed to the specification and documentation of user interfaces, considering the aspects of communication, interaction and functionality in an integrated manner.

**Keywords:** User Interface, UML, IMML and Visual IMML.

# S U M Á R I O

# Lista de Figuras

# Lista de Tabelas

# Lista de Siglas

| | |
|---|---|
| IU | Interface de Usuário |
| IHC | Interação Humano - Computador |
| UML | Linguagem de Modelagem Unificada |
| IMML | *Interactive Message Modeling Language* |
| XML | *eXtensible Markup Language* |
| DIUBM | Desenvolvimento de Interface de Usuário Baseado em Modelos |
| OMG | *Object Management Group* |
| UMLi | UML para Aplicações Interativas |
| WAE | *Web Application Extension for UML* |
| TADEUS | *Task Analisis / Design / End User System* |
| OVID | *Object, View and Interaction Design* |

# Capítulo 1

# Introdução

O uso de modelos é fundamental para o sucesso de uma atividade de engenharia. É fácil observar que as disciplinas de engenharia usam extensivamente a modelagem para representar seus projetos. A figura do engenheiro civil, por exemplo, é a de alguém envolvido com plantas e diagramas, gerenciando uma construção. Uma planta de uma obra civil é uma representação gráfica do produto final: o edifício, uma casa, uma ponte, uma estrada, etc. Essa planta permite que o cliente avalie o produto e garanta que o resultado final é o que se deseja [Booch *et al*, 2000].

Projetar software nada mais é do que construir um modelo do software. Na engenharia de software a modelagem é fundamental, pois o software é uma entidade abstrata e invisível. Os modelos na Engenharia de Software fornecem aos desenvolvedores uma visualização do sistema a ser construído. O uso desses modelos mostra não só os requisitos do problema, mas também como eles serão solucionados. Eles permitem ainda avaliar a qualidade da solução e simular o resultado, de modo a reduzir a incerteza do software, aproximar a expectativa da realização, facilitar a padronização e a automatização dos projetos e incentivar a reutilização dos componentes de software [Booch *et al*, 2000].

A Linguagem de Modelagem Unificada (Unified Modeling Language – UML) foi lançada com o objetivo de ser uma linguagem padrão para elaboração de modelos de software [Booch *et al*, 2000]. A UML fornece recursos para a modelagem de diversos tipos de sistemas, incluindo sistemas de informação, sistemas *web* e sistemas de tempo real. Sua expressividade permite diversas visões necessárias desde o desenvolvimento até a implantação desses sistemas. Além disso, ela é fácil de compreender e utilizar.

A Interface de usuário (IU) é a parte de um sistema computacional que tem como objetivo melhorar a qualidade de uso de um sistema computacional interativo. Esta qualidade, a usabilidade, envolve facilidade de uso, o aprendizado, a memorização, entre outros. Uma boa interface torna a interação com o sistema mais fácil de aprender e usar. Com isso, em relação ao usuário, a interface pode influir no sucesso do software [De Souza *et al*, 1999].

A importância das interfaces é justificada, por vários autores, na atenção que deve ser dispensada em sua elaboração e projeto. Entretanto, existem muitas dificuldades em relação à utilização de métodos para o desenvolvimento de uma boa interface. Como pode ser observado, em uma pesquisa realizada por [Campos, 2004], é estimado que a maioria das falhas dos sistemas desenvolvidos possa ser atribuída a problemas na interação entre os usuários e o sistema, ou melhor, problemas na interface do sistema.

No desenvolvimento de interfaces de usuário, modelos são necessários para descrever as tarefas de um usuário, para descrever a estrutura de diálogo humano-computador que suportam essas atividades ou para descrever a aparência e organização dos objetos de interface utilizados. No entanto, como será visto no decorrer deste trabalho, a grande maioria das técnicas e linguagens de modelagem de software não possui uma preocupação direta com o desenvolvimento da Interface de Usuário e com a usabilidade do Sistema.

A UML tem uma grande aceitação pelos desenvolvedores de aplicações, mas não é muito utilizada por designers de interfaces de sistemas interativos, pois não oferece recursos adequados para a modelagem da interação humano-computador e da interface de usuário. Embora ela proporcione um número elevado de conceitos e notações particularmente concebidos de forma a satisfazer os requisitos típicos da modelagem de software, não é fácil identificar como os elementos das IU podem ser descritos em diagramas UML.

Diversas extensões a UML foram e estão sendo propostas. Neste trabalho, analisamos várias destas linguagens e nenhuma delas foi proposta como sendo um perfil UML. Um perfil UML é uma extensão da linguagem UML representada por um pacote que contém um conjunto de elementos de modelo da UML que foram customizados para um propósito ou domínio específico utilizando os mecanismos de extensão da UML padrão. Esse pacote está diretamente relacionado com o núcleo da linguagem UML padrão. Além disso, nenhuma delas também está fundamentada em alguma teoria específica. Assim, faz-se necessário o desenvolvimento de um perfil UML que seja baseado em teoria.

## 1.1 A IMML

A *Interactive Message Modeling Language* - IMML [Leite, 2003] é uma linguagem de especificação abstrata de interface que possibilita o desenvolvimento de IU baseado em modelos. Ela está fundamentada na teoria da Engenharia Semiótica o que a diferencia da maioria das linguagens com propósitos equivalentes.

A Engenharia Semiótica é uma teoria para IHC fundamentada em Semiótica [De Souza, 1993]. Ela considera que um sistema interativo é um artefato de meta-comunicação, que conduz uma mensagem do designer ao usuário.

Uma variante desta proposta considera que esta mensagem descreve os aspectos de funcionalidade, interatividade e comunicabilidade [Leite 2005]. A funcionalidade refere-se às tarefas que o designer considera que usuário quer realizar com a aplicação. A interatividade refere-se à forma como o designer pretende que o usuário utilize a aplicação para realizar as tarefas. A comunicabilidade refere-se à maneira como o designer comunica os aspectos de funcionalidade e interatividade.

A IMML considera estes três importantes aspectos de um sistema interativo e oferece elementos para descrevê-los em três modelos distintos: o modelo de domínio, que descreve a funcionalidade da IU; o modelo de interação, que representa o processo de interação entre usuário e interface, e por fim; o modelo de comunicação, que descreve os elementos da IU que comunicam os outros aspectos. Todos esses modelos estão intrinsecamente ligados aos aspectos teóricos da Engenharia Semiótica.

A IMML é baseada em XML (*eXtensible Markup Language*) com o propósito de ter uma linguagem estruturada através de marcações, fácil de construir e estender, e possível de ser processada por ferramentas computacionais. Desta forma, é possível especificar interfaces de usuário de forma estruturada, de acordo com o modelo que a linguagem deve representar.

No entanto, isto possui algumas desvantagens. Por ser textual e baseada em marcação, o processo de construir uma especificação é bastante trabalhoso. Essa dificuldade surge, pois a IMML é composta por vários comandos, o que torna difícil de aprender e memorizar. Existe, também a dificuldade de implementar uma interface a partir de uma especificação IMML, pois os programadores precisariam entender e compreender bem a linguagem para obter sucesso na implementação.

## 1.2 OBJETIVO DO TRABALHO

O objetivo deste trabalho é propor o desenvolvimento de uma versão visual da IMML como uma extensão da UML e que enfatiza os aspectos propostos pela Engenharia Semiótica. A Visual IMML, como é denominada, é uma proposta que vem melhorar o processo de especificação da IMML através de uma modelagem visual (diagramática). Esta modelagem é feita como uma extensão da UML, que é uma linguagem padrão da indústria de software. Isto permite mais facilidade em integrar a modelagem de interface de usuário com a modelagem de restante do software.

A Visual IMML propõe um conjunto de novos elementos de modelagem (estereótipos) elaborados para a especificação e documentação de interfaces de usuário, considerando os aspectos de comunicação, interação e funcionais de forma integrada.

## 1.3 ORGANIZAÇÃO DO TRABALHO

O trabalho está organizado em sete capítulos. A introdução é o primeiro deles e o restante está estruturado da seguinte forma:

O Capítulo 2 apresenta o Desenvolvimento Baseado em Modelos e a IMML com seus principais conceitos relacionados ao desenvolvimento de IU baseado em modelos, além de apresentar um *framework* que é composto pelos cinco principais modelos de IU e que foi escolhido como base para a análise feita em alguns trabalhos relacionados.

O capítulo 3 introduz os conceitos básicos da UML, mostrando que essa linguagem, também, foi desenvolvida de uma forma aberta, podendo ser estendida se seus elementos não são capazes de atingir o objetivo desejado. Além disso, nesse capítulo mostramos como a UML é falha na modelagem de interfaces de usuário.

O capítulo 4 apresenta uma discussão e análise sobre as principais extensões da UML para IU encontradas. Essa análise está baseada no *framework* descrito no Capítulo 2, onde se verifica o potencial e as falhas de cada extensão na especificação dos modelos do *framework* adotado.

O capítulo 5 é o capítulo principal do trabalho. Nele é apresentado a Visual IMML. Esta apresentação é feita em conjunto com a descrição detalhada de cada elemento da

IMML. Primeiramente descreve-se um elemento da IMML junto com um exemplo e, logo após, apresenta-se o elemento da Visual IMML correspondente.

O capítulo 6 apresenta uma especificação IMML que será utilizada como estudo de caso para a aplicação dos novos elementos que compõem a Visual IMML. Nesse capitulo poderão ser observados os novos diagramas que são a proposta principal deste trabalho.

O capítulo 7 apresenta considerações sobre o novo perfil da UML desenvolvido, a Visual IMML, e indica possíveis trabalhos futuros que podem dar continuidade e grande ajuda na modelagem de IU.

# Capítulo 2

# Desenvolvimento Baseado em Modelos e a IMML

Nesse capítulo, serão descritos os principais conceitos relacionados ao desenvolvimento de IU baseado em modelos, ressaltando a sua fundamentação teórica, seus objetivos e vantagens. Aqui também é apresentada a IMML, que é uma linguagem que segue a abordagem baseada em modelos. No entanto, ela foi desenvolvida para permitir a especificação e modelagem de IU, considerando os aspectos de funcionalidade, interatividade, e comunicabilidade de forma integrada. Cada um destes aspectos é descrito através de modelos específicos.

## 2.1 Desenvolvimento de Interfaces de Usuário Baseado em Modelos

As propostas de utilizar modelos para direcionar o processo de desenvolvimento de uma IU são conhecidas como Desenvolvimento de Interface de Usuário Baseado em Modelos – DIUBM. Os modelos podem ser utilizados por ferramentas especializadas para construção e refinamento dos modelos.

No DIUBM duas decisões precisam ser tomadas: o que modelar e qual linguagem de modelagem utilizar. A maioria das propostas do DIUBM utiliza linguagens baseadas em XML ou extensões da UML para a modelagem de IU.

Em relação a “o que modelar”, essas notações não permitem modelar todos os aspectos de IHC de forma unificada, especialmente a visualização da estrutura dos elementos da interface associada à interação do usuário e à funcionalidade do sistema. Assim, é necessário modelar outros aspectos da interface que não sejam apenas a interação ou diálogo.

Um importante trabalho do DIUBM é o *framework* apresentado por [Puerta & Eisentein, 1998], chamado de general *computational framework for model-based interface development systems*. Esse *framework* apresenta uma tentativa de unificar os diversos modelos que precisam ser elaborados no DIUBM. Cada modelo, segundo os autores, é considerado como um conjunto de todos os elementos que podem descrever os vários aspectos da IU.

Esse *framework* de análise será tomado como base para este trabalho e considera cinco tipos de modelos de componentes de IU. Cada um desses modelos descreve um determinado aspecto da IU em uma determinada visão. A seguir veremos cada um dos tipos [Puerta & Eisentein, 1998]:

1. Modelo de Usuário: Usado para caracterizar os diferentes tipos de usuários de uma aplicação e especificar suas preocupações acerca das tarefas e organização do trabalho, seus privilégios de acesso aos objetos do domínio e suas preferências de interação.

2. Modelo de Domínio: Define os objetos que os usuários podem ver, acessar e manipular através da interface do sistema. Esse modelo é usado para representar explicitamente os atributos dos objetos e os relacionamentos entre os vários objetos do domínio.

3. Modelo de Tarefas: Este modelo descreve as tarefas que o usuário realiza no uso da IU. O modelo de tarefas envolve os elementos como metas, ações e objetos de domínio. Uma meta descreve o que o usuário deseja realizar na interface. As seqüências de ações definem o procedimento que o usuário deve executar para atingir o objetivo. Por fim, os objetos de domínio representam os elementos usados na interface pelo usuário para realizar a tarefa. É importante no suporte ao design de IU centrado no usuário.

4. Modelo de Apresentação: Descreve como a interface de usuário está organizada, ou melhor, como a interface está sendo apresentada para o usuário. Esse modelo descreve a apresentação externa da interface de usuário, envolvendo as relações estruturais e organizacionais de seus elementos.

5. Modelo de Diálogo: Descreve como o usuário final pode interagir com a apresentação da IU. Esse modelo contém, usualmente, algumas especificações de entrada do usuário e de saída que o sistema conduz através da interface de usuário. Com isso, pode-se dizer resumidamente que esse modelo descreve as entradas (ações do usuário) e as saídas do sistema (ações do software). Em outras palavras, representa as ações que um usuário pode iniciar através dos elementos que compõem uma IU e reações do sistema. Representa a comunicação do usuário com o sistema e vice-versa.

## 2.2 A IMML NO DESENVOLVIMENTO BASEADO EM MODELOS

Muitas abordagens teóricas têm apresentado propostas para o processo de design de IU que visam aumentar a sua usabilidade. Entretanto, a maioria destas abordagens apresentam como solução a este desafio modelos e técnicas para o design pautadas em características cognitivas dos usuários. Estes modelos cognitivos são insuficientes para o desafio de usabilidade por não articularem o papel do projetista e da interface na aquisição do modelo de usabilidade pelo usuário.

A Engenharia Semiótica é uma nova abordagem para IHC fundamentada na teoria semiótica [De Souza, 1993]. Essa abordagem apresenta uma perspectiva para IHC na qual o sistema computacional é um artefato de meta-comunicação e, através dele, o designer envia uma mensagem para os usuários. Esses usuários exercem o duplo papel de interagir com o sistema e interpretar uma mensagem enviada pelo designer.

Com base na proposta original da Engenharia Semiótica, [Leite 2005] considera três aspectos da interação humano-computador que devem ser modelados:

- Funcionalidade ou Domínio: Consiste no conjunto de elementos que representam os objetos e funções do domínio com os quais o usuário deve interagir.

- Interação ou Diálogo: compreende o conjunto de comandos que o usuário utiliza para interagir com o sistema e as respostas diretas que o sistema apresenta como resultados da interação. Esse aspecto também inclui as tarefas, que consistem na descrição lógica dos passos do usuário para executar uma determinada atividade até alcançar seu objetivo, ou seja, descreve as tarefas que o usuário realiza no uso da IU.

- Comunicação: Compreende a estrutura dos objetos de interface que comunicam e organizam a funcionalidade e interação. Ela deve considerar a melhor maneira que o designer tem para realizar este processo comunicativo com o usuário .

Dentro desta perspectiva, os desenvolvedores de interfaces necessitam de um formalismo que permita especificar uma interface como uma mensagem interativa.

A *Interactive Message Modeling Language* - IMML [Leite, 2003] é uma linguagem de descrição abstrata de interface que segue a abordagem baseada em modelos. No entanto, ela foi desenvolvida fundamentada nas bases teóricas da Engenharia Semiótica [De Souza, 1993] para permitir a especificação e modelagem da interface, considerando os aspectos de funcionalidade, interatividade e comunicabilidade de forma integrada, tendo isso como o seu principal diferencial.

A IMML é uma linguagem baseada na linguagem XML com a intenção de tornar mais fácil o seu processamento por ferramentas computacionais e permitir uma descrição estruturada da especificação de Interfaces de Usuário [Leite, 2003]. Ela também foi desenvolvida para a especificação de interface de usuário com um alto nível de abstração e baseada em modelos que guiam o designer para especificar a interface de usuário de forma flexível e estruturada.

A IMML fornece um modelo semântico que guia o desenvolvedor no processo de especificação da interface, com a vantagem de integrar os aspectos funcionais, interativos e comunicativos da interface numa descrição abstrata e estruturada . Dentro da perspectiva da Engenharia Semiótica, a IMML permite a especificação abstrata de três modelos de IU, de acordo com as dimensões mostradas anteriormente, são eles [Leite, 2005]:

- Modelo de Domínio: Ele descreve a funcionalidade de uma interface. A modelagem dos aspectos funcionais tem por objetivo uma integração com a modelagem funcional da engenharia de software. Esse modelo é composto pelos conceitos de objeto de domínio e função de domínio. Representa os objetos do domínio e as funções que as manipulam, alterando o estado do sistema. O estado do sistema é definido pelas propriedades e relacionamentos entre os objetos do domínio.

- Modelo de Interação: Define as ações executadas pelo usuário para comandar uma ou mais funções do domínio, a fim de alcançar seus objetivos. Representa o processo de interação entre usuário e interface. A modelagem dos aspectos de interação tem por objetivo a descrição das tarefas e do diálogo, como proposto pelas abordagens de design

centrado-no-usuário. O modelo de interação é formado por blocos de interação, descrevendo os espaços de interação, por isso, ele é descrito através dos conceitos de comando de função, resultado de função, tarefas, interação básica e estruturas de interação.

- Modelo de Comunicação: Está diretamente ligado ao conceito de comunicabilidade, considerado como uma qualidade chave num sistema interativo. A modelagem dos aspectos de comunicação segue a teoria da Engenharia Semiótica que considera que no design existe um processo comunicativo entre o designer e o usuário. Neste processo, o designer deve ter como objetivo comunicar, da melhor forma possível, os aspectos funcionais e interativos do sistema. Assim, esse modelo é responsável por integrar e comunicar ao usuário, os modelos de domínio e interação.

No entanto, por ser altamente textual, é complicado na hora de construir uma especificação IMML, pois ela é extensa, difícil de aprender e composta por vários comandos. Existe, também uma dificuldade de implementar uma interface a partir de uma especificação IMML, pois os programadores precisariam entender e compreender bem a linguagem para obter sucesso na implementação. Essas dificuldades foram descobertas a partir de um estudo e avaliação desenvolvida em cima da linguagem IMML pelo grupo de estudo de IHC da Faculdade Federal do Rio Grande do Norte e que resultou em uma de monografia [Fonseca, 2005].

Por isso, existe a necessidade de ter uma forma de descrever estes modelos de forma visual e utilizando a UML, para que a nova linguagem desenvolvida seja mais fácil de aprender e que possa ser integrada facilmente com a modelagem de software convencional.

# Capítulo 3

# A UML e seus Mecanismos de Extensão

Como o objetivo desse trabalho é a construção de um Perfil UML para a modelagem de IU, para atingi-lo se faz necessário apresentar neste capítulo: (i) a UML, uma linguagem de modelagem de software usada como padrão dentro da comunidade de desenvolvimento de software; (ii) seus mecanismos de extensão, pois essa linguagem não é completa e permite ser estendida, e por fim; (iii) o passo a passo do desenvolvimento de um Perfil UML que será utilizado para atingir o foco do trabalho.

## 3.1 A UML

A Linguagem de Modelagem Unificada – UML é uma linguagem visual utilizada para modelar software por meio do paradigma da Orientação a Objetos. Essa linguagem tornou-se, nos últimos anos, a linguagem padrão de modelagem de software adotada internacionalmente pela indústria de produção de software [Booch *et al*, 2000].

A UML é uma linguagem para especificar, visualizar, construir e documentar os artefatos envolvidos em sistemas de software, assim como modelar negócios e outros sistemas que não sejam de software. A UML representa uma coleção de melhores práticas de engenharia que se mostraram eficientes na modelagem de sistemas grandes e complexos [Booch *et al*, 2000].

A UML surgiu da união três metodologias de modelagem. O esboço inicial do projeto da UML começou com a união do método de Booch como o método OTM de Jacobson, o que resultou no lançamento do Método Unificado no final de 1995. Logo em seguida, Rumbaugh juntou-se a Booch e Jacobson e seu método OOSE também começou a ser incorporado à nova metodologia. O trabalho dos "Três Amigos", como eram conhecidos os pesquisadores, resultou no lançamento, em 1996, da primeira versão da UML.

Depois da primeira versão lançada, as grandes empresas atuantes na área de modelagem de software passaram a contribuir com o projeto, fornecendo sugestões para melhorar a nova linguagem. Com essa contribuição, a UML foi adotada pela OMG (*Object Management Group*) em 1997, como uma linguagem padrão de modelagem [Guedes, 2004].

Até pouco tempo atrás, a UML se encontrava na versão 1.5, tendo sido muito recentemente substituída pela versão 2.0. Esta nova versão traz grandes novidades em relação à estrutura geral da linguagem, principalmente com relação à possibilidade de se desenvolver "perfis" particulares a partir da UML [Medeiros, 2004] [Booch *et al*, 2000].

Os objetivos que levaram os desenvolvedores da linguagem UML, a laçar sua versão 2.0 foi reestruturá-la (nova infra-estrutura e nova super-estrutura) de maneira a torná-la mais fácil de aplicar, implementar e adaptar, melhorando sua precisão e capacidade de re-utilização [Guedes, 2004].

O objetivo da nova infra-estrutura da UML 2.0 é definir um núcleo de metalinguagem que possa ser utilizado para definir a UML, além de definir mecanismos que permitam a customização e a adaptação da UML, de maneira a criar "dialetos" para plataformas e domínios específicos [OMG UML 2.0 – Infrastructure].

A Super-estrutura da UML 2.0 define as construções ao nível de usuário, desta forma, a superestrutura representa a linguagem UML propriamente dita, onde são definidos os diversos diagramas que compõem a linguagem [OMG UML 2.0 – Superstructure].

A Super-estrutura da UML 2.0 propõe treze novos diagramas para modelagem de softwares, a maioria dos quais já existia ou era extensão de diagramas já existentes. Estes diagramas se dividem em Diagramas Estruturais e Diagramas Comportamentais.

Os Diagramas Estruturais abrangem os Diagramas de Classes, de Objetos, de Implantação, de Pacotes, de Componentes e, o novo, de Estrutura Composta, enquanto os Diagramas Comportamentais englobam os Diagramas de Caso de Uso, de Atividades, de

Seqüência e, os novos, de Máquina de Estados, de Comunicação, de Interação Geral e de Tempo [OMG UML 2.0 – Superstructure].

## 3.2 LIMITAÇÕES DA UML PARA MODELAGEM DE IUS

Como descrito anteriormente, a UML tem uma grande aceitação pelos desenvolvedores de aplicações, mas não é muito utilizada por designers de Interfaces de Usuário, pois não propõe nenhum diagrama, originalmente, para ser utilizado para modelar interface de usuário.

Porém, alguns pesquisadores tentaram mostrar que é possível modelar IU usando a UML padrão. Segundo Silva e Paton [Silva & Paton – 1, 2000], na modelagem de IU através da UML padrão, podem-se utilizar cinco diagramas. São eles: o diagrama de casos de uso, o diagrama de classes, o diagrama de atividades, o diagrama de seqüência e, por fim, o diagrama de objetos.

Para modelar as tarefas do usuário, os autores de [Silva & Paton – 1, 2000], usam o diagrama de caso de uso e o de atividades. O primeiro diagrama descreve um conjunto de seqüências de ações, inclusive variantes, que um sistema executa para produzir um resultado de valor observável por um ator. Já o segundo diagrama, representa a modelagem do fluxo de ações de uma atividade para outra no sistema. Porém, no caso da IU, pode-se descrever passo a passo como o usuário irá realizar uma determinada tarefa do sistema. Como pode ser observado na Figura 3.1, onde se modela o fluxo de ações do usuário para estabelecer um simples *login* em um sistema.

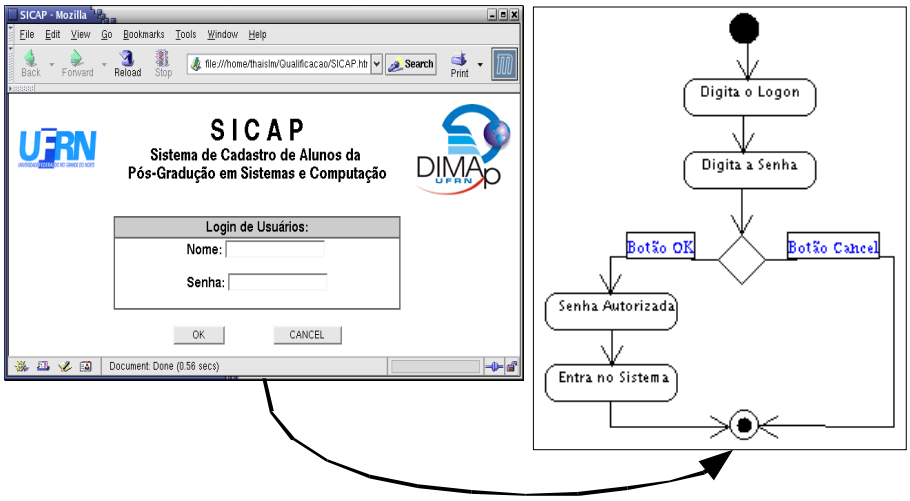



**Figura 3.1: Diagrama de Atividades de uma IU de Login.**

Para a modelagem da apresentação da interface, usa-se o diagrama de objetos que representa a modelagem de instâncias de classes de um sistema em um determinado ponto e momento de execução. Porém, em relação à modelagem de IU os objetos representaram elementos que fazem parte da interface e que serão percebidos pelo usuário. Levando em conta o exemplo passado na figura 3.1, a modelagem de sua apresentação usando o diagrama de objetos ficará como mostrada na figura 3,2.

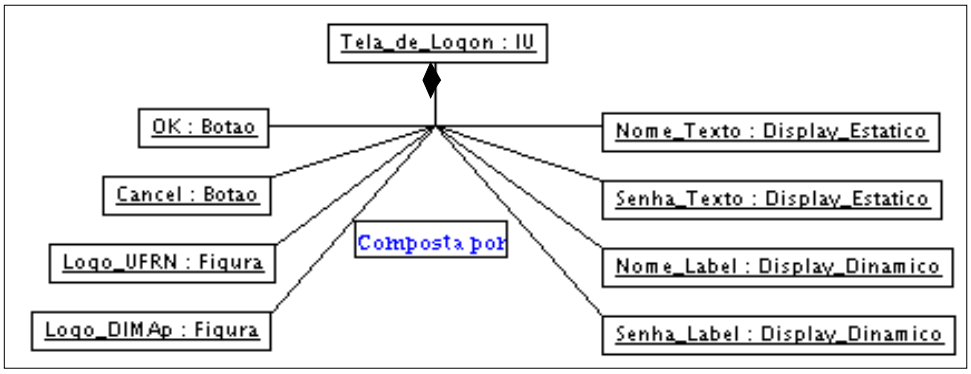



**Figura 3.2: Diagrama de Objetos da IU de *Login* do SICAP.**

Por fim, para o modelo de diálogo, de uma IU, utiliza-se o diagrama de seqüência, que dá ênfase à ordenação temporal das mensagens que são trocadas entre o usuário e o sistema. Ainda, aplicando ao exemplo da figura 3.1, a modelagem do diálogo pode ser observada na figura 3.3.

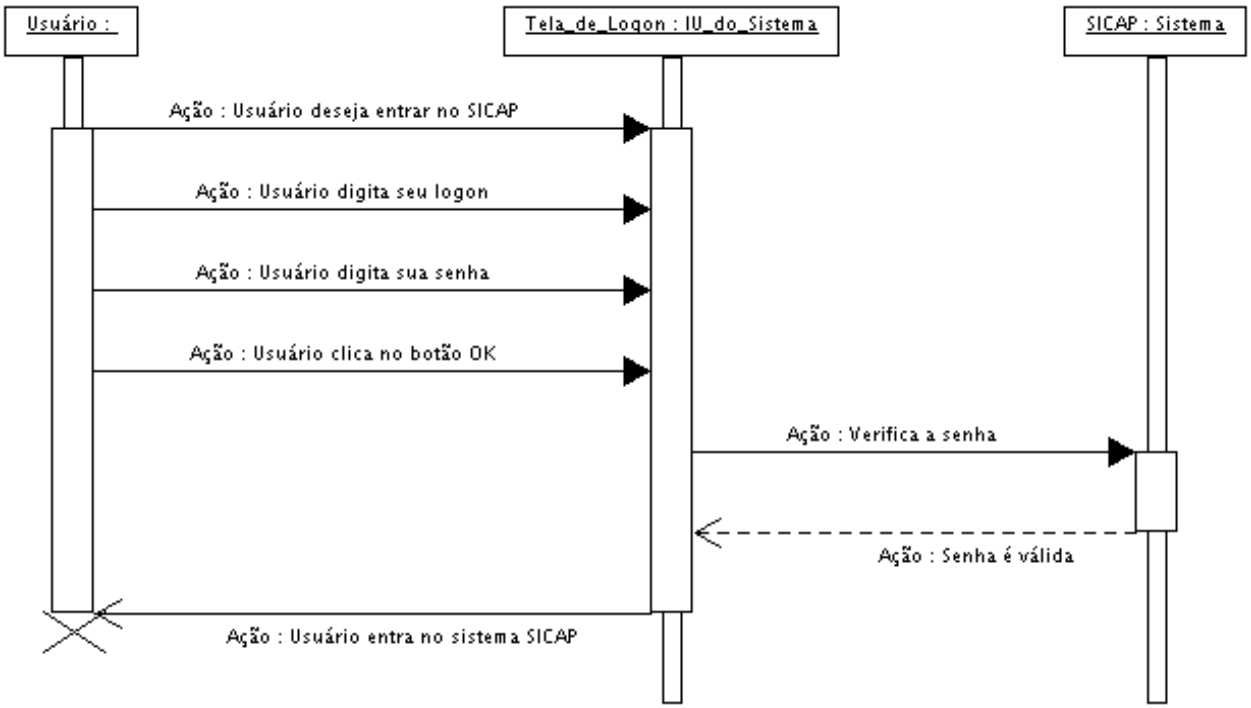



**Figura 3.3: Diagrama de Seqüência da Senha Válida**

Normalmente quando a UML padrão é usada para descrever modelos de IU, os resultados obtidos, como podem ser observados, não são claramente compreendidos pelos desenvolvedores, pois o resultado da modelagem deixa os detalhes essenciais de uma IU fracos, obscuros e sem padronização.

