UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

WILANE CARLOS DA SILVA

Gerência de Interfaces para Sistemas de Informação: uma abordagem baseada em modelos

Goiânia
2010

WILANE CARLOS DA SILVA

Gerência de Interfaces para Sistemas de Informação: uma abordagem baseada em modelos

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em Mestrado em Ciência da Computação.

Área de concentração: Engenharia de Software.

Orientador: Prof. Juliano Lopes de Oliveira

Goiânia
2010

WILANE CARLOS DA SILVA

Gerência de Interfaces para Sistemas de Informação: uma abordagem baseada em modelos

Dissertação defendida no Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Mestre em Mestrado em Ciência da Computação, aprovada em 16 de Abril de 2010, pela Banca Examinadora constituída pelos professores:

Prof. Juliano Lopes de Oliveira
Instituto de Informática – UFG
Presidente da Banca

Prof. Hugo Alexandre Dantas do Nascimento
Universidade Federal de Goiás – UFG

Prof. Gleison dos Santos Souza
Universidade Federal do Estado do Rio de Janeiro – UNIRIO

Wilane Carlos da Silva

Graduou–se em Ciências da Computação na UFG - Universidade Federal de Goiás. Durante sua graduação, foi pesquisador do CNPq em um trabalho de iniciação científica no INF/UFG - Instituto de Informática/UFG, sobre Representação de Documentos Acadêmicos em uma rede de Alta Velocidade. Durante o mestrado, na UFG, foi bolsista do CNPq e desenvolveu um trabalho teórico e prático sobre Geração Dinâmica de Interfaces Gráficas Baseadas em Modelos. Atualmente encontra-se vinculado à UFG como servidor público federal, enquadrado funcionalmente como Analista de TI.

Aos meus avós, Divino da Silva, cuja lembrança estará sempre em minha memória, e Maria Furtado da Silva, por se constituírem e agirem diferentemente enquanto pessoas, por guiarem de forma sábia, única e honrosa os filhos, netos e quem deles precisasse.

Agradecimentos

Primeiramente à Deus pela saúde, oportunidade e persistência para lutar.

À minha família pela motivação de seguir em frente.

Especialmente à minha mãe, Ana Helena da Silva, uma heroína cujos exemplos de dedicação e esforço mantiveram minha cabeça erguida nos momentos mais difíceis e à minha madrinha, na prática, Geralda Maria da Silva, por sempre ter me mostrado que o estudo é a base de tudo.

A nosso Orientador Prof. Juliano Lopes de Oliveira pelo incentivo e presteza no auxílio às atividades e discussões no desenvolvimento deste trabalho.

Aos professores do Instituto de Informática da Universidade Federal de Goiás pela aprendizagem.

Particularmente ao Professor Dr. Eduardo Simões de Albuquerque, pessoa que sempre orientou-me com muita sabedoria e simplicidade, acreditando em meu potencial e sempre disposto a ajudar.

A todos os funcionários do Instituto de Informática, especialmente ao João Bosco Carmo Moraes pelas brincadeiras e discontrações, ao Enio Perez Rodrigues Barbosa pela gentileza e prontidão, Edir de Jesus Borges Pinto pela eficiência e conselhos, e Justiniana César da Silva pela presteza.

Carinhosamente à Berenice Vieira Silva (Berê), também funcionária do INF e, muito mais que isso, a "mãezona" durante a graduação e mestrado que tanto ajudou, não só a mim, mas a todos os alunos da computação, por tratar-nos como seus filhos.

Aos amigos de mestrado Alexandre Cláudio de Almeida, Glauber Boff, Patrícia Fernandes, Fabiana Freitas Mendes, Victor Ribeiro e Elisângela Silva Dias pelas brincadeiras, coleguismo, incentivos, ajudas e motivações.

A todos os funcionários do Cercomp - Centro de Recursos Computacionais da UFG, especialmente aos amigos Dannyel Cardoso da Fonseca, Pablo Leonardo Mendes da Cruz Lima, Euler Robério Sena Santos, Daiany de Castro Vieira, Rosângela Divina de Sousa Santana e Vívia Bessa de Oliveira Soares pela ajuda na coleta de dados para o trabalho, pelos risos, motivação e pelo apoio incondicional.

