CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

PROFILE EM UML PARA MODELAGEM SIMPLIFICADA DE INTERFACES GRÁFICAS EM APLICATIVOS

ANDRÉ SANDRI

CANOAS, 2006.

ANDRÉ SANDRI

PROFILE EM UML PARA MODELAGEM SIMPLIFICADA DE INTERFACES GRÁFICAS EM APLICATIVOS

Trabalho de conclusão apresentado à banca examinadora do curso de Ciência da Computação do UNILASALLE - Centro Universitário La Salle, como exigência parcial para obtenção do grau de Bacharel em Ciência da Computação, sob orientação do Prof. Me. Carlos Michel Betemps.

CANOAS, 2006.

TERMO DE APROVAÇÃO

ANDRÉ SANDRI

PROFILE EM UML PARA MODELAGEM SIMPLIFICADA DE INTERFACES GRÁFICAS EM APLICATIVOS

Trabalho de conclusão aprovado como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação do Centro Universitário La Salle - UNILASALLE, pela seguinte banca examinadora:

Prof. Me. Javier García Lopez
UNILASALLE

Prof. Me. Mozart Lemos de Siqueira
UNILASALLE

Prof. Me. Abraham Lincoln Rabelo de Souza
UNILASALLE

Canoas, 30 de junho de 2006.

AGRADECIMENTOS

Aos meus pais Sírio e Lourdes pelo amor, carinho e dedicação, e por me ensinarem os princípios que hoje norteiam a minha vida.

À amada Carina, minha namorada, pelo amor e companheirismo durante todo o desenvolvimento deste trabalho.

Ao professor Carlos Michel Betemps pela orientação, paciência, colaboração e por sempre acreditar que faríamos um bom trabalho.

Aos revisores, avaliadores e amigos, que colaboraram para a melhoria deste documento.

Muito obrigado a todos!

RESUMO

Este documento demonstra a especificação de um *profile* UML para a modelagem de interfaces gráficas com foco na apresentação de *widgets* sob o ponto de vista da usabilidade para ambientes de desenvolvimento de *software* que utilizam MDD, mais especificamente para MDA. Para avaliação do modelo, o *profile* foi implementado e testado em cima de duas ferramentas de modelagem, *Borland Together Architect* e *Sparx Systems Enterprise Architect*. Além disso, descreve os principais conceitos, problemas e justifica a solução apresentada.

Palavras-chaves: UML, Profile, GUI, Interface Gráfica, MDA, MDD, MDE, Usabilidade, OCL.

ABSTRACT

This document demonstrates the specification of a UML profile for graphical interfaces modeling with focus in the presentation of widgets under the point of view of the usability for software development environments that use MDD, specifically for MDA. To validate the model, the profile was implemented and tested in top of two modeling tools, Borland Together Architect and Sparx Systems Enterprise Architect. Moreover, it describes the main concepts, problems and justifies the presented solution.

Keywords: UML, Profile, GUI, Graphic Interface, MDA, MDD, MDE, Usability, OCL.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

CORBA	*Common Object Request Broker Architeture*
DCOM	*Distributed Component Object Model*
DI	*Diagram Interchange*
DSL	*Domain-Specific Languages*
ECO	*Enterprise Core Objects*
EJB	*Enterprise Java Beans*
EMF	*Enhanced Metafile*
GUI	*Graphical User Interface*
IUI	*Inductive User Interface*
MDA	*Model Driven Architecture*
MDD	*Model Driven Development*
MDE	*Model Driven Engineering*
MDI	*Multiple Document Interface*
MOF	*Meta-object Facility*
OCL	*Object Construct Language*
OMG	*Object Management Group*
PIM	*Platform-independent Model*
PSM	*Platform-specific Model*
QVT	*Query, Views, and Transformations*
RIA	*Rich Internet Applications*
SOA	*Service-Oriented Architecture*
SVG	*Scalable Vectorial Graphics*
UI	*User Interface*
UML	*Unified Modeling Language*
W3C	*World Wide Web Consortium*
WMF	*Windows Metafile Format*
XMI	*XML Metadada Interchange*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Durante uma pesquisa realizada na disciplina de Pesquisa em Ciência da Computação (SANDRI, 2005), foi constatado que não existe um modelo padrão para a modelagem de interfaces gráficas. Por não existir um padrão, alguns fabricantes de ferramentas para MDD – *Model Driven Development* (MDD é apresentado na seção 2.2) e MDA – *Model Driven Architecture* (seção 2.3) criaram suas próprias soluções para a especificação da camada de apresentação, enquanto que outros não disponibilizaram nenhum recurso para este fim.

É comum ver que a maioria dos sistemas modelados em UML não contempla a interface gráfica, somente utilizam esta linguagem para definir a camada de negócio, a camada de persistência, entre outros. A camada da interface, nestes casos, é modelada com técnicas proprietárias e finalmente construída em cima de uma ferramenta de programação.

Existem algumas propostas de *profiles* em UML para esta finalidade (o conceito de *profile* é apresentado na seção 2.3.2.1, e algumas destas propostas são apresentadas no capítulo 3), porém são propostas que não utilizaram como abordagem principal os conceitos de usabilidade, além de algumas tentarem englobar o máximo de características possíveis, tornando difícil sua compreensão e rápida utilização.

Este trabalho de conclusão do curso de bacharelado em Ciência da Computação irá sugerir um recurso para possibilitar a modelagem de interfaces gráficas de forma simplificada com ênfase em características de usabilidade para utilização em projetos de sistemas empresariais, podendo ser estendido para outras abordagens de desenvolvimento de *software*.

1.1 Motivação e Justificava

A relevância deste assunto afeta o futuro e o provável sucesso da arquitetura MDA, proposta pela OMG – *Object Management Group* (OMG, 2006), pois o modelo aqui proposto poderá servir como inspiração ou referência para a conscientização e a futura padronização de um diagrama específico para modelagem de interface gráfica para a camada PIM – *Platform-independent Model*, pois até o momento é um problema em aberto nesta área. Todos os conceitos, como MDA, OMG e PIM, serão apresentados no capítulo 2.

O *profile* UML apresentado neste documento poderá também ser aproveitado imediatamente como recurso para equipes de desenvolvimento de *softwares* empresariais que utilizam abordagens MDD.

1.2 Objetivos Gerais

O objetivo principal deste trabalho é demonstrar a criação de um *profile* UML para possibilitar a modelagem de interfaces gráficas de forma simplificada com ênfase em características de usabilidade para utilização em projetos de sistemas empresariais, podendo ser estendido para outras abordagens de desenvolvimento de *software*.

Enfim, refinando o objetivo de forma mais específica, pode-se definir que o objetivo deste trabalho é criar um *profile* UML para possibilitar a modelagem de interfaces gráficas com foco na apresentação de *widgets* sob o ponto de vista da qualidade e da usabilidade para ambientes de desenvolvimento de *software* que utilizam MDD.

1.3 Metodologia de Desenvolvimento

Este trabalho de pesquisa possui caráter empírico, pois se baseia em pesquisa e relatos de implementações realizadas em diversas organizações e, em constatações oriundas dos estudos realizados. Além disso, também utiliza um conhecimento que é caracterizado como senso comum que, neste caso, são certas funcionalidades ou requisitos que um recurso deste tipo deve possuir.

Como base teórica para o desenvolvimento deste trabalho foram utilizados livros e artigos científicos, além de toda a documentação publicada pela OMG (OMG, 2006) e por

outras instituições que visam atingir excelência no desenvolvimento de *software* a partir de métodos e técnicas consagrados.

Este estudo contemplará o modelo do *profile* em UML, apresentando o problema e a solução, com suas justificativas, considerações e conclusões. Neste trabalho serão também apresentados os conceitos e termos pertinentes a este propósito.

A análise e o projeto do modelo foram feitos utilizando o paradigma de orientação a objetos e usou-se a linguagem UML para a construção dos diagramas.

Para avaliar o modelo proposto, foram utilizados os recursos oferecidos por duas ferramentas de modelagem que aceitam a criação e a utilização de *profiles* UML: o *Together Architect* 2006 (BORLAND, 2006) e o *Enterprise Architect* 6.1 (SPARX SYSTEMS, 2006).

1.4 Organização do trabalho

O presente trabalho está estruturado da seguinte forma: inicialmente é apresentado o referencial teórico necessário para o entendimento e contextualização do trabalho, portanto o capítulo 2 descreve os principais conceitos envolvidos no objetivo deste trabalho; o capítulo 3 demonstra o problema detectado e as abordagens existentes; o capítulo 4 apresenta a solução proposta e, abrange desde os objetivos e requisitos desejados até o modelo especificado, passando pela análise, modelagem e demonstrando alguns exemplos de utilização; o capítulo 5 apresenta a avaliação do modelo com uso de duas ferramentas de modelagem; e por fim, o capítulo 6 encerra com as conclusões e considerações finais sobre os estudos realizados.

2 PRINCIPAIS CONCEITOS

O modelo apresentado aqui poderá ser utilizado na camada PIM da arquitetura MDA, publicada pela OMG, além de poder ser utilizada também em outras abordagens MDE, mais especificamente em abordagens MDD. Como existem muitos conceitos e tecnologias envolvidos nesta proposta, este capítulo foi redigido para mostrar de forma sucinta os principais conceitos que serão utilizados ao longo deste documento.

A seguir, na seção 2.1 deste capítulo, é descrito o que é MDE e, logo após, é apresentada outras terminologias como MDD e MDA. No final deste capítulo, na seção 2.4, foram destacados os conceitos de usabilidade utilizados na solução proposta.

2.1 MDE – *Model Driven Engineering*

MDE, engenharia baseada em modelo, é uma técnica emergente de engenharia de *software* baseada no uso sistemático de modelos. Nesta abordagem, busca-se geralmente manter o foco do projeto e do desenvolvimento no modelo (diagramas), e não centrada no código-fonte, comum em abordagens tradicionais.

As principais abordagens existentes até o momento são:

- MIC - *Model-Integrated Computing*, que vem sido desenvolvido há uma década no ISIS (*Vanderbilt University*), com foco na construção de sistemas embarcados, e atualmente está em fase de padronização pela OMG (OMG, 2006);

- *Software Factories Initiative* da Microsoft, que oferece ferramentas DSL - *Domain-Specific Language*, tecnologia proprietária recente que utiliza a plataforma .NET com *Visual Studio* 2005;

- MDA - *Model-driven Architecture*, padronizada pela OMG, arquitetura que será apresentada nas próximas seções.

Tecnologias MDE oferecem uma abordagem promissora para tratar a incapacidade das linguagens de terceira geração de diminuir a complexidade das diferentes plataformas, buscando expressar de forma eficaz os conceitos do domínio do problema (SCHMIDT, 2006).

2.2 MDD – *Model Driven Development*

MDD está cada vez mais ganhando a atenção da indústria e de comunidades de pesquisa. MDD visa utilizar modelos na maior parte do tempo durante um processo de desenvolvimento de *software*, e prevê automação através da execução de modelos, transformações e técnicas de geração de código (KLEPPE, WARMER, BAST, 2003).

A abordagem MDD promete aumentar a produtividade daqueles que tem a tarefa de desenvolver e manter um sistema de *software* (SEARCHWEBSERVICES, 2004). Se um maior índice de produtividade for desejado durante a manutenção de sistemas, uma pesquisa da Compuware indica que MDD é a resposta (COMPUWARE, 2005). Resultados desta pesquisa indicam ganhos de 70% durante a fase de manutenção do *software*. A equipe que utilizou MDD completou cinco recursos 37% mais rápido do que a equipe tradicional, em 165 horas contra 260 horas.

Um dos projetos do Eclipse, o projeto MDDi - *Model Driven Development Integration*, que está em sua fase inicial de desenvolvimento, é dedicado a fortalecer a sua plataforma de desenvolvimento em Java oferecendo tecnologias de integração necessárias para a aplicação da abordagem MDD. Este projeto irá produzir *frameworks*[1] extensíveis e ferramentas para suportar várias linguagens de modelagem (UML, *Domain-Specific Languages*) e

[1] *Framework*: No desenvolvimento de software, um framework é uma estrutura de suporte definida em que um outro projeto do software pode ser organizado e desenvolvido. Tipicamente, um *framework* pode incluir programas de apoio, bibliotecas de código, linguagens de script e outros softwares para ajudar a desenvolver e juntar diferentes componentes do seu projeto.

metodologias. Na prática, é um esforço de criar uma alternativa totalmente gratuita com código aberto que suporte MDD com MDA (MDDI, 2005).

2.3 MDA – *Model Driven Architecture*

O OMG é uma associação de várias empresas que se uniram para criar padrões visando facilitar a integração de recursos de TI. Inicialmente, quando o OMG iniciou suas atividades em 1989, o foco era desenvolver padrões de *middleware*[2] orientados a objeto. O resultado disso foi a criação do CORBA, uma tecnologia que permitiu que as empresas conectassem toda e qualquer aplicação com qualquer outra, escrito em qualquer linguagem orientada a objetos, indiferente de plataforma ou sistema operacional. Com o passar dos anos o OMG trabalhou com seus membros para criar uma variedade de outros padrões, incluindo o UML - *Unified Modeling Language*, a notação padrão hoje utilizada por desenvolvedores de *software* no mundo todo (OMG, 2006).

Em 2001, após uma revisão do estado da arte das tecnologias, o OMG concluiu que a heterogeneidade de tecnologias existentes no mercado era permanente. Conforme o tempo passa, cada grande empresa é forçada a suportar novas linguagens de programação, outros sistemas operacionais, além de uma variedade de protocolos de rede. Então, decidiram que o principal objetivo, a interoperabilidade, poderia somente ser conseguido com um sistema padronizado e público de modelos e interfaces que são independentes de linguagem, sistema ou protocolo.

MDA é este sistema. Essencialmente, o OMG está padronizando diversos modelos abstratos para que as empresas usem para representar aplicações. Alguns destes padrões resultam em modelos que são independentes dos detalhes da implementação. Outros padrões complementam as características existentes em linguagens de programação, de sistemas operacionais ou de redes. Todos os modelos que uma empresa define podem ser integrados por interfaces e mapeamentos que asseguram que uma empresa possa inicialmente desenvolver um modelo independente de plataforma de uma aplicação, e, em seguida, possa gerar um modelo específico para uma linguagem e um sistema operacional escolhido.

[2] *Middleware*: Software de interface que permite interação de diferentes aplicações de softwares, geralmente sobre diferentes plataformas de hardware e infra-estrutura, para troca de dados.

Futuramente, caso a empresa decidir executar a aplicação em uma linguagem ou sistema operacional diferente, basta reutilizar o modelo independente de plataforma para gerar um modelo novo, específico para a aplicação nova. Assim, enquanto que as empresas evoluem e novas tecnologias são introduzidas, as aplicações, documentadas como modelos independentes da plataforma, serão preservadas e podem ser reutilizadas (OMG, 2006).

Os membros do OMG acreditam que esta solução possa resultar em uma arquitetura que sirva a uma empresa por uma década, independendo das mudanças nas linguagens de programação, dos sistemas operacionais e dos protocolos de rede.

A solução MDA engloba diversos padrões e conceitos que serão utilizados neste documento, os quais serão apresentados nas seções a seguir.

2.3.1 MOF – *Meta-object Facility*

MOF é um padrão da OMG que define a linguagem utilizada para definir modelos padronizados. É a linguagem em que as definições de UML e de CWM (que será apresentado a seguir), ou seja, os meta-modelos de UML e de CWM são escritas. Ele não é usado somente para definir modelos padronizados, mas também para permitir a construção de ferramentas que auxiliam nesta atividade (OMG, 2006).

UML é uma das linguagens de modelagem definida com MOF. O meta-modelo de UML descreve exatamente como um modelo de UML é estruturado. A figura abaixo demonstra uma parte simplificada deste meta-modelo.

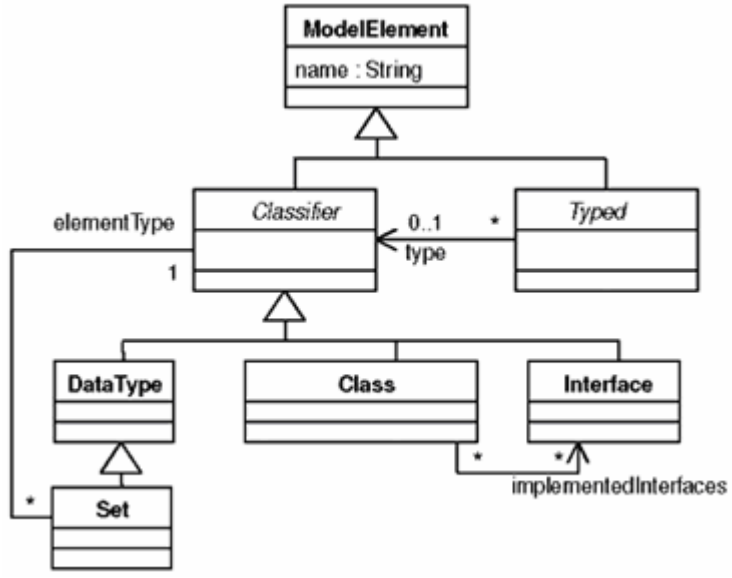


Figura 2.1 - Meta-modelo simplificado de UML
Fonte: KLEPPE, WARMER, BAST, 2003

O meta-modelo de UML descreve exatamente como um modelo de UML é estruturado. Do meta-modelo (Figura 2.1), pode-se deduzir o seguinte:

- No topo podemos ver que tudo em UML é um elemento do modelo (*ModelElement*) e possui um nome (*name*).
- A meta-classe abstrata *Classifier* é uma generalização de uma classe (*Class*), interface, e tipo de dado (*DataType*). Eles possuem muitas características comuns, mas algumas características particulares também.
- Uma classe (*Class*) pode implementar uma interface, mas não pode implementar um tipo de dado (*DataType*) ou uma interface.