## 3.3 MECANISMOS DE EXTENSÃO DA UML

Muitas vezes se faz necessário a adição de novos conceitos na UML padrão. Essa adição é realizada através da utilização dos mecanismos de extensões da UML, que habilita novos elementos utilizados na modelagem de partes específicas do software que a UML padrão não abrange.

Algumas das razões para estender e adaptar a UML padrão são [OMG UML 2.0 – Superstructure]:

- O aparecimento de uma nova terminologia e vocabulário próprios de um domínio de aplicação ou de uma plataforma de implementação concreta.
- Definir uma sintaxe para construções que não tenham uma notação própria.
- Definir uma nova notação para símbolos já existentes, mas que possam ser usados de acordo com um domínio de aplicação específica.
- Adicionar certa semântica que não aparece determinada de forma precisa no metamodelo da UML.
- Adicionar certa semântica que não existe no metamodelo da UML.
- Adicionar restrições já existentes no metamodelo, restringindo sua forma de utilização.
- Adicionar informações que podem ser úteis na hora de transformar os modelos em outros modelos ou códigos.

A UML padrão é composta por três mecanismos de extensão. Esses mecanismos foram criados para permitir que sejam desenvolvidos novos conceitos que não tenham como origem um conceito já existente na UML ou que seja contraditório com algum conceito já presente [Booch *et al*, 2000]. Os mecanismos de extensão são:

- Os Estereótipos permitem que se classifique um elemento de modelo estendido de acordo com um elemento de modelo base já existente na UML (como Classe, por exemplo), definindo novas restrições e atributos, além de poder possuir uma nova representação gráfica. Todos os elementos estendidos do modelo que possuírem um ou mais estereótipos terão os valores adicionais, restrições definidos pelos estereótipos além dos atributos, associações e super-classes que o elemento já possui na UML. Os nomes dos estereótipos não podem conflitar com nomes já existentes no metamodelo UML.

- As Restrições ampliam as semânticas dos novos estereótipos, permitindo acrescentar novas regras ou modificar regras já existentes. Elas são definidas em linguagens para construção de restrições e regras, linguagem de programação, notação matemática ou linguagem natural. Uma restrição é adicionada diretamente ao estereótipo, sendo também, aplicada a todos os elementos estendidos por esse estereótipo.

- O Valor Atribuído estende as propriedades de um elemento da UML, permitindo a criação de novas informações na especificação desse elemento. Todo valor atribuído conta com um nome e um tipo e se associa ao determinado estereótipo.

Em conjunto, esses três mecanismos de extensão possibilitam a construção de um Perfil UML que permite dar forma e desenvolver a UML de acordo com as necessidades de projeto, adaptando a UML a novas tecnologias de software [Guedes, 2004] [Booch *et al*, 2000].

## 3.4 UM MÉTODO PARA ELABORAR PERFIS UML

Os Perfis UML foram definidos originalmente na versão 1.2 da UML, embora fosse difícil de saber se estavam sendo aplicados corretamente devido a sua definição ambígua. Em sua nova versão (UML 2.0), a definição de Perfil é melhorada substancialmente, assim, especificando de forma mais clara como elaborar passo a passo e estruturar um Perfil UML [Abdullah *et al*. 2004] [OMG UML 2.0 – Superstructure].

É importante deixar claro que um Perfil UML define os mecanismos usados para adaptar o metamodelo da UML a novas plataformas ou a um domínio específico. A responsabilidade do metamodelo é definir uma linguagem que será utilizada para elaborar um modelo. Assim, os estereótipos de um Perfil UML serão instanciados a partir de um

determinado elemento do metamodelo UML. O metamodelo UML é composto pelos conceitos de classes, atributos, associação e etc.

Atualmente, já estão definidos vários perfis UML, alguns dos quais já bem sucedidos e validados pela OMG: os *Profiles* UML para CORBA e para CCM (*CORBA Component Model*), o *Profile* UML para EDOC (*Enterprise Distributed Object Computing*), o *Profile* UML para EAI (*Enterprise Application Integrations*), o *Profile* UML para *Scheduling, Performance*, e *Time*, e por fim, o Perfil UML para *Web Applications Extension* – WAE. Os outros muitos perfis ainda estão sendo validados pela OMG.

A especificação da UML 2.0 [OMG UML 2.0 – Superstructure] só se limita a definir os conceitos e objetivos do perfil UML. Porém, o trabalho desenvolvido por Lídia Fuentes e Antonio Vallecillo em [Fuentes & Vallecillo, 2004] descreve um método para a construção de perfis UML 2.0. Esses autores apresentam cinco passos para a elaboração de um perfil UML. Esse trabalho foi adotado como guia na elaboração de um perfil UML para a plataforma de aplicação IMML, que chamamos de Visual IMML.

Para ilustrar esse trabalho, vamos considerar como estudo de caso um exemplo de uma interface de usuário (figura 3.4). Como é um exemplo simples, a interface de usuário será composta apenas por: janela, tabelas, botões, legendas e caixas de texto. Porém, a aplicação desses passos poderá ser acompanhada no capítulo cinco desse trabalho, na elaboração da VISUAL IMML.

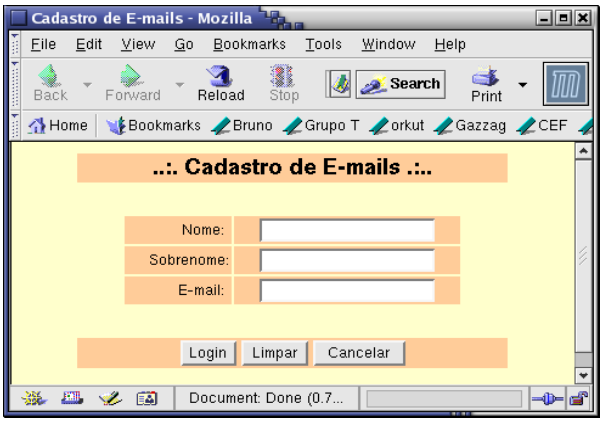



**Figura 3.4: IU simples de login**

**Passo 1 – Criação do metamodelo de domínio**

Antes de começar a elaborar um perfil, é necessário saber qual será o metamodelo de domínio utilizado. Esse metamodelo define a modelagem do domínio da aplicação que será modelada pelo Perfil UML desenvolvido. A responsabilidade primária

desse metamodelo será especificar os modelos que serão criados a partir do domínio da aplicação.

O metamodelo será definido utilizando os mecanismos da própria UML (classes, relacionamentos, associações, etc.) de forma usual como se o objetivo não fosse definir um perfil UML. Dentro do pacote <<*metamodel*>>, deve-se incluir as definições das entidades próprias do domínio, as relações entre elas, assim como suas restrições que limitam o uso dessas entidades e seus relacionamentos.

Para ilustrar esse passo, a figura 3.5 mostra um metamodelo de domínio que descreve a IU mostrada na figura 3.4, formada por uma janela e componentes básicos de uma interface. Então, o pacote <<*metamodel*>> será chamado de *My_IU* e composto por cinco classes, onde a classe Janela pode ser composta pelas classes Tabela, Botão, Legenda e Caixa de Texto, e por uma associação entre as classes Legenda e Caixa de texto.

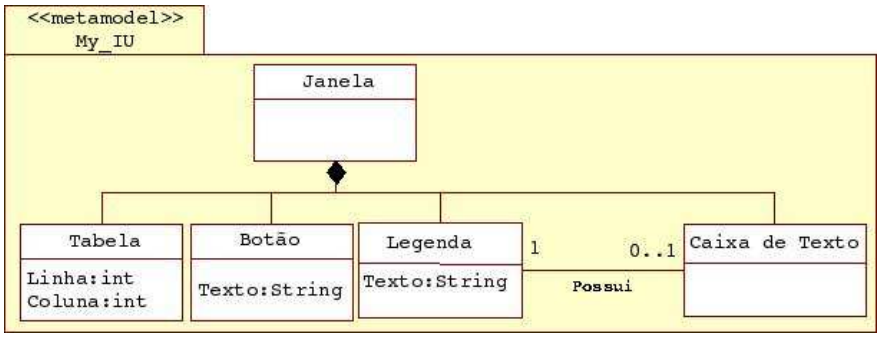



**Figura 3.5: metamodelo da IU simples de login**

Adicionalmente, como parte deste metamodelo, também é necessário expressar o conjunto de restrições definidas para ele, que são as restrições passadas pelo domínio. Em nosso estudo de caso, a restrição é:

- Restrição do metamodelo de domínio: Para cada componente caixa de texto deve haver um componente legenda explicando o que será preenchido dentro da caixa de texto.

**Passo 2 – Definição do perfil UML**

Depois de elaborar o metamodelo do domínio, o próximo passo é a definição do Perfil UML. Dentro de um pacote chamado <<*profile*>>, será incluso um estereótipo para cada um dos elementos do metamodelo que se deseja incluir no Perfil, definido no passo anterior. Estes estereótipos terão o mesmo nome que os elementos do metamodelo, estabelecendo assim uma relação entre o metamodelo e o Perfil, pois, cada um dos elementos

que foi definido no metamodelo será um estereótipo no Perfil UML instanciados por uma determinada metaclasse.

Como pode ser observado na figura 3.5, o pacote <<*metamodel My_IU*>> é composto por cinco classes e uma associação. Então, o pacote <<*profile*>> da IU será composto pelos seis estereótipos que correspondem às classes e associações definidas no metamodelo, associadas pelas metaclasses da UML que são aplicadas sobre cada estereótipo. O tipo das metaclasses corresponde ao tipo de elementos do metamodelo UML. Como pode ser observado na figura 3.6.

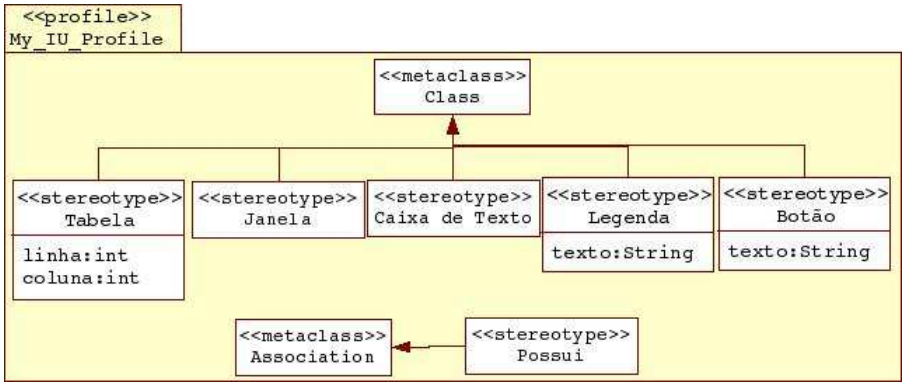



**Figura 3.6: Pacote *Profile* da IU simples de *login***

Outra opção, sem ter que criar novos estereótipos, é adaptar ou unir diagramas da UML padrão na criação e elaboração de novos diagramas para um novo Perfil UML. Tudo isso é permitido através do metamodelo de domínio da aplicação.

**Passo 3 – Descrição dos estereótipos**

É importante deixar claro quais são os elementos do metamodelo da UML que estão sendo estendidos e sobre o que é possível aplicar a um estereótipo. Exemplo de tais elementos são as classes, suas associações, seus atributos, as operações, as transações, os pacotes e etc. Desta forma, cada estereótipo se aplicará à metaclasse de UML que se utilizará no metamodelo do domínio para definir uma relação.

Como dito no passo anterior, este Perfil UML é composto por seis estereótipos. São eles: janela, tabela, caixa de texto, legenda, botão e associação possui. Vamos ilustrar apenas o estereótipo Tabela (figura 3.7):

- Estereótipo Tabela: Utilizado para definir como alguns elementos da interface podem estar organizados. Composto por um número de colunas e linhas. É instanciado da metaclasse do tipo classe. Tem como ícone a figura 3.7.

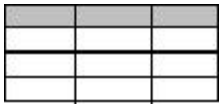


**Figura 3.7: Estereótipo Tabela**

**Passo 4 – Definição dos valores atribuídos**

Nesse passo, serão definidos os valores atribuídos dos estereótipos do perfil, que serão os atributos que aparecerão no metamodelo. Inclui, também, a definição dos seus tipos e seus possíveis valores iniciais.

Para ilustrar esse passo, em relação ao exemplo passado na figura 3.4, apenas três dos estereótipos definidos anteriormente contém valores atribuídos e um deles é o elemento Tabela.

- O estereótipo Tabela é composto por dois valores atribuídos. Um chamado de linha, onde recebe um número do tipo int que determina a quantidade de linhas que a tabela será composta e o outro é chamado de coluna que, também, recebe um número do tipo int que determina a quantidade de colunas que a tabela será composta.

**Passo 5 – Definição das Restrições**

Por fim, devem-se definir as restrições que fazem parte do Perfil, a partir das restrições do domínio. Por exemplo, as multiplicidades das associações que aparecem no metamodelo do domínio ou as próprias regras da aplicação que deve ser traduzida em restrições.

No exemplo, apenas um dos estereótipos definidos anteriormente contém restrição, que é o estereótipo Caixa de Texto, onde para cada estereótipo desse tipo deve haver, obrigatoriamente, um estereótipo legenda, que explica ao usuário o que deverá ser preenchido neste campo da IU.

É importante observar o contexto para definir as restrições, que são elementos do metamodelo da UML que está sendo adaptado para o modelo desejado. Desta forma, está sendo adicionada nova restrição semântica sobre o metamodelo da UML.

# Capítulo 4

# Propostas de Modelagem de IU Baseadas na UML

Nessa seção serão apresentadas e analisadas algumas das extensões da UML para modelagem IU. Também é analisada uma nova proposta para incluir no processo de desenvolvimento de software, a modelagem e design de IU. A análise das extensões da UML será feita verificando-se como cada uma delas permite a construção dos modelos propostos no *framework* descrito no Capítulo 2 (modelos de apresentação, diálogo e tarefas). Não serão levados em conta os modelos de usuário e domínio, pois não dizem respeito diretamente à IU, uma vez que nosso objetivo é verificar o potencial das linguagens na modelagem de IU.

## 4.1 UMLI

O Projeto UML para *Aplicações Interativas*, chamado de UMLi, aborda o problema de design e implementação de interfaces de usuário através da combinação da UML e de ambientes de desenvolvimento baseado em modelos [Silva & Paton, 2003] [Silva & Paton – 2, 2000] [Silva, 2000]. Segundo seus autores, esse projeto tem como principal propósito o de acabar com a divisão entre a interface de usuário e o núcleo principal da aplicação. Isso acontece através da integração de novas facilidades oferecidas pelos novos diagramas com as técnicas, já existentes, de modelagem da UML. Um outro objetivo da UMLi é a identificação de como o processo de design da interface e o processo de desenvolvimento da aplicação podem compartilhar modelos da interface com o modelo da aplicação subjacente.

A UMLi oferece novos diagramas que permitem a integração de toda a equipe de desenvolvimento em torno de uma notação única integrada às ferramentas de desenvolvimento. Esta extensão propõe aos designers dois novos diagramas. O primeiro é o diagrama de interface de usuário que modela a parte visual da interface do sistema. O outro é o diagrama de atividades, que toma como base o diagrama de atividades da UML padrão, porém com algumas modificações suportando assim a modelagem das atividades (seu comportamento) da IU. Para modelar uma IU usando a UMLi, seguem-se os seguintes passos (figura 4.1):

- No domínio do desenvolvedor de interface de usuário:
    1. Modelo de requisitos do usuário: criado a partir das especificações passadas pelo usuário e é usado o diagrama de caso de uso da UML padrão.
    2. Modelo abstrato de apresentação: é elaborado utilizando o diagrama de classes da UML padrão e o diagrama de interface de usuário, um dos diagramas proposto pela UMLi.
    3. Modelo concreto de apresentação: construído a partir do diagrama de interface de usuário da UMLi, que descreve a estrutura da IU.
- No domínio do desenvolvedor da aplicação:
    1. Modelo de tarefas: é elaborado com a ajuda dos cenários dos casos de uso obtidos no primeiro passo e utilizando o diagrama de atividades proposto pela UMLi.
    2. Modelo de aplicação: é desenvolvido através do uso do diagrama de classes da UML padrão, descrevendo quais objetos juntamente com seus valores serão utilizados através das classes da UML .

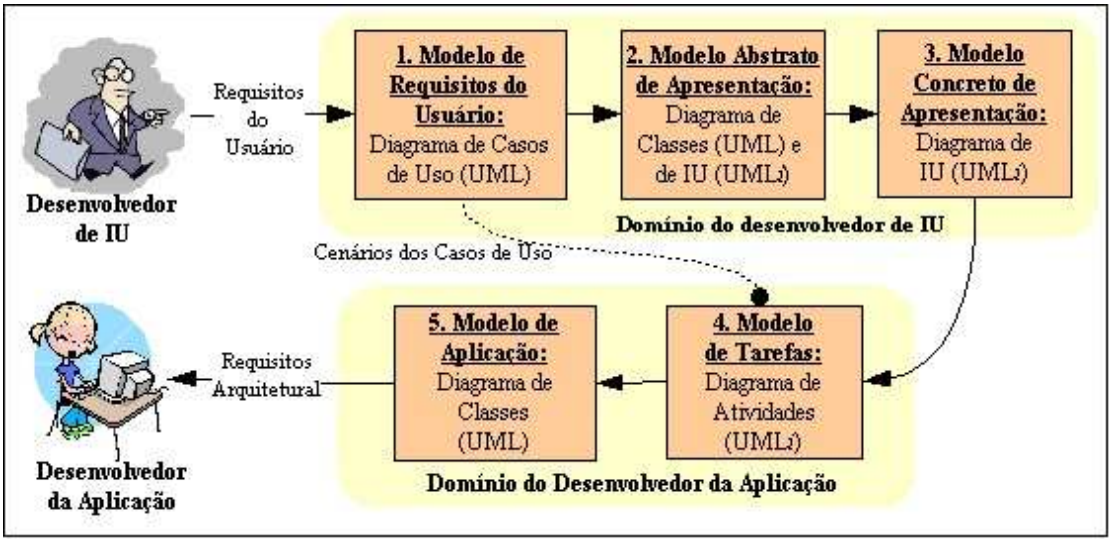



**Figura 4.1: Passos da Modelagem usando a UMLi.**

### 4.1.1 Análise

Considerando o *framework* que foi tomando como base para essa analise e proposto pelos autores de [Puerta & Eisentein, 1998], veremos agora como a UMLi permite construir os modelos de tarefas, apresentação e dialogo, onde cada um desses modelos descreve um determinado aspecto da IU em uma determinada visão.

#### *4.1.1.1 Modelo de tarefas na UMLi*

O modelo de tarefas pode ser construído em UMLi utilizando uma extensão do diagrama de atividades [Silva & Paton, 2003] [Silva & Paton – 2, 2000]. Essa extensão adiciona ao diagrama de atividades da UML um conjunto de estereótipos, como *<<presents>>*, *<<cancels>>*, *<<interacts>>* e *<<confirms>>*, que são baseados no uso dos objetos da IU que interagem com o usuário e reduzem a necessidade de modelar estado das ações.

A extensão adiciona ainda um outro conjunto de três símbolos (Selection State) que simplificam a modelagem do fluxo de controle de uma atividade para uma outra. O primeiro é representado por um círculo com um sinal de positivo dentro, chamado de order *independent*, indicando que duas atividades são executadas obrigatoriamente sem uma ordem pré-estabelecida. O segundo é chamado de *optional*, que é representado por um círculo com o sinal negativo. Ele indica que a escolha das atividades ligadas a esse círculo não importa. Por fim, o terceiro é o *repeatable*, que é um círculo com um x dentro, onde sua função é executar a tarefa indicada várias vezes. Como pode ser observado na figura 4.2.

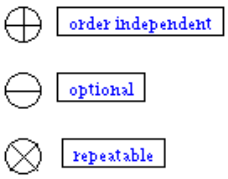



**Figura 4.2: Os três símbolos Selection State.**

### *4.1.1.2 Modelo de apresentação na UMLi*

Para a modelagem da apresentação, a UMLi descreve um novo diagrama, o diagrama de interface de usuário [Silva & Paton, 2003] [Silva & Paton – 2, 2000]. Este diagrama é uma especialização do diagrama de classes, usado para dar suporte ao modelo da apresentação.

O diagrama de interface de usuário é composto por um desenho do cubo tracejado, chamado de *FreeContainer*, que representa um alto nível dos objetos de interação. O cilindro tracejado, chamado de *Containers*, agrupa os objetos de interação que não pertencem ao *FreeContainer*. Os dispositivos que recebem informações do usuário são representados por um triângulo com a ponta para baixo, chamado de *Inputters*. Os dispositivos responsáveis por enviar informações visuais para o usuário são representados pelo triângulo de ponta para cima, chamado de *Displayers*. As setas, chamadas de *ActionsInvokers*, são responsáveis por receber informações dos usuários em forma de eventos. Como pode ser visto na figura 4.3.

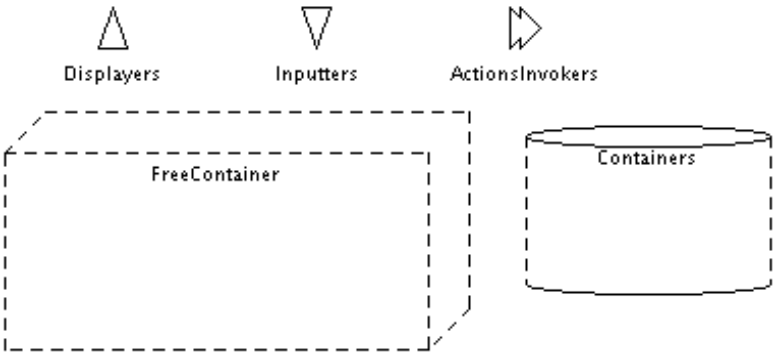



**Figura 4.3: Componentes do Diagrama de IU da UMLi**

### *4.1.1.3 Modelo de diálogo na UMLi*

Em relação à modelagem do diálogo, a UMLi utiliza o diagrama de seqüência da UML padrão, dando ênfase à ordenação temporal nas quais as mensagens são trocadas entre o usuário e o sistema.

## 4.2 WAE

A UML em seu estado original, não possui estereótipos suficientes para modelar aspectos específicos de aplicações *Web*. A WAE (*Web Application Extension for UML*) é uma extensão para a UML criada por Jim Conallen [Conallen, 2000], para auxiliar a modelagem de aplicações *Web*.

A WAE estende a notação UML trazendo novos estereótipos com semântica e restrições adicionais que permitem a modelagem de elementos específicos de uma aplicação *Web*. Os principais estereótipos são: página cliente, página servidor, *link*, *frame*, formulário, entre outros.

### 4.2.1 Análise

Considerando o *framework* de análise passado por [Puerta & Eisentein, 1998], veremos como a WAE se comporta na construção dos modelos de tarefas, apresentação e diálogo. Porém, no decorrer desta análise, foi concluído que a extensão WAE não oferece recursos para modelar as tarefas e os diálogos de uma IU, por ser uma extensão específica para o desenvolvimento de ambientes *Web*.

#### *4.2.1.1 Modelo de apresentação na WAE*

A modelagem da apresentação usando a WAE só obtém sucesso quando a interface é composta apenas por *links* para outras páginas, *frame* ou formulários [Conallen, 2000]. Quando a página *web* é construída a partir de um script lado-servidor, a WAE fornece duas classes distintas estereotipadas por <<*server page*>> e <<*cliente page*>>.

Para o uso de formulários HTML são definidos novos estereótipos. Deste modo, os formulários são representados no modelo através de uma classe estereotipada por <<*form*>> que tem como atributos os campos do formulário. Por fim, o frame que permite a representação de múltiplas páginas *Web* ao mesmo tempo e são implementados em HTML pela definição de um *frameset*, sendo representado no modelo através de uma classe estereotipada por <<*frameset*>>.

## 4.3 WISDOM

O Wisdom é um método para desenvolvimento de sistemas interativos para pequenos grupos de desenvolvedores [Jardim, 2002] [Jardim & Cunha, 2000]. Esse método é caracterizado por usar a prototipação evolutiva com a definição clara dos objetivos a serem atingidos em cada etapa de implementação.

O método introduz um processo no qual o desenvolvimento não é guiado pela funcionalidade interna do sistema, mas sim pelos Casos de Usos UML. Para isso, o Wisdom apresenta uma nova arquitetura (figura 4.4), na qual a organização dos modelos favorece a separação entre os conceitos de núcleos funcionais e os componentes utilizados por um sistema interativo [Jardim, 2002].

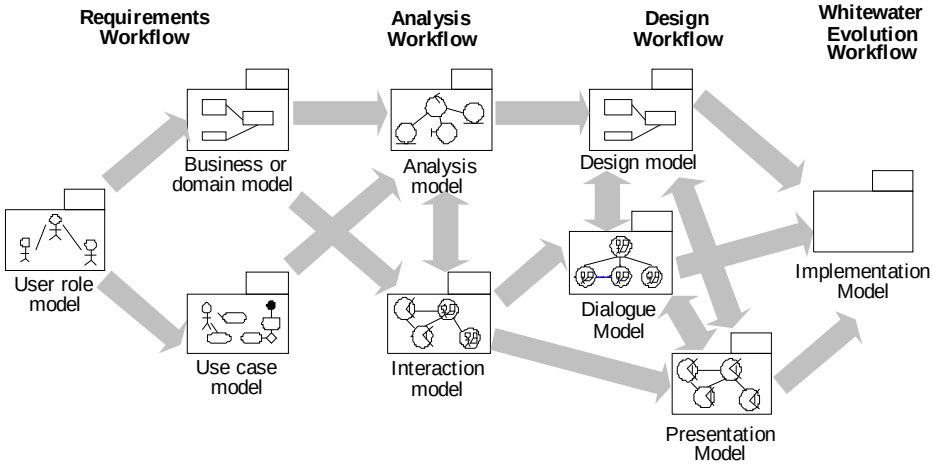



**Figura 4.4: Arquitetura do modelo Wisdom [Jardim, 2002]**

Esse método estende a UML padrão através de novos diagramas e estereótipos. Os novos diagramas permitem construir os modelos de perfil do usuário, de interação, de diálogo e de apresentação. Eles podem ser visto na arquitetura do Wisdom, mostrada na figura 4.4.

### 4.3.1 Análise

Nesta seção será visto como o método Wisdom se comporta considerando o *framework* que foi tomando como base para essa análise e proposto por [Puerta & Eisentein, 1998]. Será analisado como esse método permite construir os modelos de tarefas, apresentação e diálogo.

No Wisdom existe o modelo de interação que permite separar e identificar os elementos de diálogo e espaço que compõem uma interface. Esse modelo introduz dois novos estereótipos. O estereótipo <<*task*>> é representado por uma esfera com um desenho de um usuário e de um computador. O estereótipo <<*Interaction Space*>> pode ser representado por uma esfera com algumas linhas. O primeiro permite modelar o diálogo entre os atores e o sistema, já o segundo representa o ambiente que o usuário interage com o sistema em todas as suas funções, conteúdos e informações.

#### *4.3.1.1 Modelo de tarefas e diálogo no Wisdom*

O método de diálogo do Wisdom unifica os modelos de tarefas e de diálogo do *framework*. Este modelo representa as tarefas realizadas pelo usuário do sistema e suas relações, ou melhor interações com o sistema, que são completamente independentes dos detalhes de implementação, deixando esse detalhes para o programador decidir a melhor forma de implementar o que foi modelado pelo *designer* Ele introduz quatro novos estereótipos de dependência, necessários para as relações, são eles:

1. <<*infopass*>> - que denota que uma tarefa envia informação para outra tarefa;
2. <<*seq*>> - denota que a tarefa dependente é ativada quando a tarefa independente termina;
3. <<*seqi*>> - denota que a tarefa independente ativa a tarefa dependente com trocas de informações;
4. <<*deact*>> - denota que a tarefa dependente é desativada definitivamente quando a tarefa independente termina.

#### *4.3.1.2 Modelo de apresentação no Wisdom*

O modelo de apresentação do Wisdom é a representação física dos elementos que compõem o estereótipo <<*Interaction Space*>> apresentado no modelo de interação e introduz cinco novos estereótipos:

1. <<*navigate*>> - que identifica um espaço por onde o usuário move a sua atenção;
2. <<*contains*>> - uma classe que contém outra;

3. <<*input element*>> - informação mostrada ao usuário, que pode ser manipulada;

4. <<*output element*>> - informação mostrada ao usuário, que não pode ser manipulada;

5. <<*action*>> - ação do usuário que pode provocar alteração no sistema de forma significante.

Por isso, com base nos modelos descritos acima e através do desenvolvimento de um protótipo é possível chegar a apenas um esboço de uma boa interface.

## 4.4 RUPI

O RUPi é uma extensão do *Rational Unified Process* (RUP) proposta em [Souza & Furtado, 2003], que considera os fatores e conceitos relativos a IHC durante o ciclo de desenvolvimento. O RUP é um processo de desenvolvimento de *software* proposto para ser um modelo de referência e um padrão de mercado [Souza & Furtado, 2003].

Essa extensão toma como ponto de partida a falta de preocupação dos processos de desenvolvimento de software com a usabilidade. Visando solucionar essa falha, o RUPi propõe a inserção de novos detalhes aos *workflows* já existentes do RUP, definindo um conjunto de novas atividades.