Todo o modelo de UML deve cumprir estas regras; senão, não é uma instância correta do meta-modelo de UML. Por causa disto, pode-se utilizar modelos de UML sabendo exatamente como são representados e como são estruturados.

O papel principal do MOF dentro do MDA é que ele dá os conceitos e as ferramentas em relação aos modelos padronizados. Usando a definição de MOF sobre uma linguagem de modelagem pode-se definir transformações entre os modelos. E já que as transformações são definidas utilizando os meta-modelos da linguagem de modelagem envolvida, podem ser

aplicadas a todo o modelo escrito em qualquer linguagem. Sem uma linguagem padrão para descrever os meta-modelos, as transformações não poderiam ser definidas corretamente e a utilização do MDA seria muito difícil de realizar. Enfim, o MOF é a tecnologia principal para o MDA (KLEPPE, WARMER, BAST, 2003).

Um dos padrões incorporados ao MOF é o OCL – *Object Construct Language*, que é a linguagem utilizada para descrever restrições em modelos baseados em MOF (WARMER, KLEPPE, 2004). Será avaliada a utilização da linguagem OCL para definir as regras e restrições do modelo, e por isso, será apresentada em um capítulo a parte.

Atualmente um padrão novo está sob fase de finalização na OMG, chamado de QVT - *Query, Views, and Transformations*. Este padrão contemplará uma linguagem formal padrão para descrever as transformações que serão aplicadas entre os modelos, cujas linguagens são definidas usando o MOF. Será parte do MOF, e terá principalmente os seguintes componentes (QVT, 2005):

- Uma linguagem para a criação de visões sobre os modelos;
- Uma linguagem para efetuar pesquisas sobre os modelos;
- Uma linguagem declarativa para descrever transformações;
- Uma linguagem visual para descrever transformações.

Este novo padrão, que foi recentemente aprovado, está atualmente em fase de finalização. Será oficialmente disponibilizado no dia 07 de julho de 2006 (OMG, 2006). Apesar de ainda não estar concluído, já existem ferramentas que permitem utilizá-lo, de forma experimental. E existem também ferramentas que realizam transformações a partir de linguagens proprietárias.

2.3.2 UML – *Unified Modeling Language*

UML é a especificação mais utilizada da OMG, pois é a maneira mais utilizada para modelar um sistema com seu comportamento, sua arquitetura, seus processos de negócio e estruturas de dados.

Esta especificação auxilia o desenvolvedor a especificar, visualizar e documentar modelos de sistemas, incluindo sua estrutura e *design*[3], de maneira que comporte todas as exigências relatadas por requisitos. Pode também ser utilizado para modelar processos de negócio e sistemas que não são *softwares*, muitas vezes utilizando perfis (*profiles*) para estes casos.

Atualmente o UML está em sua segunda versão - UML 2.0, que teve como principal evolução a modelagem visual. Os novos recursos permitem que esta nova versão contemple novos elementos que existem atualmente em novas tecnologias de *software*, como MDA e *SOA - Service-Oriented Architecture* (OMG, 2006).

A especificação do UML 2.0 é extensa, e por isso foi divida em quatro partes:

- Superestrutura: É a especificação que define os diagramas e os elementos do UML. Basicamente, especifica seis diagramas de estrutura, três diagramas de comportamento, quatro diagramas de interação e os elementos compreendidos entre eles.

- Infra-estrutura: É a especificação que define as classes básicas que formam a fundação da superestrutura do UML e do MOF 2.0.

- OCL - *Object Constraint Language*: É a especificação da linguagem formal utilizada para descrever restrições, condições, e para expressar referências aos objetos modelados.

- *Diagram Interchange*: É a especificação que trata principalmente do intercâmbio de diagramas entre ferramentas de modelagem fabricadas por diferentes empresas. Ela estende o meta-modelo UML com um pacote (*package*) suplementar para manter informações orientadas a gráficos, permitindo assim que os modelos sejam importados e exportados, e também arquivados para depois serem restaurados, e assim serem exibidos de forma similar a qual foram originalmente feitas.

[3] *Design*: é um termo da língua inglesa que se refere a um determinado esforço criativo, seja bidimensional ou tridimensional, no qual se projetam objetos ou meios de comunicação diversos para o uso humano. Devido a este fato, ela pode ser traduzida como "desenho", mas não se refere ao ato de desenhar. Devido à dificuldade de tradução, costuma-se adotar a palavra original, ou, de forma ambígua, utilizar o termo “projeto”.

A especificação da superestrutura contém o conceito de *profiles*, *stereotypes* (estereótipos) e *tagged values* (valores etiquetados), que serão apresentados a seguir.

2.3.2.1 *Profiles*

Os sistemas orientados a objetos possuem muitas características estruturais em comum: possuem classes, interações entre elas, e assim por diante. Mas no momento em que são implementadas em alguma tecnologia, estas características similares ganham diferentes terminologias. Por exemplo, a plataforma .NET da Microsoft possui como tecnologia para componentização remota chamada de *Remoting.NET*, a plataforma Win32 utiliza normalmente DCOM e J2EE da SUN utiliza normalmente EJB.

Para facilitar a compreensão dos modelos UML por especialistas nestas tecnologias, é preferível utilizar a terminologia aplicada à tecnologia utilizada do que generalizar. Por exemplo, é preferível modelar para um sistema Java um componente EJB do que modelar um componente genérico utilizando a classe *Component* do meta-modelo UML.

Apesar de a linguagem UML ter sido projetada para atingir qualquer plataforma ou tecnologia utilizando elementos genéricos, a OMG criou um mecanismo para possibilitar modelagem de artefatos utilizando terminologias de mais alto nível, permitindo utilizar UML de acordo com necessidades específicas em cima de tecnologias. Este mecanismo é chamado de *Profile*.

Este recurso provê uma forma de definir um vocabulário comum para uma determinada tecnologia ou domínio, para depois ser utilizado como novos elementos de um ou mais modelos. Dessa forma, também será mais fácil automatizar o processo da transformação do modelo em código-fonte, pois um componente nomeado como DCOM poderá ser convertido em código-fonte para a criação deste componente em cima desta tecnologia.

Enfim, é um mecanismo especializado definido como parte do UML. Um *profile* define uma maneira específica de utilização do UML. Por exemplo, "CORBA *Profile*" define uma maneira específica de utilizar UML para modelar interfaces CORBA, e "Java *Profile*" define uma maneira de modelar código-fonte Java em UML (OMG, 2006).

Profiles foram introduzidos no UML 1.3 como uma maneira de estender esta linguagem de modelagem. Antes do UML 2.0, a idéia de estender UML estava limitada a estereótipos (*stereotypes*) e *profiles*. A idéia de estender mudou ligeiramente no UML 2.0. Existe agora um novo mecanismo, o meta-modelo. Um meta-modelo é um mecanismo subjacente que define uma linguagem para expressar um modelo. Em outras palavras, o meta-modelo do UML é um modelo que é utilizado para definir o UML, e é possível adicionar novas regras ao meta-modelo de forma a estender o UML (KLEPPE, WARMER, BAST, 2003).

Para especificar melhor estas regras, é previsto a utilização de colocar restrições no modelo. Restrição (*constraint*) é uma forma de expressar uma regra específica para um determinado elemento do *profile*, de forma a permitir que sejam definidas regras restritivas de utilização de um elemento em particular.

Restrições podem ser escritas em linguagem natural, como por exemplo:

- idade deve igual ou maior que zero

Restrições podem também ser escritas utilizando uma linguagem de programação, como por exemplo, em Java:

- idade >= 0

Restrições podem ser escritas em uma linguagem formal padronizada para este fim, a OCL. Neste trabalho será avaliada a utilização desta linguagem para a definição das restrições, pois assim será possível que a ferramenta de modelagem interprete o *profile* UML e verifique as consistências dos elementos aqui modelados. A seção 2.3.2.4 apresenta um resumo sobre esta linguagem.

2.3.2.2 *Stereotypes* (estereótipos)

Estereótipo significa que um elemento do modelo tem uma utilização especial. É normalmente exibido nos diagramas com seu nome entre dois "*guillemots*", como por exemplo: <<DCOM>>

Caso o estereótipo tiver um ícone associado, é possível exibi-lo com seu ícone. Muitas ferramentas UML oferecem a possibilidade de escolher entre uma ou mais formas de visualizar um estereótipo (Figura 2.2).

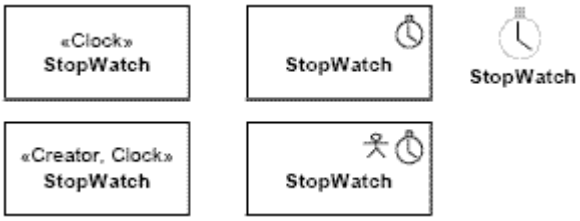


Figura 2.2 – Opções de apresentação de um estereótipo
Fonte: Unified Modeling Language: Superstructure (OMG, Versão 2.0, formal/05-07-04, p. 667, 2006)

2.3.2.3 *Tagged values* (propriedades)

Estereótipos podem ou não ter um ou mais valores associados. Este recurso oferece uma forma de possibilitar a colocação de informações pertinentes ao novo elemento.

Cada *tagged value* (valor etiquetado), ou propriedade, possui um nome, um tipo e opcionalmente um valor inicial. Por exemplo, a expressão abaixo se refere ao nome “Ano”, com tipo “*Integer*”, e valor inicial igual a “2006”:

- *Ano*: *Integer* = 2006

2.3.2.4 OCL – *Object Constraint Language*

OCL, inicialmente desenvolvida pela IBM (IBM, 2006) para modelagem de negócio, foi escolhida como a linguagem padrão para a expressão de restrições em UML, e tem sido aprimorada pela OMG. Como OCL é uma linguagem formal, ela pode fornecer automatização das restrições em ferramentas UML ou até em ferramentas de execução de sistemas construídos a partir da abordagem MDD.

Apesar de não ser necessário utilizar OCL para expressar restrições (*constraints*), a utilização desta linguagem padronizada traz alguns benefícios, dos quais é possível destacar que expressões OCL consistem em elementos do modelo, constantes e operadores. Elementos do modelo podem ser atributos, operações e membros relacionados por associações.

Atualmente já existem ferramentas de modelagem em UML que utilizam OCL, e uma destas ferramentas é o *Together Architect* 2006 (BORLAND, 2006). OCL é utilizado nesta ferramenta para expressar restrições nos diagramas (nos modelos), restrições nos *profiles* UML (nos meta-modelos), e também para expressar regras de auditoria e métricas. Além disso, OCL também é utilizado em tempo de execução para executar sistemas baseados em modelo UML utilizando diagramas do *Together* juntamente com os componentes ECO - *Enterprise Core Objects*, suíte de componentes que acompanha as últimas versões do *Delphi*.

Em diagramas UML, OCL pode ser também utilizado para armazenar e trocar valores de propriedades nos diagramas de estado e de atividade.

Normalmente cada expressão OCL é fisicamente anexada ao elemento do modelo a qual a expressão se refere. É possível também definir restrições OCL que não estão fisicamente anexadas a um elemento. Neste caso, é necessário informar qual é o contexto da expressão, ou seja, a qual elemento a expressão se refere. Por exemplo:

```
Context AccountFee

inv: self.baseCost >= 0.0
```

A palavra-chave "*Context*" informa qual é o elemento do modelo a qual a expressão se refere. Sempre que for declarado o contexto, deve-se declarar também qual é o tipo da restrição. A palavra-chave "*inv*:" indica o tipo da expressão, e este tipo indica que a expressão OCL é invariável (*invariant*), ou seja, uma restrição condicional que deve resultar sempre como sentença verdadeira, caso contrário, o elemento do modelo estará em um estado inconsistente.

Além deste tipo de restrição, existe também o tipo pré-condição (*precondition*) e o tipo pós-condição (*postconditions*).

A pré-condição, palavra-chave "*pre*:", é uma restrição que pode ser definida em métodos do modelo e é utilizada antes da execução do método associado. É normalmente utilizado para validar os parâmetros de entrada de um método.

Já a pós-condição, palavra-chave "*pos*:", é utilizada após a execução de um método, normalmente utilizado para validar os parâmetros de saída de um método, ou até para descrever como os valores de entrada são modificados após a execução do método.

2.3.3 XMI – XML *Metadata Interchange*

XMI é um formato padrão recomendado pela OMG desde 1999, que tem como objetivo o intercâmbio de dados possibilitando o compartilhamento de modelos entre ferramentas de modelagem diferentes de uma forma padronizada, trazendo, assim, consistência e compatibilidade em sistemas criados em ambientes diferentes. XMI possibilita a transferência de modelos UML e meta-modelos baseados em MOF através do padrão XML (OMG, 2006).

XMI é uma tecnologia da OMG baseado no padrão XML da W3C. Por isso ele é conhecido por integrar três padrões: XML da W3C[4], UML e MOF.

2.3.4 CWM - *Common Warehouse Metamodel*

CWM é uma linguagem de modelagem que foi projetada especificamente para modelar aplicações de *data warehousing*[5]. Possui interfaces e relações padronizadas que podem ser usadas para permitir um fácil intercâmbio de meta-dados de sistemas de *Data Warehouse* e *Business Intelligence* entre ferramentas, plataformas e repositórios em ambientes heterogêneos distribuídos (OMG, 2006).

O meta-modelo tem muito em comum com o meta-modelo de UML, mas tem um número de meta-classes especiais, como por exemplo, para modelar bases de dados relacionais. Os autores do CWM removeram tudo do UML que não era necessário para suas necessidades, e adicionaram detalhes específicos para *data warehousing*.

Pelo fato de *data warehousing* ser uma tecnologia que combina informação de diferentes fontes, este meta-modelo inclui diversos meta-modelos simplificados para bancos de dados relacionais, registros e estruturas, OLAP, XML, transformações, visualização da informação, mineração de dados, banco de dados multidimensionais, processos e operações de *data warehousing*, entre outros.

---

[4] W3C: *World Wide Web Consortium*, foi criado em 1994 para levar a Web para o seu potencial máximo, através do desenvolvimento de protocolos comuns e fóruns abertos que promovem sua evolução e asseguram a sua interoperabilidade. O W3C desenvolve tecnologias, denominadas Web *Standards* (ou padrões Web) para a criação e a interpretação dos conteúdos para Web.

[5] *Data warehousing*: O processo de projetar, construir e manter um sistema de *Data Warehouse*.

Todos estes meta-modelos foram modelados em MOF. Assim, todos podem ser utilizados como fonte ou destino para as transformações dentro de um processo de MDA (KLEPPE, WARMER, BAST, 2003).

2.3.5 PIM – *Platform-independent Model*

O padrão MDA define três etapas para a construção de um *software*. A primeira etapa consiste na criação de um modelo independente de plataforma chamado de PIM – *Platform-independent Model* (OMG, 2006).

Um PIM descreve um sistema completo para uma determinada necessidade de negócio. No PIM, o sistema é modelado a partir do ponto de vista que melhor representa o sistema: o negócio, e apenas o negócio. Neste modelo não existem indícios de que o sistema será implementado em um *mainframe* com um banco de dados relacional, ou se será implementado em uma aplicação J2EE sobre um sistema Linux (KLEPPE, WARMER, BAST, 2003).

Características principais:

- O PIM deve ser escrito para ser compreendido e corrigido por outros profissionais. Para isso, é importante que a linguagem de modelagem utilizada pelo PIM seja muito bem elaborada, pois deve ser compreendida por seres humanos e por máquinas.
- O PIM deve ser independente de qualquer tecnologia de execução.

2.3.6 PSM – *Platform-specific Model*

A segunda etapa definida pelo padrão MDA para a construção de um *software* consiste na criação automática de um ou mais modelos dependentes de plataforma, chamados de PSM – *Platform-specific Model*. Isto é feito utilizando transformações automatizadas do modelo representado pelo PIM (OMG, 2006).

Um PSM, inversamente ao PIM, deve refletir de forma muito próxima os conceitos e as construções utilizados na tecnologia correspondente a ser utilizada para a implementação do sistema. Em um PSM destinado a bases de dados, para citar um exemplo, os conceitos de tabela, coluna, e chaves estrangeiras devem estar claramente visíveis (KLEPPE, WARMER, BAST, 2003).