O RUPi é composto por quatro *workflows*, que são baseados nos *workflows* do RUP e adaptados aos conceitos de IHC. Os quatros *workflows* do RUPi são: requisitos, análise e design, implementação e testes. Para cada *workflow*, existem novos artefatos que são baseados no paradigma do DIUBM.

Os requisitos são descritos através dos cenários, modelo de usuário e de tarefas e a construção de um protótipo como artefatos para a elaboração de uma IU. No *workflow* de análise e design usa-se o modelo de tarefa e domínio. O *workflow* de implementação tem como artefato apenas os protótipos da IU. Por fim, o *workflow* de teste usa o modelo de tarefa. Utilizando a UML padrão como linguagem para a modelagem desses artefatos.

Com isso, pode ser observado que a RUPi tem os mesmos problemas, apontados anteriormente, na utilização da UML padrão para a modelagem de IU através dos modelos de passados no *framework* adotado, assim não se faz necessário a elaboração dos modelos baseados no RUPi.

## 4.5 TADEUS

A abordagem TADEUS (Task Analis / Design / End User System) consiste de um framework baseado em modelos para representação de IU, associado a uma metodologia e um ambiente para o desenvolvimento das interfaces de usuário [Stiber & Stary, 2000] [Stary, 2000], como mostra a figura 4.5.

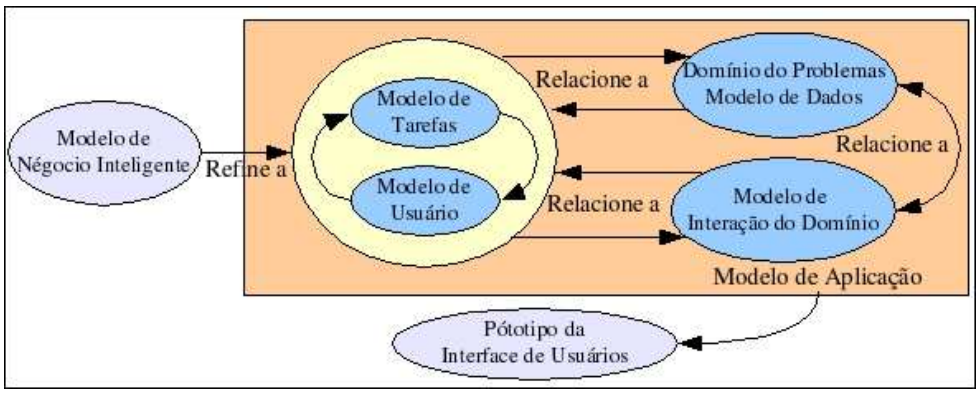



**Figura 4.5: TADEUS Framework**

TADEUS permite a especificação e modelagem unificada de vários componentes associados por meio de modelos. Cada um desses modelos permite uma especificação estruturada e dinâmica e captura a descrição de tarefas, processo de negócios, papéis de usuário, dados do domínio do problema e modalidades de interação.

Segundo Chris Stary em [Stiber & Stary, 2000] [Stary, 2000], a abordagem TADEUS vem propondo aos designers três novos diagramas. Esses diagramas são resultados da união das linguagens de modelagem UML e OSA (Object-Oriented Systems Analysis) e são denominados de:

- Object Relation Diagram (ORD) – Descreve as relações estruturais entre as classes e os objetos, usado na apresentação das IU. É derivado do diagrama de classes da UML padrão.

- Object Behavior Diagram (OBD) – Descreve o comportamento dos objetos que compõem a IU e utiliza o diagrama de estados da UML padrão.

- Object Interaction Diagram (OID) – Descreve a interação entre o ciclo de vida dos objetos. Esse diagrama foi derivado dos diagramas de atividades e de colaboração da UML original.

### 4.5.1 Análise

Nas subseções a seguir será verificado qual o procedimento adotado pela abordagem TADEUS em relação ao *framework* que foi adotado como base para essa análise [Puerta & Eisentein, 1998]. Será analisado como essa abordagem permite construir os modelos de tarefas, apresentação e diálogo.

#### *4.5.1.1 Modelo de tarefas em TADEUS*

Comparando com o nosso *framework* de análise, as tarefas no TADEUS são derivadas do modelo de negócio (figura 4.5), onde tais tarefas que são relevantes para suportar os negócios interativos e são modeladas usando a notação ORD. As tarefas podem ser divididas em sub-tarefas (ações, operações) ou herdadas de 'super'-tarefas. Com a especificação dinâmica das tarefas, elas podem ser modeladas usando o diagrama OBD.

#### *4.5.1.2 Modelo de apresentação em TADEUS*

Quando se completa toda a especificação da IU, antes mesmo da prototipagem, é criado o modelo de aplicação (figura 4.5), que equivale ao modelo de apresentação do nosso *framework* de análise. Na modelagem da aplicação é usado o diagrama OID onde será exposto sincronicamente o comportamento de todos os objetos da IU.

#### *4.5.1.3 Modelo de diálogo em TADEUS*

O modelo de diálogo do *framework*, no TADEUS é denominado modelo de interação. Na maioria das vezes, ele é baseado em um projeto da interface de usuário e pode ser modelado através do diagrama OBD, que permite modelar a comunicação entre os objetos que fazem parte do projeto da IU.

## 4.6 OVID

A metodologia OVID (*Object, View and Interaction Design*) é um conjunto de técnicas para o design de interface orientado a objetos [Azevedo *et al*, 2000]. Essa metodologia tem como principal objetivo permitir o mapeamento entre os requisitos, identificados por uma análise prévia de tarefas, e a especificação do artefato a ser desenvolvido.

De acordo com [Azevedo *et al*, 2000], a OVID é uma metodologia formal para o design da experiência do usuário baseada na análise das metas e tarefas dos usuários. O

resultado da aplicação da metodologia é um diagrama abstrato que descreve a arquitetura do software do ponto de vista do usuário.

A OVID não propõe nenhum novo diagrama. Essa metodologia tira proveito dos diagramas já existentes da UML padrão para modelar os três elementos do design de interfaces de usuário, propostos pelos autores em [Azevedo *et al*, 2000], que são:

- Os objetos: são os objetos que o usuário percebe como as ações ou os papéis por elas desempenhados, documentos, acontecimentos, entre outros.
- As visões: são as interfaces que representam os objetos, pois cada classe necessitará de pelo menos uma forma para ser visualizada e/ou de editar os seus atributos.
- As interações: são as atividades que os usuários realizam com os objetos. As interações com o sistema podem, ainda, ser detalhadas especificando os eventos possíveis de gerar através das diversas visões e o seu tratamento pelo sistema.

Esses três elementos entram num ciclo para o desenvolvimento das interfaces (figura 4.6) que segue os seguintes passos:

1. Um conjunto inicial de objetos é formado depois da análise das tarefas;
2. As visões são definidas através da visão apropriada do usuário em relação aos aspectos de todos os objetos;
3. As novas tarefas são descritas em termos de novos objetos e novas visões;
4. As interações entre os usuários e os objetos são descritas detalhadamente.

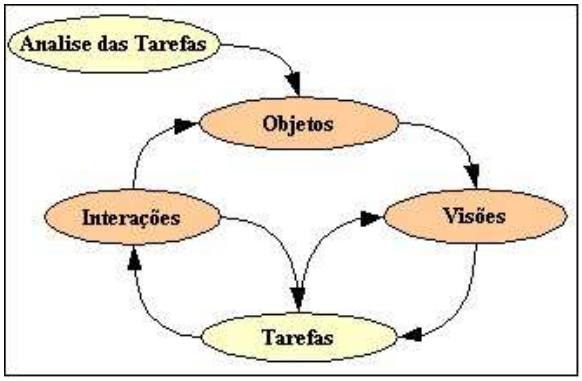



**Figura 4.6: Ciclo do desenvolvimento de interface do OVID.**

### 4.6.1 Analise

Considerando o *framework* que foi tomando como base para análise em [Puerta & Eisentein, 1998], veremos como a metodologia OVID se comporta na construção dos modelos de tarefas, apresentação e diálogo, onde cada um deles descreve um determinado aspecto da IU.

#### *4.6.1.1 Modelo de tarefas em OVID*

Na metodologia OVID para modelar as tarefas usa-se o diagrama de seqüências da UML padrão, onde é representada a disposição das tarefas através das visões dos objetos.

#### *4.6.1.2 Modelo de Apresentação em OVID*

Em relação ao modelo de apresentação do *framework* de análise adotado, as visões como são denominadas no OVID, são representadas em um diagrama de objetos e seus relacionamentos, onde cada visão pode ser considerada uma representação de um objeto em uma IU.

#### *4.6.1.3 Modelo de Diálogo em OVID*

Por fim, o Modelo de Diálogo é representado pelas interações entre os objetos em cada visão e pode ser modelado usando o diagrama de estados da UML padrão.

## 4.7 CONSIDERAÇÕES SOBRE A UML E SUAS EXTENSÕES

A Linguagem UML, apresentada no capítulo anterior, quando é usada para modelar IU, foge um pouco de sua proposta original, pois como ela não tem nenhum diagrama específico para a modelagem de interfaces de usuário deixa os designers utilizá-la sem nenhuma padronização. Além disso, os diagramas da UML padrão não demonstra para os designers os detalhes necessários para a construção de uma boa IU. Assim, podemos dizer que essa linguagem não é adequada para modelar as tarefas, apresentação e diálogo de uma IU.

A UMLi só soluciona parte do problema da modelagem de Interfaces, pois a apresentação e as tarefas estão bem modeladas através do uso dos novos diagramas propostos por essa extensão, ficando de fora o modelo de diálogo, que utiliza o diagrama de seqüência

da UML padrão, trazendo a tona todos os problemas existentes nessa linguagem em relação às IU.

A WAE é uma extensão UML ideal para modelar as características das aplicações *Web*, trazendo novos estereótipos com semântica e restrições adicionais. Porém, a WAE não oferece nenhum recurso para modelar as tarefas e o diálogo. Além disso, a modelagem da apresentação só é bem sucedida quando a IU é composta por alguns estereótipos particulares, pois se uma IU de um sistema Web for composta por outros tipos de componentes, como caixas de diálogo, barras de ferramentas, etc., a sua modelagem é bastante deficiente, não atingindo completamente seu objetivo.

Em sua essência, o WISDOM é uma abordagem baseada em tarefas de modo a permitir a especificação apenas do aspecto dinâmico da interação com o sistema, esquecendo o aspecto estático das IU.

A abordagem TADEUS permite uma modelagem abrangente de vários aspectos da comunicação entre os objetos da IU (as relações, os comportamentos e as interações). As tarefas e o diálogo são bem modelados por essa extensão. Porém, ela não mostra como estão estruturados os componentes que fazem parte da interface, por isso a modelagem da apresentação é muito deficiente neste aspecto.

A OVID utiliza os diagramas da UML padrão para modelar os três elementos do design de IU propostos por seus autores em [Azevedo *et al*. 2000]. Usando os diagramas de interação e de objetos, resultando em uma IU abstrata que descreve a arquitetura do software e um protótipo de sua apresentação.

Após a análise de quais os modelos do *framework* podem ser elaborados em cada extensão da UML, chega-se à conclusão que para modelar uma IU completamente é necessário a combinação de várias dessas propostas.

A Tabela 4.1 apresenta uma síntese dos modelos atendidos pelas extensões UML estudadas.

Usando a UML Padrão e a OVID (que utiliza os diagramas da UML Padrão) para a modelagem de IU será obtido um resultado inadequado, pois não teria a padronização defendida por sua política de modelagem além, de não demonstrar detalhes suficientes da IU modelada para o designer.

A WAE não modela completamente a apresentação da IU, pois só obtém sucesso quando modela uma IU composta por alguns elementos. O WISDOM não atende completamente os modelos de apresentação e de diálogo por ter seu foco no aspecto dinâmico da interação humano-sistema, esquecendo o aspecto estático das IU. O TADEUS não mostra como estão estruturados os componentes da IU modelada, por isso a modelagem da apresentação é deficiente neste aspecto.

A última coluna da Tabela 4.1 apresenta como nossa proposta se comportará em relação aos modelos atendidos no *framework* de análise.

**Tabela 4.1: Comportamento das Extensões UML para IU em relação aos Modelos**

| **Modelos**<br>**Linguagem** | **Modelo de Tarefas** | **Modelo de Apresentação** | **Modelo de Diálogo** |
|---|---|---|---|
| **UML Padrão** | Atende<br>(Não adequadamente) | Atende<br>(Não adequadamente) | Atende<br>(Não adequadamente) |
| **UML*i*** | Atende | Atende | Não Atende |
| **WAE** | Não Atende | Atende<br>(Não completamente) | Não Atende |
| **WISDOM** | Atende | Atende<br>(Não completamente) | Atende<br>(Não completamente) |
| **TADEUS** | Atende | Atende<br>(Não completamente) | Atende |
| **OVID** | Atende<br>(Não adequadamente, usa a UML Padrão) | Atende<br>(Não adequadamente, usa a UML Padrão) | Atende<br>(Não adequadamente, usa a UML Padrão) |
| **IMML** | Atende (Detalhado no próximo capítulo) | Atende (Detalhado no próximo capítulo) | Atende (Detalhado no próximo capítulo) |

# Capítulo 5

# A IMML e a VISUAL IMML

A Visual IMML foi desenvolvida com a proposta de construir uma versão diagramática da IMML, já que essa linguagem é textual, baseada na XML, considerada de difícil aprendizado e compreensão. Ela tem por objetivo melhorar o processo de especificação com a IMML através de uma modelagem visual. Além disso, a Visual IMML foi desenvolvida como uma extensão da UML, já que essa linguagem não oferece recursos adequados para a modelagem da interação humano-computador e da interface de usuário. Isto permite mais facilidade em integrar a modelagem de interface de usuário com a modelagem de restante do software.

A Visual IMML, como poderá ser observado nesse capítulo, vem propondo um conjunto de novos diagramas e estereótipos adaptados para a especificação, visualização, modelagem e documentação de IU, permitindo descrever aspectos de comunicação, interação e funcionais de forma integrada, sua principal vantagem e diferencial em relação às outras extensões da UML para IU.

Neste capítulo, serão descritas a IMML e, paralelamente, a proposta desse trabalho, a Visual IMML. Primeiramente, de acordo com os passos a serem seguidos para o desenvolvimento de um Perfil UML, será mostrado o metamodelo de domínio da Visual IMML, logo após, pode-se observar os novos diagramas e estereótipos propostos para os três modelos que a Visual IMML é formada.

Para um melhor entendimento dos novos elementos propostos pela Visual IMML, utilizaremos um exemplo simples que será modelado pela IMML e pela Visual

IMML. Esse exemplo será descrito durante sua modelagem, dividido em três parte: Modelo de Domínio, Modelo de Interação e Modelo de Comunicação.

## 5.1 O METAMODELO DE DOMÍNIO DO PERFIL VISUAL IMML

Antes de começar a elaborar o perfil, é necessário saber qual será o metamodelo de domínio utilizado. Como foi especificado anteriormente, esse metamodelo vai definir o domínio da aplicação que será considerado pelo Perfil UML desenvolvido. Sua responsabilidade será especificar os modelos que serão criados a partir do domínio da aplicação.

O metamodelo de domínio da Visual IMML será o modelo conceitual da linguagem IMML. A IMML será estabelecida no seu nível conceitual de modelagem utilizando o diagrama de classes da UML. Ele tem por objetivo documentar claramente o significado e o uso de seus elementos de vocabulário, relações e restrições da IMML.

O Modelo Conceitual da IMML e o metamodelo da Visual IMML serão definidos dentro de um pacote UML <<*metamodel*>> que será composto pelos modelos de Domínio, de Interação e de Comunicação. Nele devem-se incluir as definições das entidades próprias da IMML, as relações entre suas entidades, assim como suas restrições que limitam o uso dessas entidades e seus relacionamentos. O metamodelo da Visual IMML será elaborado a partir de como a linguagem IMML está definida e de como os elementos desse modelo estão estruturados.

Como já se sabe a IMML é composta por três modelos (domínio, interação e comunicação), então para uma melhor visualização o metamodelo da Visual IMML foi dividido em três pacotes, como pode ser observado na figura 5.1, que mostra o modelo de domínio, na figura 5.2, onde pode ser observado o modelo de interação e, por fim, a figura 5.3, o modelo de comunicação.

No diagrama conceitual do modelo de domínio da figura 5.1 pode-se observar que ele é formado por dois conceitos principais, o de objeto de domínio e o de função de domínio. Onde a função de domínio é composta pelos controles, estados, pré e pós condições e os operandos de entrada e saída. Esse operandos, como pode ser observado, são os objetos de domínio usados na função de domínio relacionada.

No modelo conceitual do modelo de interação, mostrado na figura 5.2, observa-se que esse modelo é composto por três conceitos importantes, as tarefas, os comandos de função e os resultados de função. Eles podem ser estruturados internamente pelas estruturas de interação (*join*, *combine*, *sequence*, *select* e *repeat*) e/ou utilizando as interações básicas (*perceive-information*, *enter-informatio*, *select-information*, *active*, *go* e *do*).

Por fim, a figura 5.3 apresenta o metamodelo do modelo de comunicação, que é composta por seis signos de composição (*task-environment*, frame, *command-pannel*, *display-area*, *group*, *visualization elements* e *interactive sign*), esses signos podem ser compostos por outros signos de acordo com o que pode ser observado na figura.

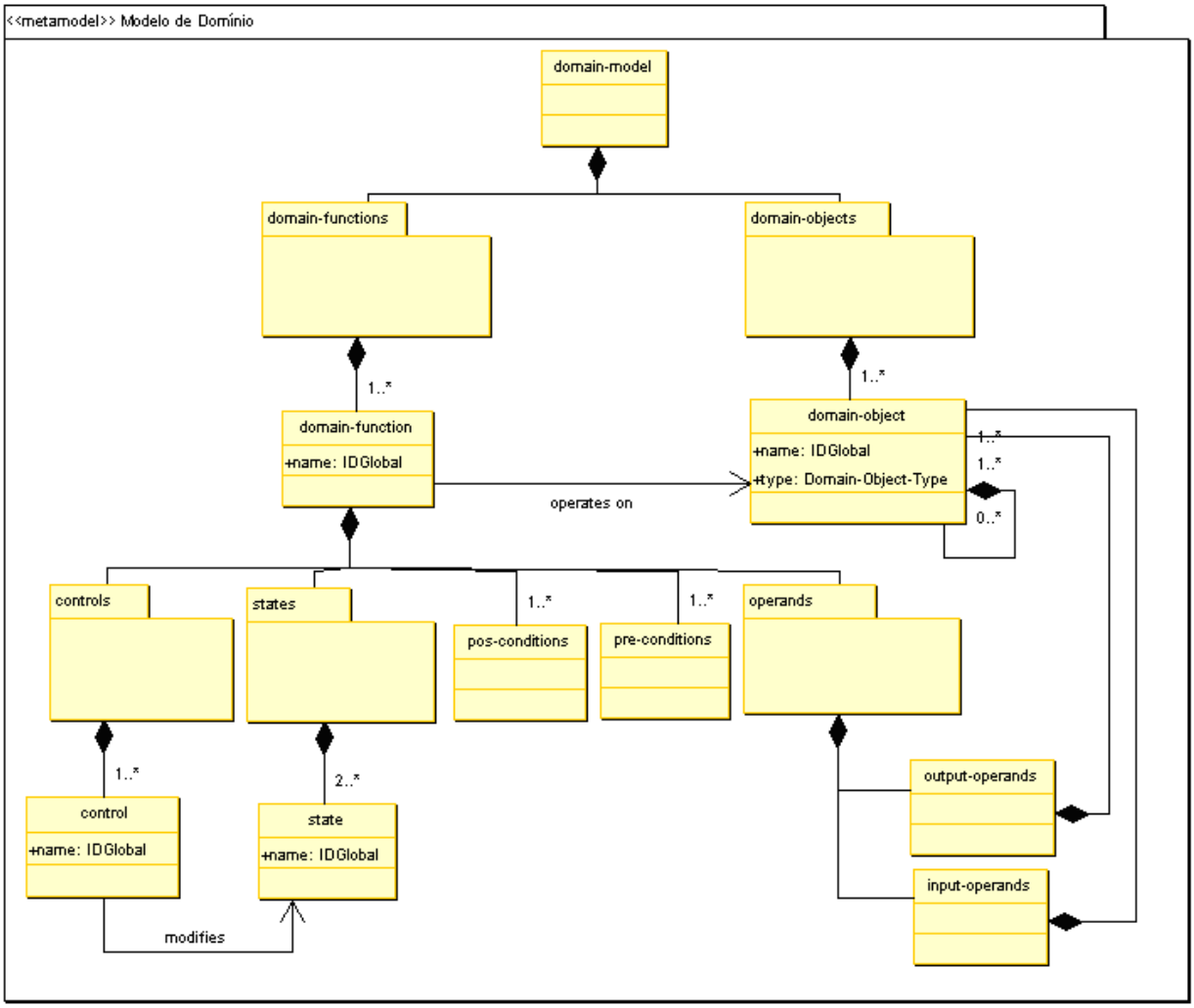



**Figura 5.1: Pacote do metamodelo do modelo de domínio da Visual IMML**

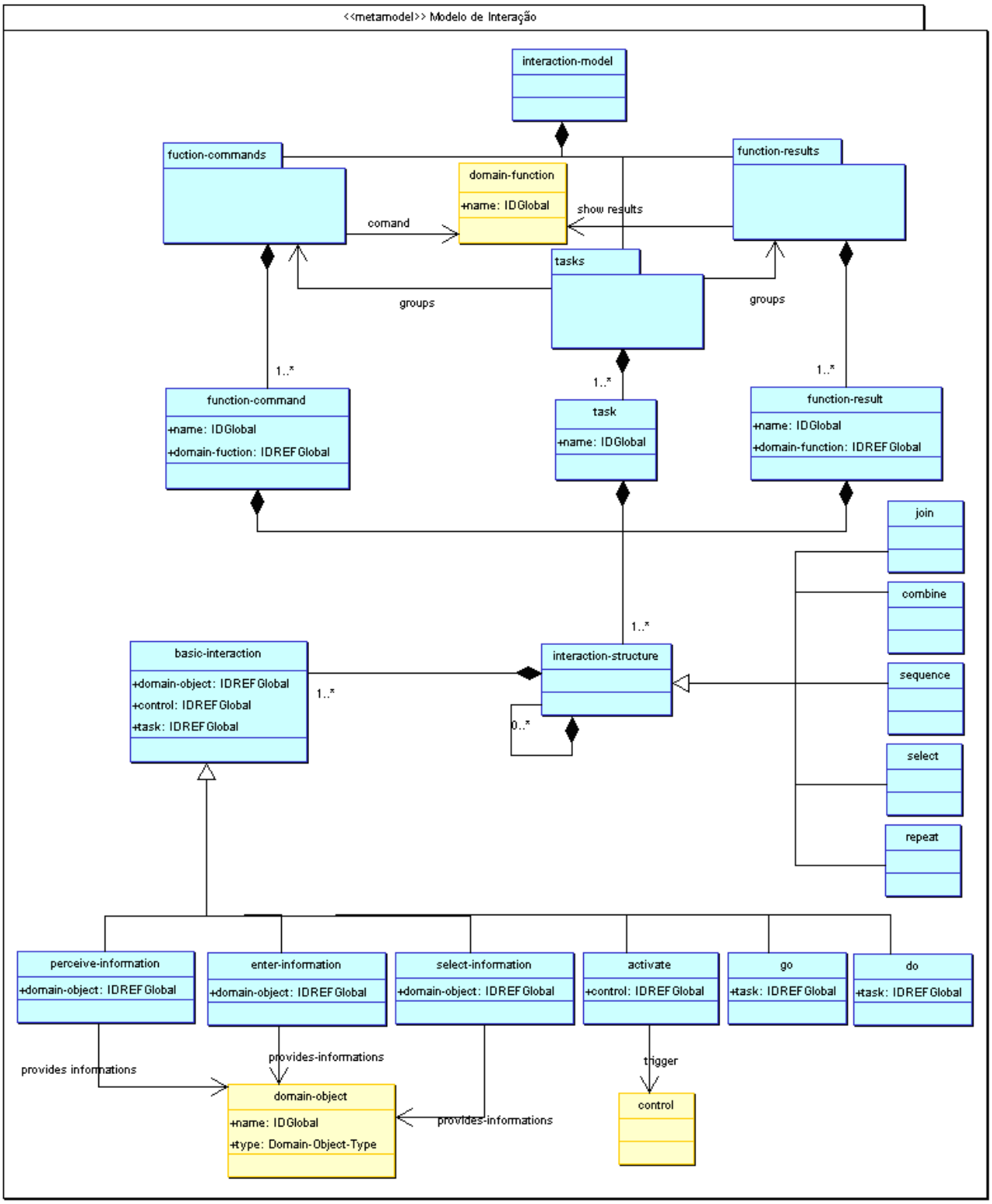



**Figura 5.2: Pacote do metamodelo do modelo de interação da Visual IMML**

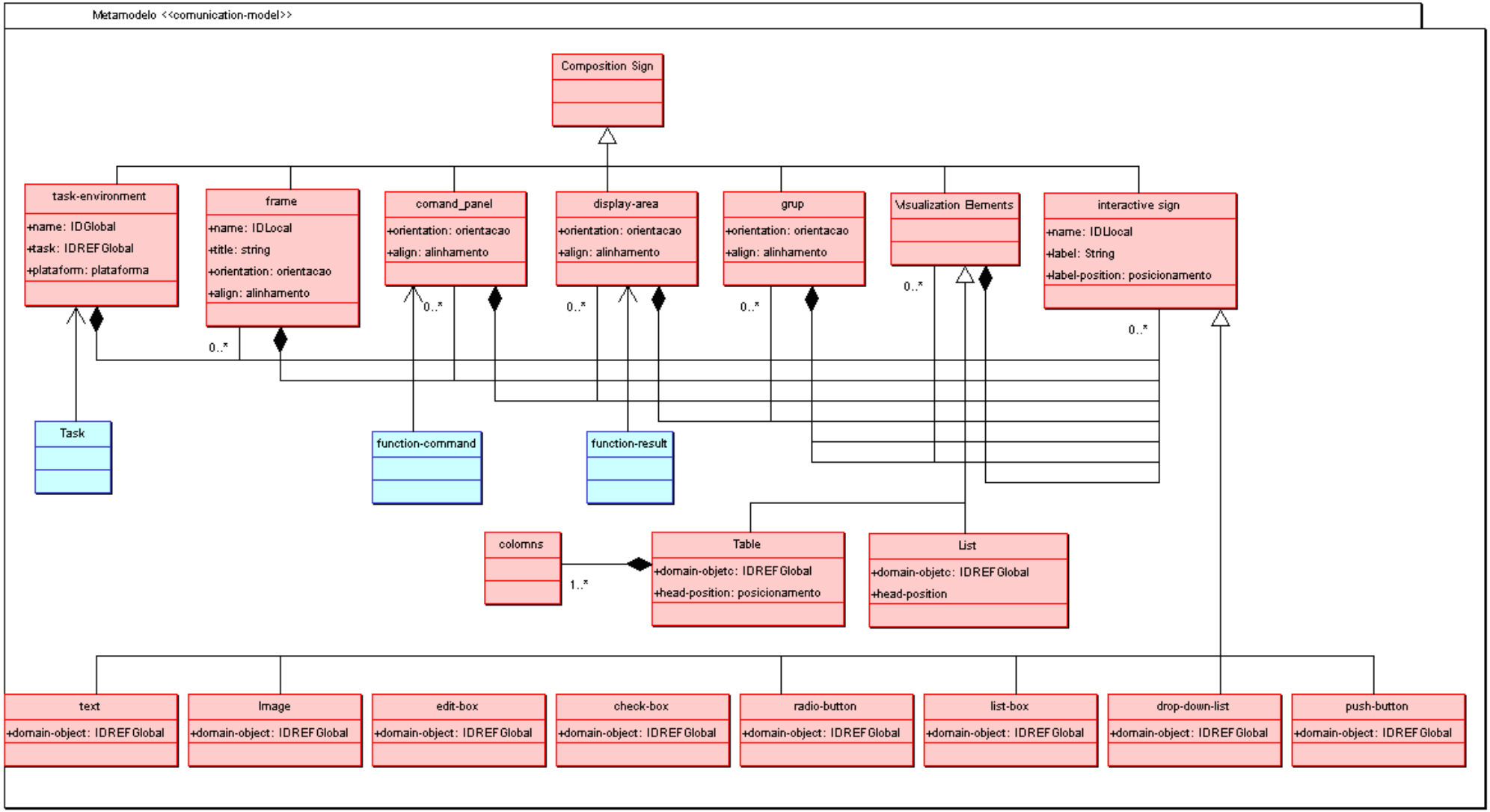



**Figura 5.3: Pacote do metamodelo do modelo de Comunicação da Visual IMML**

## 5.2 O PERFIL (UML) VISUAL IMML

Após especificar o modelo conceitual da linguagem IMML e utilizando-os como o metamodelo do domínio da VISUAL IMML, o próximo passo é a definição do Perfil da Visual IMML, como foi especificado no capítulo 3.

Nas próximas subseções, será descrito juntamente com os novos diagramas e estereótipos da Visual IMML, os detalhes da linguagem IMML. No apêndice, segue uma descrição geral de todos os novos elementos da Visual IMML.

### 5.2.1 Sintaxe inicial da IMML

A IMML é dividida em três modelos e poderá ser observada utilizando-se do diagrama de pacotes da UML Padrão para documentar claramente o significado e o uso de seus modelos (figura 5.4). Esse diagrama pode ser representado de duas maneiras, a primeira da forma de composição (figura 5.4 (a)) e a segunda é utilizando o relacionamento de composição da UML padrão (figura 5.4(b)). Será optado pela primeira forma, neste trabalho, mas é possível utilizar as duas formas.