A terceira etapa é a geração do código-fonte ou da criação de uma base de dados em um SGBD a partir do PSM gerado. Como os PSMs são automaticamente gerados, cada PSM necessita ser compreendido apenas por ferramentas automatizadas de transformação e por peritos da tecnologia para a qual o PSM foi gerado. Para facilitar isso, atualmente já existem *profiles* em UML para tecnologias específicas, como por exemplo, o *EJB Profile for UML* (KLEPPE, WARMER, BAST, 2003).

2.4 Usabilidade

Usabilidade é uma metodologia científica aplicada na criação e remodelação de interfaces de *softwares*, *sites*, aplicativos, jogos e produtos de modo a torná-las fáceis de aprender e de usar, com foco no usuário.

O termo usabilidade foi definido na norma ISO/IEC 9126, que define características de qualidade para produtos de softwares, como "um conjunto de atributos de software relacionado ao esforço necessário para seu uso e para o julgamento individual de tal uso por determinado conjunto de usuários." (ISO/IEC 9126)

Este conceito evoluiu e foi redefinido na norma ISO 9241-11, que define guias de utilização, como "a capacidade de um produto ser usado por usuários específicos para atingir objetivos específicos com eficácia, eficiência e satisfação em um contexto específico de uso." (ISO 9241)

Os estudos de usabilidade começaram de fato em 1970 com a fundação da PARC - *Palo Alto Research Center*, da *Xerox Corporation*, com o objetivo de desenvolver novas tecnologias com o intuito de produzir produtos mais fáceis de vender, ou seja, mais atraentes para marketing. A partir destes estudos, foi primeiramente criado o *Alto workstation*, que foi desenvolvido inicialmente para uso interno da *Xerox*.

Em 1977, aproveitando tecnologias desenvolvidas pela PARC, o departamento de desenvolvimento de sistemas da *Xerox* contratou alguns cientistas da PARC para desenvolver e lançar em 1981 o *Star workstation*, oficialmente conhecido como *8010 Star Information System*. Foi o primeiro sistema comercializado que utilizou várias tecnologias que hoje são utilizadas em *softwares* de computadores pessoais, como interfaces gráficas baseadas em

janelas, *widgets*, pastas, *mouse*, rede *Ethernet*, servidores de arquivos, servidores de impressão e correio eletrônico.

A partir destes sistemas, vários outros sistemas foram criados e comercializados, como os sistemas Lisa, Macintosh, Amiga, OS/2 e Microsoft Windows, fato que a usabilidade tem cada vez mais ganhando importância no desenvolvimento de *software*.

Principais benefícios da usabilidade:

- Maior compreensão do *software* por parte do usuário;
- Diminuição da evasão de usuários por desistência;
- Aumento da eficiência da utilização do *software*;
- Menor custo de suporte e treinamento;
- Maior fidelidade do usuário ao *software*;
- Percepção positiva da empresa que desenvolveu o *software*.

Conforme SHNEIDERMAN, “os bons desenvolvedores sabem que tomar atenção cuidadosa em detalhes de *design* para usuários nos primeiros estágios do desenvolvimento reduz dramaticamente o tempo de desenvolvimento e seu custo, pois gerarão menos problemas durante o desenvolvimento e, assim, haverá menor necessidade de manutenção durante o ciclo de vida do *software*.” (SHNEIDERMAN, PLAISANT, 2004, p. 177)

Devido à importância da usabilidade em *softwares* modernos, é comum encontrar empresas que padronizaram a usabilidade em seus aplicativos ou até em suas arquiteturas de desenvolvimento. Como principal exemplo, temos a IBM e a Microsoft.

A IBM tem o CUA - *Common User Access* (BERRY, 1988), que estabelece um nível de padronização compatível com seus diferentes ambientes computacionais previstos no SAA - *Systems Application Architecture*, permitindo, assim, reaproveitar a experiência em usabilidade de seus usuários e projetistas independente da plataforma em utilização. CUA é baseado em uma arquitetura de interface gráfica que identifica os elementos estruturais fundamentais utilizados normalmente em sistemas gráficos, especificando assim os componentes, as regras e guias de utilização, como por exemplo, o "*CUA Basic Interface*

Design Guide" publicado em 1989 (IBM, 2006), para serem utilizados por desenvolvedores e por criadores de ferramentas de modelagem gráfica. Cada componente é especificado em termos de sua aparência e como os usuários interagem com ele. Além disso, sugere formas de utilização dos componentes para atividades que ocorrem frequentemente.

A Microsoft publicou sua padronização de usabilidade anos mais tarde, em 1999 (MICROSOFT, 2006). Chamada de "*Microsoft Windows User Experience*" é a literatura padronizada desta empresa que serve como guia para a criação de *softwares* com ênfase em uma boa usabilidade.

Atualmente existem estudos e implementações com técnicas emergentes de construção de interfaces gráficas. Como principais exemplos, temos os *Rich Clients* e as interfaces de usuário indutivas.

Rich Client é a camada de interação (GUI) de aplicativos RIA - *Rich Internet Applications*, que são aplicativos *Web* que possuem características e funcionalidades semelhantes aos aplicativos tradicionais (*desktop*). Este tipo de aplicativo normalmente transfere toda a camada GUI para o *browser* mantendo no servidor apenas a parte bruta dos dados. Neste tipo de aplicativo é comum encontrar *frameworks* especializados com bibliotecas de *widgets* portáveis. O atual sucesso dessa plataforma se deve a possibilidade de construir *softwares* para a *Web* de forma a possuírem funcionalidades quase idênticas a *softwares* desenvolvidos para rodarem nativamente em ambiente *desktop*.

Interface de usuário indutivas (IUI - *Inductive User Interface*) é um modelo que sugere novas técnicas de criar interfaces gráficas mais simples separando recursos em diversas telas ou páginas de forma que fiquem mais fáceis de utilizar e de compreender (MICROSOFT, 2006). Sua principal diferença com o modelo oposto, que pode ser chamado de dedutivo, é que o modelo indutivo procura evitar que o usuário tenha que deduzir o propósito da interface e como utilizar seus controles para concluir uma tarefa. São também baseadas em *widgets*, que será apresentado na próxima seção.

2.4.1 *Widgets*

Um *widget* (ou controle) é um componente da interface gráfica que um usuário de computador interage, como uma janela, uma imagem, ou uma caixa de texto. *Widgets* são qualificados como objetos virtuais que imitam ou representam um objeto físico.

Widgets são conjuntos de unidades básicas de componentes utilizados em interfaces gráficas (GUI). Normalmente são oferecidas em bibliotecas ou em *frameworks*. A maioria das ferramentas de programação oferece bibliotecas de *widgets* reutilizáveis para sua pronta utilização, como *Delphi* da *Borland*, *Java* (ferramentas oferecidas por várias empresas) e *Visual Studio* da Microsoft.

Widgets são conhecidos também como pequenos aplicativos especializados em oferecer fácil acesso a funcionalidades utilizadas freqüentemente, como, por exemplo, acessórios da área de trabalho do *Mac OS*, sistema operacional gráfico da *Apple*. O *Mac OS X* (versão 10.4) possui suporte nativo a *widgets*, via *Dashboards*[6]. A nova versão do sistema operacional da *Microsoft*, o *Windows Vista*, oferecerá recurso similar, via *Windows Sidebar*[7], onde cada mini-aplicativo será chamado de *Desktop Gadgets*.

Nesta monografia o conceito de *widget* será restrito a controles, ou seja, componentes utilizados para a construção de interfaces gráficas. Os componentes mais conhecidos e utilizados são janelas, botões, caixas de seleção, *radio buttons*, ícones, imagens, barras de ferramentas, menus, menus de contexto, *Tree views*, grades, tabelas, barras de rolagem, barras de progresso, textos, campos de edição, barras de *status*, entre outros.

---

[6] *Dashboards*: são pequenos aplicativos baseados em *widgets* para o sistema operacional *Mac OS X* 10.4 da *Apple*.

[7] *Windows Sidebar*: um painel que ficará no lado direito ou esquerdo da área de trabalho, que oferecerá recursos semelhantes aos *Dashboards* da *Apple*.

3 O PROBLEMA

Como introduzido no capítulo 1, o problema enfatizado nesta monografia é a inexistência de uma técnica ou modelo padronizado para a modelagem de interfaces gráficas em aplicativos desenvolvidos sobre a abordagem MDD. Porém, existem soluções alternativas que contemplam essa necessidade, as quais serão apresentadas nas próximas seções deste capítulo.

Conforme confirmado por BLANKENHORN (2004, p. 17), interfaces gráficas são importantes para os usuários de um sistema de *software*, que por definição é a única parte do sistema que é visível a eles, e têm um grande impacto no custo e na produtividade do desenvolvimento de *software*, pois foi demonstrado que a programação dessa parte pode consumir 50% do código-fonte de um aplicativo.

Conforme GOULD, "algumas das mais sérias dificuldades podem ser evitadas se, em uma etapa inicial, os clientes e usuários tenham uma impressão realística de como o sistema finalizado será exibido.” (GOULD, LEWIS, 1985)

Sem a existência de um recurso especializado que permita modelar interfaces gráficas durante a fase de análise e projeto, com um nível de abstração aproximado ao resultado final, não é possível modelar aspectos referentes à usabilidade, como posições dos elementos, informações textuais, especificar quais widgets serão utilizados, como serão utilizados pelo *software*, entre outros motivos.

Conforme SHNEIDERMAN, "o objetivo principal de projetistas de menu, de preenchimento de formulários, e de caixas de diálogo é de criar interfaces sensíveis, compreensíveis, memorizáveis, com uma organização conveniente às tarefas dos usuários." (SHNEIDERMAN, PLAISANT, p. 117)

Uma maneira mais imediata de oferecer um novo recurso para a modelagem de interfaces gráficas dentro da arquitetura MDA é de criar um *profile* UML para esta finalidade.

Atualmente, para a modelagem de interfaces gráficas, existem abordagens que utilizam *profiles* UML entre outras técnicas. Nas próximas seções, serão apresentadas as principais abordagens pesquisadas.

3.1 Abordagens que não utilizam *profiles* UML

Todas as soluções que não são baseadas em *profile* são proprietárias, ou seja, são específicas de cada ferramenta. A seguir será apresentada a técnica oferecida pelas ferramentas de modelagem *Rose Web Modeler* e *System Architect*.

3.1.1 *Rose Web Modeler*

A ferramenta de modelagem *Rational Rose Enterprise Edition* 2003 (IBM, 2006) oferece como recurso para modelagem de interfaces gráficas de aplicativos *Web*, o *Web Modeler* (Figura 3.1). Este recurso é específico para modelar aplicativos *Web* e inclui estereótipos para páginas geradas dinamicamente no servidor, páginas geradas no lado do cliente (no browser) e um estereótipo limitado para a modelagem de formulários *Web*.

Utilizando este recurso, não é possível modelar aspectos referentes à usabilidade, como posições dos elementos, informações textuais, especificar quais widgets serão utilizados, como serão utilizados pelo *software*, entre outras características vitais de usabilidade.

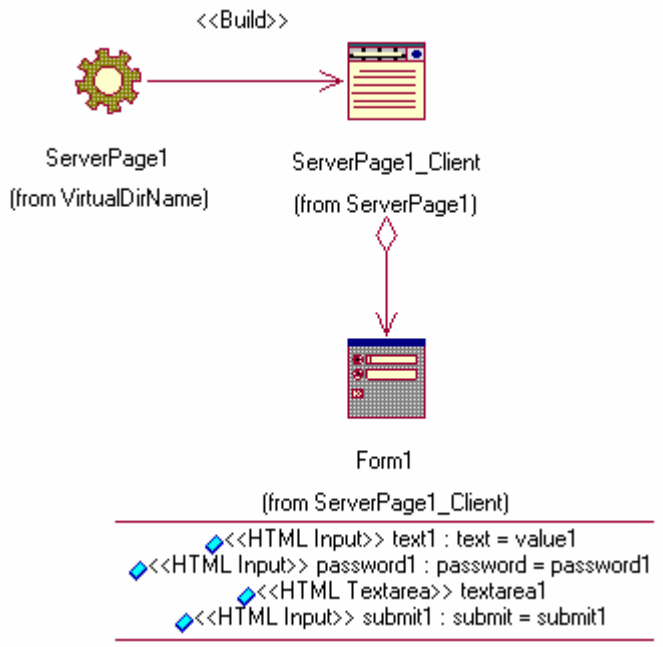


Figura 3.1 – *Rose Web Modeler*

Fonte: *Rational Rose Enterprise Edition* 2003

3.1.2 *Screen Painter Diagram* do *System Architect*

A ferramenta de modelagem *System Architect* 2001 (TELELOGIC, 2006), da *Popkin Software*, que foi adquirida pela *Telelogic* em abril de 2005, oferece o recurso *Screen Painter* (Figura 3.2) como recurso de modelagem de interfaces gráficas. É uma alternativa similar às alternativas oferecidas em ferramentas de programação como *Delphi* e *Visual Studio*, com possibilidades de expressão muito limitadas.

Esta abordagem permite a modelagem de interfaces gráficas sobre a ótica da usabilidade, pois permite definir o posicionamento dos objetos, utilizar *widgets*, inserir informações textuais, entre outros.

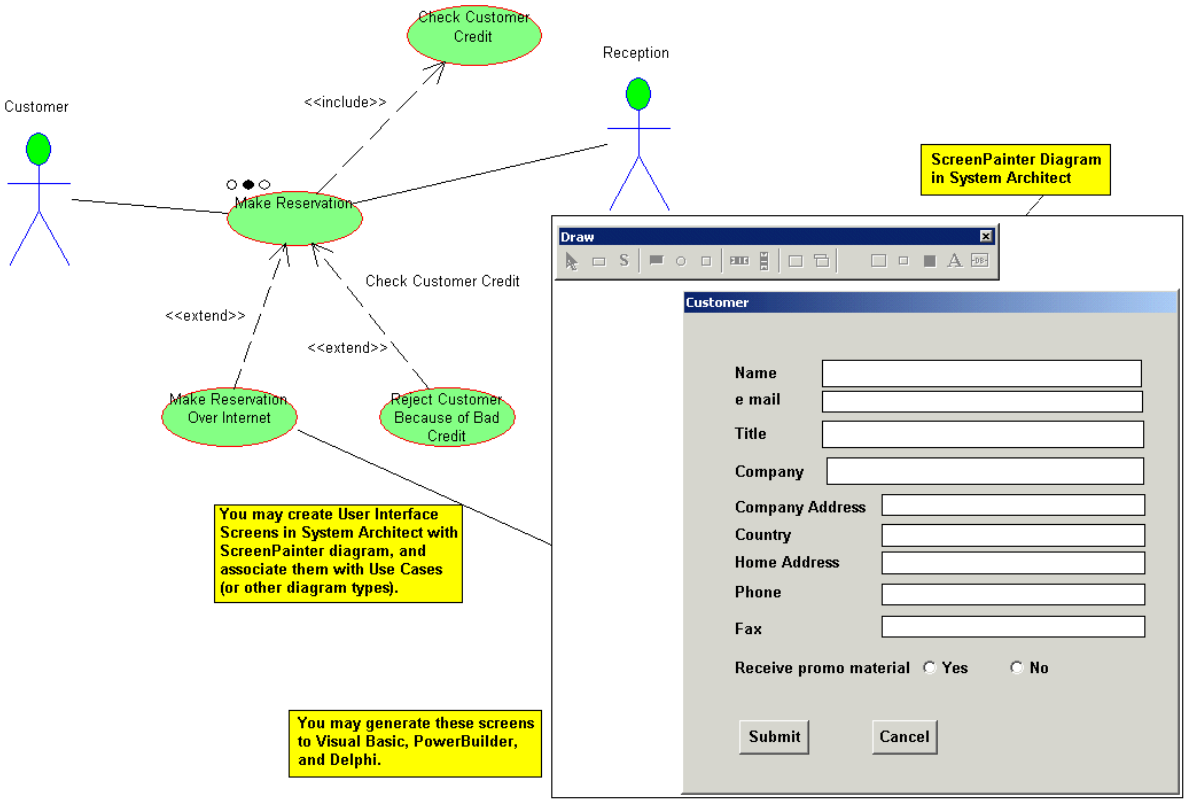


Figura 3.2 – *Screen Painter*

Fonte: Telelogic *Web Site*

O problema desta abordagem é que ela não é padronizada e nem aberta – ela é específica desta ferramenta, e depende de funcionalidades específicas para a conseqüente geração de código para a criação do *software* em si.

3.2 Abordagens que utilizam *profiles* UML

Todas as soluções que são baseadas em *profile* podem ser reutilizadas em outras ferramentas, desde que a ferramenta onde o *profile* foi criado permita a exportação da definição do *profile*, preferencialmente via arquivo XMI.