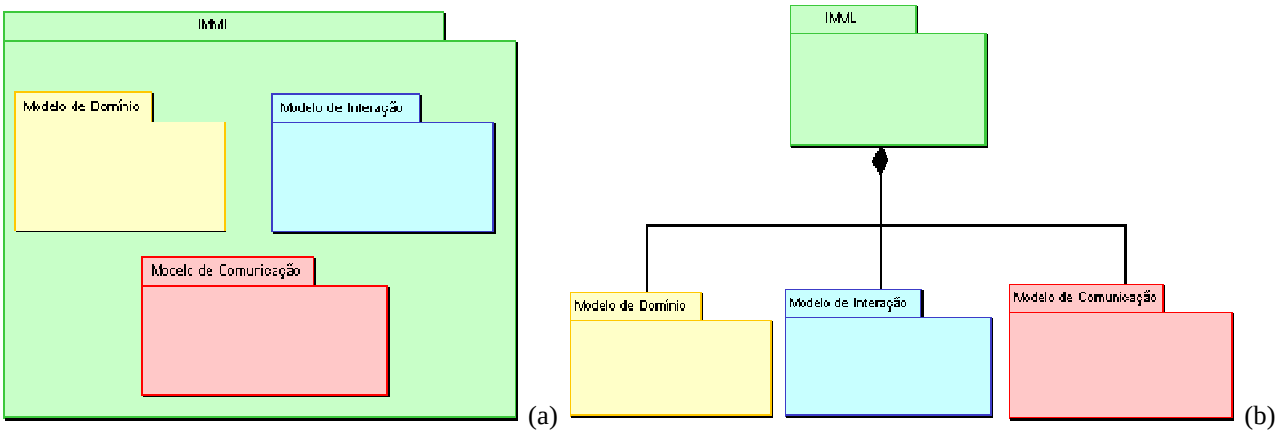



**Figura 5.4: Pacote IMML: (a) Forma de Composição; (b) Uso do relacionamento de composição.**

Todo arquivo de especificação IMML deverá ser um arquivo XML "bem-formado" e deverá conter como elemento principal o elemento <*imml*>. Este elemento é o nó raiz numa especificação IMML. Ele deve possuir os elementos <*domain-model*>, <*interaction-model*> e <*communication-model*>, como pode ser visto na figura 5.5, que mostra como está estruturado esse elemento no código da IMML.

```
<imml>
     <domain-model> ... </domain-model>
     <interaction-model> ... </interaction-model>
     <communication-model> ... </communication-model>
</imml>
```

**Figura 5.5: Especificação IMML do elemento <*imml*>**

### 5.2.2 Modelo de Domínio

O Modelo de Domínio está baseado no conceito da funcionalidade de uma interface. Na IMML, este modelo é composto pelos conceitos de objeto de domínio e função de domínio [Leite, 2003]. O primeiro conceito faz uma referência aos elementos do domínio de aplicação e o segundo conceito faz referência a um processo executado pelo computador que opera sobre estes objetos.

Na Visual IMML, os detalhes dos objetos e funções de domínio serão especificados com um novo diagrama. Esse novo diagrama será o resultado de uma mistura entre os diagramas de objetos, estados e casos de uso da UML padrão. Nos subitens abaixo serão descritos detalhadamente os conceitos que fazem parte desse modelo, são eles:

#### *5.2.2.1 Objeto de Domínio*

Um objeto de domínio é uma referência abstrata feita a elementos concretos da aplicação, como base de dados, registros, arquivos e mensagens, ou seja, toda e qualquer informação que o usuário poderá manipular por meio da interface de uma aplicação para a realização das ações necessárias para a conclusão de tarefas.

Existem vários tipos do objeto de domínio, são eles:

- Objetos de Domínio Simples: fazem referência a um único elemento do domínio que pode ser do tipo: texto, número, data, imagem e etc.
- Objetos de Domínio Composto: faz referencia a um objeto que é composto por outros objetos.de diferentes tipos
- Objetos de Domínio Conjunto-finito (*Finite-set*): faz referência a um grupo limitado e pré-defindo de objetos do domínio de mesmo tipo.
- Objeto de Domínio Lista (List): faz referência a um conjunto não definido de elemento do domínio de um mesmo tipo.

- Objetos de Domínio Tabela (*Table*): faz referencia a um grupo de objetos de domínio organizado de uma maneira tabular que o usuário pode manipular na IU.

Na IMML, os objetos de domínio estão agrupados dentro de um elemento chamado domain-objects e possuem a seguinte sintaxe (figura 5.6):

```
<domain-objects>
  <domain-object name="” type="" />
     ...
  </domain-object>
</domain-objects>
```

**Figura 5.6: Sintaxe IMML do objeto de domínio da IMML**

Na Visual IMML, os objetos de domínio são representados pelo diagrama de objetos da UML padrão, com as instâncias e os relacionamentos entre eles. Para ficar mais organizado, todos os objetos de domínio da IU modelada pela Visual IMML são observados dentro de um elemento estereótipo do tipo pacote da UML padrão, representado pelo elemento <<*domain-objects*>> que está representado no modelo conceitual do modelo de domínio (figura 5.1).

Para um melhor entendimento segue um exemplo da modelagem de uma biblioteca digital básica, que tem uma função simples de busca. Para isso, essa especificação deve constar com alguns objetos de domínio, são eles: número do ISBN, nome do título, nome do autor, ano de publicação, entre outros como pode ser observado na figura 5.7. Já a figura 5.8, usando Visual IMML, mostra como os objetos de domínio do exemplo passado na figura anterior serão especificados. Eles serão representados pelo diagrama de objetos e organizados em um estereótipo <<*domain-objects*>> que é do tipo pacote da UML Padrão. Como pode ser observado existem vários objetos de domínio simples, um do tipo *finit-set* e outro do tipo *compose-set* que se faz uso do relacionamento de composição da UML.

```
<domain-model>
  <domain-objects>
    <domain-object name="ISBN" type="text" />
    <domain-object name="Title" type="text" />
    <domain-object name="Author" type="text" />
    <domain-object name="Publisher" type="text" />
    <domain-object name="Year" type="number" />
    <domain-object name="Edition" type="number" />
    <domain-object name="KeyWords" type="text" />
    <domain-object name="SearchKeys" type="text" />
    <domain-object name="SearchFields" type="finite-set">
      <item>ISBN</item>
      <item>Title</item>
      <item>Author</item>
      <item>Publisher</item>
      <item>KeyWords</item>
    </domain-object>
    <domain-object name="BooksList" type="composed-object">
```

```
        <item domain-object="Title" />
        <item domain-object="Author" />
        <item domain-object="Publisher" />
        <item domain-object="KeyWords" />
      </domain-object>
   </domain-objects>
<domain-model>
```

**Figura 5.7: Exemplo dos objetos de domínio de uma biblioteca digital, especificado em IMML**

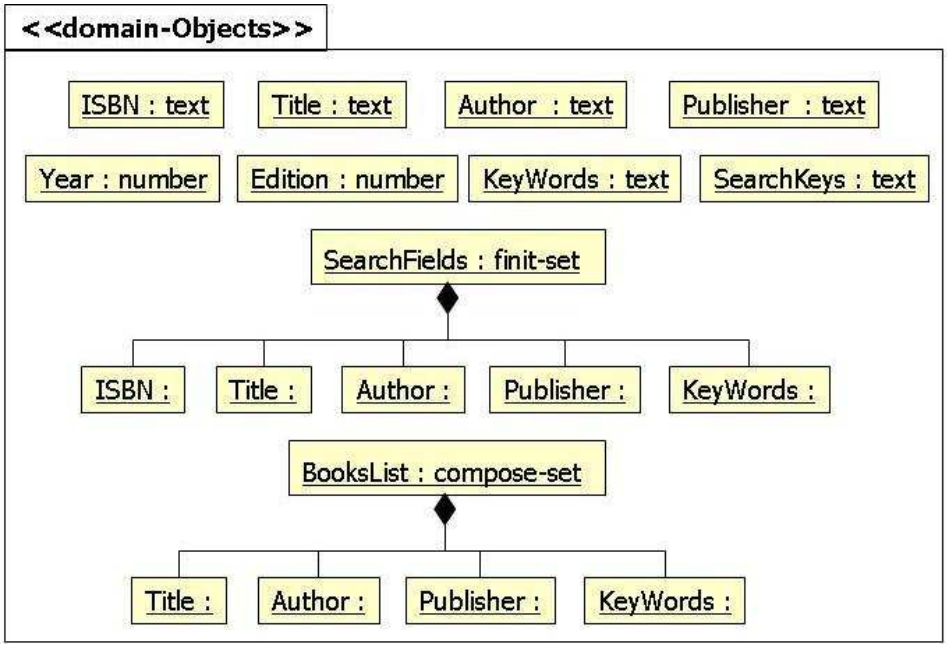



**Figura 5.8: Exemplo dos objetos de domínio de uma biblioteca digital, especificado em Visual IMML**

### *5.2.2.2 Função de Domínio*

Uma função de domínio é uma referência abstrata feita a um processo executado pelo computador, capaz de mudar o estado da aplicação, através de mudanças nos objetos de domínio. A função de domínio descreve as ações que serão realizadas pelo usuário para alcançar seu objetivo na IU. Os elementos das funções de domínio são:

- Operandos de Entrada (objetos de domínio): usados para especificar os operandos de entrada de uma função de domínio.
- Operandos de Saída (objetos de domínio): usados para especificar os operandos de saída de uma função de domínio.
- Pré-condições: usados para especificar informalmente as pré-condições de uma função de domínio. Descrevem quais são as pré-condições que devem ser atendidas para realização da função.

- Pós-condições: usados para especificar informalmente as pós-condições de uma função de domínio. Descrevem as pós-condições que devem ser alcançadas após a realização da função.

- Controles: usado para especificar um controle de uma função de domínio. Ele é responsável por levar a função de domínio de um estado ao outro.

- Estados: usado para especificar um estado de uma função de domínio.

Na IMML, o elemento *<domain-function>* deverá possuir elementos *<controls>*, *<states>*, *<input-operands>*, *<output-operands>*, *<pré-conditions>* e *<post-conditions>*. Os elementos devem obedecer à seqüência sintática, como pode ser observado na figura 5.9.

```
<domain-function name="">
     <input-operands> ... </input-operands>
     <output-operands> ... </output-operands>
     <pre-conditions> ... </pre-conditions>
     <post-conditions> ... </post-conditions>
     <controls> ... </controls>
     <states> ... </states>
</domain-function>
```

**Figura 5.9: Sintaxe IMML da função de domínio da IMML**

Na Visual IMML, as funções do domínio serão representadas por um novo estereótipo derivado do diagrama de caso de uso da UML padrão, como pode ser observado na figura 5.10.

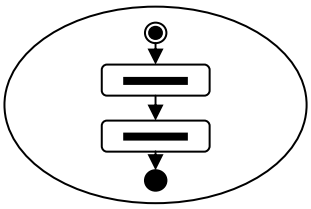


**Figura 5.10: Estereótipo da Visual IMML do elemento *<domain-function>***

Para cada caso de uso (função do domínio) existirá um relacionamento de realização junto ao elemento *<<domain_function_models>>* que descreve os detalhes da função de domínio. Este relacionamento tem como função indicar que a função de domínio realizará o caso de uso. O elemento *<<domain_function_models>>* é um estereótipo derivado de um elemento pacote. Ele é composto por dois outros elementos: *<<operands>>* e *<<state-model>>*.

O elemento <<*operands*>> contém os elementos <<*input_operands*>> e <<output_operands>>. Ambos são uma extensão do estereótipo <<*domain_object*>>. Eles descrevem os operandos de entrada e saída de cada função do domínio. Eles estarão organizados na Visual IMML dentro de um estereótipo derivado do pacote UML chamado <<*operands*>>.

O elemento <<*state-model*>> é um estereótipo derivado de um elemento pacote e contém o diagrama de estados através do qual serão descritos os estados do sistema. O estado inicial terá uma nota chamada pré-condição e o estado final, que terá uma nota chamada pós-condição. A transição entre os estados será ativada por controles.

Para ilustrar as funções de domínio, será dada continuidade ao exemplo passado anteriormente da biblioteca digital. Essa biblioteca tem uma única função que é a de busca (*Search*). A figura 5.11 mostra como a função de domínio desta interface está especificada utilizando a IMML. Já as figuras 5.12 e 5.13 mostram como a função de domínio será especificada usando a Visual IMML. A primeira será representada pelo diagrama de casos de uso. A segunda usará os diagramas de objetos e estados, organizados em pacotes da UML Padrão.

```
<domain-model>
  <domain-functions>
    <domain-function name="Search">
      <input-operands>
        <item domain-object="SearchKeys" />
        <item domain-object="SearchFields" />
      </input-operands>
      <output-operands>
        <item domain-object="BooksList" />
      </output-operands>
      <pre-conditions>Term NOT empty</pre-conditions>
      <pos-conditions>Show the result list OR error message</pos-
conditions>
      <controls>
        <control name="Search" from-state="Idle" to-state="Running" />
        <control name="Cancel" from-state="Running" to-state="Idle" />
      </controls>
      <states>
        <state name="Idle" />
        <state name="Running" />
      </states>
    </domain-function>
  <domain-functions>
<domain-model>
```

**Figura 5.11: Exemplo da função de domínio da biblioteca digital, especificado em IMML**

A figura 5.12 apresenta a primeira parte da modelagem das funções de domínio de uma IU. Nesta parte, poderá ser observada toda a função de domínio da IU especificada. Ela é representada pelo novo estereótipo derivado dos casos de uso da UML padrão. A

realização desta função de domínio é descrita por um conjunto de modelos que são agrupados em um pacote. Este pacote é também um estereótipo que faz parte da Visual IMML.

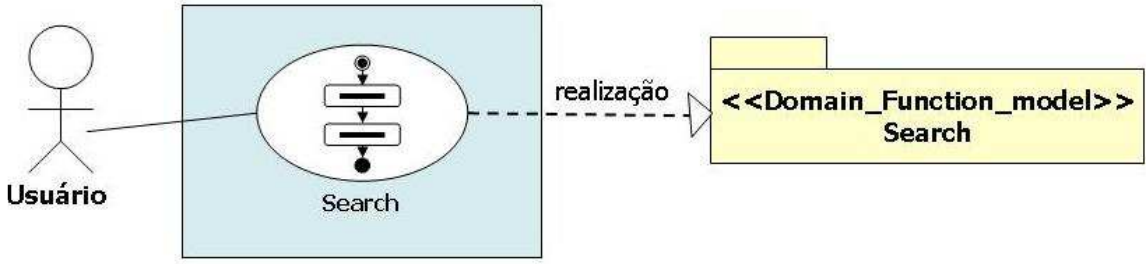



**Figura 5.12: Exemplo da parte I da função de domínio da biblioteca digital, especificado em Visual IMML**

Por fim, a segunda parte deste novo diagrama define o que contém o pacote que realiza os casos de uso. Esse pacote é dividido em duas partes, ou melhor, é composto por outros dois pacotes. Um é o pacote de operandos mostra quais objetos de domínio serão utilizados como operandos de entrada e de saída, esse pacote é uma extensão do pacote que contém os objetos de domínio, e o outro, é o pacote de estados que mostra através do diagrama de estados da UML padrão, quais são os estados que a IU vai assumir, quais os controles que vão modificar esses estado e quais as pré e pós-condições que devem existir para a modificações dos estados. Tudo isso pode ser observado na figura 5.13 que ilustra esse diagrama usando o exemplo passado anteriormente.

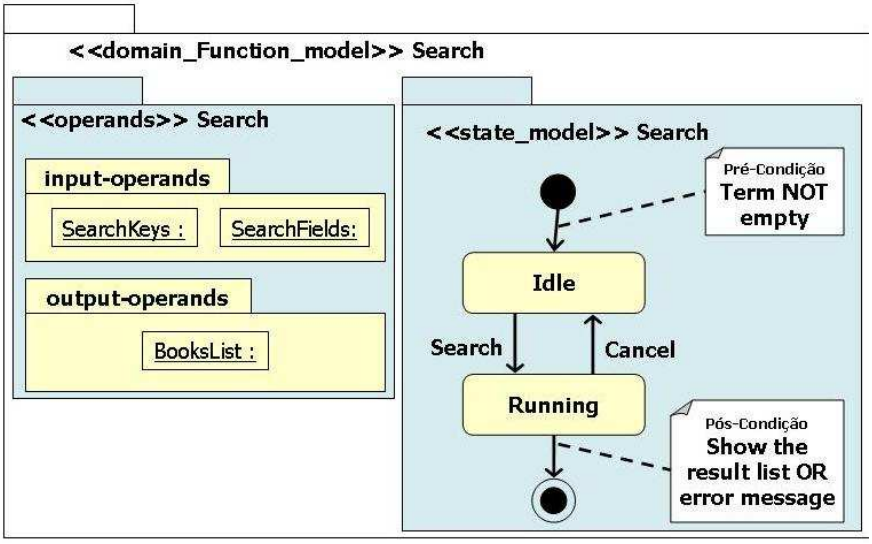



**Figura 5.13: Exemplo da parte II da função de domínio da biblioteca digital, especificado em Visual IMML**

### 5.2.3 Modelo de Interação

O modelo de interação define as ações executadas pelo usuário para comandar uma ou mais funções de domínio, a fim de alcançar seus objetivos. Ele representa o processo de interação entre usuário e interface [Leite, 2003].

Este modelo é formado por cinco elementos. São eles: tarefas, comando de função, resultado de função, interação básica e estruturas de interação. Isto pode ser observado no modelo conceitual de interação passado na figura 5.2. As estruturas de interação são compostas por elementos responsáveis por organizar os comandos e resultados de função dentro de uma tarefa, e as interações básicas dentro dos comandos e resultados de função. Nos subitens abaixo serão descritos os conceitos que fazem parte do modelo de interação detalhadamente.

#### *5.2.3.1 Tarefas*

Uma tarefa é uma composição estruturada de comandos de função, resultados de função e/ou outras tarefas. Esta composição é organizada através das estruturas de interação.

Na especificação da IMML o conjunto de tarefas é descrito de forma agrupada pelo elemento <*tasks*>, que é um repositório para organizar as tarefas de uma IU. Na Visual IMML, o elemento <*tasks*> é representado pelo estereótipo <<*tasks*>> que é representado por um pacote que é uma derivação do diagrama de pacotes da UML padrão. O elemento <*task*> possui a seguinte sintaxe na IMML (figura 5.14):

```
<tasks>
  <task name="">
       estrutura de interação
            ou
       interação básica
  </task>
</tasks>
```

**Figura 5.14: Sintaxe IMML do elemento <*task*>**

Na VISUAL IMML, o elemento tarefa será representada pelo estereótipo <*task*> que é apresentado no ícone mostrado na figura 5.15. Esse elemento será utilizado juntamente com o relacionamento de composição da UML, para descreve os elementos que compõem a tarefa. No metamodelo de domínio a tarefa é representada pelo elemento classe, como pode ser visto na figura 5.2 e tem como restrição poder ser composta por comandos e

resultados de função, e por estruturas de interação ou interações básicas, que também serão usadas para organizar as interações.

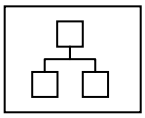


**Figura 5.15: Estereótipo da Visual IMML do elemento <*task*>**

### *5.2.3.2 Comando de Função*

Um comando de função é formado por um conjunto de interações básicas. Ele é usado na entrada de informações e no controle da execução de uma função de domínio. Cada comando de função deve ser associado a uma função de domínio. Já uma função de domínio poderá ter vários comandos de função associados a ela.

As interações básicas que compõem um comando de função devem ser organizadas através das estruturas de interação.

Na especificação da IMML o conceito de Comando de Função é representado pelo elemento <*function-command*>. A sua sintaxe está descrita na figura 5.16.

```
<function-commands>
  <function-command name="" domain-function="">
       estrutura de interação
              ou
       interação básica
  </function-command>
</function-commands>
```

**Figura 5.16: Sintaxe IMML do elemento <*function-command*>**

O elemento <*function-commands*> é um repositório utilizado para organizar os comandos de função de uma IU, que na Visual IMML será representado por um estereótipo <<*function_commands*>> que é derivado do elemento pacote da UML padrão.

Na Visual IMML, um comando de função (<*function-command*>) será representado por um novo estereótipo (figura 5.17), que deverá organizar as interações do usuário usando as estruturas de interações e/ou interações básicas. Esse estereótipo é representado no metamodelo de domínio como uma classe e tem como restrição ter pelo menos uma estrutura de interação e/ou interação básica.

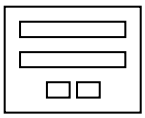


**Figura 5.17: Estereótipo da Visual do elemento <*function-command*>**

#### *5.2.3.3 Resultado de Função*

Um resultado de função é usado para descrever como o usuário irá interagir com as saídas das funções de domínios, que podem ser objetos do domínio, mensagens de erros ou alertas. Ou seja, ele é responsável pelo *feedback* do sistema ao usuário a respeito do processo de interação.

Na especificação da IMML esse conceito é descrito pelo elemento <*function-result*> e possuir a seguinte sintaxe (figura 5.18). O elemento <*function-results*> é um repositório para organizar os resultados de função de uma IU numa especificação IMML.

```
<function-results>
  <function-result name="" domain-function="">
       estrutura de interação
            ou
       interação básica
  </function-result>
</function-results>
```

**Figura 5.18: sintaxe IMML do elemento <*function-result*>**

Na VISUAL IMML o elemento repositório (<*function-results*>) será um estereótipo <<*function_results*>> derivado de pacote da UML padrão. O elemento resultado de função (<*function-result*>) é representado por um novo estereótipo descrito na figura 5.19. Esse elemento será utilizado juntamente com o relacionamento de composição da UML, para descrever os elementos que compõem os resultados de funções. Os resultados de função são representados no metamodelo de domínio como uma classe (figura 5.2) e tem como restrição ser composto por pelo menos uma estrutura de interação e/ou interação básica.

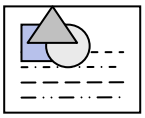


**Figura 5.19: Estereótipo da Visual IMML do elemento <*function-result*>**

#### *5.2.3.4 Interação Básica*

Uma interação básica especifica uma ação executada pelo usuário ao interagir com uma interface. São exemplos de interações básicas: clicar num botão, selecionar um elemento numa lista, digitar um texto ou número, visualizar um resultado, e outras. Existem várias interações básicas entre tarefas e/ou comandos/resultados de função, são elas:

A interação de *Entrar* com *Informação* é usada para especificar uma entrada de informação, como por exemplo, o usuário digitar as informações pedidas pela IU. Na IMML

ela é representada pelo elemento <*enter-information*> e possui a sintaxe mostrada na figura 5.20, já na Visual IMML será representada pelo estereótipo da figura 5.21 e é representado como uma classe no metamodelo de domínio (figura 5.2).

```
<enter-information domain-object="" />
```

**Figura 5.20: Sintaxe IMML do elemento <*enter-information*>.**

**Figura 5.21: Estereótipo da Visual IMML do elemento <*enter-information*>.**

A interação de *Selecionar Informação* é usada para especificar uma seleção de informação, como por exemplo, o usuário selecionar umas das alternativas mostradas pela IU.

Na IMML será representada pelo elemento <*select-information*> e possui a sintaxe mostrada na figura 5.22, já na Visual IMML será representada pelo estereótipo da figura 5.23 e pode ser visualizado no metamodelo de domínio como uma classe (figura 5.2)..

```
<select-information domain-object="" />
```

**Figura 5.22: Sintaxe IMML do elemento <*select-information*>.**

**Figura 5.23: Estereótipo da Visual IMML do elemento <*select-information*>.**

A interação *Ativar* controles é para especificar uma ativação de um controle, como por exemplo, o usuário ativer um controle através de um botão na IU. É representada na IMM pelo elemento <*activate*> (figura 5.24) e na Visual IMMl pelo estereótipo da figura 5.25 e é representado por uma classe no mentamodelo de domínio (figura 5.2).

```
<activate control="" />
```

**Figura 5.24: Sintaxe IMML do elemento <*activate*>.**

**Figura 5.25: Estereótipo da Visual IMML do elemento <*activate*>.**

A interação *Visualizar Informação* é usado especificar uma percepção de informação, como por exemplo, o usuário perceber um texto informativo na IU. Essa interação é representada pelo elemento <*perceive-information*> e tem como sintaxe na IMML a figura 5.26 e na Visual IMML tem como estereótipo a figura 5.27, é representado como uma

classe no metamodelo de domínio (figura 5.2) e poderá possuir, como restrição, apenas texto para a percepção direta e estática de uma informação, dessa forma, seu atributo é opcional.

```
<perceive-information domain-object="" />
          ou
<perceive-information>Texto</perceive-information>
```

**Figura 5.26: Sintaxe IMML do elemento <*perceive-information* >.**

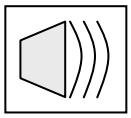


**Figura 5.27: Estereótipo da Visual IMML do elemento <*perceive-information*>.**

Por fim, a interação de *Navegação* é usada para especificar navegação entre tarefas e/ou comandos/resultados de função, como por exemplo, o usuário perceber que é necessário navegar na IU para atingir um objetivo.

Na IMML, é representada pelo elemento <*go*> e possui a sintaxe como mostrado na figura 5.28. Na Visual IMML terá como estereótipo a Figura 5.29, é representado no metamodelo de domínio como uma classe, tem como restrição possuir pelo menos uma interação básica, ou recursivamente, outra estrutura de interação e, por fim, pode receber três tipos de valores atribuídos a sua direção, que são: *away*, *ahead* e *back*

```
<go task="" />
       ou
<go function-command="" />
       ou
<go function-result="" />
```

**Figura 5.28: Sintaxe IMML do elemento <*go*>.**

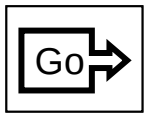



**Figura 5.29: Estereótipo da Visual IMML do elemento <go>.**

### *5.2.3.5 Realização de Função*

Além das interações básicas, a IMML dispõe de um elemento para especificar a realização de comandos de função ou resultados de função. O elemento realização é usado para especificar a realização de uma tarefa, um comando de função ou um resultado de função. Deverá ser usado apenas em tarefas para indicar a realização de comandos de função, resultados de função e/ou outras subtarefas, dentro da tarefa em questão.

O elemento da IMML que representa esta interação é <*do*>, tem como sintaxe a figura 5.30 e na Visual IMML terá como estereótipo o ícone observado na figura 5.31,é representado como uma classe no metamodelo de domínio (figura 5.2) e como restrição ser usado apenas em tarefas para indicar a realização de comandos/resultados de função e/ou outras tarefas, dentro da tarefa em questão.

```
<do task="" />
       ou
<do function-command="" />
       ou
<do function-result="" />
```

**Figura 5.30: Sintaxe IMML do elemento <*do*>.**

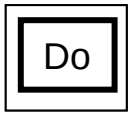



**Figura 5.31: Estereótipo da Visual IMML do elemento <do>.**

### *5.2.3.6 Estruturas de Interação*

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. A linguagem possui cinco estruturas de interação e na Visual IMML cada uma delas é representada por novos estereótipos, são eles: seqüência, repetição, agrupamento, seleção, combinação e junção.

A Seqüência especifica que o usuário deverá executar as interações de maneira ordenada. Ela tem como estereótipo, na Visual IMML, o ícone mostrado na figura 5.32, terá no metamodelo (figura 5.2) a representação de uma classe e como restrição deverá possuir pelo menos uma interação básica, ou recursivamente, outra estrutura de interação. Na IMML, o elemento <*sequence*> a sua sintaxe está mostrada na figura 5.33.

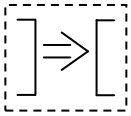


**Figura 5.32: Estereótipo da Visual IMML do elemento <*sequence*>**

```
<sequence>
     interação básica
          ou
     estrutura de interação
</sequence>
```

**Figura 5.33: Sintaxe IMML do elemento <sequence>**

A Repetição é usada quando o usuário precisa repetir várias vezes às interações. Na IMML o elemento <*repeat*> deverá conter pelo menos uma interação básica, ou recursivamente, outra estrutura de interação. A sua sintaxe está mostrada na figura 5.35. Ela tem como estereótipo, na Visual IMML, a figura 5.34, é representado por uma classe no metamodelo (figura 5.2) e como restrição deverá possuir pelo menos uma interação básica, ou recursivamente, outra estrutura de interação.

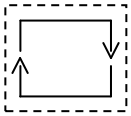


**Figura 5.34: Estereótipo da Visual IMML do elemento <*repeat*>**

```
<repeat>
     interação básica
          ou
     estrutura de interação
</repeat>
```

**Figura 5.35: Sintaxe IMML do elemento <*repeat*>**

A Seleção é usada quando o usuário precisa escolher uma dentre várias interações e, na VISUAL IMML, tem como estereótipo o ícone mostrado na figura 5.36, é representado por uma classe no metamodelo de domínio (figura 5.2) e tem como restrição possuir pelo menos uma interação básica, ou recursivamente, outra estrutura de interação. Já na IMML, possui a sintaxe da figura 5.37.

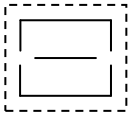


**Figura 5.36: Estereótipo da Visual IMML do elemento <*select*>**

```
<select>
     interação básica
          ou
     estrutura de interação
</select>
```

**Figura 5.37: Sintaxe IMML do elemento <*select*>**

A Combinação é usada quando duas ou mais interações têm alguma dependência entre si. Na sintaxe da IMML, o elemento <*combine*> está representado como pode ser visto na figura 5.38. Na Visual IMML esse elemento tem como estereótipo o ícone representado na figura 5.39, poderá ser visualizado no metamodelo (figura 5.2) como uma classe e tem como restrição possuir pelo menos uma interação básica, ou recursivamente, outra estrutura de interação.