A seguir serão apresentados os *profiles* mais relevantes ao problema apresentado neste documento. Estes *profiles* foram apresentados em pesquisas acadêmicas, e não foram encontradas implementações destas especificações em ferramentas comerciais de modelagem.

3.2.1 *UML Web Profile*

O *UML Web Profile*, proposto para a modelagem de interfaces gráficas de *Web* sites, é um *profile* criado para ser utilizado com a metodologia UWE - *UML Web Engineering*, que define os modelos conceitual, de navegação e de apresentação (HENNICKER, KOCH, 2001).

Esta abordagem possibilita a modelagem dos *widgets* para ambientes *Web*, o que inclui posição, tamanho e aninhamento (qual componente está dentro de outro componente). Inclui estereótipos para menus, itens de menus, janelas, imagens, *links*, formulários, entre outros (Figura 3.3).

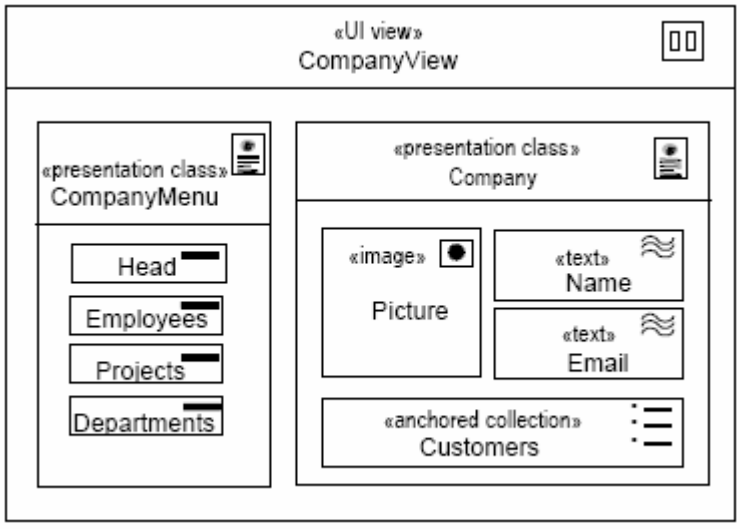


Figura 3.3 – Exemplo de apresentação do UML *Web Profile*
Fonte: HENNICKER, KOCH, 2001

Apesar de este profile pertencer a uma metodologia completa, ele é restrito a ambientes *Web*, ou seja, em uma arquitetura MDA ele deve ser utilizado dentro do PSM para plataformas *Web*, ao invés do PIM para possibilitar seu uso em qualquer plataforma. Outro fator negativo é quanto à usabilidade, pois mesmo permitindo a modelagem ser parecida ao visual final do *software* a ser gerado, não possibilita a modelagem de informações textuais como rótulos e textos. Além disso, não existem estereótipos definidos para modelagem de *widgets* comuns em formulários *Web*, como campos de edição, caixas de seleção, e assim por diante. Para isso, existe apenas um estereótipo definido chamado de <<*Form*>> que é utilizado como um *container* destes componentes, sem a possibilidade de especificá-los.

3.2.2 *GUILayout*

O *GUILayout*, também proposto para a modelagem de interfaces gráficas de *Web* sites, é um *profile* criado para a representação abstrata de telas baseadas em esboços de designers gráficos (BLANKENHORN, 2004, p.7).

Esta abordagem, que oferece como principais estereótipos *ScreenArea*, *Screen*, *Form*, *Navigation*, *Link*, *Heading*, *Image*, *Logo*, *Text* e *WorkSpace*, possibilitando a modelagem do *layout* dos componentes comumente encontrados em páginas *Web*, o qual também inclui posição, tamanho e aninhamento (Figura 3.4).

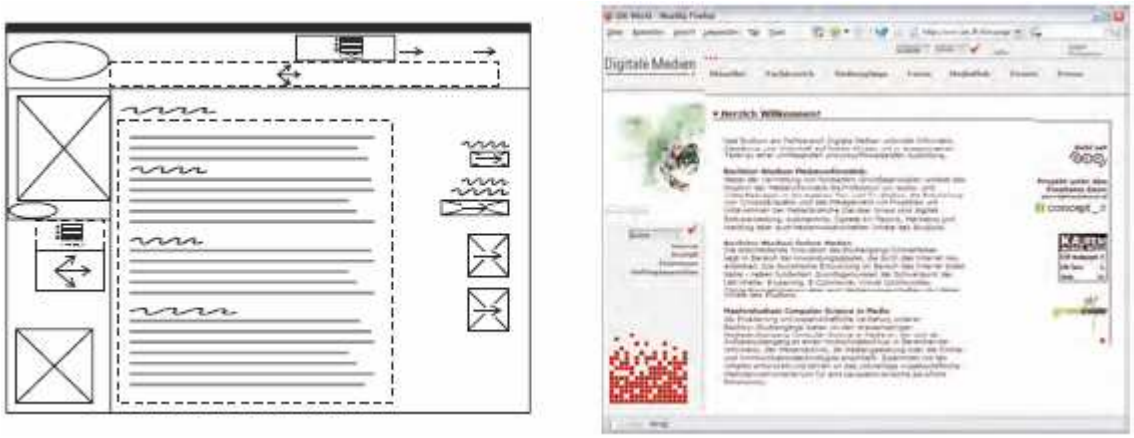


Figura 3.4 - *GUILayout* e página *Web* correspondente
Fonte: BLANKENHORN, 2004, p.98

Além de possibilitar a modelagem de elementos gráficos, a especificação de *GUILayout* contempla também navegação e interação com outros diagramas.

Como pontos negativos, similar ao *UML Web Profile* (3.2.1), ele também é restrito a ambientes *Web*, não apresenta no modelo informações textuais, e não existem estereótipos previstos para modelagem de *widgets* comuns em formulários *Web*, como campos de edição, caixas de seleção, e assim por diante. De forma semelhante ao *UML Web Profile*, existe apenas um estereótipo chamado de <<*Form*>> que é utilizado como um *container* limitador destes componentes, sem a possibilidade de especificá-los

Além disso, este modelo não foi avaliado em uma ferramenta de modelagem com suporte a *profiles* - a avaliação foi feita foi em cima de um protótipo programado pelo autor, o qual elimina a prova de utilização dos padrões documentados pela OMG, os quais são implementados em ferramentas de modelagem e dificilmente em pequenos protótipos.

Um exemplo da necessidade de avaliação do *profile* em cima de uma ferramenta de modelagem é que estes dois modelos, *GUILayout* e *UML Web Profile*, estenderam todos os estereótipos com funcionalidade do tipo *container* (para possibilitar aninhamento de determinados elementos) a partir da meta-classe *Class*, a qual não suporta esta funcionalidade. Ou seja, ambos os modelos, se forem implementados em uma ferramenta de modelagem, simplesmente não funcionarão de acordo com os objetivos previstos. Acredito que, caso estes autores tivessem feito uma avaliação deste tipo, teriam corrigido a utilização da meta-classe *Class* para outras meta-classes que suportam esta funcionalidade.

No próximo capítulo será apresentada uma solução que contempla a utilização de um profile UML para modelagem de interfaces gráficas utilizando conceitos de usabilidade, e, logo após, será feita a avaliação do modelo em cima de duas ferramentas de modelagem comerciais.

4 A SOLUÇÃO PROPOSTA

Conforme introduzido na seção 1.2, o objetivo deste trabalho é demonstrar a criação de um *profile* UML para possibilitar a modelagem de interfaces gráficas com foco na apresentação de *widgets* sob o ponto de vista da qualidade e da usabilidade para ambientes de desenvolvimento de *software* que utilizam MDD.

Para a criação deste meta-modelo para modelagem de interfaces gráficas foi escolhido a técnica de *profile*, ao invés de outras alternativas, as quais estão descritas nesta mesma seção. Esta técnica permite compatibilidade com ferramentas existentes, além de possibilitar sua integração sem depender da interação do fabricante da ferramenta de modelagem. Teoricamente, qualquer ferramenta que oferecer o recurso de *profile* em conformidade com a especificação da superestrutura do UML 2.0 e seguir a especificação do *Diagram Interchange* permitindo obter a posição, a hierarquia e o tamanho dos elementos modelados, será capaz de utilizar com sucesso o modelo aqui apresentado.

Este modelo poderia ter sido criado utilizando diretamente a linguagem MOF, a mesma linguagem a qual UML foi expressa. Porém, MOF exige que muitos detalhes sejam especificados, e a incorporação de um modelo expresso em MOF em cima de uma ferramenta exigiria a interação dos fabricantes de ferramentas. Apesar de esta alternativa eliminar em muito as limitações existentes na abordagem do *profile*, foi escolhido este caminho visando verificar o quanto suas limitações podem prejudicar ou não a utilização desta técnica para o objetivo proposto. As limitações aqui citadas serão apresentadas na avaliação do modelo (capítulo 5).

Uma alternativa seria criar um modelo através da utilização de um conjunto de outras técnicas, como utilizar editores específicos para a modelagem sem utilizar padrões da OMG, ou com a utilização de *templates* ou outras tecnologias. Essa alternativa foi descartada pois dificultaria em muito uma padronização entre diversas ferramentas de modelagem, a qual é um objetivo desta proposta.

Conforme informado por BLANKENHORN (2004, p. 61), a solução apresentada por ele foi a primeira solução desta área a incorporar os novos recursos do UML 2.0 juntamente com o DI - *Diagram Interchange*. O principal diferencial desta abordagem é que com DI é possível manter informações de layout automaticamente ao invés de ter que prever *tagged values* para a manual especificação da posição e do tamanho dos elementos modelados, técnica que era utilizada sobre versões antigas do UML, já que estas versões não ofereciam uma forma automática de fazê-lo.

Já que a proposta deste documento utiliza técnicas similares utilizados no modelo de BLANKENHORN (2004, p. 67), é possível destacar que a solução deste documento é a segunda solução proposta na mesma área com os seguintes diferenciais:

- O foco é na usabilidade (representação qualitativa da interface final), e não no layout (esboços de *Web designers*);
- Os componentes oferecidos serão *widgets*, e não elementos gráficos da *Web*;
- O modelo é genérico para várias plataformas, ao invés de ser específico para a *Web*;
- Este *profile* foi validado em ferramentas de modelagem comerciais de UML 2.0, e não em um protótipo criado exclusivamente pelo autor.

Um outro importante requisito do modelo aqui apresentado é possibilitar sua utilização como um diagrama de modelagem GUI genérica para ser utilizado na camada PIM em uma arquitetura MDA (requisito informado na seção 4.1), pois até o momento não existe padronização para tal. Acredita-se que a padronização de um modelo voltado para a modelagem de interfaces gráficas será essencial para o futuro e o sucesso desta arquitetura.

O principal diferencial desta proposta é a especificação de um modelo que permita a representação de uma interface gráfica com aspectos e conceitos estudados pela área da

usabilidade, ou seja, um meta-modelo que permita representar o aspecto final da interface gráfica de forma mais semelhante possível, onde apenas as restrições impostas pela tecnologia de *profile* do UML alterarão o formato de como essa interface poderá ser modelada, ou seja, forçará diferenças visuais entre a interface modelada e a interface final construída durante a construção do *software*.

Aspectos de usabilidade serão incorporados no modelo de forma a abstrair ou apresentar certos conceitos, como por exemplo:

- Permitir a modelagem do posicionamento e da hierarquia de *widgets*;
- Permitir modelar as expressões textuais utilizadas nas interfaces gráficas, como títulos, rótulos, textos de campos, textos de ajuda de contexto, entre outros;
- Permitir a utilização de *boxes* e *containers*, utilizados para o agrupamento de informações e controles correlacionados;
- Oferecer *widgets* comumente utilizados em ferramentas atuais de programação, possibilitando assim uma fácil transformação do modelo em código-fonte, ou em *software* executável;
- Oferecer propriedades que permitam a modelagem de troca de estados destes componentes;
- Oferecer propriedades que possam determinar o conteúdo de cada componente, estaticamente ou dinamicamente, através de simples expressões textuais (do tipo *String*) ou através de expressões OCL.

Enfim, este *profile* poderá ser utilizado de forma imediata em processos de desenvolvimento de *software* em ferramentas que suportem a utilização deste recurso, e poderá servir como base para outros estudos.

Nas próximas seções, serão apresentados os requisitos definidos para este *profile*, e logo após será demonstrado o modelo elaborado que atende estes requisitos.

4.1 Requisitos do *profile*

Para contemplar os objetivos deste trabalho, foram definidos dez requisitos para que o *profile* seja fácil de utilizar, para ser utilizável em ferramentas de modelagem, para permitir a modelagem de interfaces gráficas quanto a utilização de *widgets*, entre outros. A seguir estão enumerados os requisitos que contemplam a solução apresentada neste trabalho:

1. Um diagrama deve modelar o *layout* (disposição) de uma interface gráfica;
2. O diagrama deverá ser fácil de aprender, utilizar e entender;
3. A criação de um digrama em uma ferramenta especializada de modelagem deverá ser uma tarefa de poucos minutos;
4. A criação de um diagrama não deve exigir muito trabalho adicional;
5. A simbologia utilizada no diagrama deverá representar os elementos fundamentais utilizados em interfaces gráficas (*widgets*);
6. O modelo deverá oferecer alguma forma de extensão para utilização direta no diagrama para a representação de *widgets* não contemplados nesta especificação;
7. O modelo deverá seguir as premissas do UML, ou seja, ser simples, compreensível para pessoas que não são da área de desenvolvimento de *software*, e o diagrama poderá ser impresso em papel sem perder informações essenciais para tomadas de decisão;
8. O modelo deverá ser abstrato o suficiente para possibilitar utilizá-lo na camada PIM da arquitetura MDA, ou seja, independente de tecnologia destino (*Web*, *Desktop*, PDA, POS, etc);
9. Prever a utilização de OCL nos *widgets* possibilitando assim automatização do funcionamento do sistema gerado em ferramentas que suportam a execução do modelo, como por exemplo a execução de diagramas modelados no *Together Architect* 2006 (BORLAND, 2006) a partir da utilização da suíte de componentes chamada de ECO III, atualmente distribuída juntamente com o *Delphi* da *Borland*.

10. O modelo deverá ser criado para utilização em ferramentas de modelagem que suportem a criação e utilização de *profiles* UML de acordo com a especificação da OMG, e para isso o *profile* especificado neste documento deverá ser avaliado em cima de pelo menos uma ferramenta com este tipo de suporte.

Note que não existem requisitos com a preocupação de possibilitar navegação a partir da interface gráfica modelada, interação entre outros diagramas, prototipação, pois são características não tratadas neste trabalho. Estas características poderão ser solucionadas em futuros trabalhos.

4.2 O projeto

A partir de um levantamento dos principais *widgets* utilizados por *softwares* e oferecidos por ferramentas de programação, foram selecionados os seguintes *estereótipos* para pertencer ao modelo: *Window*, *Panel*, *GroupBox*, *TabSheet*, *TextBox*, *MemoBox*, *RichBox*, *ComboBox*, *Media*, *Table*, *Gauge*, *TrackBar*, *RadioButton*, *CheckBox*, *Button*, *Label* e *LinkedLabel*. Cada estereótipo foi definido na seção 4.3.

Além destes, foram incluídos estereótipos para extensão, ou seja, para suportar de forma genérica *widgets* não previstos neste modelo. Os estereótipos de extensão são *ExtendedContainer*, *ExtendedBox* e *ExtendedText*.

O modelo e os estereótipos foram definidos sempre tendo em vista que, futuramente, para possibilitar a transformação do modelo em *software* executável, existirá um *framework* de execução encarregado de instanciar cada estereótipo em seu respectivo *widget*, realizar a conversão entre as propriedades de cada elemento do modelo para as propriedades do *widget* (conforme a plataforma gráfica em execução) e prover a validação e a persistência dos conteúdos dos controles.

Os seguintes *widgets* chegaram a ser avaliados, mas não são contemplados neste modelo pelos motivos abaixo justificados:

- *PopupMenu*: controle para possibilitar a modelagem de um menu *popup* associado a um *widget* em particular. Esse controle não foi incluído nesta solução pois esse menu pode ser definido pelo *framework* para cada *widget* de

forma genérica. Além disso, outro motivo é que a proposta deste trabalho não contempla a especificação da navegação e da interação da interface gráfica.

- *TreeView*: controle que possibilita exibir elementos encadeados em uma relação hierárquica. Um *widget* específico para este comportamento não é contemplado no modelo por ser um elemento que exige certa complexidade para a definição de seu conteúdo. Porém, foi previsto no modelo que o *widget* chamado de *Table* funcione de forma semelhante para apresentar relações hierárquicas quando for associada uma classe que apresente relações hierárquicas em seu conteúdo.

- *ListView*: controle especializado em exibir listas de informações a partir de formatos pré-configurados (por ícones, detalhada, em miniaturas, lado a lado, entre outros). A justificativa de não ter incluído esse elemento é a mesma do *widget TreeView*.

Além disso, os seguintes recursos não foram especificados pois espera-se que sejam oferecidos pelo *framework* de execução, e não pelo modelo:

- Ajuda de contexto por *widget*: não haverá um atributo específico para referenciar um índice associado ao *widget* no arquivo de ajuda. Este recurso poderá ser automatizado pelo *framework*, onde cada elemento poderá ser referenciado pelo nome do elemento ou pela sua posição hierárquica no modelo.

- *Drag* & *Drop*: controles para a especificação de arrastar e soltar. Esse recurso não foi incluído nesta solução pois a proposta deste trabalho não contempla modelagem de interações da interface gráfica. Este recurso pode ser automaticamente oferecido pelo *framework*, de forma genérica.

- Formas gráficas: linhas, retângulos, ou qualquer outra representação gráfica proprietária utilizada para separar ou agrupar os controles a partir de sua similaridade ou contexto. A utilização destes objetos gráficos pode ser automaticamente oferecida pelo *framework* de execução ou pode ser modelada através do *widget* oferecido com o estereótipo *Media*.

Para a especificação das restrições dos elementos do modelo foi avaliada a utilização de OCL. Infelizmente, durante os primeiros testes de avaliação do modelo, foi constatado que

nenhuma das ferramentas de modelagem utilizadas neste projeto (apresentados no capítulo 5) suportam a utilização de OCL com os meta-modelos utilizados neste *profile*. Já que não é possível validar estas restrições em OCL, as restrições serão especificadas em texto comum. A conversão destas regras em OCL e sua validação poderão ser realizadas em futuros trabalhos.

4.3 O modelo proposto

Nesta seção são apresentados todos os estereótipos que representam os principais *widgets* utilizados por *softwares* e oferecidos por ferramentas de programação. Cada elemento possui uma descrição de seu uso, os atributos definidos para possibilitar sua integração com um *framework* de execução, as restrições referentes a outros elementos do modelo (quando for necessário), a notação visual (quando aplicável) e exemplos de utilização.

Para efeitos de referência a este modelo será utilizado o nome *UGUI Profile*, que é um acrônimo para *Usability Graphical User Interface Profile*.

Todos os elementos diretamente utilizáveis deste *profile* serão ligados ao estereótipo <<*paletteContribution*>>, que é a forma disponibilizada pelo *Together* para indicar quais são os estereótipos que ficarão disponibilizados na paleta de elementos de um diagrama de componente, permitindo assim sua futura utilização. O diagrama de componentes foi escolhido como digrama para a modelagem de interfaces gráficas pois as interfaces gráficas são basicamente compostas de componentes (*widgets*).

O modelo completo da definição do *profile* pode ser consultado no Apêndice A.

4.3.1 *Container*

Estereótipo abstrato que é utilizado como classe ancestral dos elementos que possuem comportamento do tipo *container*.

4.3.1.1 Descrição

Este estereótipo foi definido como uma classe abstrata, ou seja, não é um elemento que pode ser utilizado diretamente para modelagem da interface gráfica. O tipo *container* é o comportamento onde o elemento pai, ou hospedeiro, tem a característica de permitir a

inclusão de outros elementos visualmente dentro dele, ou seja, possibilita aninhar elementos relacionados. Esta característica existe na meta-classe *Component* do UML, que permite que sejam adicionados outros elementos *Component* dentro dele (entre outras meta-classes como o *Artifact*) com a finalidade de apresentar a complexidade existente com a conjunção de sub-componentes que compõem um componente.

Este estereótipo estende a meta-classe *Component* para possibilitar o comportamento de um *container*, visto que essa característica é oferecida por este elemento do meta-modelo do UML 2.0, e será estendida pelos estereótipos *Window*, *Panel*, *GroupBox*, *TabSheetGroup* e *TabSheet* apresentados nas próximas seções.

4.3.1.2 Atributos

- *Visible*: *Boolean = true*. Utilizado para definir a visibilidade do elemento durante o ciclo de execução do *software*.
- *Enabled*: *Boolean = true*. Utilizado para definir se o elemento estará ativo durante o ciclo de execução do *software*. Caso o elemento estiver visível e não ativo, o *framework* poderá exibir o objeto, mas não permitir a interação com os elementos internos deste elemento. Neste caso, os elementos internos estarão disponíveis apenas para visualização.
- *Resizable*: *Boolean = true*. Indica se é possível redimensionar o controle durante a execução do *software*. O *framework* deverá providenciar o redimensionamento e a realocação dos objetos aninhados deste elemento.

4.3.1.3 Restrições

1. Deverá haver pelo menos um estereótipo estendido de *container* no diagrama de modelagem de interfaces gráficas.

4.3.2 *Window*

Estereótipo utilizado para representar uma janela. Estende a classe abstrata *Container*.

4.3.2.1 Descrição

Window é um estereótipo do tipo *container* utilizado para representar uma janela, a qual poderá aninhar outros *widgets*. Na maioria dos casos, este elemento será o *container* principal para ser utilizado na modelagem de um artefato de interface gráfica.

4.3.2.2 Atributos

- *Title*: *String*. Utilizado para determinar um título para a janela durante a execução do *software*.
- *IconFilename*: *String*. Utilizado para determinar um ícone, a partir de um arquivo que contenha uma imagem com formato suportado pelo *framework*, para ser exibido na barra de título da janela conforme sua plataforma de execução.
- *Style*: *WindowStyle = Normal*. Configura a forma que a janela será construída e apresentada. O tipo *WindowStyle* é determinada a partir de um *Enumeration* especificado no modelo. Os itens possíveis são *Normal*, *Float* (para janelas flutuantes não modais), *Modal* (para janelas com comportamento modal), *MDIParent* (para janelas principais MDI), *MDIChild* (para janelas filhas MDI) e *StayOnTop* (para janelas que devem ficar obrigatoriamente na frente de qualquer outra janela).

4.3.2.3 Restrições

1. Um *Window* não pode aninhar outro *Window*.

4.3.2.4 Notação

Este estereótipo é representado visualmente conforme a figura abaixo, onde possui como cabeçalho o nome do estereótipo, o nome do componente definido pelo projetista, um ícone representando uma janela e uma área interna que poderá ser utilizada como *container* para o aninhamento de outros estereótipos previstos neste projeto.

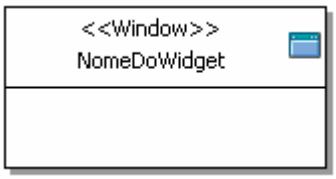


Figura 4.1 – Estereótipo *Window*

4.3.3 *Panel*

Estereótipo utilizado para representar uma área virtual para agrupar *widgets*. Estende a classe abstrata *Container*.

4.3.3.1 Descrição

Panel é um estereótipo do tipo *container* utilizado para representar uma área virtual, ou seja, uma área limitada com ou sem a apresentação de borda, a qual poderá aninhar outros *widgets*. Poderá ser utilizada como uma forma de possibilitar a ocultação ou a desativação de um conjunto de *widgets* incluídos em sua estrutura interna através das propriedades *Visible* e *Enabled*, respectivamente.

4.3.3.2 Atributos

- *BorderVisible*: *Boolean* = *true*. Utilizado para definir se será exibido borda nos limites da área representada por este elemento.
- *Caption*: *String*. Utilizado para informar opcionalmente um texto neste elemento, que será exibido durante a execução somente se este elemento não possuir nenhum *widget* aninhado.
- *Alignment*: *AlignmentStyle* = *Center*. Define o alinhamento do *Caption*. O tipo *AlignmentStyle* é um *enumeration* definido no modelo com os valores *Left*, *Center*, *Right* e *Justify*.

4.3.3.3 Restrições

1. Um *Panel* não pode aninhar um *Window*.

4.3.3.4 Notação

Este estereótipo é representado visualmente conforme a figura abaixo, onde possui como cabeçalho o nome do estereótipo, o nome do componente definido pelo projetista, um ícone representando painel e uma área interna que poderá ser utilizada como *container* para o aninhamento de outros estereótipos previstos neste projeto.

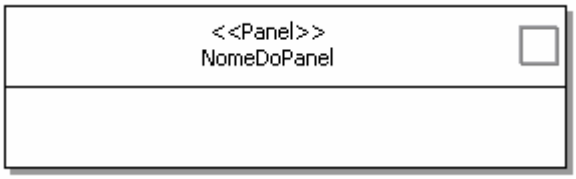


Figura 4.2 – Estereótipo *Panel*

4.3.4 *GroupBox*

Estereótipo utilizado para representar um *widget* do tipo *group box*. Estende a classe abstrata *Container*.

4.3.4.1 Descrição

Este elemento é utilizado para agrupar elementos relacionados. Durante sua execução, é exibida uma borda nos limites da área deste elemento, e é também exibido opcionalmente um título na parte superior que indica a finalidade dos elementos aninhados por ele.

4.3.4.2 Atributos

- *Caption*: *String*. Utilizado para informar opcionalmente um texto de título deste elemento.

4.3.4.3 Restrições

1. Um *GroupBox* não pode aninhar um *Window*.
2. Um *GroupBox* não pode aninhar um *Panel*.

4.3.4.4 Notação

Este estereótipo é representado visualmente conforme a figura abaixo, onde possui como cabeçalho o nome do estereótipo, o nome do componente definido pelo projetista, um ícone representando este *widget* e uma área interna que poderá ser utilizada como *container* para o aninhamento de outros estereótipos previstos neste projeto.

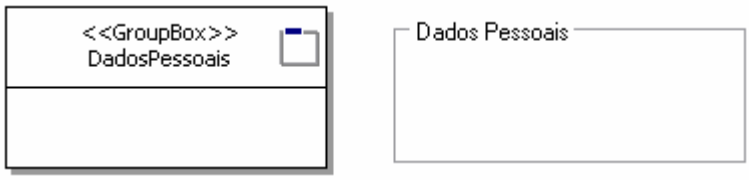


Figura 4.3 – Estereótipo *GroupBox* e um exemplo de seu *widget* correspondente

4.3.5 *TabSheetGroup*

Estereótipo utilizado para agrupar logicamente um ou mais elementos *TabSheet*. Estende a classe abstrata *Container*.

4.3.5.1 Descrição

Este elemento é utilizado para definir logicamente a modelagem de *widgets* do tipo *TabSheet* que serão utilizados em conjunto, ou seja, elementos que serão sobrepostos na mesma área de visualização conforme uma nova aba (título) for selecionada. Utilize este elemento para agrupar várias áreas com abas nomeadas, onde apenas a área (página) da aba atualmente selecionada (parte do *TabSheet*) durante a execução poderá ser exibida por vez.

4.3.5.2 Atributos

- *ActivePage*: *String*. Propriedade opcional que pode ser utilizada para indicar qual é a *TabSheet* ativa durante a execução, onde o conteúdo é o nome do elemento *TabSheet* selecionado. Se esta propriedade permanecer em branco, a aba ativa será a primeira aba modelada no elemento hospedeiro de acordo com sua posição.

- *TabPosition*: *TabSheetStyle* = *Top*. Indica a posição onde as abas (títulos dos *TabSheet*) serão exibidas. Pode ser *Top*, *Left*, *Bottom* e *Right*.

4.3.5.3 Restrições

1. Este elemento deve possuir pelo menos um elemento *TabSheet*.

2. Somente o elemento *TabSheet* pode ser incluído neste container, nada mais.

4.3.5.4 Notação

A notação deste elemento será apresentada na próxima seção (com o estereótipo *TabSheet*).

4.3.6 *TabSheet*

Estereótipo utilizado para representar um *widget* do tipo *TabSheet*. Estende a classe abstrata *Container*.

4.3.6.1 Descrição

TabSheet é uma área que possui uma aba como título e uma área que serve como container para outros *widgets* (página). De forma resumida, é uma página individual dentro de uma coleção de páginas contidas em um elemento *TabSheetGroup*. Sempre que a aba for selecionada durante sua execução, a página deste elemento será exibida sobre outras páginas de outros elementos *TabSheet* contidos em um *TabSheetGroup*. Caso este elemento for definido com o atributo *Visible* igual a *false*, a página e a guia não serão exibidas. Caso este

elemento for configurado como *false* na propriedade *Enabled*, a página será exibida como somente-leitura, ou seja, todos os elementos contidos nesta página estarão visíveis e com a propriedade *Enabled* igual a *false*, que é uma forma de automaticamente evitar alterações nos conteúdos dos *widgets* contidos neste elemento.

4.3.6.2 Atributos

- *Caption*: *String*. Utilizado para informar o texto de título deste elemento, que será exibido em execução na sua respectiva aba. Caso esse valor não for informado, o *framework* de execução deverá atribuir automaticamente um número de índice para esta aba.

- *IconFilename*: *String*. Utilizado para determinar um ícone, a partir de um arquivo que contenha uma imagem com tipo suportada pelo *framework*, para ser exibido na aba juntamente com o título, desde que isso seja suportado pelo ambiente de execução.

4.3.6.3 Restrições

1. Todo *TabSheet* deve pertencer a um *TabSheetGroup*.
2. Um *TabSheet* não pode conter um *Window*.
3. Um *TabSheet* não pode conter um *TabSheetGroup*.

4.3.6.4 Notação

Este estereótipo é representado visualmente conforme a figura abaixo, onde possui como cabeçalho o nome do estereótipo, o nome do componente definido pelo projetista, um ícone representando este *widget* e uma área interna que poderá ser utilizada como *container* para o aninhamento de outros estereótipos previstos neste projeto. Na figura abaixo é demonstrada um exemplo de utilização deste estereótipo com o estereótipo *TabSheetGroup*.

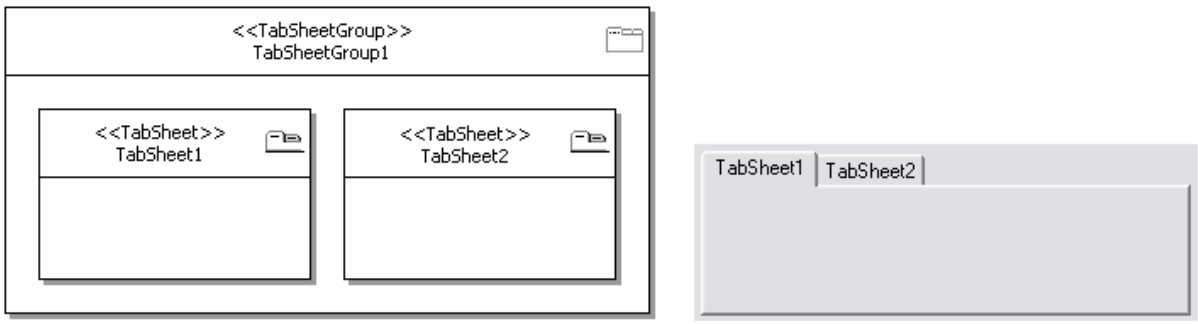


Figura 4.4 – Estereótipo *TabSheetGroup* aninhando estereótipos *TabSheet*, e um exemplo.

4.3.7 *ExtendedContainer*

Estereótipo de extensão utilizado para representar outros *widgetds* do tipo *Container*. Estende a classe abstrata *Container*.

4.3.7.1 Descrição

ExtendedContainer é um estereótipo de extensão para *widgets* com comportamento do tipo *container*, ou seja, pode ser utilizado quando nenhum dos estereótipos previstos no modelo atende às necessidades do projeto. Dependendo da ferramenta de modelagem, pode ser possível adicionar atributos padronizados de acordo com as características deste novo componente para sua integração com o *framework* de execução.

4.3.7.2 Atributos

- *Type*: *String*. Utilizado para informar o tipo do widget que se pretende utilizar. O tipo informado nesta propriedade deverá ser suportado pelo *framework* de execução.

4.3.8 *BoxedWidget*

Estereótipo abstrato que é utilizado como classe ancestral dos elementos que possuem comportamento do tipo *widget* não textual.

4.3.8.1 Descrição

Este estereótipo foi definido como uma classe abstrata, ou seja, não é um elemento que pode ser utilizado diretamente para modelagem da interface gráfica. Comportamento do tipo não textual é o comportamento em que os *widgets* possuem quando o seu conteúdo será definido a partir de um valor definido por sua entidade de representação de dados, e sua representação em execução não necessariamente conterá inicialmente alguma informação textual. Este estereótipo estende a meta-classe *Artifact* para possibilitar o comportamento de um elemento visual retangular limitado, visto que essa característica é oferecida por este elemento do meta-modelo do UML 2.0.

4.3.8.2 Atributos

- *Visible*: *Boolean* = *true*. Utilizado para definir a visibilidade do elemento durante o ciclo de execução do *software*.
- *Enabled*: *Boolean* = *true*. Utilizado para definir se o elemento estará ativo durante o ciclo de execução do *software*. Caso o elemento estiver visível e não ativo, o *framework* poderá exibir o objeto, mas não permitir a interação com os elementos internos deste elemento. Neste caso, os elementos internos estarão disponíveis apenas para visualização.
- *Hint*: *String*. Usado para informar um texto de ajuda de contexto que será exibido quando for passado o foco para este elemento ou quando o ponteiro do *mouse* for posicionado sobre este elemento.