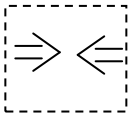


**Figura 5.38: Estereótipo da Visual IMML do elemento <*combine*>.**

```
<combine>
     interação básica
          ou
     estrutura de interação
</combine>
```

**Figura 5.39: Sintaxe IMML do elemento <*combine*>**

A Junção é usada para agrupar interações que tem algum relacionamento, mas não requerem uma ordem de realização. Na IMML, o elemento <*join*> tem como sintaxe a figura 5.41. Na Visual IMML esse elemento tem como estereótipo o ícone na figura 5.40, pode ser observado com o uma classe no mentamodelo de domínio e deverá possuir, como restrição, pelo menos uma interação básica, ou recursivamente, outra estrutura de interação.

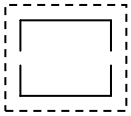


**Figura 5.40: Estereótipo da Visual IMML do elemento <*join*>**

```
<join>
     interação básica
            ou
     estrutura de interação
</join>
```

**Figura 5.41: Sintaxe IMML do elemento <*join*>**

### *5.2.3.7 Exemplo dos Elementos do Modelo de Interação*

Para exemplificar o modelo de interação, será utilizado o exemplo passado no modelo anterior. O exemplo da biblioteca digital que contem uma única função de busca.

O código IMML que especifica o exemplo pode ser observado na figura 5.42. Esse modelo é composto apenas por uma tarefa, uma função de comando e um resultado de comando. Esses três blocos são organizados utilizando os conceitos de interações básicas e estruturas de interações.

```
<interaction-model>
  <tasks>
    <task name="Searching">
      <sequence>
        <do function-command="SearchingCommand" />
        <do function-result="SearchingResult" />
      </sequence>
    </task>
  </tasks>
```

```
  <function-commands>
    <function-command name="SearchingCommand" domain-function="Search">
      <join>
        <perceive-information>Please enter the search
             parameters</perceive-information>
        <sequence>
          <select-information domain-object="SearchFields" />
          <enter-information domain-object="SearchKeys" />
          <select>
            <activate control="Search" />
            <activate control="Cancel" />
          </select>
        </sequence>
      </join>
    </function-command>
  </function-commands>
  <function-results>
    <function-result name="SearchingResult" domain-function="Search">
      <perceive-information domain-object="BooksList" />
    </function-result>
  </function-results>
</interaction-model>
```

**Figura 5.42: Especificação em IMML do modelo de interação da biblioteca digital.**

A figura 5.43 mostra como fica o código da figura 5.42 especificado usando a Visual IMML. Como pode ser observado, o modelo de interação é compostos por três repositórios, representados pelos respectivos estereótipos. No estereótipo *<<tasks>>*, representado por um pacote UML, está representada a tarefa do exemplo, que é composta por uma seqüência de dois elementos *<do>* para a realização dessa tarefa. No estereótipo *<<function-command>>*, está modelado o comando de função do exemplo e por fim, no estereótipo *<<function-result>* >pode-se observar o resultado da função. Todos organizados utilizando as interações básicas e estruturas de interações.

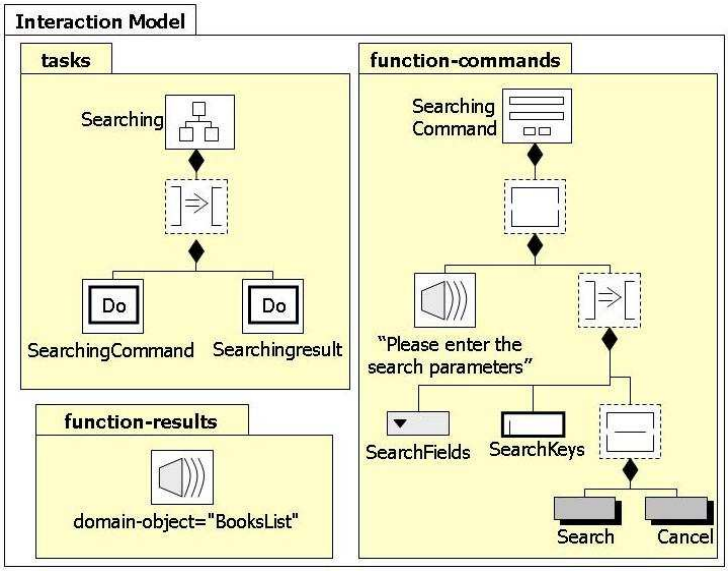



**Figura 5.43: Especificação em Visual IMML do modelo de interação da biblioteca digital.**

### 5.2.4 Modelo de Comunicação

O Modelo de Comunicação da IMML é responsável por integrar e comunicar ao usuário os modelos de domínio e interação. Ele possui alguns conceitos essenciais para sua compreensão e especificação, que serão absorvidos pela Visual IMML. São eles: signos de composição, signos interativos e os elementos de visualização de objetos de domínio.

Na Visual IMML o Modelo de Comunicação descreve, através de estereótipos, a organização espacial dos elementos da IU que serão comunicados ao usuário. Além disso, através de relacionamentos, o comportamento da interface é representado de uma forma diagramática.

Abaixo, descreveremos cada um dos elementos do modelo de comunicação, sua especificação em IMML e a sua representação na Visual IMML.

#### *5.2.4.1 Signos de Composição*

Os Signos de Composição são responsáveis pela organização estrutural dos signos interativos. Existem cinco unidades de composição, descritas a seguir.

O elemento *<task-environment>* (ambiente de tarefas) é usado para especificar um ambiente de realização de uma tarefa. Esse elemento é um conjunto de signos de composição que serão apresentados ao usuário ao longo do processo de interação, para realização da tarefa a ele associado. Na IMML esse elemento deverá possuir a sintaxe como mostrada na figura 5.44. Já na Visual IMML esse elemento será representado por um novo estereótipo cujo ícone pode ser observado na figura 5.45, é representado no metamodelo (figura 5.3) como uma classe e deverá está associado à tarefa para a qual ele permitirá sua realização e à plataforma que o suportará.

Esse elemento contém uma propriedade muito importante, é a propriedade de plataforma (*platform*), que passa ao implementador para qual plataforma a IU vai ser modelada. Essa propriedade pode ser vista na figura 5.44.

```
<task-environment name="" task="" platform="">
    ...
</task-environment>
```

**Figura 5.44:Sintaxe IMML do elemento *<task-environment>***

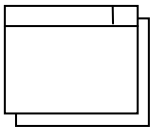


**Figura 5.45: Estereótipo da Visual IMML do elemento <*task-environment*>**

O elemento <*frame*> (quadro) é usado para especificar os signos que serão apresentados ao usuário num dado momento de interação. Ele será responsável por apresentar ao usuário os signos interativos necessários para comandar uma função de domínio ou apresentar seus resultados. Na linguagem IMML ele está especificado como na figura 5.46, já na Visual IMML esse elemento terá como estereótipo o ícone mostrado na figura 5.47 e é representado como uma classe no metamodelo de domínio (figura 5.3).

```
<frame name="" orientation="" align="">
     ...
</frame>
```

**Figura 5.46:Sintaxe IMML do elemento <*frame*>**

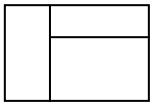


**Figura 5.47: Estereótipo da Visual IMML do elemento <*frame*>**

O elemento <*command-panel*> (painel de comando) é utilizado para especificar uma composição de signos interativos que irão comunicar as interações básicas de um comando de função. Esse elemento terá como sintaxe, na IMML, como se pode ver na figura 5.48 e terá como estereótipo o ícone mostrado na figura 5.49 e pode ser observado como uma classe no metamodelo (figura 5.3)

```
<command-panel       name=""       function-command=""       orientation=""
        align="">
     ...
</command-panel>
```

**Figura 5.48: Sintaxe IMML do elemento <*command-panel*>**

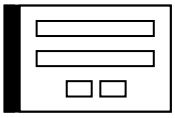


**Figura 5.49: Estereótipo da Visual IMML do elemento <*command-panel*>**

O Elemento <*display-area*> (área de visualização) é usado para especificar uma composição de signos interativos que irão comunicar as interações básicas de um resultado de função. Ele terá como sintaxe da IMML a especificação mostrada na figura 5.50

e será representado pelo estereótipo, na Visual IMML, mostrado na figura 5.51 e é representado no metamodelo (figura 5.3) como uma classe.

```
<display-area name="" function-result="" orientation="" align="">
    ...
</display-area>
```

**Figura 5.50: Sintaxe IMML do elemento <*display-area*>**

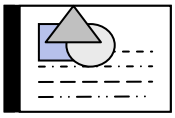


**Figura 5.51: Estereótipo da Visual IMML do elemento <*display-area*>**

O elemento <*group*> (grupo) é usado para especificar uma composição de signos interativos, comunicando a existência de alguma relação entre eles. Na IMML, esse elemento possui a sintaxe mostrada na figura 5.52, e pode ser visualizado na figura 5.53 o seu estereótipo da Visual IMML e é representado como uma classe no metamodelo de domínio (figura 5.3).

```
<group name="" orientation="" align="">
    ...
</group>
```

**Figura 5.52: Sintaxe IMML do elemento <*group*>**

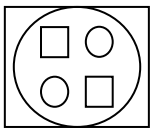


**Figura 5.53: Estereótipo da Visual IMML do elemento <*group*>**

Todos os signos de composição possuem uma propriedade chamada "*orientação*" que passa para o desenvolvedor se aqueles grupos de signos estão na orientação vertical ou horizontal. Para que essa propriedade seja visualizada na especificação da Visual IMML, os estereótipos serão utilizados juntamente com um retângulo pontilhado que mostrará a orientação passada pelo designer de IU. Para um melhor entendimento do uso destes retângulos pode-se observar o exemplo passado no *Subitem 5.2.3.4* e na figura 5.79.

#### *5.2.4.2 Signos Interativos*

Um Signo interativo é responsável por comunicar como realizar as interações básicas pelo usuário. Os signos interativos são descritos a seguir.

O elemento <*text*> (texto) especifica um texto que o usuário pode perceber na interface. Ele tem como restrição estar associado um objeto de domínio que é informação de saída de uma função de domínio ou ter seu conteúdo explicitamente definido pelo designer. Esse elemento tem como código IMML o que se pode observar na figura 5.54 e tem como estereótipo da Visual IMML o ícone mostrado na figura 5.55 e é representado por uma classe no metamodelo (figura 5.3).

```
<text name="" label="" label-position="" domain-object="" />
     ou
<text>Texto</text>
```

**Figura 5.54: Sintaxe IMML do elemento <*text*>**


**Figura 5.55: Estereótipo da Visual IMML do elemento <*text*>**

O elemento <*image*> (imagem) é usado para especificar um conteúdo gráfico ou imagem na interface que o usuário deve perceber como informação de saída de uma função de domínio Ele restringi-se a sempre ser associado a um objeto de domínio. Na IMML, esse elemento possui a sintaxe mostrada na figura 5.56. A figura 5.57 mostra o ícone do seu estereótipo da Visual IMML e é representado por uma classe no metamodelo de domínio (figura 5.3).

```
<image name="" label="" label-position="" domain-object="" />
```

**Figura 5.56: Sintaxe IMML do elemento <*image*>**

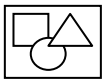


**Figura 5.57: Estereótipo da Visual IMML do elemento <*image*>**

O elemento <*edit-box*> (caixa de texto) permitirá ao usuário informar dados utilizando o teclado, que devem ser fornecidos como entrada em um comando de função (associado a uma função de domínio). Ele restringi-se a sempre ser associado a um objeto de domínio. Ele terá como sintaxe da IMML a especificação mostrada na figura 5.58. O ícone do seu estereótipo, na Visual IMML, está mostrado na figura 5.59 e é uma classe no metamodelo de domínio (figura 5.3).

```
<edit-box name="" label="" label-position="" domain-object="" />
```

**Figura 5.58: Sintaxe IMML do elemento <*edit-box*>**

**Figura 5.59: Estereótipo da Visual IMML do elemento <*edit-box*>**

O elemento <*text-area*> (área de texto) é usado para especificar um elemento que permita ao usuário fornecer várias linhas de texto e editá-lo com informações que devem ser fornecidas como entrada em um comando de função. Esse elemento restringi-se a sempre ser associado a um objeto de domínio. Esse elemento terá como sintaxe, na IMML, como se pode ver na figura 5.60, terá como estereótipo o ícone mostrado na figura 5.61 e representa uma classe no metamodelo (figura 5.3).

```
<text-area name="" label="" label-position="" domain-object="" />
```

**Figura 5.60: Sintaxe IMML do elemento <*text-area*>**


**Figura 5.61: Estereótipo da Visual IMML do elemento <*text-area*>**

O elemento <*check-box*> (caixa de checagem) é usado para permitir ao usuário perceber, marcar e desmarcar a seleção de um objeto de domínio que deve ser fornecido como entrada em um comando de função. Porém quando o objeto de domínio for um conjunto finito, a seleção dos itens poderá ser múltipla. Na linguagem IMML ele está especificado como na figura 5.62. Na Visual IMML esse elemento terá como estereótipo o ícone mostrado na f igura 5.63 e é representado como uma classe no metamodelo de domínio (figura5.3).

```
<check-box name="" label="" label-position="" domain-object="" />
```

**Figura 5.62: Sintaxe IMML do elemento <*check-box*>**

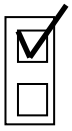


**Figura 5.63: Estereótipo da Visual IMML do elemento <*check-box*>**

O elemento <*radio-button*> (botão de rádio) permitirá que o usuário possa perceber, marcar e desmarcar a seleção de um objeto de domínio que deve ser fornecido como entrada em um comando de função. Porém quando o objeto de domínio for um conjunto finito, a seleção dos itens será exclusiva. Esse elemento terá como código IMML a especificação mostrada na figura 5.64 e seu estereótipo na Visual IMML será o ícone

mostrado na figura 5.65 e é representado por uma classe no metamodelo de domínio (figura 5.3).

```
<radio-button name=""label=""label-position="" domain-object="" />
```

**Figura 5.64: Sintaxe IMML do elemento <*radio-button*>**

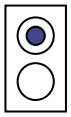


**Figura 5.65: Estereótipo da Visual IMML do elemento <*radio-button*>**

O elemento <*list-box*> (caixa-lista) é usado para o usuário perceber e selecionar múltiplos itens de uma lista de opções. Ele restringe-se a estar associado a elementos de um objeto de domínio e podem ser utilizados como entrada de um comando de função ou saída em um resultado de função. Na IMML, esse elemento possui a sintaxe mostrada na figura 5.66, e pode ser visualizado na figura 5.67 o ícone de seu estereótipo da Visual IMML e é uma classe no metamodelo de domínio (figura 5.3).

```
<list-box name="" label="" label-position="" domain-object="" />
```

**Figura 5.66: Sintaxe IMML do elemento <*list-box*>**

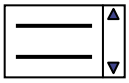


**Figura 5.67: Estereótipo da Visual IMML do elemento <*list-box*>**

O elemento <*drop-down-list*> tem a mesma aplicação de um <*list-box*>. Apenas a sua aparência é diferente. Esse elemento terá como sintaxe da IMML a especificação mostrada na figura 5.68 e será representado pelo estereótipo, na Visual IMML, mostrado na figura 5.69 e é uma classe do metamodelo de domínio (figura 5.3).

```
<drop-down-list name="" label="" label-position="" domain-object="" />
```

**Figura 5.68: Sintaxe IMML do elemento <*drop-down-list*>**

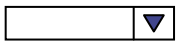


**Figura 5.69: Estereótipo da Visual IMML do elemento < *drop-down-list* >**

O elemento <*push-button*> (botão) permitirá ao usuário perceber e ativar alguma ação, que poderá ser uma transição para o próximo quadro e/ou ativação do controle de alguma função. Neste caso, ele estará associado a um controle de uma função de domínio.

Na IMML, esse elemento possui a sintaxe mostrada na figura 5.70.e pode ser visualizado na figura 5.71 o ícone de seu estereótipo da Visual IMML e na figura 5.3 ele é representado como uma classe do metamodelo.

Esse elemento é composto por uma propriedade muito importante, a de transição (transition). Com essa propriedade, quando um usuário ativa alguma ação através do elemento <*push-button*>, se realiza a transição de um signo de composição para outro, como por exemplo, o usuário querer ver o resultado da realização de uma tarefa apóes clicar em um determinado botão.

```
<push-button control="" transition=""/>
```

**Figura 5.70: Sintaxe IMML do elemento <*push-button*>**

**Figura 5.71: Estereótipo da Visual IMML do elemento <*push-button*>**

### *5.2.4.3 Elementos de Visualização de Objetos de Domínio*

Os objetos de domínio compostos podem ser organizados em estruturas visuais pré-definidas, tais como tabelas ou listas. A IMML dispõe de elementos para descrever como os objetos compostos podem ser comunicados ao usuário. Os elementos <*table*> e <*list*> permitem descrever, respectivamente, os objetos do domínio tabelas ou listas.

O elemento <*table*> deve ser usado para especificar uma composição de signos interativos, que deverá ser apresentada de maneira tabular. Caso esteja associada a um objeto de domínio que seja um conjunto de outros objetos, terá um elemento <*columns*> contendo os signos interativos para os elementos do conjunto. Na IMML, esse elemento possui a sintaxe mostrada na Figura 5.72, e pode ser visualizado na Figura 5.73 o seu estereótipo da Visual IMML e é uma classe no metamodelo de domínio (figura 5.3).

```
<table name="" domain-object="" head-position="">
  <columns> ... </columns>
</table>
```

**Figura 5.72: Sintaxe IMML do elemento <*table*>**

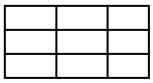


**Figura 5.73: Estereótipo da Visual IMML do elemento <*table*>**

O elemento <*columns*> será usado para especificar quais serão as colunas de uma tabela, possuirá signos interativos, onde cada um corresponder á a uma coluna na tabela e estará associado a um objeto componente do objeto associada à tabela. Na IMML, esse elemento possui a sintaxe mostrada na figura 5.74, e pode ser visualizado na figura 5.75 o seu estereótipo da Visual IMML e é uma classe no metamodelo de domínio (figura 5.3)..

```
<columns>
   ...
</columns>
```

**Figura 5.74: Sintaxe IMML do elemento <*columns*>**

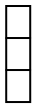


**Figura 5.75: Estereótipo da Visual IMML do elemento <*columns*>**

O elemento <*list*> é usado para especificar uma composição de signos interativos, que deverá ser apresentada de maneira ordenada. Esse elemento terá como sintaxe da IMML a especificação mostrada na figura 5.76 e será representado pelo estereótipo, na Visual IMML, mostrado na figura 5.77 e é uma classe no metamodelo de domínio (figura 5.3).

```
<list>
   ...
</list>
```

**Figura 5.76: Sintaxe IMML do elemento <*list*>**

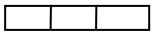


**Figura 5.77: Estereótipo da Visual IMML do elemento <*list*>**

#### *5.2.4.4 Exemplo do Modelo de Comunicação*

Para ilustrar esse modelo, usaremos o exemplo da biblioteca digital que é composta pela função de busca. Neste item será mostrado o código IMML para o modelo de comunicação desse exemplo e sua modelagem usando a Visual IMML correspondente ao código passado.

A figura 5.78 mostra o código IMML deste modelo. Ele é composto por apenas um ambiente de tarefas. Nele pode-se observar dois *frames*: um no qual o usuário irá realizar a busca e outro que irá mostrar o resultado da busca realizada.

```
<communication-model>
   <task-environment name="SearchingOnDesktop" task="Searching"
```

```
          platform="desktop">
     <frame name="frameSearchForm" orientation="vertical" align="left">
       <command-panel function-command="SearchingCommand"
                orientation="vertical" align="left">
          <text name="SearchMessage">Please enter the search
                parameters</text>
          <drop-down-list name="SearchFiels" label="Search fiels:" label-
                position="left" domain-object="SearchFields" />
          <edit-box name="SearchKeys" label="Search keys:" label-
                position="left" domain-object="SearchKeys" />
          <group orientation="horizontal" align="center">
            <push-button   name="Search"   label="Search"   control="Search"
transition = “frameBooksList”/>
            <push-button   name="Cancel"   label="Cancel"   control="Cancel"
hide=”this”/>
          </group>
       </command-panel>
     </frame>
     <frame name="frameBooksList" orientation="vertical" align="left">
       <text name="SearchResultMessage">The search result(s) is(are)</text>
       <display-area function-result="SearchingResult"
               orientation="vertical" align="left">
          <table domain-object="BooksList" head-position="top">
            <columns>
              <text name="Title" domain-object="Title" />
              <text name="Author" domain-object="Author" />
              <text name="Publisher" domain-object="Publisher" />
            </columns>
          </table>
       </display-area>
     </frame>
   </task-environment>
</communication-model>
```

**Figura 5.78: Especificação em IMML do modelo de comunicação da biblioteca digital.**

A figura 5.79 mostra como fica o novo diagrama da Visual IMML que corresponde ao código IMML passado na figura 5.78. Como pode ser observado, o ambiente de tarefas é composto por dois *frames* na horizontal. O primeiro é composto por um painel de comando onde o usuário irá realizar a busca. O segundo é formado por uma área de visualização, onde poderá ser observado o resultado da busca feita pelo usuário. Para o usuário passar de um frame para outro existe a transição através da ativação do comando do botão *Search*, como pode ser visto na figura abaixo.

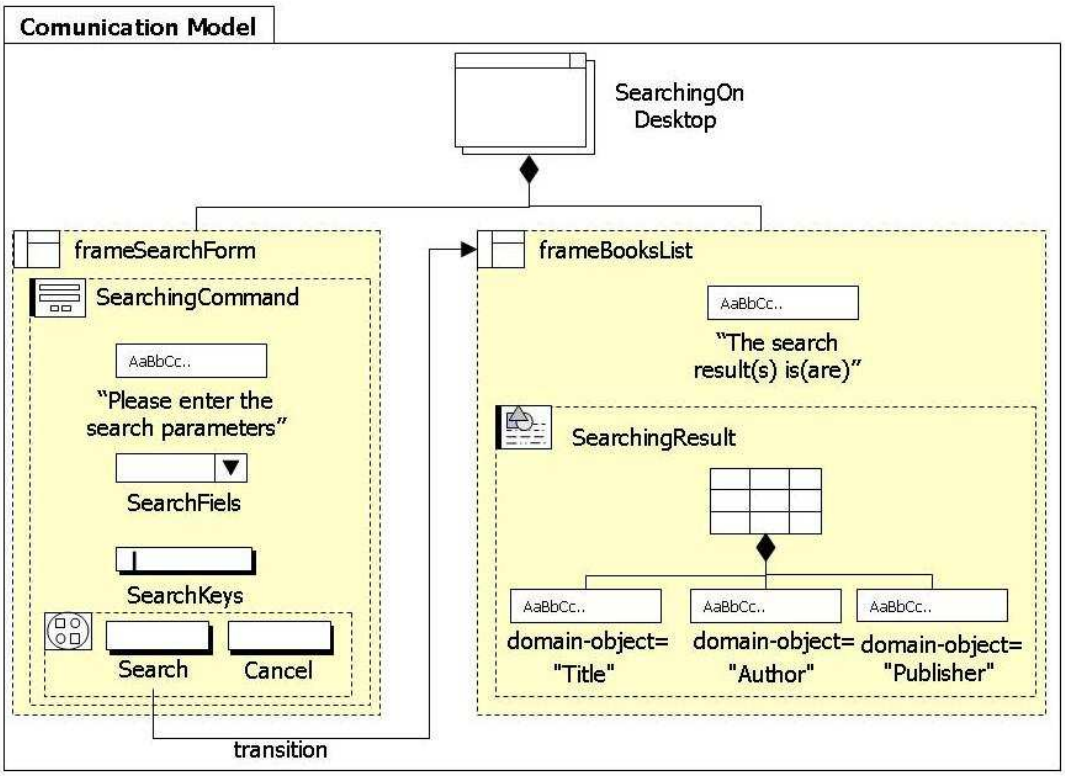



**Figura 5.79: Especificação em Visual IMML do modelo de comunicação da biblioteca digital.**

# Capítulo 6

# Aplicação da Visual IMML

Neste capítulo será apresentada uma especificação IMML como estudo de caso. Essa especificação foi desenvolvida pelo grupo de estudos de IHC da Universidade Federal do Rio Grande do Norte para a avaliação da linguagem IMML [Fonseca, 2005]. Agora a especificação do estudo de caso será utilizada para a modelagem utilizando a versão diagramática da IMML, a Visual IMML.

## 6.1 DESCRIÇÃO DO ESTUDO DE CASO

O estudo de caso descrito neste capítulo considera um banco como domínio de aplicação. O Banco Rico, como foi denominado, estava interessado em facilitar o atendimento ao cliente com o investimento de tecnologias. Assim, foi solicitado à equipe de desenvolvimento uma pesquisa de possíveis situações nos quais os clientes necessitam fazer uso dos serviços dos bancos. As situações analisadas foram:

- Utilização de um terminal de auto-atendimento, onde é possível: emitir extrato impresso ou para visualização na tela, consultar saldo, efetuar pagamento e efetuar pagamento apresentando o saldo restante na conta.
- Utilização de um aparelho de telefone celular, onde é possível: consultar saldo, efetuar pagamento e efetuar pagamento apresentando o saldo restante na conta.

- Utilização de um sistema *Web* do Banco Rico, onde é possível: emitir extrato (visualização na tela), consultar saldo, efetuar pagamento e efetuar pagamento apresentando o saldo restante na conta.

Para ilustrar estes casos, a equipe desenvolveu os seguintes cenários:

- **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 ele 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 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 isso, os designers foram encarregados de desenvolver diferentes interfaces de usuário que permitissem 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*. As diferentes interfaces deveriam utilizar um mesmo núcleo funcional para os serviços. Ou seja, as funções de consulta e pagamentos deveriam ser as mesmas para todas as formas de acesso.

A partir dos requisitos (cenários e casos de uso) descritos anteriormente, serão apresentadas nas subseções a seguir as modelagens destas interfaces de usuário utilizando a IMML e a Visual IMML.

## 6.2 APLICAÇÕES DA IMML E DA VISUAL IMML NO ESTUDO DE CASO

Nesta seção, será descrita a modelagem do estudo de caso usando a IMML e, logo após, a modelagem utilizando os diagramas e estereótipos da Visual IMML. Para uma melhor visualização, a modelagem estará dividida em três partes, ou melhor, nos três modelos que compõem as linguagens: Modelo de Domínio, Modelo de Interação e Modelo de Comunicação.

Inicialmente, pode-se observar como está especificado o cabeçalho de um documento IMML, já que essa linguagem está baseada na linguagem XML (figura 6.1).

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<imml      application="BancoRico      xmlns="http://www.dimap.ufrn.br/imml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <domain-model>
   <interaction-model>
   <communication-model>
</imml>
```

**Figura 6.1: Cabeçalho IMML da especificação do Banco Rico**

### 6.2.1 Modelo de Domínio

Para mostrar como ficará o modelo de domínio dessa interface, o código IMML será dividido em duas partes. A primeira será a dos objetos de domínio e a segunda especifica as funções de domínio desse modelo. Logo após cada figura descrevendo o código, poderá ser observado o diagrama da Visual IMML correspondente, ficando assim, mais fácil à visualização.

A figura 6.2, apresenta tosos os objetos de domínio que fazem parte do modelo de domínio da IU do nosso estudo de caso, o Banco Rico, modelada usando a IMML.

```
<domain-objects>
  <domain-object name="Conta" type="number" />
  <domain-object name="Agencia" type="number" />
  <domain-object name="Senha" type="number" />
  <domain-object name="DataInicio" type="date" />
  <domain-object name="DataFim" type="date" />
  <domain-object name="DataTransacao" type="date" />
  <domain-object name="NomeTransação" type="text" />
  <domain-object name="ValorTransação" type="number" />
  <domain-object name="TipoTransaçao" type="text" />
  <domain-object name="Saldo" type="number" />
  <domain-object name="CodigoPagamento" type="number" />
  <domain-object name="DataPagamento" type="date" />
  <domain-object name="ValorPagamento" type="number" />
  <domain-object name="NomeDoMes" type="text" />
  <domain-object name="EstatusPagamento" type="finite-set">
    <item>Pagamento Efetuado com Sucesso!</item>
    <item>Pagamento Não Efetuado!</item>
  </domain-object>
  <domain-object name="Extrato" type="composed-object">
    <item domain-object="Transação" />
    <item domain-object="Saldo" />
  </domain-object>
  <domain-object name="Transação" type="table">
    <item domain-object="DataTransacao" />
    <item domain-object="NomeTransação" />
    <item domain-object="ValorTransação" />
    <item domain-object="TipoTransação" />
  </domain-object>
  <domain-object name="Mes" type="table">
    <item domain-object="NomeDoMes" />
  </domain-object>
</domain-objects>
```

**Figura 6.2: Código IMML dos Objetos de Domínio do Banco Rico.**

A figura 6.3 mostra o diagrama da Visual IMML que corresponde ao código mostrado na Figura acima (figura 6.2). Esse diagrama é composto por um pacote, que tem a função de organizar; pelos objetos de domínio, que são representados pelo estereótipo de objetos da UML padrão; e, por fim, pelo relacionamento de composição que existe entre os objetos de domínio que são do tipo composto, do tipo *finite-set* ou do tipo *table*.

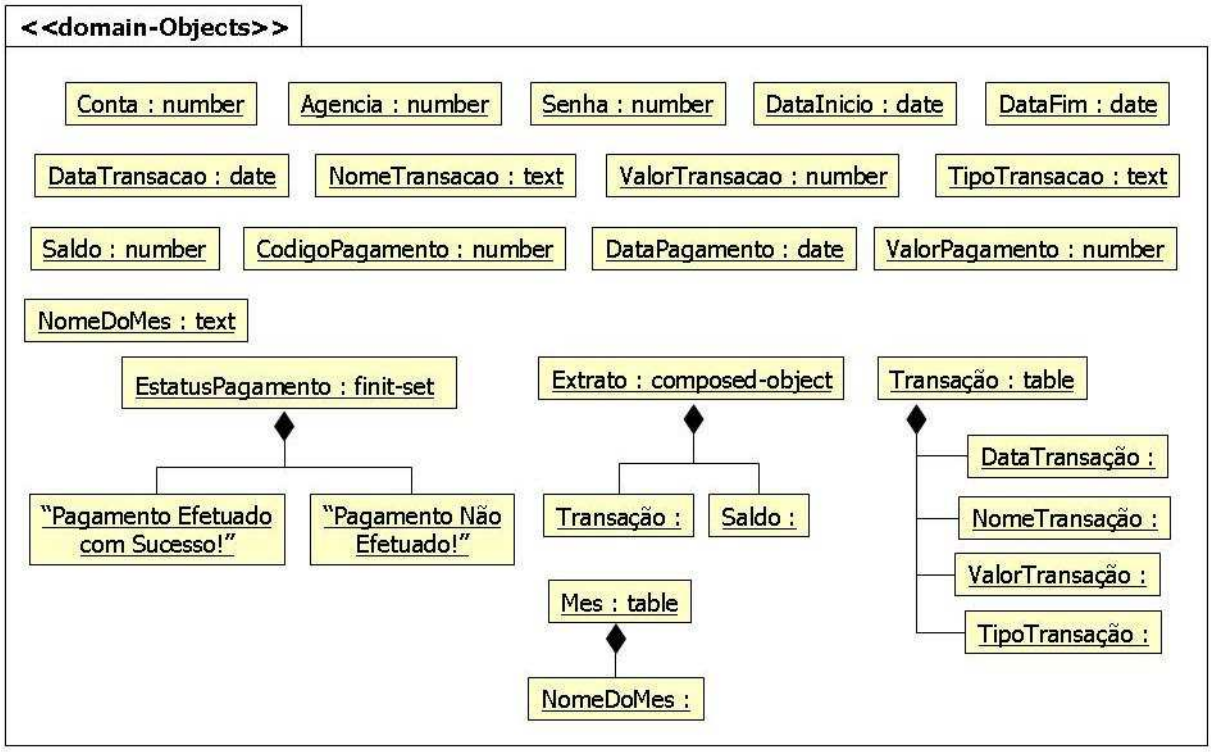



**Figura 6.3: Diagrama Visual IMML dos Objetos de Domínio do Banco Rico.**

Já a figura 6.4, apresenta as funções de domínio do exemplo do Banco Rico modelada usando a IMML. As funções do domínio correspondem aos casos de uso referentes aos cenários descritos acima. Por isso, a figura 6.4 está dividida em quatro funções de domínio: a primeira é chamada de Emitir Extrato; a segunda é denominada Consultar Saldo; a terceira é a de Efetuar Pagamento; e, por fim, a quarta é Efetuar Pagamento com Saldo. Cada uma dessas funções são compostas pelos seus operandos de entrada, operandos de saída, seus controles e estados.

```
<domain-functions>
  <!-- EMITIR EXTRATO -->
  <domain-function name="EmitirExtrato">
    <input-operands>
      <item domain-object="Conta" />
      <item domain-object="Agencia" />
      <item domain-object="Senha" />
      <item domain-object="DataInicio" />
      <item domain-object="DataFim" />
    </input-operands>
    <output-operands>
      <item domain-object="Extrato" />
    </output-operands>
    <controls>
      <control name="Consultar" from-state="initial"
        to-state="Consultando" />
      <control automatic="yes" from-state="Consultando"
        to-state="final" />
    </controls>
    <states>
      <state name="Consultando" />
    </states>
```

```
   </domain-function>
   <!--  CONSULTAR SALDO  -->
   <domain-function name="ConsultarSaldo">
     <input-operands>
       <item domain-object="Conta" />
       <item domain-object="Agencia" />
       <item domain-object="Senha" />
       </input-operands>
     <output-operands>
       <item domain-object="Saldo" />
     </output-operands>
     <controls>
       <control name="Consultar" from-state="Disponível"
         to-state="Consultando" />
       <control automatic="yes" from-state="Consultando"
         to-state="ExibindoSaldo" />
     </controls>
     <states>
       <state name="Disponivel" />
       <state name="Consultando" />
       <state name="Exibindo" />
     </states>
   </domain-function>
   <!--  EFETUAR PAGAMENTO  -->
   <domain-function name="EfetuarPagamento">
     <input-operands>
       <item domain-object="Conta" />
       <item domain-object="Agencia" />
       <item domain-object="Senha" />
       <item domain-object="CodigoPagamento" />
       <item domain-object="DataPagamento" />
       <item domain-object="ValorPagamento" />
     </input-operands>
     <output-operands>
       <item domain-objects="Saldo" />
     </output-operands>
     <controls>
     <control name="Pagar" from-state="Disponível"
       to-state="EfetuandoPagamento" />
     <control automatic="yes" from-state="EfetuandoPagamento"
       to-state="ExibindoResultadoDoPagamento" />
     </controls>
     <states>
       <state name="Disponivel" />
       <state name="EfetuandoPagamento" />
       <state name="ExibindoResultadoDoPagamento" />
     </states>
   </domain-function>
   <!--  EFETUAR PAGAMENTO COM SALDO  -->
   <domain-function name="EfetuarPagamentoComSaldo"
       extends="EfetuarPagamento">
     <controls>
       <control name="Consultar" from-state="Disponivel"
         to-state="ConsultandoSaldo" />
       <control automatic="yes" from-state="ConsultandoSaldo"
         to-state="ExibindoSaldo" />
       <control name="Pagar" from-state="ExibindoSaldo"
         to-state="EfetuandoPagamento" />
-   <!--   control   automatic="yes"   from-state="EfetuandoPagamento"   to-
state="ExibindoSaldo"      este  controle  já  foi  definido  na  função  mais
abstrata -->
     </controls>
     <states>
       <state name="ConsultandoSaldo" />
       <state name="ExibindoSaldo" />
     </states>
</domain-function>
```

**Figura 6.4: Código IMML das funções de Domínio do Banco Rico.**

A figura 6.5, abaixo, descreve a primeira parte da modelagem das funções de domínio. Como pode ser observado, cada função de domínio corresponde a um caso de uso do nosso diagrama. Cada um dos casos de uso são detalhados pelos elementos da Visual IMML descritos pelo estereótipo *<<domain_function_model>>*. Esse elementos estão detalhados na próxima figura (figura 6.6).

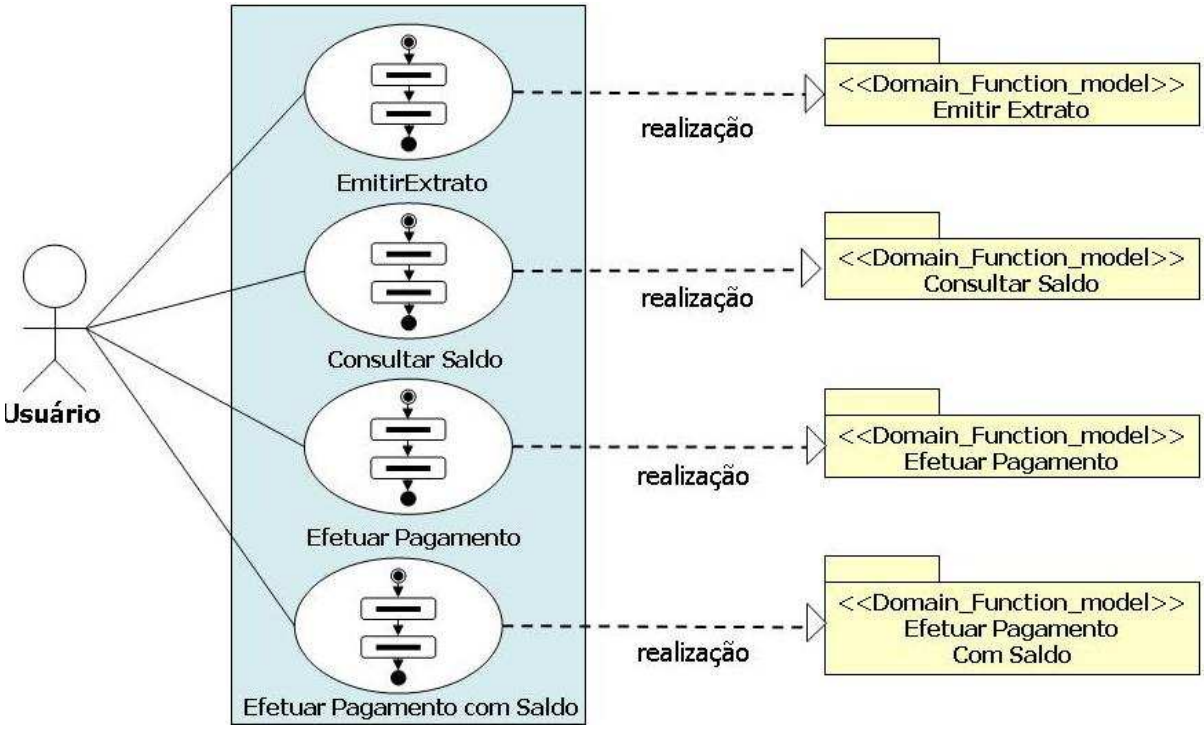



**Figura 6.5: Funções de domínio do Banco Rico, especificadas em Visual IMML**

Os estereótipos de realização dos casos de usos, mostrados na figura 6.5, serão detalhados para descrever os elementos componentes. Estes componentes estão organizados em dois pacotes. O primeiro contém o modelo de estados, que detalha as condições, os estados e controles que fazem parte da função de domínio correspondente. O segundo contém os operandos de entrada e saída da função de domínio. Como podem ser observados, esses operandos na verdade são os objetos de domínio que fazem parte da função.

O detalhamento do caso de uso Emitir Extrato pode ser observado na figura 6.6, onde em seu modelo de estado têm-se apenas dois controles e um estado. Nos estereótipos dos operandos podem-se observar quais objetos de domínio essa função vai utilizar como operandos de entrada e saída. Não é necessário descrever os tipos dos objetos, uma vez que os mesmo estão descritos no modelo de objetos de domínio. A transição *yes* indica que a mudança de estados ocorre automaticamente.

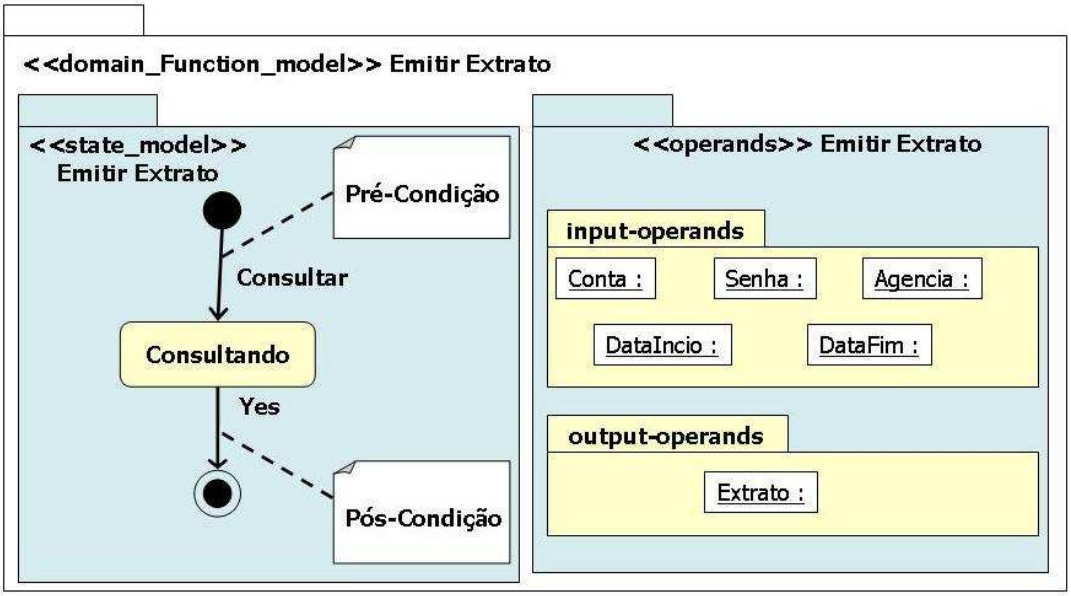



**Figura 6.6: Detalhamento da função de domínio Emitir Extrato.**

A Figura 6.7 descreve o detalhamento da função de domínio Consultar Saldo. O diagrama do modelo de estados descreve os três estados: *disponível*, *consultando* e *exibindo saldo*. O controle *consultar*. No diagrama de operandos se podem observar quais objetos de domínio essa função vai utilizar como operando de entrada e saída.

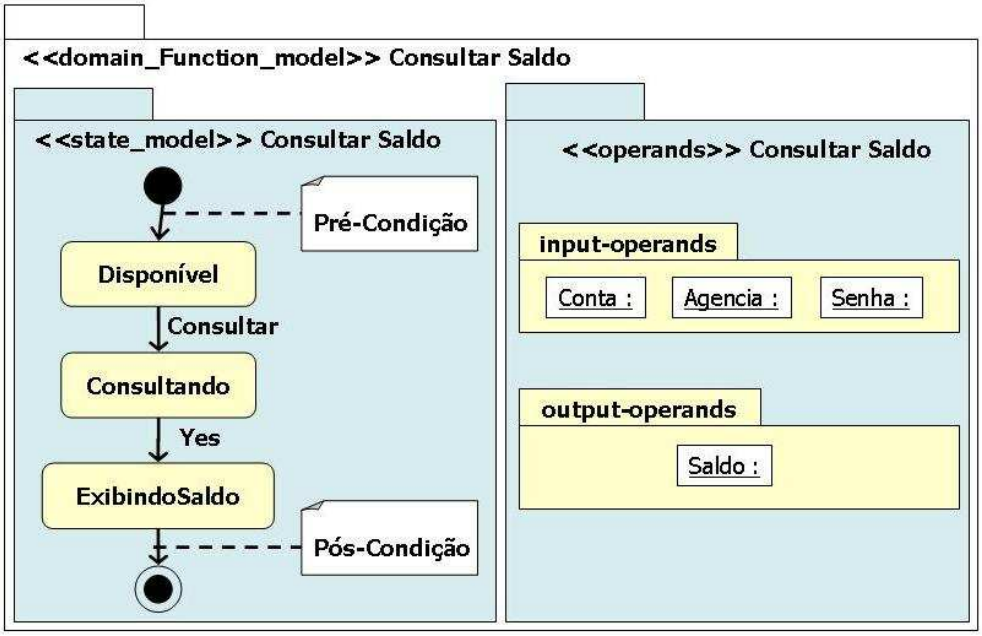



**Figura 6.7: Detalhamento da função de domínio Consultar Saldo.**

Na Figura 6.8 pode-se observar o detalhamento da função de domínio Efetuar Pagamento. O diagrama do modelo de estados é semelhante ao da função Consultar Saldo. Ele contém dois controles: o de *pagar* e o *yes*; e três estados: *disponível*, *efetuar pagamento* e

*exibindo resultado do pagamento*. O modelo de operandos especifica quais objetos de domínio essa função vai utilizar como operandos de entrada e saída.

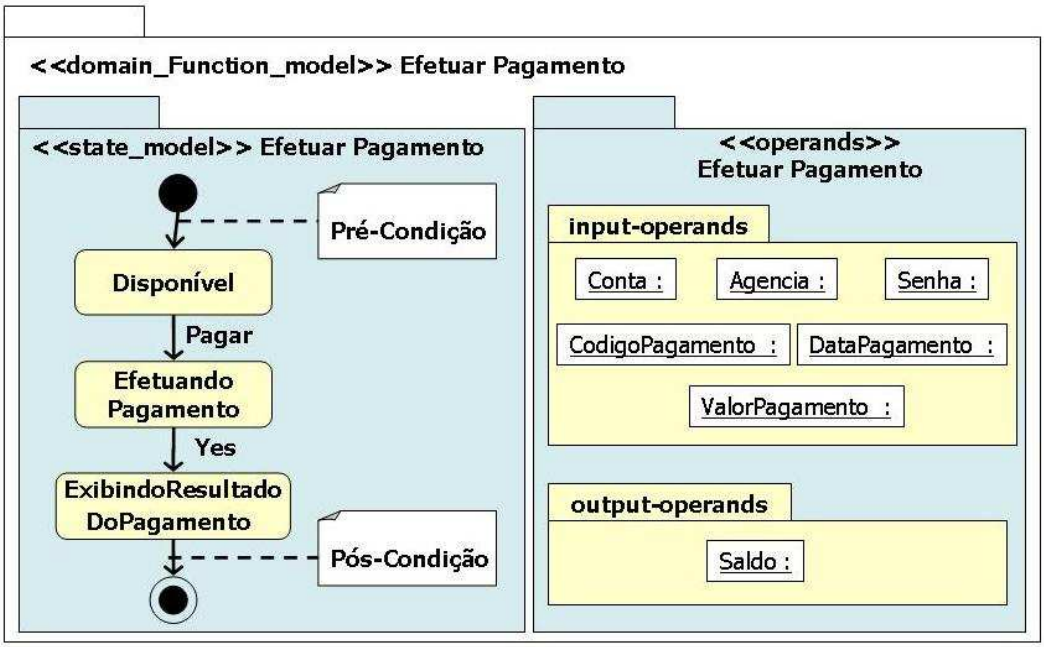



**Figura 6.8: Detalhamento da função de domínio Efetuar Pagamento.**

Por fim, a Figura 6.9 mostra os detalhes da função de domínio Efetuar Pagamento Com Saldo. O modelo de estados contém três controles: o de *consultar*, o *yes* e o de *pagar*; e é composto por quatro estados: *disponível*, *consultar saldo*, *exibindo saldo* e *efetuar pagamento*. O modelo de operandos podem ser observados quais objetos de domínio essa função vai utilizar como operandos de entrada e saída.

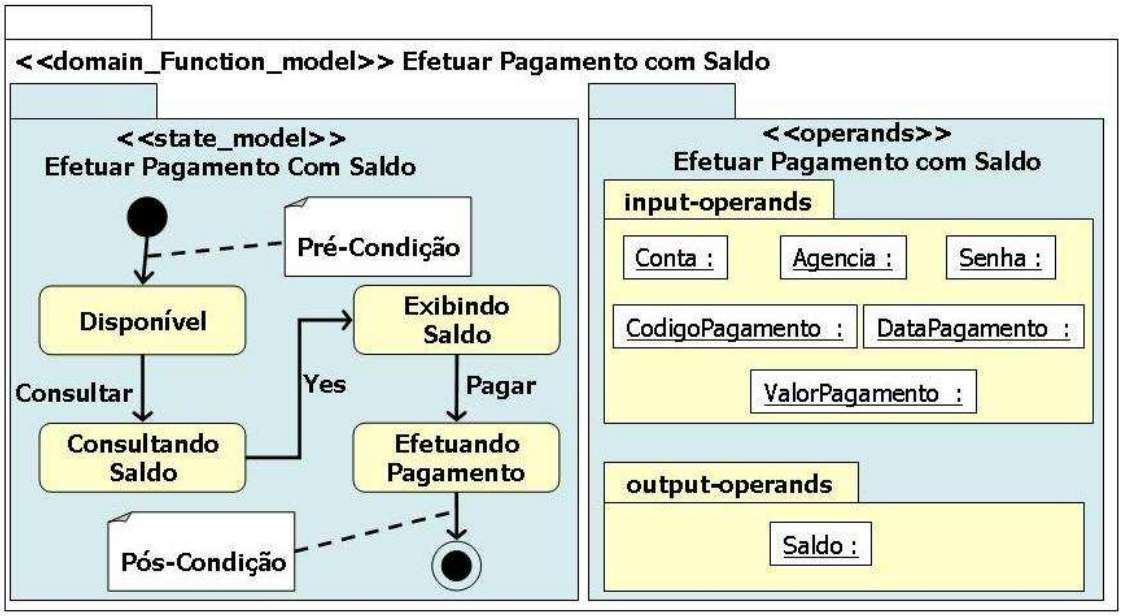



**Figura 6.9: Detalhamento da função de domínio Efetuar Pagamento com Saldo.**

### 6.2.2 Modelo de Interação

A estrutura da especificação IMML de um modelo de interação pode ser vista na figura 6.10:

```
<interaction-model>
  <tasks>
  </tasks>
  <function-commands>
  </function-commands>
  <function-results>
  </function-results>
</interaction-model>
```

**Figura 6.10: Especificação IMML do Modelo de Interação.**

Na Visual IMML, para uma melhor organização e leitura da modelagem, o modelo de interação será dividido em três pacotes, são eles o repositório das tarefas, das funções de comandos e dos resultados de comando, como pode-se ver na figura 6.11.

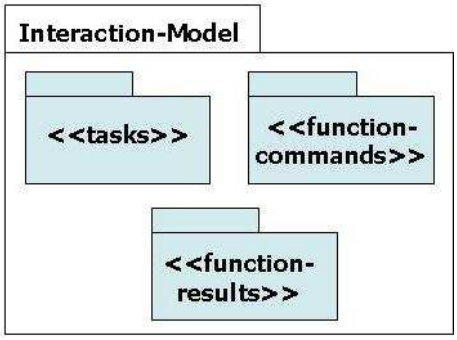



**Figura 6.11: Especificação Visual IMML do Modelo de Interação.**

Para ilustrar o modelo de interação desta interface, o código IMML será dividido em três partes. A primeira especifica as tarefas deste modelo. A segunda especifica os comandos de função. A última especifica os resultados de função do modelo de interação. Logo após cada figura que apresenta o código, pode ser observado o diagrama da Visual IMML correspondente, ficando assim, mais fácil à visualização.

A figura 6.12 mostra as tarefas que fazem parte do modelo da IU do Banco Rico modelada usando a IMML. Esse modelo é composto por cinco tarefas: tarefa Principal, tarefa Emitir Extrato, tarefa Consultar Saldo, tarefa Efetuar Pagamento Simples e, por fim, tarefa Emitir Pagamento com Saldo Antes.

```
<tasks>
  <task name="Pincipal">
    <select>
      <do task="EmitirExtrato" />
      <do task="ConsultarSaldo" />
      <do task="EfetuarPagamentoSimples" />
      <do task="EfetuarPagamentoComSaldoAntes" />
    </select>
  </task>
  <task name="EmitirExtrato">
    <sequence>
      <do function-command="Extrato" />
      <do function-result="SaidaExtrato" />
    </sequence>
  </task>
  <task name="ConsultarSaldo">
    <sequence>
      <do function-command="Saldo" />
      <do function-result="SaidaSaldo" />
    </sequence>
  </task>
  <task name="EfetuarPagamentoSimples">
    <sequence>
      <do function-command="PagamentoSimples" />
      <do function-result="SaidaPagamento" />
    </sequence>
  </task>
  <task name="EfetuarPagamentoComSaldoAntes">
    <sequence>
      <do function-command="Saldo" />
      <do function-command="PagamentoSimples" />
      <do function-result="SaidaPagamento" />
    </sequence>
  </task>
</tasks>
```

**Figura 6.12: Especificação IMML das tarefas do Banco Rico.**

A Figura 6.13 mostra o diagrama com que descreve as tarefas. O conjunto de tarefas é agrupado dentro do estereótipo <<*tasks*>>. Cada tarefa é modelada com os estereótipos da Visual IMML que representam os detalhes das tarefas que fazem parte da IU do Banco Rico. De forma geral, estas tarefas contêm uma estrutura de interação que organiza as atividades a serem realizadas. Cada atividade, seja ela comando de função ou resultado de função, é referenciada pelo estereótipo <<do>>.

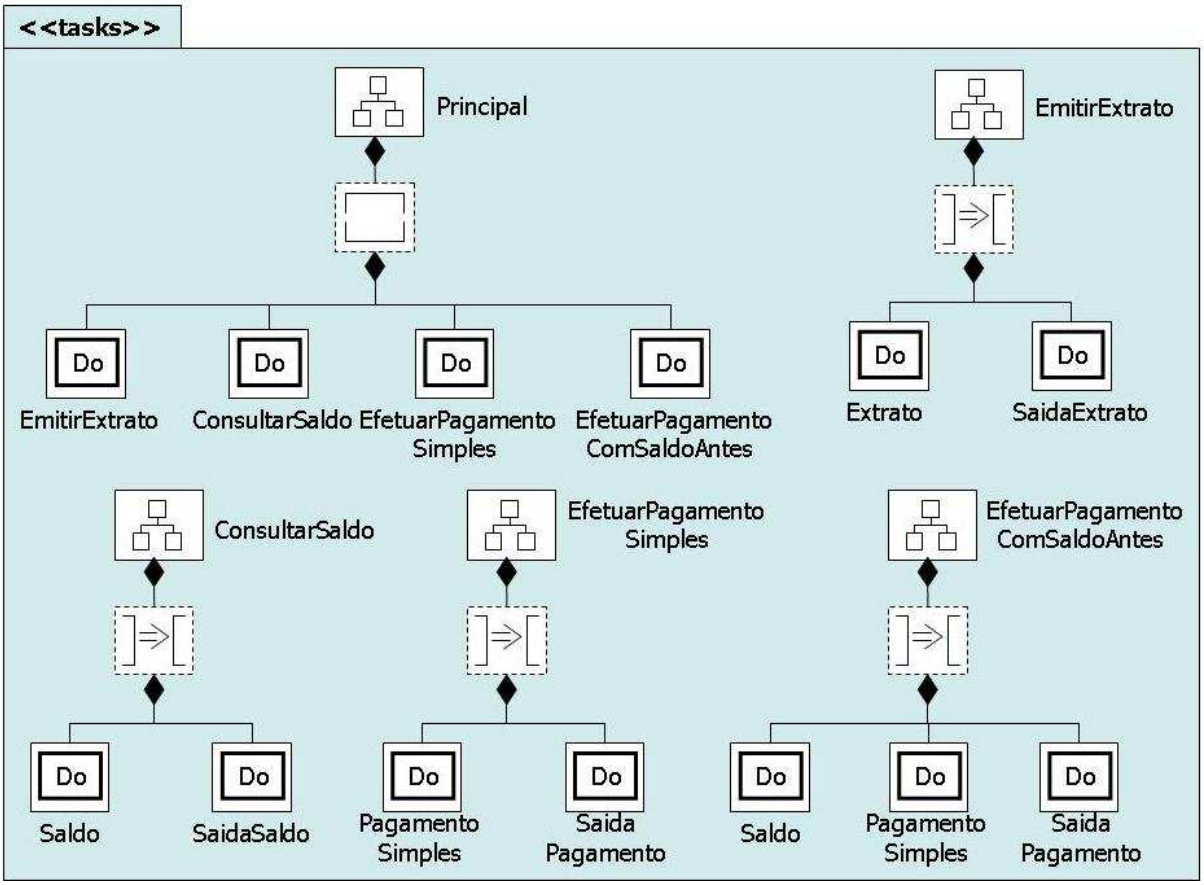



**Figura 6.13: Diagrama Visual IMML das tarefas do Banco Rico.**

A figura 6.14 apresenta a especificação IMML dos comandos de função que fazem parte do modelo de interação da IU do Banco Rico. Essa IU é composta por quatro funções de comando, são elas: *extrato*, *saldo*, *pagamento simples* e *efetuar pagamento com saldo antes*. Essas funções são organizadas utilizando as interações básicas e as estruturas de interações.