4.3.8.3 Restrições

1. Todo *BoxedWidget* deverá ser incluído dentro de um *Window*, *Panel*, *GroupBox* ou *TabSheet*.

4.3.9 *TextBox*

Estereótipo utilizado para representar um *widget* do tipo *edit box*. Estende a classe abstrata *BoxedWidget*.

4.3.9.1 Descrição

Um *edit box* é um controle de edição de informação textual em uma única linha. Esse controle é utilizado para recuperar e alterar texto informado pelo usuário. Pode também ser utilizado para mostrar texto aos usuários, permitindo operações de copiar, colar e recortar.

4.3.9.2 Atributos

- *Value*: *String*. Valor do conteúdo deste controle.
- *MaxLength*: *Integer*. Número máximo de caracteres possíveis de serem digitados como valor deste controle. Durante a execução, o *framework* deverá impedir a digitação de um número maior do que a quantidade máxima especificada.
- *MinLength*: *Integer* = 0. Número mínimo de caracteres informados como valor válido deste controle. A validação deverá preferencialmente ocorrer quando o *framework* tentar persistir (gravar) esta informação.
- *PasswordMode*: *Boolean* = *false*. Indica se o controle funcionará de forma protegida, ou seja, caso o valor for *true*, o conteúdo será mascarado com a utilização de um caractere especial para esconder o valor da entrada, não permitirá operações de copiar o valor para a área de transferência e, se suportado pelo *framework*, impedirá a interceptação das teclas pressionadas para o sistema operacional, impedindo assim o furto de informação privilegiada a partir de cavalos de tróia ou outros tipos de *softwares* indesejados.
- *Required*: *Boolean* = *true*. Indica se a digitação do conteúdo deste campo é obrigatória. A validação deverá preferencialmente ocorrer quando o *framework* tentar persistir (gravar) esta informação.
- *Mask*: *String*. Utilizado para definir uma máscara para facilitar e também forçar o teor e a formatação do conteúdo digitada. O formato da máscara deverá ser especificado pelo *framework* de execução.
- *FieldName*: *String*. Utilizado para informar qual é o atributo da classe que persiste o conteúdo deste controle. Se o *framework* possibilitar, poderá ser utilizada uma expressão OCL para indicar qual é a classe e atributo relacionado.

4.3.9.3 Notação

Este estereótipo é representado visualmente conforme a figura abaixo, onde possui como cabeçalho o nome do estereótipo, o nome do componente definido pelo projetista e um ícone representando um controle de edição de texto.

Figura 4.5 – Estereótipo *EditBox*

4.3.10 *MemoBox*

Estereótipo utilizado para representar um *widget* do tipo *edit box* com possibilidade de edição de várias linhas de texto. Estende a classe abstrata *BoxedWidget*.

4.3.10.1 Descrição

Este *widget* é um controle para a visualização e a edição de texto com várias linhas, ou seja, permite que seja inserido mais que uma linha de texto, ao contrário do *TextBox* que permite somente uma única linha. Este controle é normalmente utilizado para representar grande quantidade de informação.

4.3.10.2 Atributos

- *Value*: *String*. Valor do conteúdo deste controle.
- *MaxLength*: *Integer*. Número máximo de caracteres possíveis de serem digitados como valor deste controle. Durante a execução, o *framework* deverá impedir a digitação de um número maior do que a quantidade máxima especificada.
- *MinLength*: *Integer* = 0. Número mínimo de caracteres informados como valor válido deste controle. A validação deverá preferencialmente ocorrer quando o *framework* tentar persistir (gravar) esta informação.

- *Required*: *Boolean = true*. Indica se a digitação do conteúdo deste campo é obrigatória. A validação deverá preferencialmente ocorrer quando o *framework* tentar persistir (gravar) esta informação.

- *FieldName*: *String*. Utilizado para informar qual é o atributo da classe que persiste o conteúdo deste controle. Se o *framework* possibilitar, poderá ser utilizada uma expressão OCL para indicar qual é a classe e atributo relacionado.

- *WordWrap*: *Boolean = false*. Indica se, durante a exibição ou a edição do conteúdo deste controle, se o texto será automaticamente movido para a próxima linha caso passar do limite direito do controle. Caso esta propriedade for *false*, serão habilitadas barras de rolagem para permitir a visualização e edição do texto que passar dos limites do controle.

4.3.10.3 Notação

Este estereótipo é representado visualmente conforme a figura abaixo, onde possui como cabeçalho o nome do estereótipo, o nome do componente definido pelo projetista e um ícone representando este *widget*.

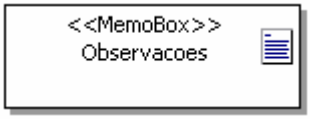


Figura 4.6 – Estereótipo *MemoBox*

4.3.11 *RichBox*

Estereótipo utilizado para representar um *widget* semelhante ao *MemoBox*, com a principal diferença de possibilitar a visualização de conteúdo formatado (rico). Estende a classe abstrata *BoxedWidget*.

4.3.11.1 Descrição

Este *widget* é um controle para a visualização e a edição de texto em várias linhas que inclui variação em formatação de fontes e parágrafos. O tipo de formato do conteúdo é

definido pela propriedade *ContentType* onde, conforme o tipo configurado, o controle deverá exibir durante sua execução uma barra de ferramentas oferecendo botões de atalho para a manipulação da formatação do conteúdo.

4.3.11.2 Atributos

- *Value*: *String*. Valor do conteúdo deste controle.
- *MaxLength*: *Integer*. Número máximo de caracteres possíveis de serem digitados como valor deste controle. Durante a execução, o *framework* deverá impedir a digitação de um número maior do que a quantidade máxima especificada.
- *MinLength*: *Integer* = 0. Número mínimo de caracteres informados como valor válido deste controle. A validação deverá preferencialmente ocorrer quando o *framework* tentar persistir (gravar) esta informação.
- *Required*: *Boolean* = *true*. Indica se a digitação do conteúdo deste campo é obrigatória. A validação deverá preferencialmente ocorrer quando o *framework* tentar persistir (gravar) esta informação.
- *FieldName*: *String*. Utilizado para informar qual é o atributo da classe que persiste o conteúdo deste controle. Se o *framework* possibilitar, poderá ser utilizada uma expressão OCL para indicar qual é a classe e atributo relacionado.
- *ContentType*: *String*. Indica o tipo do conteúdo. O tipo informado deverá ser suportado pelo *framework* de execução, e poderá ser HTML, PDF, RTF, entre outros.

4.3.11.3 Notação

Este estereótipo é representado visualmente conforme a figura abaixo, onde possui como cabeçalho o nome do estereótipo, o nome do componente definido pelo projetista e um ícone representando este *widget.*

Figura 4.7 – Estereótipo *RichBox*

4.3.12 *ComboBox*

Estereótipo utilizado para representar um *widget* do tipo *combo box*. Estende a classe abstrata *BoxedWidget*.

4.3.12.1 Descrição

Este *widget* é um controle para a seleção de um item a partir de uma lista de itens. É semelhante ao *widget TextBox* com uma lista *drop-down* com *scroll* anexada a este controle. Conforme o item selecionado, este controle retornará o índice selecionado conforme sua posição da lista de entrada, onde o primeiro item recebe o valor 0 (zero).

4.3.12.2 Atributos

- *Items*[*]: *String*. É uma lista de valores de entrada. Cada linha de texto será apresentada em uma linha na lista de *drop-down* durante a execução deste controle. Caso o *framework* permitir, será possível configurar aqui uma expressão OCL que ditará de qual classe e atributo buscar o conteúdo desta lista. O atributo, neste caso, deverá retornar um vetor do tipo *String* ou um *enumeration*.

- *SelectedIndex*: *Integer*. Utilizado para pré-configurar um valor de índice selecionado inicial, ou para permitir a troca de estado (do índice selecionado).

- *Required*: *Boolean* = *true*. Indica se a seleção de um item deste campo é obrigatória. A validação deverá preferencialmente ocorrer quando o *framework* tentar persistir (gravar) esta informação.

- *FieldName*: *String*. Utilizado para informar qual é o atributo da classe que persiste o conteúdo deste controle. Se o *framework* possibilitar, poderá ser utilizada uma expressão OCL para indicar qual é a classe e atributo relacionado.

- *Sorted*: *Boolean = false*. Caso o valor desta propriedade for *true*, a lista será classificada em ordem alfabética durante a execução, sem alterar a posição dos índices originalmente configurados.

- *Editable*: *Boolean = false*. Caso o valor desta propriedade for *true*, o controle permitirá a edição do valor da linha selecionada. O *framework* deverá suportar a persistência desta modificação caso for configurada uma expressão OCL na propriedade *Items*.

4.3.12.3 Notação

Este estereótipo é representado visualmente conforme a figura abaixo, onde possui como cabeçalho o nome do estereótipo, o nome do componente definido pelo projetista e um ícone representando este *widget.*

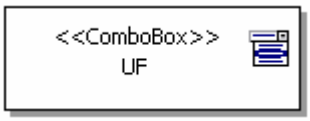


Figura 4.8 – Estereótipo *ComboBox*

4.3.13 *Media*

Estereótipo utilizado para representar um *widget* com suporte a conteúdo multimídia. Estende a classe abstrata *BoxedWidget*.

4.3.13.1 Descrição

Este *widget* pode ser utilizado para exibir imagens, ícones, animações, vídeos e sons. Pode também ser configurado para permitir a troca deste conteúdo. Os tipos suportados por este *widget* dependerão do *framework* de execução, conforme a plataforma de execução.

4.3.13.2 Atributos

- *Value*: *String*[*]. Valor do conteúdo deste controle, que em execução corresponde a um vetor de bytes. Está disponibilizado para permitir trocas de conteúdo, conforme regras que podem ser estipuladas no modelo.

- *Required*: *Boolean = true*. Caso a propriedade *Editable* for *true*, esta propriedade indica que este controle deverá ter conteúdo. A validação deverá preferencialmente ocorrer quando o *framework* tentar persistir (gravar) esta informação.

- *FieldName*: *String*. Utilizado para informar qual é o atributo da classe que persiste o conteúdo deste controle. Se o *framework* possibilitar, poderá ser utilizada uma expressão OCL para indicar qual é a classe e atributo relacionado.

- *ContentType*: *String*. Indica o tipo do conteúdo. O tipo informado deverá ser suportado pelo *framework* de execução, e poderá ser GIF, JPG, ICO, MP3, WAV, MOV, SWF, entre outros.

- *Editable*: *Boolean = false*. Caso o valor desta propriedade for *true*, o controle permitirá a alteração do conteúdo deste controle, conforme seu tipo.

- *Stretch*: *Boolean = false*. Caso o valor desta propriedade for *true* e o conteúdo for uma imagem ou animação, o controle reduzirá imagens maiores do que o tamanho padrão deste controle para que caibam dentro dos limites da área de visualização.

4.3.13.3 Notação

Este estereótipo é representado visualmente conforme a figura abaixo, onde possui como cabeçalho o nome do estereótipo, o nome do componente definido pelo projetista e um ícone representando este *widget.*

Figura 4.9 – Estereótipo *Media*

4.3.14 *Table*

Estereótipo utilizado para representar um *widget* com suporte a conteúdo tabular. Estende a classe abstrata *BoxedWidget*.

4.3.14.1 Descrição

Este *widget* representa um controle em forma de grade para simplificar o manuseio de conteúdos que podem ser apresentados em formato tabular. O tipo de conteúdo pode ser variável, ou seja, para cada célula pode conter tanto texto como vetores de texto, imagens, ou até outras tabelas. Cada célula será exibida em formatação apropriada conforme seu conteúdo.

4.3.14.2 Atributos

- *TotalCols*: *Integer*. Número total de colunas. A configuração deste valor é opcional, pois este tamanho é automaticamente configurado a partir do conteúdo da variável *Titles*. Caso for configurado um valor menor do que o tamanho oferecido por *Titles*, as colunas restantes não serão exibidas.

- *Titles*: String[*]. Esta propriedade pode conter um vetor de *String* para possibilitar a configuração da linha de título da tabela. Pode também ser utilizada uma expressão OCL contendo uma classe e propriedade que retorne o vetor de títulos, ou um enumeration. Caso o valor desta propriedade for vazio, a linha de título não será exibida durante a execução deste controle.

- *Items*: *String*[*]. É uma lista de valores de entrada. Cada item deste vetor será apresentado em uma célula, com quebra de linha automática conforme o número de colunas definida por *Titles* ou por *TotalCols* (esta última somente será utilizada caso *Titles* estiver em branco). Pode também ser utilizada uma

expressão OCL contendo uma classe e propriedade que retorne uma matriz de conteúdo.

- *Editable*: *Boolean = false*. Caso o valor desta propriedade for *true*, o controle permitirá a edição das células da linha selecionada. Cada célula será editada conforme seu conteúdo, ou seja, caso em uma célula existir uma lista, este célula funcionará como uma lista *drop-down*.

- *FieldName*: *String*. Utilizado para informar qual é o atributo da classe que persiste o conteúdo deste controle. Se o *framework* possibilitar, poderá ser utilizada uma expressão OCL para indicar qual é a classe e atributo relacionado. Se for vazia e a propriedade *Items* for uma expressão OCL, o valor de *Items* será utilizado para a persistência do controle.

- *SelectedIndex*: *Integer*. Utilizado para pré-configurar um valor inicial do índice da linha selecionada, ou para permitir a troca de estado deste índice. A primeira linha corresponde ao índice de valor 0 (zero).

4.3.14.3 Notação

Este estereótipo é representado visualmente conforme a figura abaixo, onde possui como cabeçalho o nome do estereótipo, o nome do componente definido pelo projetista e um ícone representando este *widget.*

Figura 4.10 – Estereótipo *Table*

4.3.15 *Gauge*

Estereótipo utilizado para representar um *widget* do tipo *gauge*. Estende a classe abstrata *BoxedWidget*.

4.3.15.1 Descrição

O *widget gauge* é um controle para mostrar informação quantitativa em um formato visual fácil de compreender. Apresenta a informação em porcentagem. Este controle não permite a alteração do valor, pois é utilizado somente para visualização.

4.3.15.2 Atributos

- *Style*: *GaugeStyle* = *HorizontalBar*. Define o formato de representação. Pode ser configurado como *HorizontalBar* (barra horizontal), *VerticalBar* (barra vertical), *Pie* (formato de torta), *Needle* (formato de agulha) ou *Text* (texto).
- *Progress*: *Integer* = 0. Valor numérico do controle. Preferencialmente, deve variar de 0 a 100, embora que valores abaixo ou acima disso podem ser suportados pelo *framework*.
- *ShowText*: *Boolean* = *true*. Caso o *Style* for configurado para uma representação gráfica, esta propriedade indicará a existência conjunta do texto indicando o valor do atributo *Progress*. Este valor será apresentado em percentual, ou seja, com o caractere “%” após o valor.
- *FieldName*: *String*. Utilizado para informar qual é o atributo da classe que indica o conteúdo numérico deste controle.

4.3.15.3 Notação

Este estereótipo é representado visualmente conforme a figura abaixo, onde possui como cabeçalho o nome do estereótipo, o nome do componente definido pelo projetista e um ícone representando este *widget*.

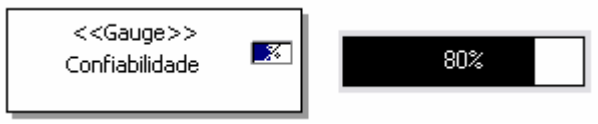


Figura 4.11 – Estereótipo *Gauge* e um exemplo do *widget* relacionado

4.3.16 *TrackBar*

Estereótipo utilizado para representar um *widget* do tipo *track bar*. Estende a classe abstrata *BoxedWidget*.

4.3.16.1 Descrição

O *widget TrackBar* é um controle utilizado para visualizar e alterar um valor quantitativo limitado. Durante execução, é exibida uma barra com um ponteiro que pode ser deslocado para aumentar ou diminuir o valor numérico.

4.3.16.2 Atributos

- *MinValue*: *Integer* = 0. Valor mínimo. Será o valor configurado caso o ponteiro for deslocado totalmente à esquerda.
- *MaxValue*: *Integer* = 10. Valor máximo. Será o valor configurado caso o ponteiro for deslocado totalmente à direita.
- *Orientation*: *TrackBarOrientation* = *Horizontal*. Orientação de visualização do controle. Pode ser *Horizontal* ou *Vertical*.
- *Value*: *Integer*. Valor numérico do conteúdo para efeitos de inicialização ou troca de estado. Deve ser um valor compreendido entre o valor mínimo e o valor máximo.
- *ShowText*: *Boolean* = *false*. Se o valor for *true*, será exibido o valor atual de forma centralizada logo abaixo ou à direita do componente.

- *FieldName*: *String*. Utilizado para informar qual é o atributo da classe que persiste o conteúdo deste controle. Se o *framework* possibilitar, poderá ser utilizada uma expressão OCL para indicar qual é a classe e atributo relacionado.