```
<function-commands>
  <function-command name="Extrato" domain-function="EmitirExtrato">
    <select>
      <sequence>
        <enter-information domain-object="Conta" />
        <enter-information domain-object="Agencia" />
        <enter-information domain-object="Senha" />
        <select>
          <join>
            <perceive-information>Escolha o mes que deseja
              consultar extrato
            </perceive-information>
            <enter-information domain-object="Mes" />
          </join>
          <join>
            <perceive-information>Digite a data de inicio e
              final, ou somente a data de inicio
            </perceive-information>
            <enter-information domain-object="DataInicio" />
```

```
            <enter-information domain-object="DataFim" />
          </join>
        </select>
        <activate control="Consultar" />
      </sequence>
      <go direction="away" />
    </select>
  </function-command>
  <function-command name="Saldo" domain-function="ConsultarSaldo">
    <select>
      <sequence>
        <enter-information domain-object="Conta" />
        <enter-information domain-object="Agencia" />
        <enter-information domain-object="Senha" />
        <activate control="Consultar" />
      </sequence>
      <go direction="away" />
      </select>
  </function-command>
  <function-command name="PagamentoSimples" domain-
      function="EfetuarPagamento">
    <select>
      <sequence>
        <enter-information domain-object="Conta" />
        <enter-information domain-object="Agencia" />
        <enter-information domain-object="Senha" />
        <enter-information domain-object="CodigoPagamento" />
        <enter-information domain-object="DataPagamento" />
        <enter-information domain-object="ValorPagamento" />
        <activate control="Pagar" />
      </sequence>
      <go direction="away" />
    </select>
  </function-command>
  <function-command name="PagamentoComSaldoAntes" domain-
      function="EfetuarPagamentoComSaldoAntes">
    <select>
      <sequence>
        <join>
          <enter-information domain-object="Conta" />
          <enter-information domain-object="Agencia" />
          <enter-information domain-object="Senha" />
        </join>
        <activate control="Consultar" />
        <join>
          <perceive-information>O seu saldo é:
          </perceive-information>
          <perceive-information domain-object="Saldo" />
          <perceive-information>Deseja continuar o pagamento?
          </perceive-information>
          <select>
            <go direction="away" />
            <join>
              <go direction="ahead" />
              <sequence>
                <enter-information domain-object=
                  "CodigoPagamento"/>
                <enter-information domain-object=
                  "DataPagamento" />
                <enter-information domain-object=
                  "ValorPagamento" />
              </sequence>
              <activate control="Pagar" />
            </join>
          </select>
        </join>
      </sequence>
      <go direction="away" />
    </select>
```

```
  </function-command>
</function-commands>
```

**Figura 6.14: Código IMML dos comandos de função do Banco Rico.**

A partir da figura 6.14 foram elaboradas as figuras 6.15 e 6.16 que correspondem à especificação da IU utilizando a Visual IMML. Para uma melhor visualização, a figura 6.14 mostra apenas os dois primeiros comandos de função (extrato e saldo) da IU do Banco Rico e a figura 6.15 corresponde aos dois últimos (pagamento simples e pagamento com saldo antes).

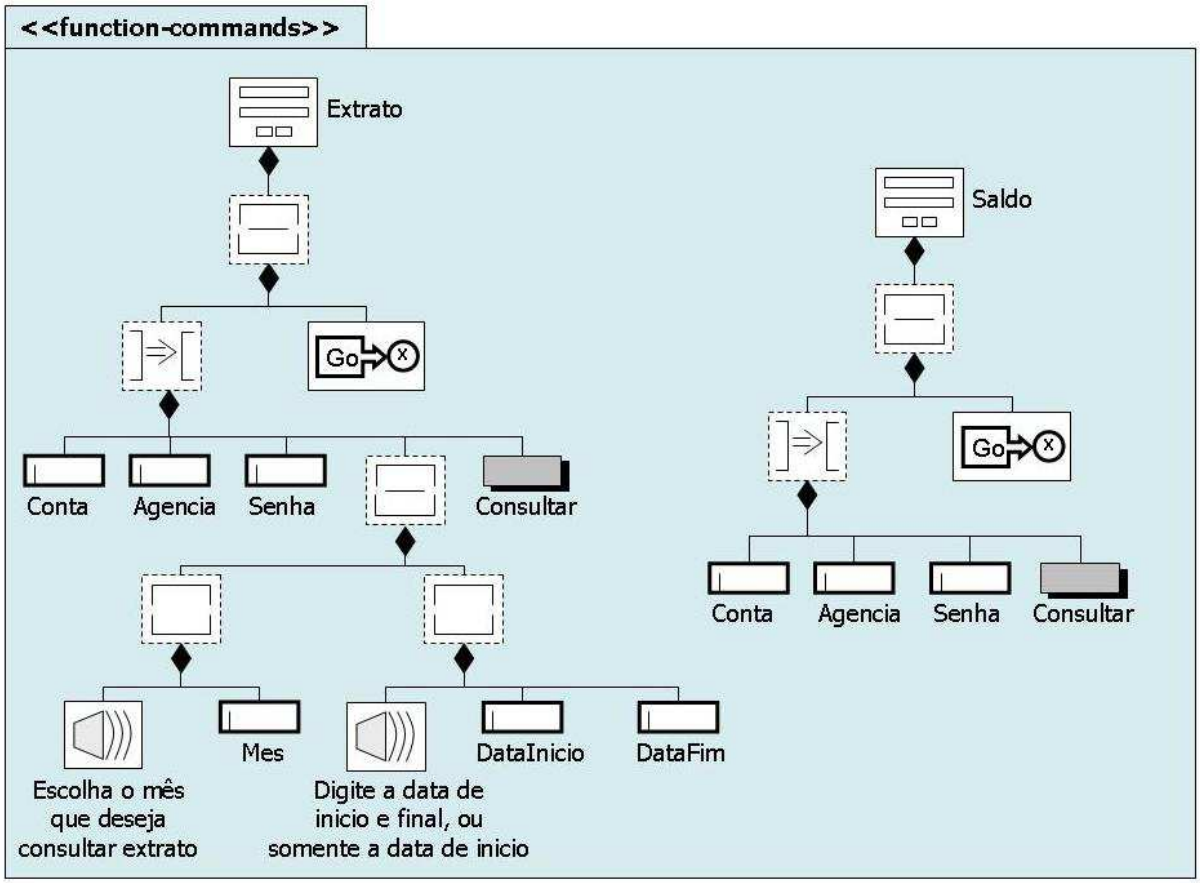



**Figura 6.15: Diagrama Visual IMML dos comandos de função do Banco Rico – Parte I.**

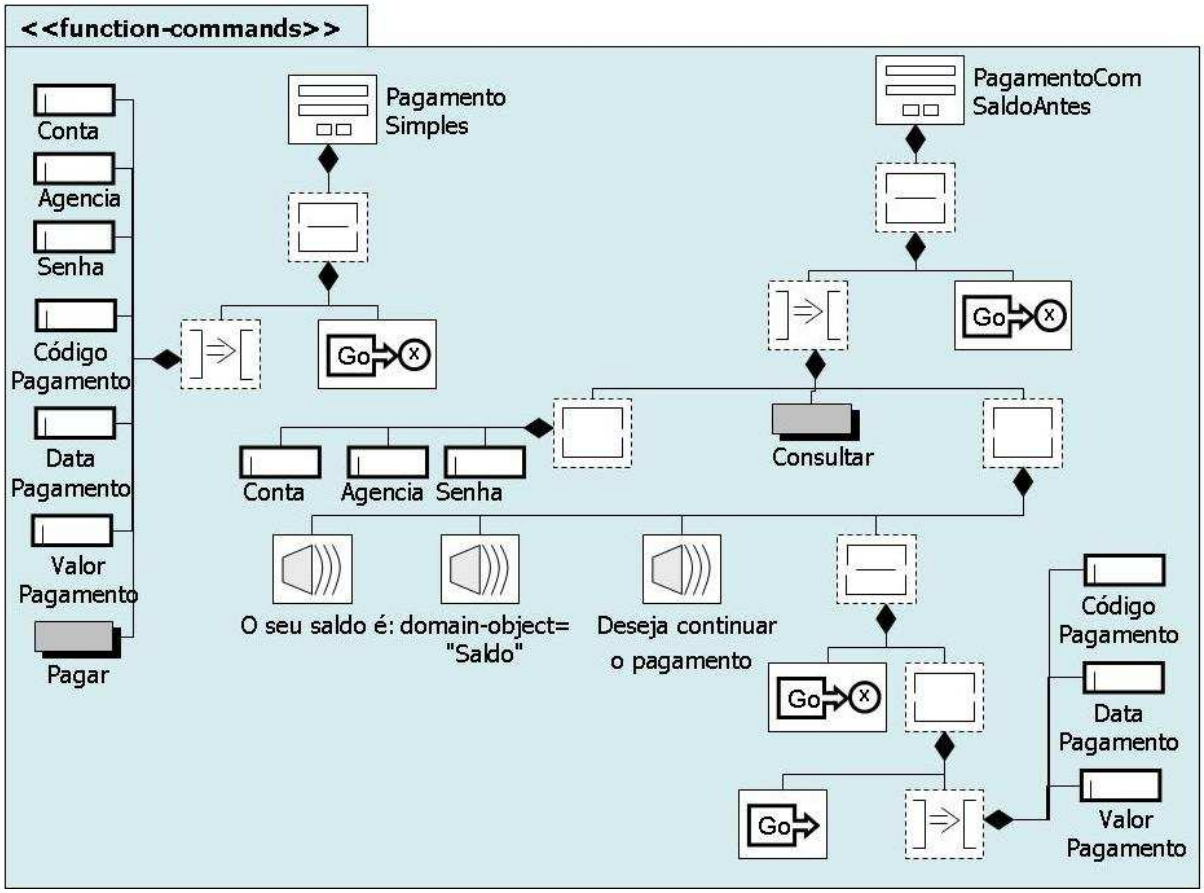



**Figura 6.16: Diagrama Visual IMML dos comandos de função do Banco Rico – Parte II.**

Para finalizar a especificação do modelo de interação, será observado o código IMML dos resultados de função que fazem parte da IU do Banco Rico. Essa especificação é composta por quatro resultados de funções, são elas: *SaidaExtrato*, *SaidaSaldo*, *SaidaPagamento* e *SaidaPagamentoComSaldo*. Os resultados de função, igualmente como as tarefas e comandos de função, são organizados usando as interações básicas e as estruturas de interação. Como pode ser observado na figura 6.17.

```
<function-results>
  <function-result name="SaidaExtrato" domain-
    function="EmitirExtrato">
    <perceive-informaiton domain-object="Extrato" />
  </function-result>
  <function-result name="SaidaSaldo" domain-
    function="ConsultarSaldo">
    <sequence>
      <perceive-information>O seu saldo é:</perceive-information>
      <perceive-informaiton domain-object="Saldo" />
    </sequence>
  </function-result>
  <function-result name="SaidaPagamento" domain-
    function="EfetuarPagamento">
    <sequence>
      <perceive-information>Pagamento efetuado com
        Sucesso!</perceive-information>
      <perceive-informatio>O seu saldo é:</perceive-informatio>
      <perceive-informaiton domain-object="Saldo" />
```

```
      </sequence>
   </function-result>
   <function-result name="SaidaPagamentoComSaldo" domain-
      function="EfetuarPagamentoComSaldoAntes">
      <sequence>
         <perceive-information>Pagamento efetuado com
           Sucesso!</perceive-information>
         <perceive-information>O seu saldo é:</perceive-information>
         <perceive-informaiton domain-object="Saldo" />
      </sequence>
   </function-result>
</function-results>
```

**Figura 6.17: Código IMML dos resultados de função do Banco Rico.**

Por fim, na figura 18 pode-se observar o diagrama da Visual IMML correspondente aos resultados de função que fazem parte do modelo de interação da IU do Banco Rico.

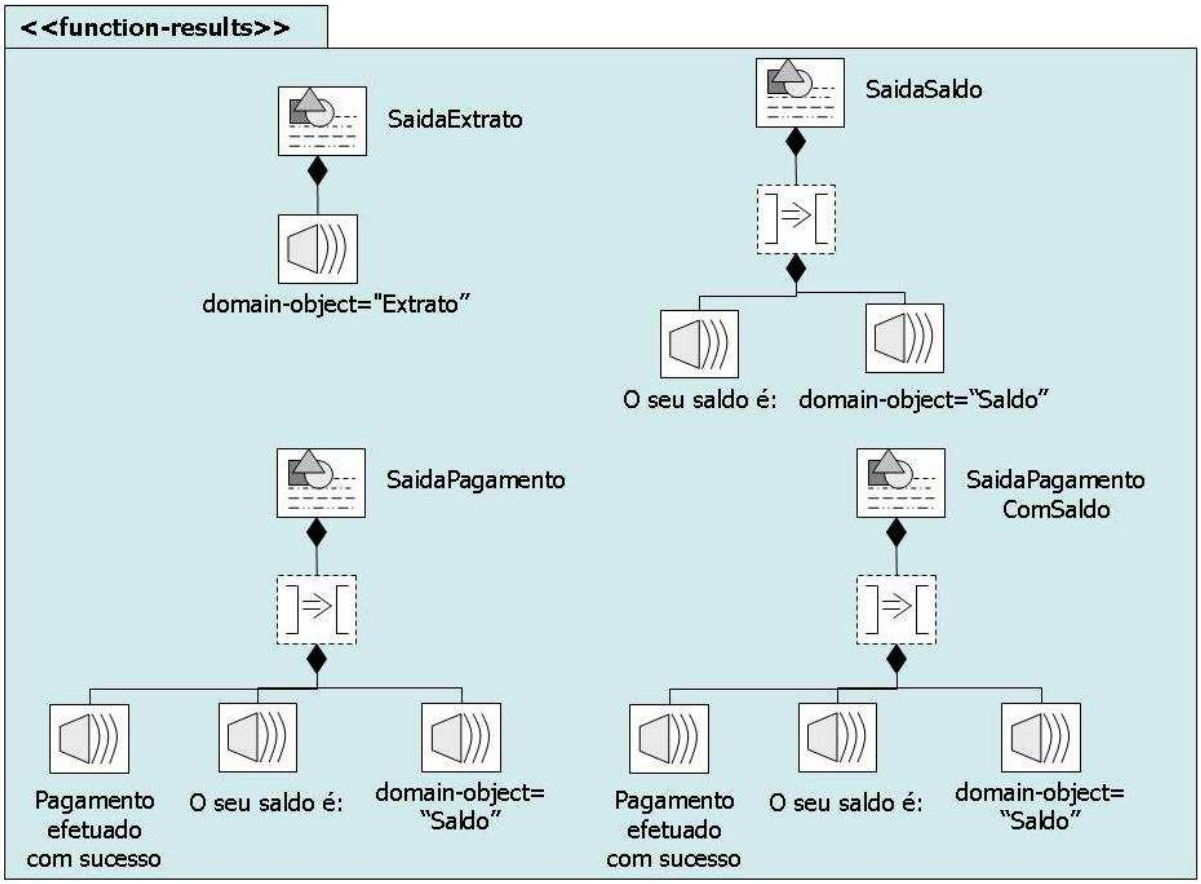



**Figura 6.18: Diagrama Visual IMML dos resultados de função do Banco Rico.**

### 6.2.3 Modelo de Comunicação

Para visualizar a especificação do modelo de comunicação da IU do Banco Rico, o código IMML será dividido pelos ambientes de tarefas existentes. Neste exemplo, em

apenas duas partes. Logo após cada figura do código IMML dos ambientes de tarefas pode-se observar o diagrama da Visual IMML correspondente.

A figura 6.19 mostra o código IMML do primeiro ambiente de tarefa que faz parte do modelo de comunicação da IU do Banco Rico. Esse ambiente é responsável pela retirada de extrato no terminal e está ligada a tarefa principal dessa IU. Ela é composta por quatro painéis de comando que estão diretamente ligadas aos comandos de função do modelo de interação, por mais quatros áreas de visualização que mostram os resultados de funções do modelo de interação e, também, por vários signos interativos e pelos elementos de visualização de objetos de domínio.

```
<task-environment name="ExtratoNoTerminal" task="Principal"
  platform="Terminal">
  <frame>
    <frame name="grupoSuperior" title="Banco Rico"
      orientation="horizontal" align="left">
      <push-button label="Saldo" show="frameSaldo" />
      <push-button label="Extrato" show="frameExtrato" />
      <push-button label="Pagamento" show="framePagamento" />
      <push-button label="Pagamento Com Saldo Antes"
        show="framePagamentoComSaldo" />
    </frame>
    <frame name="grupoInferior" />
  </frame>
  <command-panel function-command="Saldo" name="frameSaldo"
    title="Banco Rico" 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"
        transition="MostraSaldo" target="grupoInferior" />
      <push-button label="Cancelar" hide="this" />
    </group>
  </command-panel>
  <display-area function-result="SaidaSaldo" name="MostraSaldo"
    orientation="vertical" align="left">
    <text label="Data" domain-object="Data" />
    <text label="Agencia" domain-object="Agencia" />
    <text label="Conta" domain-object="Conta" />
    <text label="Saldo" domain-object="Saldo" />
    <push-button label="Sair" hide="this" />
  </display-area>
  <command-panel function-command="Extrato" name="frameExtrato"
    orientation="vertical" align="left">
    <group orientation="horizontal">
      <edit-box label="Agencia" domain-object="Agencia" />
      <edit-box label="Conta" domain-object="Conta" />
    </group>
    <edit-box label="Senha" domain-object="Senha" />
    <group orientation="vertical" align="center">
      <text>Escolha o mes que deseja ver Extrato</text>
      <drop-down-list name="Meses" label="NomeMeses" label-
          position="Center" domain-object="Mes" />
    </group>
    <group orientation="vertical" align="center">
      <text>Ou digite no campo uma data inicial e final, ou
        somente uma final</text>
      <edit-box label="DataInicial" domain-object="DataInicio" />
      <edit-box label="DataFinal" domain-object="DataFim" />
```

```
      </group>
      <group orientation="horizontal" align="center">
        <push-button label="Consultar" control="Consultar"
          transition="MostraExtrato" target="grupoInferior" />
        <push-button label="Cancelar" hide="this" />
      </group>
    </command-panel>
    <display-area function-result="SaidaExtrato"
      name="MostraExtrato" orientation="vertical" align="left">
      <table domain-object="Extrato" head-position="top">
        <column label="NomeTransacao" domain-
          object="NomeTransacao"/>
        <column label="DataTransacao" domain-
          object="DataTransacao"/>
        <column label="ValorTransação" domain-
          object="ValorTransação" />
        <column label="TipoTransacao" domain-
          object="TipoTransacao" />
      </table>
      <text label="Saldo" domain-object="Saldo" />
      <push-button label="Sair" hide="this" />
    </display-area>
    <command-panel function-command="Pagamento"
      name="framePagamento" orientation="vertical" align="left">
      <group orientation="horizontal">
        <edit-box name="Agencia" domain-object="Agencia" />
        <edit-box name="Conta" domain-object="Conta" />
      </group>
      <edit-box label="DataPagamento" domain-
        object="DataPagamento" />
      <edit-box label="Valor do Pagemento" domain-
        object="ValorPagamento" />
      <edit-box label="CodigoPagamento" domain-
        object="CodigoPagamento" />
      <edit-box name="Senha" domain-object="Senha" />
      <group orientation="horizontal" align="center">
        <push-button label="Pagar" control="Pagar"
          transition="MostraPagamento" target="grupoInferior" />
        <push-button label="Cancelar" hide="this" />
      </group>
    </command-panel>
    <display-area function-result="SaidaPagamento"
      name="MostraPagamento" orientation="vertical" align="left">
      <text label="Resultado do Pagamento" domain-
        object="EstatusPagamento" />
      <text label="Saldo" domain-object="Saldo" />
      <push-button label="Sair" hide="this" />
    </display-area>
    <command-panel function-command="PagamentoComSaldoAntes">
      <frame name="frameSaldoComSaldo" orientation="vertical"
        align="left">
        <group orientation="horizontal">
          <edit-box label="Agencia" domain-object="Agencia" />
          <edit-box label="Conta" domain-object="Conta" />
        </group>
        <edit-box label="Senha" domain-object="Senha" />
        <group orientation="horizontal" align="center">
          <push-button label="ConsultarSaldo" control="Consultar"
            transition="MostraResultado" target="grupoInferior" />
          <push-button label="Cancelar" hide="this" />
        </group>
        <text label="Saldo" domain-object="Saldo" />
        <text>Deseja Continuar?/</text>
        <push-button label="Sim" direction="go"
          transition="frameContinuarPagamento" />
        <push-button label="Nao" hide="this" />
      </frame>
      <frame name="frameContinuarPagamento" orientation="vertical"
        align="left">
```

```
        <edit-box label="DataPagamento" domain-
          object="DataPagamento" />
        <edit-box label="ValorPagemento" domain-
          object="ValorPagamento" />
        <edit-box label="CodigoPagamento" domain-
          object="CodigoPagamento" />
        <group orientation="horizontal" align="center">
          <push-button label="Pagar" control="Pagar"
            transition="MostraPagamentoComSaldo"
            target="grupoInferior" />
          <push-button label="Cancelar" hide="this" />
        </group>
      </frame>
   </command-panel>
   <display-area function-result="SaidaPagamentoComSaldo"
      name="MostraPagamentoComSaldo" orientation="vertical"
      align="left">
      <text label="Resultado do Pagamento" domain-
        object="EstatusPagamento" />
      <text label="Saldo" domain-object="Saldo" />
      <push-button label="Sair" hide="this" />
   </display-area>
</task-environment>
```

**Figura 6.19: Código IMML do primeiro ambiente de tarefas do Banco Rico.**

A partir da Figura 6.19 foram desenvolvidas, utilizando os novos estereótipos da Visual IMML, os modelos apresentados nas figuras 6.20 e 6.21. Nelas pode ser observado como fica o novo diagrama da Visual IMML que corresponde ao ambiente de tarefas passado na figura acima. Esse ambiente de tarefa foi desenvolvido para a plataforma de um terminal. Para uma melhor visualização esse diagrama foi dividido em duas partes.

A primeira parte (apresentada na Figura 6.20) mostra que o ambiente de tarefa *ExtratoNoTerminal* contém um *frame*, que é composto por outros dois frames visualizados horizontalmente; dois painéis de comando (*FrameSaldo* e *FrameExtrato*); e, por duas áreas de visualização (*MostrarSaldo* e *MostrarExtrato*).

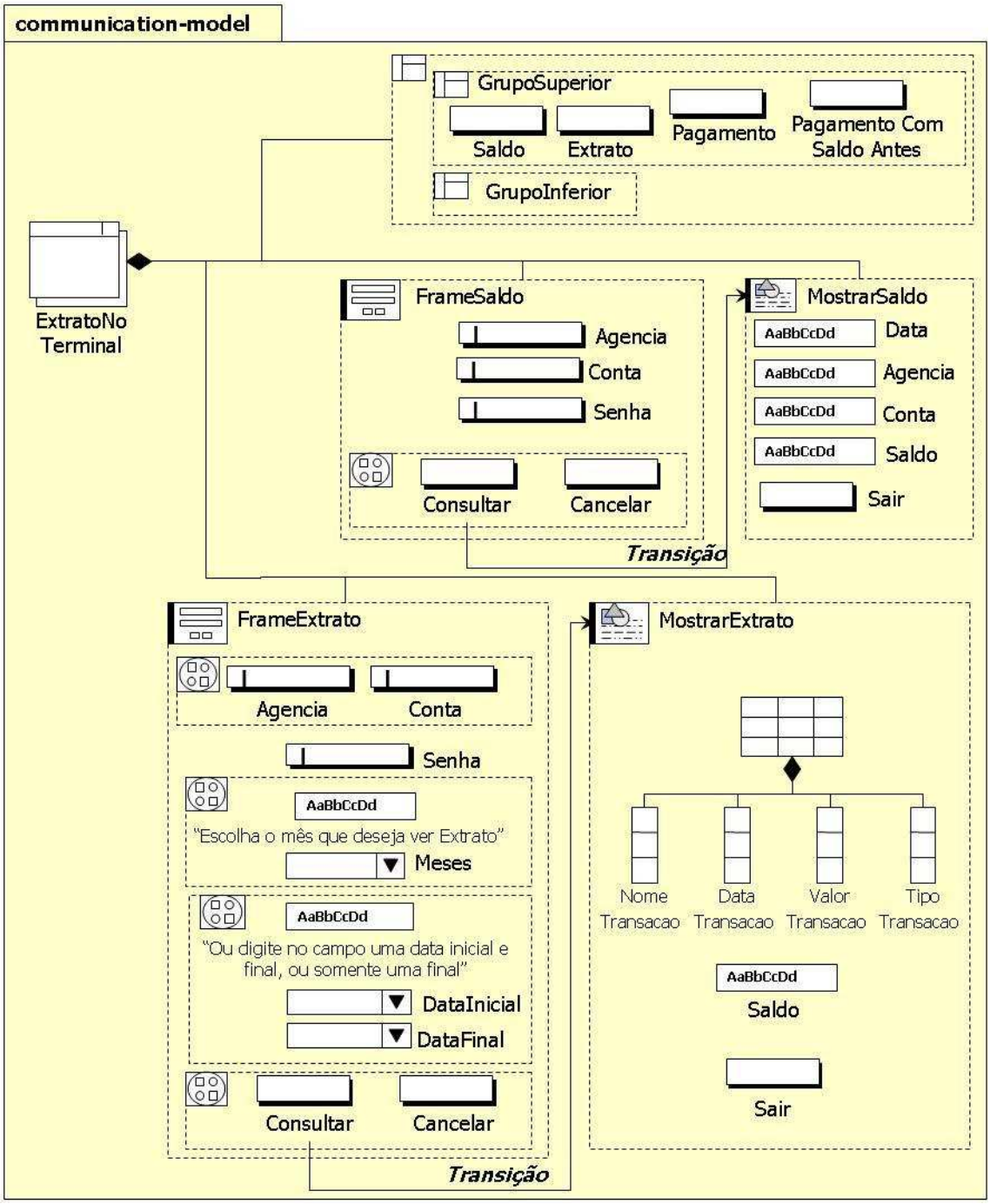



**Figura 6.20: Diagrama Visual IMML do primeiro ambiente de tarefa do Banco Rico – Parte I.**

A segunda parte (figura 6.21) mostra a continuação do ambiente de tarefa *ExtratoNoTerminal* contendo mais dois painéis de comando (*FramePagamento* e *FramePagamentoComSaldoAntes*); e, por mais duas áreas de visualização (*MostrarPagamento* e *MostrarPagamentoComSaldo*).

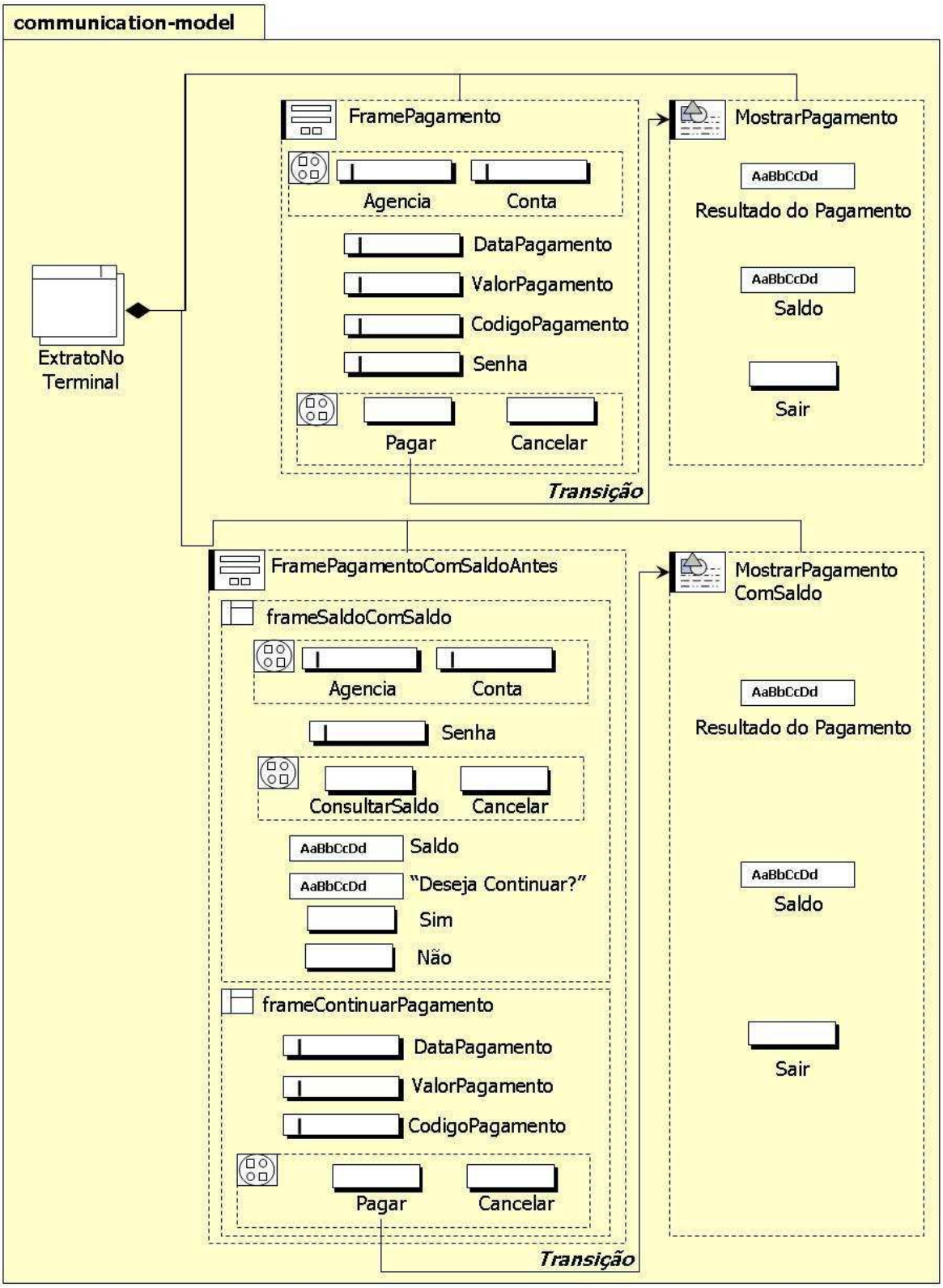



**Figura 6.21: Diagrama Visual IMML do primeiro ambiente de tarefa do Banco Rico – Parte II.**

Na figura 6.22 pode-se ver o código IMML do segundo ambiente de tarefa que faz parte do modelo de comunicação da IU do Banco Rico. Esse ambiente é responsável pelo

pagamento de uma conta simples usando a plataforma de IU de celular. Esse código é composto: (i) por um painel de comando que está ligado ao comando de função do modelo de interação; (ii) por uma área de visualização que mostra o resultado de função do modelo de interação; e, também, (iii) por vários signos interativos e elementos de visualização de objetos de domínio.

```
<task-environment name="PagamentoSimples"
   task="EfetuarPagamentoSimples" platform="celular">
   <command-panel function-command="PagamentoSimples">
      <frame name="primeiro" orientation="vertical" align="left">
         <edit-box label="Agencia" domain-object="Agencia" />
         <push-button label="Next" transition="segundo" />
      </frame>
      <frame name="segundo" orientation="vertical" align="left">
         <edit-box label="Conta" domain-object="Conta" />
         <push-button label="Next" transition="terceiro" />
      </frame>
      <frame name="terceiro" orientation="vertical" align="left">
         <edit-box label="Senha" domain-object="Senha" />
         <push-button label="Next" transition="quarto" />
      </frame>
      <frame name="quarto" orientation="vertical" align="left">
         <edit-box label="Codigo" domain-object="CodigoPagamento" />
         <push-button label="Next" transition="quinto" />
      </frame>
      <frame name="quinto" orientation="vertical" align="left">
         <edit-box label="Valor" domain-object="ValorPagamento" />
         <push-button label="Next" transition="sexto" />
      </frame>
      <frame name="sexto" orientation="vertical" align="left">
         <edit-box label="Data" domain-object="DataPagamento" />
         <push-button label="Pagar" control="Pagar"
            transition="setimo" />
      </frame>
   </command-panel>
   <display-area function-result="SaidaPagamento">
      <frame name="setimo" orientation="vertical" align="left">
         <text label="Resultado do Pagamento" domain-
            object="EstatusPagamento" />
         <text label="Saldo" domain-object="Saldo" />
         <push-button label="Sair" hide="this" />
      </frame>
   </display-area>
</task-environment>
```

**Figura 6.22: Especificação IMML do segundo ambiente de tarefas do Banco Rico.**

Por fim, na Figura 6.23 pode-se observar como fica o diagrama da Visual IMML que corresponde ao segundo ambiente de tarefa do Banco Rico, especificado em IMML na Figura 6.22. Essa figura é composta por um único painel de comando, na orientação vertical, que é composto por seis *frames*, todos na orientação vertical e, também; por uma única área de visualização, que é composta por um frame onde se pode observar o resultado da tarefa realizada pelo usuário.

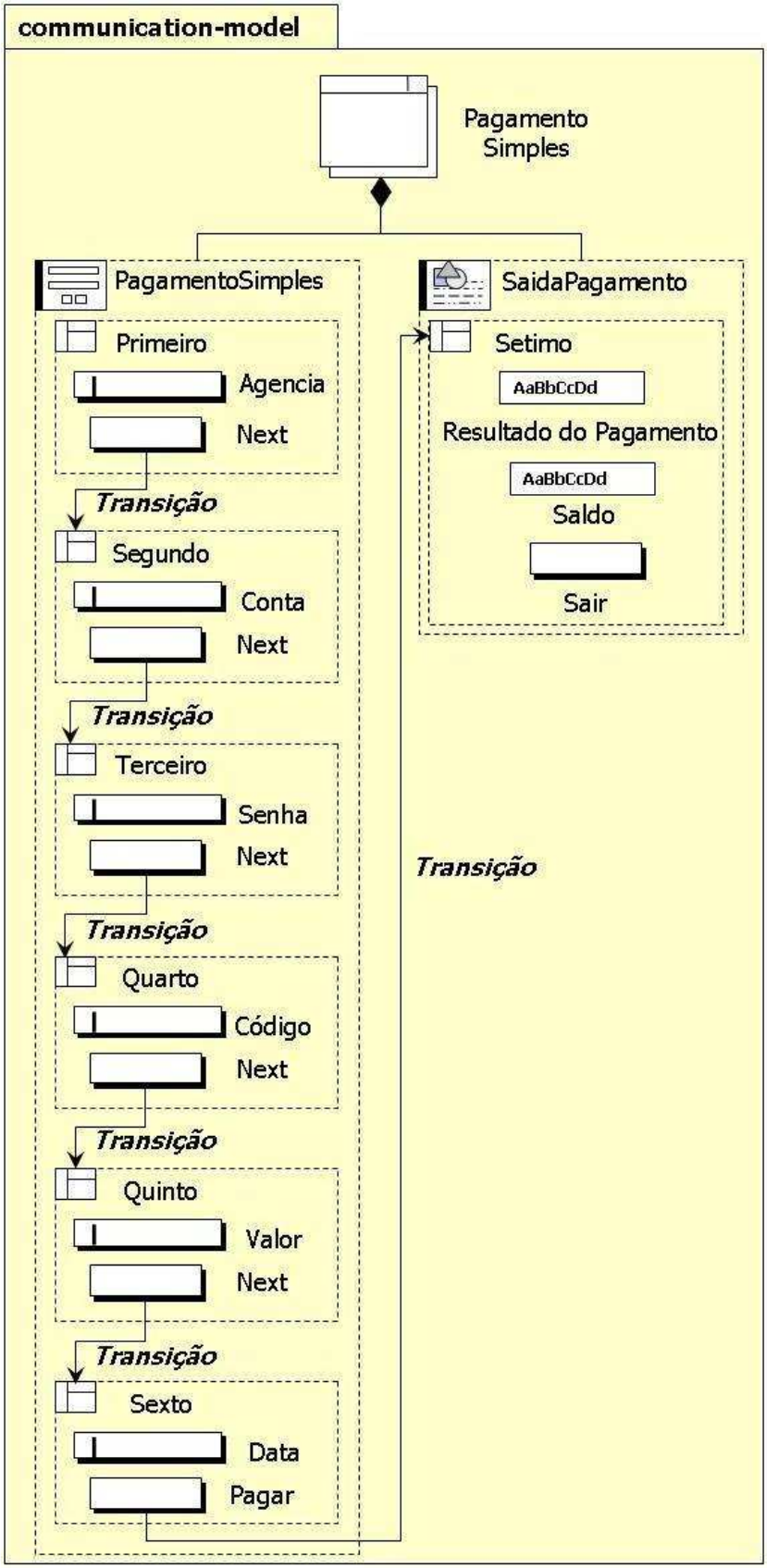



**Figura 6.23: Diagrama Visual IMML do segundo ambiente de tarefa do Banco Rico.**

# Capítulo 7

# Considerações Finais

Este trabalho apresentou a Visual IMML, um Perfil UML para a modelagem de IU. Esse perfil é uma versão visual (diagramática) da IMML. Para isso, muitas pesquisas foram realizadas e os resultados foram concretizados na escrita desta dissertação.

O primeiro capítulo apresentou as motivações, os problemas que tentamos resolver e os objetivos do trabalho. Em seguida, no segundo capítulo, apresentamos as principais fundamentações teóricas para o tema abordado neste trabalho, descrevendo a abordagem do desenvolvimento de IU baseado em modelos e a IMML, linguagem na qual este trabalho está baseado. O terceiro capítulo descreveu a UML, juntamente com os seus mecanismos de extensão para a elaboração de Perfis UML, além de mostrar o quanto essa linguagem é falha quando se diz respeito à modelagem de interfaces de usuário. No quarto capítulo, descrevemos o resultado de uma análise feita em relação a alguns trabalhos correlatos a este. No quinto capítulo, descrevemos a IMML e a Visual IMML. Já no sexto capítulo, apresentamos um estudo de caso, onde foi aplicada a modelagem de sua IU usando a IMML e a Visual IMML. Finalmente, neste sétimo e último capítulo, apresentamos algumas considerações finais acerca deste trabalho.

Como foi visto neste trabalho, a IMML possui algumas desvantagens em relação à construção de uma especificação, por ser textual. Essa dificuldade surge, pois a IMML é extensa, difícil de aprender e composta por vários comandos.

Assim, a Visual IMML é uma proposta que vem melhorar o processo de especificação da IMML através de uma modelagem visual (diagramática). Ela possui as características da IMML e incorpora a sua fundamentação teórica, baseada na Engenharia

Semiótica, focando-se principalmente nos aspectos de funcionalidade, interatividade e comunicabilidade, qualidades importantes para assegurar a usabilidade da interface de uma interface, obtendo com isso, um diferencial importante dentre as outras linguagens.

A Visual IMML foi construída de forma a manter o alto nível de abstração existente na IMML. Ela é composta por novos estereótipos para os elementos do modelo de domínio, interação e comunicação que permitem especificar a arquitetura de uma IU.

## 7.1 PRINCIPAIS CONTRIBUIÇÕES

Assim, com a realização deste trabalho pode-se observar que este Perfil UML vem trazendo grandes contribuições para a questão da modelagem de IU. Dentre essas contribuições destaca-se:

- A Visual IMML foi definida como um perfil da UML de forma a permitir uma integração com especificações UML e uma maior aceitação dos usuários, uma vez que a UML é uma linguagem padrão para modelagem de software.
- A Visual IMML define um conjunto de novos elementos adaptados para a especificação, visualização, modelagem e documentação de Interfaces de Usuário, permitindo descrever aspectos de comunicação, interação e funcionais de forma integrada, como sua principal vantagem e diferencial das outras extensões da UML para IU.
- Com a Visual IMML espera-se que o uso da IMML seja facilitado e que ela permita uma descrição de interfaces de usuários que possua as vantagens de uma linguagem de especificação com as facilidades da modelagem visual de interfaces.
- Com o uso da Visual IMML permita uma melhor comunicação entre os designers de IU e os programadores, gerando uma IU com alta qualidade.

Com isso, é importante ressaltar que tais contribuições servirão posteriormente como base para construção de novas ferramentas, além de se estabelecer como um documento que oferecerá como suporte técnico para modelagem de IU utilizando a Visual IMML.

## 7.2 LIMITAÇÕES

Esse trabalho, ainda em desenvolvimento, vem com algumas limitações, que devem ser superadas, são elas:

- A Visual IMML foi desenvolvida de forma a refletir a IMML, assim ela terá os mesmos problemas que a IMML poderá ter, pois a IMML ainda está em desenvolvimento e avaliação.
- Ainda não foram feitas avaliações do comportamento da Visual IMML para interfaces com muitas telas e/ou widgets.
- A Visual IMML, ainda não permite especificar detalhes finos do design de IUs como cores, tipos de fontes, etc.
- Ainda não foram executadas avaliações de uso da Visual IMML, especialmente sobre o quanto sua especificação é precisa e completa.
- A Visual IMML não tem por objetivo descrever protótipos da interface ou descrever telas do sistema. Seu objetivo é especificar a estrutura e comportamento.
- A descrição do comportamento de uma IU ainda está limitada às descrições de transições entre os *signos de composição* do modelo de comunicação e às descrições das estruturas de interação do modelo de interação. Ainda será avaliado se estas descrições são suficientes ou não.

A partir dessas limitações abre-se campo para estudos como perspectivas futuras da Visual IMML.

## 7.3 PERSPECTIVAS FUTURAS

Alguns ajustes na Visual IMML ainda estão sendo feitos a partir da experiência com alguns projetos. É possível que algumas alterações ainda venham a ser realizadas.

Essa linguagem deverá, futuramente, ser testada e avaliada por um grupo de designers de interfaces para obter um resultado positivo garantindo o uso dessa linguagem.

Ferramentas para a modelagem com a Visual IMML estão sendo construídas. Estas ferramentas permitem um mapeamento direto da Visual IMML com a IMML. Esta última será na verdade a forma interna de representação da ferramenta. Os elementos visuais são elaborados pelo usuário e armazenados como arquivos IMML.

# Referências


Abdullah M.S., Evans A., Benest I. & Kimble C. 2004. “Developing a UML Profile for Modelling Knowledge-Based Systems”. In Proceeding of Model-Driven Architecture: Fudations And Aplications (MDAFA'04). Pag 202-216.

Azevedo, P.; Merick, R. & Roberts, D. 2000. “OVID to AUIML - User-Oriented Interface Modelling”. TUPIS'2000, Towards a UML Profile for Interactive Systems Development. York, UK.

Booch, G; Rumbaugh, J & Jacobson, I. 2000. “UML Guia do Usuário”. Editora Campus.

Campos, J. Creissac. 2004. “The modelling gap between software engineering and human-computer interaction”. In Rick Kazman, Len Bass and Bonnie John, editor(s), ICSE 2004 Workshop: Bridging the Gaps II, pages 54-61, The IEEE, May.

Conallen, Jim. 2000. “Building Web Applications With UML”. New Jersey, Addison Wesley, June.

De Souza, C. S. 1993. “The semiotic engineering of user interface languages. In International Journal of Man-machine Studies, 39(5): 753-773.

De Souza, C.; Leite, J.; Prates, R. & Barbosa, S. 1999. “Projeto de Interfaces de Usuário: Perspectivas Cognitivas e Semióticas”. In Simpósio Brasileiro de Computação, Rio de Janeiro - RJ.

Fonseca, F.M. 2005 “Um Estudo Comparativo de Técnicas de Especificação e Modelagem de Interfaces: Medindo o Potencial da IMML”. Trabalho monográfico, UFRN.

Fuentes, L. & Vallecillo, A. 2004 “Una Introducción a los Profiles UML”. In Novática, 168:6-11, 2004

Guedes, G. 2004. “UML, Uma Abordagem Prática”. Editora Novatec.

Jardim, N. & Cunha, J.F. 2000. “Towards a UML profile for interaction design: the Wisdom approach”. In TUPIS' 2000, Towards a UML Profile for Interactive Systems Development. York, UK.

Jardim, N. 2002. “Modelação por Objetos para o desenvolvimento Centrado nos utilizadores e o desenho de Interfaces com o Utilizador” – Resumo extendido, Uma 2002.

Leite, J.C. 2003. “Specifying the User Interfaces as an Interactive Message”. In Proceedings of HCI International, vol 2, p. 716-720, June.

Leite, J.C. 2005. “Improving usability by communicating functional and task/interaction models though the user interface”. Proceedings of HCI International. (CD-Rom).

OMG UML 2.0 Superstructure Final Adopted Specification - (OMG document ptc/03-08-02) Cap 18, – disponível em: http://www.uml.org/

OMG UML 2.0 Infrastructure Final Adopted Specification - (OMG document ptc/03-08-02), disponível em: http://www.uml.org/

Puerta, A & Eisenstein, J. 1998. “Towards a general computational *framework* for model-based interface development systems”. In Proceedings of the 4th international conference on Intelligent user interfaces. December.

Phillips, C. & Kemp, E. 2002. “In Support of User Interface Design in the Rational Unified Process”. In Proc. Third Australasian User Interface Conference (AUIC2002), Melbourne, Australia. Conferences in Research and Practice in Information Technology, 7. Grundy, J. and Calder, P., Eds., ACS. 21-27.

Silva, P.P. & Paton, N. 2000. “User Interface Modelling with UML”. In Proceedings of the 10th European-Japanese Conference on Information Modelling and Knowledge Representation, Saariselka, Finland, May.

Silva, P.P. & Paton, N. 2000. “UMLi: The Unified Modeling Language for Interactive Applications”. In UML 2000 - The Unified Modeling Language: Advancing the Standard (3rd International Conference on the Unified Modeling Language, York, United Kingdom,

October, 2-6, 2000), A. Evans, S. Kent and B. Selic (Eds.). LNCS Vol 1939, pages 117-132, Springer.

Silva, P.P. 2000. “UMLi: Integrating User Interface and Application Design”. In TUPIS' 2000, Towards a UML Profile for Interactive Systems Development. York, UK.

Silva, P.P. & Paton, N. 2003. “Improving UML Support for User Interface Design: A Metric Assessment of UMLi”. In Proceedings of ICSE-2003 Workshop on Bridging the Gaps Between Software Engineering and Human-Computer Interaction, R. Kazman, L. Bass and J. Bosch (Eds.), Portland, OR, USA. IFIP, pages 76-83.

Stary, C. 2000 “Contextual prototyping of user interfaces”. In Proceedings of the conference on Designing interactive systems: processes, practices, methods, and techniques. August.

Stoiber, S & Stary, C. 2000. “UML-Support for Model- and Task-Based Development of Interactive Software Systems”. In TUPIS' 2000, Towards a UML Profile for Interactive Systems Development. York, UK.

Sousa, K.S. & Furtado, E. 2003. “RUPi – A Unified Process that Integrates Human-Computer Interaction and Software Engineering”. In Proceedings of Bridging the Gaps Between Software Engineering and Human-Computer Interaction at ICSE 03, Portland, OR. pp. 41-48.

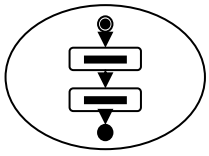
# Apêndice A – Estereótipos da Visual IMML

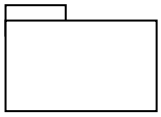
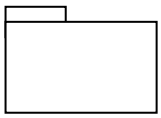
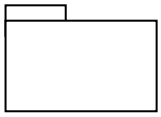
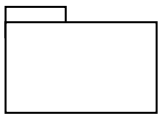
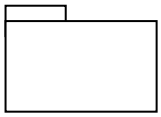
Este apêndice se concentra em apenas mostrar, resumidamente e de forma clara, os novos estereótipos propostos pela Visual IMML. Segue abaixo Tabelas que estarão divididas no nome do estereótipo, no tipo de classe do metamodelo que o estereótipo assume na sua descrição e em seu ícone. Para uma melhor visualização, vamos dividir nos três modelos que compõe a Visual IMML.

## A.1 MODELO DE DOMÍNIO

Esse modelo é, basicamente, formado pela união dos diagramas da UML padrão, porém ele propõe um novo estereótipo, que é (Tabela A.1):

**Tabela A.1: Estereótipos do Modelo de Domínio da Visual IMML.**

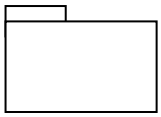
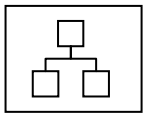
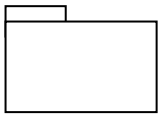
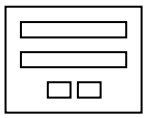
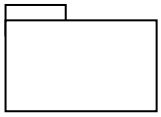
| **Caso de Uso da Função de Domínio** | |
|---|---|
| Classe do Metamodelo | Caso de Uso |
| Descrição | Estereótipo usado para representar as funções de domínio do modelo de domínio. |
| Ícone |  |
| Restrição | - |
| Valor Atribuído | - |
| **Domain-Objects** | |
| Classe do Metamodelo | Pacote |
| Descrição | Estereótipo usado como repositório e para organizar os objetos de domínio do modelo de domínio. |

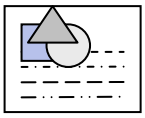
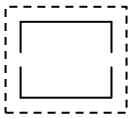
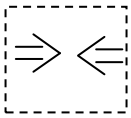
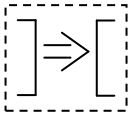
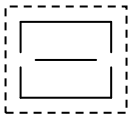
| | |
|---|---|
| Ícone |  |
| Restrição | Deverá conter pelo menos um objeto de domínio. |
| Valor Atribuído | - |
| **Domain-Functions** | |
| Classe do Metamodelo | Pacote |
| Descrição | Estereótipo usado como repositório e para organizar as funções de domínio do modelo de domínio. |
| Ícone |  |
| Restrição | Deverá conter pelo menos uma função de domínio. |
| Valor Atribuído | - |
| **Controls** | |
| Classe do Metamodelo | Pacote |
| Descrição | Estereótipo usado como repositório para os controles do modelo de domínio. |
| Ícone |  |
| Restrição | Deverá conter pelo menos um controle. |
| Valor Atribuído | - |
| **States** | |
| Classe do Metamodelo | Pacote |
| Descrição | Estereótipo usado como repositório para os estados do modelo de domínio. |
| Ícone |  |
| Restrição | Deverá conter pelo menos dois estados. |
| Valor Atribuído | - |
| **Operands** | |
| Classe do Metamodelo | Pacote |
| Descrição | Estereótipo usado como repositório para os operandos do modelo de domínio. |
| Ícone |  |
| Restrição | - |
| Valor Atribuído | - |

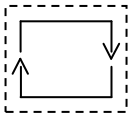
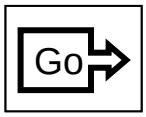
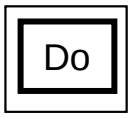
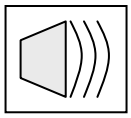
## A.2 MODELO DE INTERAÇÃO

Esse modelo é totalmente composto por novos estereótipos, são eles (Tabela A.2):

## Tabela A.2: Estereótipos do Modelo de Interação da Visual IMML.

| **Tasks** | |
|---|---|
| Classe do Metamodelo | Pacote |
| Descrição | Estereótipo usado como repositório para as tarefas do modelo de interação |
| Ícone |  |
| Restrição | Deverá possuir pelo menos uma tarefa. |
| Valor Atribuído | - |
| **Task** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*task*> é usado para especificar uma tarefa. |
| Ícone |  |
| Restrição | Deverá possuir pelo menos uma estrutura de interação ou uma interação básica. |
| Valor Atribuído | - |
| **Function-Commands** | |
| Classe do Metamodelo | Pacote |
| Descrição | Estereótipo usado como repositório para os comandos de função do modelo de interação. |
| Ícone |  |
| Restrição | Deve conter pelo menos um comando de função. |
| Valor Atribuído | - |
| **Function-Command** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*function-command*> é usado para especificar um comando de função. |
| Ícone |  |
| Restrição | Deverá possuir pelo menos uma estrutura de interação ou uma interação básica. |
| Valor Atribuído | - |
| **Function-Results** | |
| Classe do Metamodelo | Pacote |
| Descrição | Estereótipo usado como repositório para os resultados de função do modelo de interação. |
| Ícone |  |
| Restrição | Deve conter pelo menos um resultado de função. |
| Valor Atribuído | - |
| **Function-Result** | |
| Classe do Metamodelo | Classe |

| | |
|---|---|
| Descrição | O elemento <*function-result*> é usado para especificar um resultado de função. |
| Ícone |  |
| Restrição | Deverá possuir pelo menos uma estrutura de interação ou uma interação básica. |
| Valor Atribuído | - |
| **Join** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*join*> é usado para especificar uma estrutura de interação livre. |
| Ícone |  |
| Restrição | Ele deverá possuir pelo menos uma interação básica, ou recursivamente, outra estrutura de interação. |
| Valor Atribuído | - |
| **Combine** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*combine*> é usado para especificar uma estrutura de interação que deverá ser executada de forma combinada. |
| Ícone |  |
| Restrição | Ele deverá possuir pelo menos uma interação básica, ou recursivamente, outra estrutura de interação. |
| Valor Atribuído | - |
| **Sequence** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*sequence*> é usado para especificar uma estrutura de interação que deverá ser executada de forma seqüencial. |
| Ícone |  |
| Restrição | Ele deverá possuir pelo menos uma interação básica, ou recursivamente, outra estrutura de interação. |
| Valor Atribuído | - |
| **Select** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*select*> é usado para especificar uma estrutura de interação que deverá ser executada de forma seletiva. |
| Ícone |  |
| Restrição | Ele deverá possuir pelo menos uma interação básica, ou recursivamente, outra estrutura de interação. |
| Valor Atribuído | - |
| **Repeat** | |

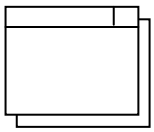
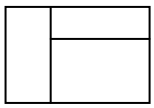
| | |
|---|---|
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*repeat*> é usado para especificar uma estrutura de interação que deverá ser executada de forma repetitiva |
| Ícone |  |
| Restrição | Ele deverá possuir pelo menos uma interação básica, ou recursivamente, outra estrutura de interação. |
| Valor Atribuído | - |
| **Go** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*go*> é usado para especificar uma interação básica de navegação entre tarefas e/ou comandos/resultados de função. |
| Ícone |   |
| Restrição | - |
| Valor Atribuído | Recebe três tipos de direcionamento: *away, ahead e back* |
| **Do** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <do> é usado para especificar a realização de uma tarefa, um comando de função ou um resultado de função. |
| Ícone |   |
| Restrição | Deverá ser usado apenas em tarefas para indicar a realização de comandos/resultados de função e/ou outras tarefas, dentro da tarefa em questão. |
| Valor Atribuído | - |
| **Perceive-Information** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*perceive-information*> é usado para especificar uma interação básica de percepção de informação. |
| Ícone |  |
| Restrição | Ele poderá possuir apenas texto para a percepção direta e estática de uma informação. |
| Valor Atribuído | Dessa forma, seu atributo é opcional. |
| **Enter-Information** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*enter-information*> é usado para especificar uma interação básica de entrada de informação. |
| Ícone |  |
| Restrição | -. |
| Valor Atribuído | - |
| **Select-Information** | |

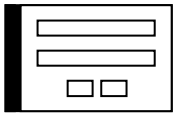
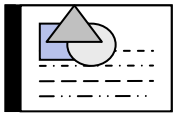
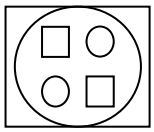
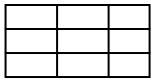
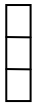
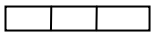
| | |
|---|---|
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*select-information*> é usado para especificar uma interação básica de seleção de informação. |
| Ícone |  |
| Restrição | - |
| Valor Atribuído | - |
| **Activate** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*activate*> é usado para especificar uma interação básica de ativação de um controle. |
| Ícone |  |
| Restrição | - |
| Valor Atribuído | - |

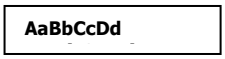
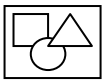
## A.3 MODELO DE COMUNICAÇÃO

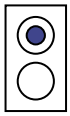
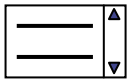
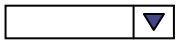
Juntamente com o modelo de interação, o modelo de comunicação é composto, totalmente, por novos estereótipos, são eles (Tabela A.3):

**Tabela A.3: Estereótipos do Modelo de Comunicação da Visual IMML.**

| | |
|---|---|
| **Task-Environment** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*task-environment*> é usado para especificar um ambiente de realização de uma tarefa. |
| Ícone |  |
| Restrição | Ele deverá está associado à tarefa para a qual ele permitirá sua realização e à plataforma que o suportará. |
| Valor Atribuído | - |
| **Frame** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*frame*> é usado para especificar um quadro que será apresentando ao usuário num dado momento de interação. |
| Ícone |  |
| Restrição | - |
| Valor Atribuído | - |
| **Command-Panel** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*command-panel*> é usado para especificar uma composição de signos interativos que irão comunicar as interações básicas de um comando de função. |

| | |
|---|---|
| Ícone |  |
| Restrição | - |
| Valor Atribuído | - |
| **Display-Area** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*display-area*> é usado para especificar uma composição de signos interativos que irão comunicar as interações básicas de um resultado de função. |
| Ícone |  |
| Restrição | - |
| Valor Atribuído | - |
| **Group** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*group*> é usado para especificar uma composição de signos interativos, comunicando a existência de alguma relação entre eles. |
| Ícone |  |
| Restrição | - |
| Valor Atribuído | - |
| **Table** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*table*> é usado para especificar uma composição de signos interativos, que deverá ser apresentada de maneira tabular. |
| Ícone |  |
| Restrição | - |
| Valor Atribuído | As colunas contidas na tabela. |
| **Columns** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*columns*> é usado para especificar quais serão as colunas de uma tabela. |
| Ícone |  |
| Restrição | - |
| Valor Atribuído | - |
| **List** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*list*> é usado para especificar uma composição de signos interativos, que deverá ser apresentada de maneira sequencialmente. |
| Ícone |  |

| | |
|---|---|
| Restrição | - |
| Valor Atribuído | - |
| **Text** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*text*> é usado para especificar que o signo interativo permitirá que o usuário perceba um texto na interface.. |
| Ícone |   |
| Restrição | Poderá está associado um objeto de domínio ou ter seu conteúdo explicitamente definido pelo designer. |
| Valor Atribuído | - |
| **Image** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*image*> é usado para especificar que o signo interativo permitirá que o usuário perceba um conteúdo gráfico ou imagem na interface. |
| Ícone |  |
| Restrição | Sempre está associado a um objeto de domínio. |
| Valor Atribuído | - |
| **Edit-Box** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*edit-box*> é usado para especificar que o signo interativo permitirá ao usuário perceber apenas uma linha (pequena quantidade) de texto e edita-lo. |
| Ícone |  |
| Restrição | Sempre está associado a um objeto de domínio. |
| Valor Atribuído | - |
| **Text-Area** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*text-area*> é usado para especificar que o signo interativo permitirá ao usuário perceber várias linhas (grande quantidade) de texto e edita-lo. |
| Ícone |   |
| Restrição | Sempre está associado a um objeto de domínio. |
| Valor Atribuído | - |
| **Check-Box** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*check-box*> é usado para especificar que o signo interativo permitirá ao o usuário perceber, marcar e desmarcar a seleção de um objeto de domínio. Quando o objeto de domínio for um conjunto finito, a seleção dos itens será múltipla. |
| Ícone |  |
| Restrição | Sempre está associado a um objeto de domínio. |
| Valor Atribuído | - |

| **Radio-Button** | |
|---|---|
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*radio-button*> é usado para especificar que o signo interativo permitirá ao usuário perceber, marcar e desmarcar a seleção de um objeto de domínio. Quando o objeto de domínio for um conjunto finito, a seleção dos itens será exclusiva. |
| Ícone |  |
| Restrição | Sempre está associado a um objeto de domínio. |
| Valor Atribuído | - |
| **List-Box** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <list-box> é usado para especificar que o signo interativo permitirá que ao usuário perceber e selecionar múltiplos itens de uma lista de opções. |
| Ícone |  |
| Restrição | Sempre está associado a um objeto de domínio. |
| Valor Atribuído | - |
| **Drop-Down-List** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*drop-down-list*> é usado para especificar que o signo interativo permitirá ao usuário perceber e selecionar exclusivamente um ítem de uma lista de opções. |
| Ícone |  |
| Restrição | Sempre está associado a um objeto de domínio. |
| Valor Atribuído | - |
| **Push-Button** | |
| Classe do Metamodelo | Classe |
| Descrição | O elemento <*push-button*> é usado para especificar que o signo interativo permitirá ao usuário perceber e ativar alguma ação, que poderá ser uma transição para o próximo quadro e/ou ativação do controle de alguma função. |
| Ícone |  |
| Restrição | Poderá ter um controle associado. |
| Valor Atribuído | - |

# Livros Grátis

( http://www.livrosgratis.com.br )

Milhares de Livros para Download:

Baixar livros de Administração
Baixar livros de Agronomia
Baixar livros de Arquitetura
Baixar livros de Artes
Baixar livros de Astronomia
Baixar livros de Biologia Geral
Baixar livros de Ciência da Computação
Baixar livros de Ciência da Informação
Baixar livros de Ciência Política
Baixar livros de Ciências da Saúde
Baixar livros de Comunicação
Baixar livros do Conselho Nacional de Educação - CNE
Baixar livros de Defesa civil
Baixar livros de Direito
Baixar livros de Direitos humanos
Baixar livros de Economia
Baixar livros de Economia Doméstica
Baixar livros de Educação
Baixar livros de Educação - Trânsito
Baixar livros de Educação Física
Baixar livros de Engenharia Aeroespacial
Baixar livros de Farmácia
Baixar livros de Filosofia
Baixar livros de Física
Baixar livros de Geociências
Baixar livros de Geografia
Baixar livros de História
Baixar livros de Línguas