4.3.16.3 Notação

Este estereótipo é representado visualmente conforme a figura abaixo, onde possui como cabeçalho o nome do estereótipo, o nome do componente definido pelo projetista e um ícone representando este *widget.*

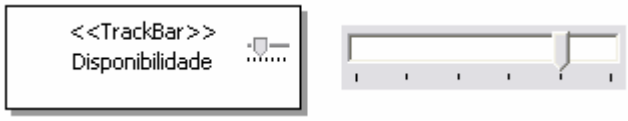


Figura 4.12 – Estereótipo *TrackBar* e um exemplo do *widget* relacionado

4.3.17 *ExtendedBox*

Estereótipo de extensão utilizado para representar outros *widgetds* do tipo *BoxedWidget*. Estende a classe abstrata *BoxedWidget*.

4.3.17.1 Descrição

ExtendedBox é um estereótipo de extensão para *widgets* com comportamento do tipo *widget* não textual, ou seja, pode ser utilizado quando nenhum dos estereótipos previstos no modelo atende às necessidades do projeto. Dependendo da ferramenta de modelagem, pode ser possível adicionar atributos padronizados de acordo com as características deste novo componente para sua integração com o *framework* de execução.

4.3.17.2 Atributos

- *Type*: *String*. Utilizado para informar o tipo do *widget* que se pretende utilizar. O tipo informado nesta propriedade deverá ser suportado pelo *framework* de execução.

4.3.18 *TextualWidget*

Estereótipo abstrato que é utilizado como classe ancestral dos elementos que possuem texto relativo a seu uso no modelo.

4.3.18.1 Descrição

Este estereótipo foi definido como uma classe abstrata, ou seja, não é um elemento que pode ser utilizado diretamente para modelagem da interface gráfica. Esta classe é utilizada para *widgets* que contenham texto associado ao elemento na visualização do modelo. Este estereótipo estende a meta-classe *Note* para possibilitar o comportamento de um elemento textual retangular, visto que essa característica é oferecida por este elemento do meta-modelo do UML 2.0.

4.3.18.2 Atributos

- *Name*: String. Utilizado para definir opcionalmente um nome ao elemento, visto que a meta-classe *Note* não possui este atributo.
- *Visible*: *Boolean = true*. Utilizado para definir a visibilidade do elemento durante o ciclo de execução do *software*.
- *Enabled*: *Boolean = true*. Utilizado para definir se o elemento estará ativo durante o ciclo de execução do *software*. Caso o elemento estiver visível e não ativo, o *framework* irá exibir o objeto sem permitir a alteração do conteúdo.
- *UseAcceleratorChar*: *Boolean = false*. Caso o valor for *true*, qualquer caractere do texto precedido do caractere “&” aparecerá sublinhado, e este caractere será utilizado durante a execução do *software* como tecla de atalho para desviar o foco no controle correspondente.

4.3.18.3 Restrições

1. Todo *TextualWidget* deverá ser incluído dentro de um *Window*, *Panel*, *GroupBox* ou *TabSheet*.

4.3.19 *RadioButton*

Estereótipo utilizado para representar um *widget* do tipo *radio button*. Estende a classe abstrata *TextualWidget*.

4.3.19.1 Descrição

Este *widget* apresenta um conjunto de opções exclusivas para o usuário, ou seja, apenas um *RadioButton* de um conjunto pode ser marcado por vez. Quando o usuário selecionar outro *RadioButton* de um mesmo conjunto, o último controle marcado será automaticamente desmarcado. Para formar um conjunto deve-se adicionar mais que um *widget RadioButton* em uma mesma janela, todos com o mesmo valor definido para o atributo *FieldName*.

4.3.19.2 Atributos

- *Hint*: *String*. Texto da ajuda de contexto, que será exibida quando este controle ganhar foco ou quando o ponteiro do *mouse* estiver sobre este componente durante a execução.
- *Checked*: *Boolean* = *false*. Determina se este controle está marcado ou não.
- *CheckedValue*: *String*. Valor do conteúdo a ser persistido caso o controle estiver marcado.
- *FieldName*: *String*. Utilizado para informar qual é o atributo da classe que persiste o conteúdo deste controle. Se o *framework* possibilitar, poderá ser utilizada uma expressão OCL para indicar qual é a classe e atributo relacionado.
- *Required*: *Boolean* = *true*. Indica se marcação do conteúdo deste campo é obrigatória. A validação deverá preferencialmente ocorrer quando o *framework* tentar persistir (gravar) esta informação. Caso este controle formar um conjunto, a existência do valor *true* em pelo menos um dos controles forçará todos os controles do conjunto com o valor *true*.

4.3.19.3 Notação

Este estereótipo é representado visualmente conforme a figura abaixo, onde possui como cabeçalho o nome do estereótipo, a informação textual relativa ao componente, definida pelo projetista, e um ícone representando este *widget*.

Figura 4.13 – Estereótipo *RadioButton*

4.3.20 *CheckBox*

Estereótipo utilizado para representar um *widget* do tipo *check box*. Estende a classe abstrata *TextualWidget*.

4.3.20.1 Descrição

Este *widget* representa um controle que pode estar ligado (checado) ou desligado (não checado).

4.3.20.2 Atributos

- *Hint*: *String*. Texto da ajuda de contexto, que será exibida quando este controle ganhar foco ou quando o ponteiro do *mouse* estiver sobre este componente durante a execução.
- *Checked*: *Boolean* = *false*. Determina se este controle está marcado ou não.
- *CheckedValue*: *String*. Valor opcional do conteúdo a ser persistido caso o controle estiver marcado.
- *FieldName*: *String*. Utilizado para informar qual é o atributo da classe que persiste o conteúdo deste controle. Se o *framework* possibilitar, poderá ser utilizada uma expressão OCL para indicar qual é a classe e atributo relacionado.

4.3.20.3 Notação

Este estereótipo é representado visualmente conforme a figura abaixo, onde possui como cabeçalho o nome do estereótipo, a informação textual relativa ao componente, definida pelo projetista, e um ícone representando este *widget*.

Figura 4.14 – Estereótipo *CheckBox*

4.3.21 *Button*

Estereótipo utilizado para representar um *widget* do tipo botão. Estende a classe abstrata *TextualWidget*.

4.3.21.1 Descrição

Este *widget* representa um botão de comando. Caso este botão for acionado (pressionado), o *framework* realizará a ação prevista na propriedade *Execute*.

4.3.21.2 Atributos

- *DefaultButton*: *Boolean* = *false*. Determina se este controle é o botão padrão da janela, ou seja, o botão que será executado caso o usuário pressionar a tecla *Enter*. Normalmente o controle é exibido com sombreamento em destaque quando esta propriedade possuir valor igual a *true*.

- *Hint*: *String*. Texto da ajuda de contexto, que será exibida quando este controle ganhar foco ou quando o ponteiro do *mouse* estiver sobre este componente durante a execução.

- *IconFilename*: *String*. Utilizado para determinar um ícone para ser exibido juntamente com o texto do botão, a partir de um arquivo que contenha uma imagem com formato suportado pelo *framework* de execução.

- *UseAcceleratorChar*: *Boolean = false*. Caso o valor for *true*, qualquer caractere do texto precedido do caractere “&” aparecerá sublinhado, e este caractere será utilizado durante a execução do *software* como tecla de atalho para pressionar o botão.

- *Execute*: *String*. Utilizado para informar um texto para indicar qual o método de execução quando o botão for pressionado. Pode ser uma expressão OCL. A forma em que o valor deste campo deverá ser informada será atrelada ao *framework* de execução.

4.3.21.3 Notação

Este estereótipo é representado visualmente conforme a figura abaixo, onde possui como cabeçalho o nome do estereótipo, a informação textual relativa ao componente, definida pelo projetista, e um ícone representando este *widget*.

Figura 4.15 – Estereótipo *Button*

4.3.22 *Label*

Estereótipo utilizado para representar um *widget* do tipo texto não-editável. Estende a classe abstrata *TextualWidget*.

4.3.22.1 Descrição

Este *widget* representa uma área com texto, sem a possibilidade de foco ou de alteração do texto por parte do usuário. Entretanto, é possível alterar o texto durante a execução por parte do *framework* de execução ou por regras definidas no modelo.

4.3.22.2 Atributos

- *Style*: *String*. Determina opcionalmente o estilo da fonte e da formatação do texto. A forma em que o valor deste campo deverá ser informada será atrelada ao *framework* de execução.

- *Alignment*: *AlignmentStyle* = *Left*. Define o alinhamento do texto. O tipo *AlignmentStyle* é um *enumeration* definido no modelo com os valores *Left*, *Center*, *Right* e *Justify*.

4.3.22.3 Notação

Este estereótipo é representado visualmente conforme a figura abaixo, onde possui como cabeçalho o nome do estereótipo, a informação textual relativa ao componente, definida pelo projetista, e um ícone representando este *widget*.

Figura 4.16 – Estereótipo *Label*

4.3.23 *LinkedLabel*

Estereótipo utilizado para representar um *widget* do tipo texto *hyperlink* não-editável. Estende a classe abstrata *TextualWidget*.

4.3.23.1 Descrição

Este *widget* representa uma área com texto do tipo *hyperlink*, ou seja, caso for acionado (pressionado) qualquer parte do texto, o *framework* realizará a ação prevista na propriedade *Execute*. Este controle pode receber foco, mas não permite alteração do texto por parte do usuário. Entretanto, é possível alterar o texto durante a execução por parte do *framework* de execução ou por regras definidas no modelo.

4.3.23.2 Atributos

- *Style*: *String*. Determina opcionalmente o estilo da fonte e da formatação do texto. A forma em que o valor deste campo deverá ser informada será atrelada ao *framework* de execução.

- *Alignment*: *AlignmentStyle* = *Left*. Define o alinhamento do texto. O tipo *AlignmentStyle* é um *enumeration* definido no modelo com os valores *Left*, *Center*, *Right* e *Justify*.

- *Execute*: *String*. Utilizado para informar um texto para indicar qual o método de execução quando o texto deste controle for pressionado. Pode ser uma expressão OCL. A forma em que o valor deste campo deverá ser informada será atrelada ao *framework* de execução.

- *Hint*: *String*. Texto da ajuda de contexto, que será exibida quando este controle ganhar foco ou quando o ponteiro do *mouse* estiver sobre este componente durante a execução.

4.3.23.3 Notação

Este estereótipo é representado visualmente conforme a figura abaixo, onde possui como cabeçalho o nome do estereótipo, a informação textual relativa ao componente, definida pelo projetista, e um ícone representando este *widget*.

Figura 4.17 – Estereótipo *LinkedLabel*

4.3.24 *ExtendedText*

Estereótipo de extensão utilizado para representar outros *widgets* do tipo *TextualWidget*. Estende a classe abstrata *TextualWidget*.

4.3.24.1 Descrição

ExtendedText é um estereótipo de extensão para *widgets* com comportamento do tipo *widget* textual, ou seja, pode ser utilizado quando nenhum dos estereótipos previstos no modelo atende às necessidades do projeto. Dependendo da ferramenta de modelagem, pode ser possível adicionar atributos padronizados de acordo com as características deste novo componente para sua integração com o *framework* de execução.

4.3.24.2 Atributos

Type: *String*. Utilizado para informar o tipo do *widget* que se pretende utilizar. O tipo informado nesta propriedade deverá ser suportado pelo *framework* de execução.

4.4 Exemplos de Utilização

Para ilustrar a utilização do UGUI *Profile*, aqui é apresentado dois exemplos de utilização, onde, para cada exemplo, foi utilizado como origem uma interface gráfica de um software existente, e o destino é o modelo da réplica da interface gráfica utilizando os elementos propostos neste trabalho.

Para o primeiro exemplo de utilização do UGUI *Profile* foi utilizado a interface gráfica de formatação de parágrafo do *WordPad* da Microsoft (Figura 4.18).

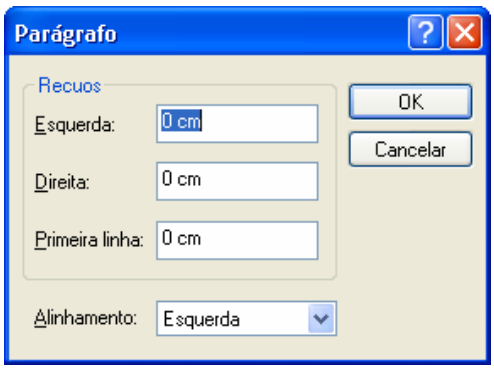


Figura 4.18 - GUI de formatação de parágrafo

Fonte: Microsoft WordPad 5.1

Esta interface gráfica basicamente possui uma janela modal, um *widget group box*, dois botões, vários rótulos de texto, vários *widgets text box* e um *widget combo box*. A réplica desta interface (o modelo) é apresentada na figura abaixo.

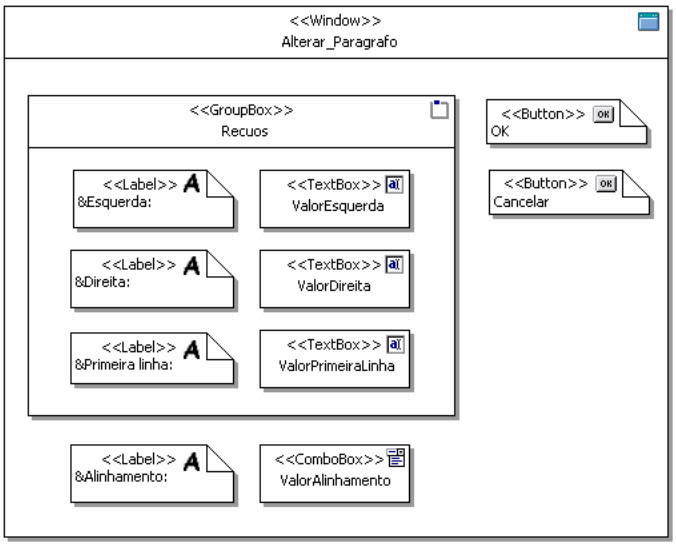


Figura 4.19 – Modelo do GUI de formatação de parágrafo

Fonte: Elaborado pelo autor

Como segundo exemplo, foi utilizado uma interface gráfica do sistema *Remote Banking* do Banrisul (Figura 4.20), que é um sistema WEB. O modelo criado a partir desta interface está no apêndice B (devido a seu tamanho).

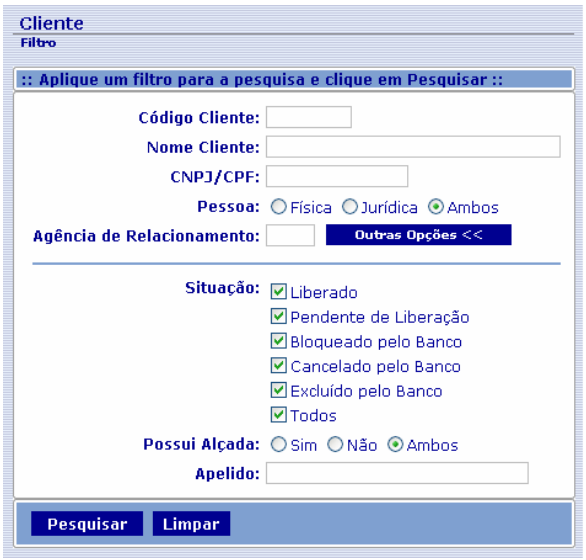


Figura 4.20 – GUI de filtro de pesquisa de cliente

Fonte: *Remote Banking* - BANRISUL

5 AVALIAÇÃO DO MODELO

Para avaliar e testar o modelo e a abordagem escolhida, foi utilizado duas ferramentas de modelagem comerciais que suportam a criação e a utilização de *profiles*, as quais serão apresentadas nas próximas seções. Para cumprir este objetivo, o modelo apresentado neste documento foi programado em cima dessas ferramentas para, em seguida, simular a modelagem de interfaces gráficas utilizando os elementos definidos na seção 4.3.

5.1 *Together Architect* 2006

A ferramenta de modelagem *Together Architect* 2006 (BORLAND, 2006) suporta a criação de *profiles* UML. Tem suporte de especificação visual do *profile* de forma muito semelhante à forma proposta pela OMG. É possível configurar um estereótipo para ter sua própria representação gráfica utilizando um arquivo gráfico no formato SVG 1.1, que é um padrão gráfico da OMG onde a imagem é especificada com o uso de um arquivo XML. Além disso, é possível associar um ícone para a representação do elemento na paleta de componentes.

Esta ferramenta também suporta a especificação de restrições em OCL com o auxílio de um editor específico para esta linguagem. Porém, a validação sistemática das expressões escritas nesta linguagem ainda não é suportada em tempo de definição do *profile*, somente são suportadas durante a utilização (instanciação) do *profile* em si.

Durante a avaliação com esta ferramenta, foram encontrados problemas referentes à utilização de imagens e ícones utilizados para especializar a notação visual dos elementos.

Conforme padronizado pela OMG:

> Quando um estereótipo possui em sua definição um ícone, este ícone pode ser graficamente anexado ao elemento do modelo estendido pelo estereótipo. Todo elemento do modelo que tiver uma representação gráfica pode ter um ícone anexado. Quando os elementos do modelo forem graficamente expressos de:
>
> - Caixas: a caixa pode ser substituída pelo ícone, e o nome do elemento do modelo aparece debaixo do ícone. Esta forma de apresentação pode ser utilizada somente quando um modelo do elemento for extensão de um único estereótipo e quando as propriedades do elemento do modelo (isto é, atributos, operações de uma classe) não são apresentadas. Como outra opção, o ícone pode ser apresentado em uma forma reduzida, dentro e no topo da caixa que representa o elemento do modelo. Quando vários estereótipos são aplicados, vários ícones podem ser apresentados dentro da caixa.
> - Links: o ícone pode ser colocado próximo ao link.
> - Notação textual: o ícone pode ser apresentado à esquerda da notação textual.
>
> (Citação traduzida de Unified Modeling Language: Superstructure. OMG, Versão 2.0, formal/05-07-04, p. 666, 2006)

Um dos problemas é que, ao utilizar um estereótipo que estende a meta-classe *Component* (para prover o comportamento do tipo *container*) que contenha uma imagem associada, não é possível visualizar os elementos aninhados de forma gráfica, apesar de que é possível verificar que os elementos continuam obedecendo a relação hierárquica aos quais foram adicionados. Este aspecto particular de comportamento não é explicitamente previsto na Superestrutura e nem na Infra-estrutura das atuais versões do UML (OMG, 2006), o qual poderia ser previsto em futuras revisões destes documentos.

Outro problema é que, ao utilizar um estereótipo que não seja do tipo *container*, e que contenha uma imagem associada, não é apresentado o nome do elemento, comportamento que é previsto na Superestrutura e na Infra-estrutura das atuais versões do UML (OMG, 2006) ao utilizar imagens em um estereótipo. Inclusive, para estereótipos com notação textual, a utilização de uma imagem impede que a informação textual seja exibida.

Uma forma de ignorar estes problemas é não utilizar imagens nos estereótipos, fato que prejudica a utilização do *profile*, visto que a notação visual especificada tem como objetivo facilitar a visualização e a compreensão dos *widgets* modelados. O apêndice C demonstra uma imagem do *Together* sendo utilizado com estas limitações.

5.2 *Enterprise Architect* 6.1

A ferramenta de modelagem *Enterprise Architect* 6.1 (SPARX SYSTEMS, 2006) também suporta a criação de *profiles* UML. Tem suporte de especificação visual do *profile* de forma semelhante ao *Together Architect*, ou via edição manual de um arquivo XML obedecendo a regras proprietárias estipuladas pela ferramenta. É possível configurar um estereótipo para ter sua própria imagem utilizando um arquivo gráfico EMF ou WMF, desde que seja especificada diretamente no arquivo XML.

Esta ferramenta também suporta a especificação de restrições em OCL. Porém, não existe um mecanismo para validação e auxílio da escrita de expressões nesta linguagem, diferentemente do *Together*.

De forma semelhante ao *Together*, esta ferramenta também apresenta problemas de visualização referentes à utilização de imagens e ícones utilizados para especializar a notação visual dos elementos. A única ressalva é que neste caso é apresentado o nome do elemento na parte superior de forma centralizada, comportamento semelhante à forma padronizada pela Superestrutura do UML 2.0 (OMG, 2006).

Existe outra forma de associar uma imagem nesta ferramenta, via recurso proprietário *Shape Script*, que é uma forma de definir linhas, caixas, elipses, entre outras, via programação utilizando uma linguagem proprietária chamada de *EAShapeScript* 1.0. Infelizmente, este recurso não possibilita a exibição do nome do elemento, ou seja, não resolve o problema.

Portanto, para ignorar estes problemas, é recomendado a não utilização de imagens até que estes problemas sejam solucionados em futuras versões desta ferramenta.

Outro problema desta ferramenta é que a meta-classe *Component*, quando utilizada como *container*, ou seja, ao incluir um *Component* dentro de outro *Component*, é possível verificar que estes elementos foram adicionados em uma relação hierárquica (um dentro do outro) pela exibição dos elementos no *Project View*, mas no diagrama o elemento filho fica inicialmente em cima do elemento pai, e a ferramenta permite movê-lo para fora dos limites do elemento pai sem alterar a estrutura hierárquica. Este comportamento pode induzir a erros de modelagem. Este problema não ocorre no *Together*.

6 CONCLUSÃO

O modelo apresentado neste documento oferece um recurso para possibilitar a modelagem de interfaces gráficas com aspectos de usabilidade para ambientes MDD, mais precisamente para a camada PIM do MDA, pois é independente de tecnologia, possibilita modelar o comportamento básico dos *widgets* ao serem executados, prevê propriedades para serem utilizadas para a validação, persistência e apresentação dos dados, e permite construir o diagrama de forma aproximada aos elementos que compõem as atuais interfaces gráficas.

Todos os requisitos propostos (seção 4.1) foram satisfeitos, pois:

1. O *profile* permite modelar o *layout* (disposição) de uma interface gráfica, pois foram criados estereótipos que permitem aninhar outros estereótipos;

2. O *profile* é fácil de aprender, utilizar e entender, pois os estereótipos são semelhante aos *widgets* disponibilizados em ferramentas de programação;

3. O *profile* permite a criação de um digrama em poucos minutos utilizando ferramentas especializadas de modelagem. Isso foi constatado durante a avaliação em cima das ferramentas de modelagem;

4. A criação de um diagrama não exige muito trabalho adicional, pois basta criar o elemento, nomeá-lo e definir sua posição, e configuração da maioria dos atributos de cada elemento são opcionais, economizando tempo;

5. A simbologia utilizada nos *widgets* representa, através de ícones, os elementos (widgets) utilizados em interfaces gráficas;

6. O *profile* oferece formas de extensão nos estereótipos *ExtendedContainer*, *ExtendedBox* e *ExtendedText*;

7. O modelo seguiu as premissas do UML de ser simples e compreensível, e permite a impressão do diagrama em papel sem perder informações essenciais para tomadas de decisão;

8. O modelo é independente de tecnologia;

9. A utilização de OCL foi prevista nos *widgets* em atributos específicos, como por exemplo o atributo *FieldName* do *widget TextBox*.

10. O modelo foi criado e avaliado em ferramentas de modelagem que suportam a criação e utilização de *profiles* de acordo com a especificação da OMG.

Com a avaliação do modelo em cima de ferramentas de modelagem foram encontrados problemas referentes à utilização de imagens nos estereótipos e da utilização de restrições OCL. Portanto, conclui-se que existe uma necessidade de amadurecimento das ferramentas quanto a estes aspectos, além de que é recomendável reforçar a padronização da OMG para evitar estes tipos de problemas, já que estas especificações não previram explicitamente estes tipos de necessidades.

Mesmo assim, é possível utilizar o UGUI *Profile* nestas ferramentas de modelagem sem a utilização da notação gráfica (ícones), pois estas ferramentas exibem o nome do estereótipo em cada elemento.

6.1 Trabalhos futuros

Como trabalhos futuros relacionados, podem-se sugerir como destaque avaliar a utilização deste *profile* durante a construção de um *software* real, preferencialmente utilizando tecnologias e ferramentas MDA. Outras sugestões são:

- Especificar e avaliar um modelo semelhante ao proposto neste documento utilizando a linguagem MOF, ou seja, sem utilizar o recurso de *profiles*, objetivando diminuir as limitações e problemas encontrados (na seção 0) com a utilização da abordagem seguida neste trabalho;

- Especificar e avaliar um modelo semelhante ao proposto neste documento utilizando DSL, comparando benefícios entre esta e abordagem apresentada neste trabalho;

- Avaliar e sugerir modificações nos padrões da OMG referentes a utilização de imagens e ícones nos elementos de um *profile*, visando eliminar os problemas que foram encontrados durante a avaliação do modelo em cima de ferramentas de modelagem;

- Avaliar e sugerir correções para as ferramentas de modelagem utilizadas na avaliação do modelo de forma que sigam corretamente e eficientemente a padronização estabelecida pela OMG;

- Especificar e construir um *framework* de execução para interfaces gráficas modeladas a partir deste *profile*. Para isso, pode-se recomendar a utilização da ferramenta de modelagem *Together* integrada ao ECO III (BORLAND, 2006) como camada de execução, suíte de componentes que é distribuída com as atuais versões do *Delphi* da *Borland*;

- Realizar a conversão das regras de restrição deste modelo utilizando OCL, e, em seguida, efetivar validações em cima de uma ferramenta de modelagem que suporte restrições expressas nesta linguagem;

- Complementar este trabalho, sugerindo recursos para a navegação a partir da interface gráfica modelada, interação entre outros diagramas, relação entre as interfaces gráficas e os casos de uso, entre outros.

REFERÊNCIAS

SANDRI, A. **Viabilidade de Construção de Software com MDD e MDA**. Artigo de Pesquisa em Ciência da Computação. UNILASALLE, 2005.

OMG 2006. **OMG Web Site**, Object Management Group. Disponível em <http://www.omg.org>. Acesso em 17 jun. 2006.

SCHMIDT, D. **Model-Driven Engineering**. IEEE Computer Society. Fev. 2006.

HAMILTON, K., MILES, R. **Learning UML 2.0**. UK, O'Reilly, 2006. 286 páginas.

SHNEIDERMAN, B., PLAISANT, C. **Designing the user interface**: strategies for effective human computer interaction. Addison-Wesley: 4 ed., 652p, 2004.

GOULD, D., LEWIS, C. **Designing for Usability**: Key Principles and what designers think. Communications of the ACM, 3 mar. 1985, p. 300-311. Disponível em <http://doi.acm.org/10.1145/3166.3170>. Acesso em 17 jun. 2006.

MICROSOFT. **Microsoft Windows User Experience**. Microsoft Professional Editions. Microsoft Press. Set. 1999.

MICROSOFT. **Official Guidelines for User Interface Developers and Designers**. Disponível em <http://msdn.microsoft.com/library/>. Acesso em 17 jun. 2006.

MICROSOFT. **Microsoft Inductive User Interface Guidelines**. Microsoft Press. 9 fev. 2001.

BERRY, R. E**. Common User Access** - A consistent and usable human-computer interface for the SAA environments. IBM SYSTEMS JOURNAL, VOL 27, NO 3. 1988. Pg 281. Disponível em <http://www.research.ibm.com/journal/sj/273/ibmsj2703E.pdf>. Acesso em 17 jun. 2006.

IBM. **CUA Basic Interface Design Guide**. Disponível em <http://publibfp.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/f29bdg00/CCONTENTS>. Acesso em 17 jun. 2006.

IBM 2006. **IBM Web Site**. IBM. Disponível em <http://www.ibm.com/>. Acesso em 17 jun. 2006.

HENNICKER, R. KOCH, N. **Modeling the User Interface of Web Applications with UML**. Institute of Computer Science, Ludwig-Maximilians-University of Munich, Alemanha, 2001.

BLANKENHORN, K. **A UML Profile for GUI Layout**. Master's Thesis. University of Applied Sciences. Furtwangen, 2004.

SEARCHWEBSERVICES 2004. **Research: Model-driven development increases productivity.** SearchWebServices.com.

MDDI 2005, **Model Driven Development Integration. A Technology Project Proposal.** Eclipse Group. Disponível em <http://www.eclipse.org/proposals/eclipse-mddi/main.html>. Acesso em 17 jun. 2006.

KLEPPE, A., WARMER, J., BAST, W. **MDA Explained: The Model Driven Architecture™**: Practice and Promise. Addison Wesley, 2003.

QVT. **Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification**. OMG, Final Adopted Specification, ptc/05-11-01, 2 nov. 2005.

WARMER, J., KLEPPE, A. **Object Constraint Language, Getting Your Models Ready for MDA**, Second Edition, Addison Wesley, 2004.

SENDALL, S., KOZACZYNSKI, W. **Model Transformation – the Heart and Soul of Model-Driven Software Development**. Swiss Federal Institute of Technology in Lausanne, 2003.

BORLAND, **Together Architect 2006**. Cupertino, EUA. Disponível em <http://www.borland.com/br/products/together/index.html >. Acesso em 17 jun. 2006.

BORLAND. **Borland Web Site**. Cupertino, EUA. Disponível em <http://www.borland.com/>. Acesso em 17 jun. 2006.

SPARX SYSTEMS, **Enterprise Architect 6.1**. Creswick, Austrália. Disponível em <http://www.sparxsystems.com/products/ea.html>. Acesso em 17 jun. 2006.

COMPUWARE 2005. **OptimalJ 4.0**. Compuware. Disponível em <http://www.compuware.com/>. Acesso em 17 jun. 2006.

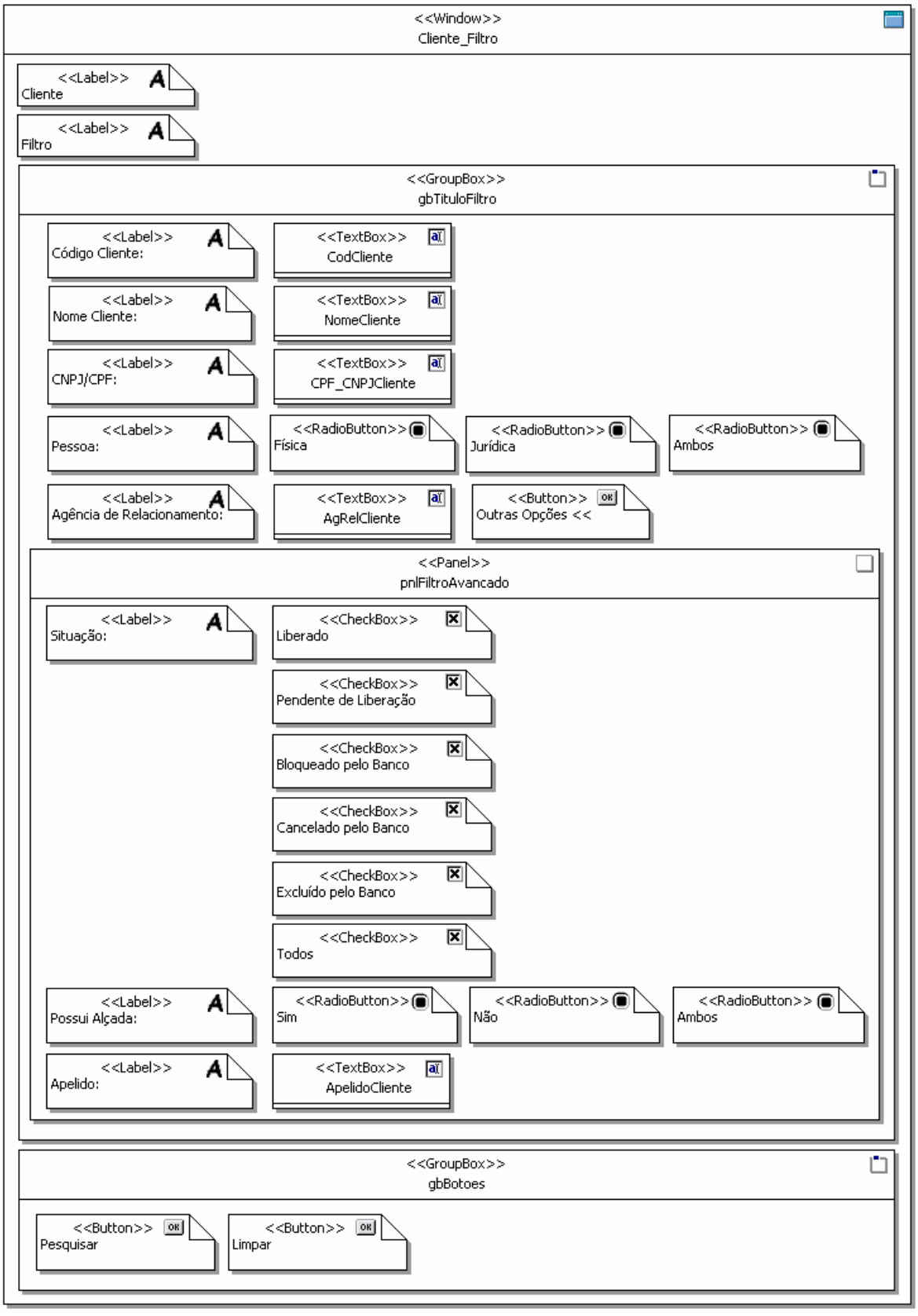
TELELOGIC. **Telelogic Web Site**. Malmö, Suécia. Disponível em <http://www.telelogic.com/>. Acesso em 17 jun. 2006.

ISO/IEC 9126. **Information Technology – Software Product Evaluation**: Quality Characteristics and Guidelines for their use. 1991.

ISO 9241. **Ergonomic Requirements For Office Work With Visual Display Terminals**: Part 11 – Guidelines on Usability. 1998.

APÊNDICE A – DIAGRAMA DO UGUI PROFILE

APÊNDICE B – EXEMPLO DE UTILIZAÇÃO



APÊNDICE C – UGUI PROFILE NO TOGETHER