Aos superiores do Cercomp e também amigos Fernando César Silva da Mota e Hugo Alexandre Dantas do Nascimento, por entendenderem a importância deste trabalho

e propiciarem totais condições para o término do mesmo.

Ao CNPp pelo apoio financeiro.

Quinze anos atrás, as empresas competiam em preço. Hoje em qualidade. Amanhã será no design.

Robert Hayes,
Professor da Harvard Business School.

Resumo

da Silva, Wilane Carlos. **Gerência de Interfaces para Sistemas de Informação: uma abordagem baseada em modelos**. Goiânia, 2010. 150p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

Construir e manter a Interface Gráfica com Usuário (GUI) para Sistemas de Informação (SI) demanda muito tempo e esforço da equipe de Engenharia de Software. Esta dissertação descreve uma abordagem dirigida por modelos para criar e gerenciar essas interfaces. Nessa abordagem, o Engenheiro de Software projeta um modelo conceitual do sistema de informação usando um meta-modelo orientado a objetos. Um conjunto de regras de mapeamento predefinidas é aplicado para refinar e transformar o modelo conceitual do SI, gerando, em tempo de execução, a sua interface gráfica, com aparência e comportamentos específicos. Esse mecanismo melhora a usabilidade do software, assegurando consistência e homogeneidade das GUIs; aumenta a produtividade da equipe de Engenharia de Software; e simplifica a manutenção do SI. Nos experimentos realizados foi observada uma média de produtividade quinze vezes maior do que os métodos tradicionais no desenvolvimento de GUIs.

Palavras–chave

Interface Gráfica com Usuário, GUI, MDD, Desenvolvimento Dirigido a Modelo, MBD, Desenvolvimento Baseado em Modelo, MDE, Engenharia Dirigida a Modelo.

Abstract

da Silva, Wilane Carlos. **Interfaces Management to Information Systems: an models based approach**. Goiânia, 2010. 150p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

Building and maintaining the Graphical User Interface (GUI) for Enterprise Information Systems usually requires much Software Engineering staff time and effort. This paper describes a model-driven approach to create and manage these GUI. In this approach, the Software Engineer designs conceptual models of the information system software using object-oriented meta-models. A set of predefined mapping rules is applied to automatically transform and refine the conceptual models in order to generate, in runtime, the look and feel of the GUI, without source code creation. This improves the software usability, assuring consistency and homogeneity of the GUI; increases the productivity of the software engineering staff; and simplifies maintenance. In our experiments we have observed an average productivity fifteen times higher than traditional methods of GUI development.

Keywords

Graphic User Interface, GUI, MDD, Model Driven Development, MBD, Model Based Development, MDE, Model Driven Engineering.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

CAPÍTULO 1

Introdução

1.1 Motivação

A Interface Gráfica com Usuário (GUIs - *Graphic User Interface*) é um dos componentes mais importantes dos Sistemas de Informação (SI), pois representa, para o usuário do sistema, a única forma de interagir com o software subjacente e obter as informações produzidas pelo SI. No entanto, as GUIs são componentes complexos e contribuem amplamente para a baixa produtividade no desenvolvimento de software, principalmente porque:

1. as modificações nas GUIs são mais frequentes em relação a outros componentes do SI, tais como funções de aplicação ou procedimentos armazenados no Banco de Dados [39].

2. demandam muito esforço para produzir um grande volume de código-fonte. Em média, cerca de 50% do código de um SI é relacionado com a criação e configuração de GUIs [30].

3. sua complexidade vem aumentando nos últimos anos, e tende a crescer ainda mais [16], requerendo recursos técnicos cada vez mais sofisticados [10].

4. exigem forte padronização de apresentação e comportamento para fornecer boa usabilidade ao usuário final [16, 39].

5. necessitam de maior cuidado para garantir manutenibilidade, visto que o código da GUI envolve componentes gerados por *toolkits* ou bibliotecas externas, e que tornam o código específico do SI mais complexo.

Há vários modelos ou paradigmas de interação com o usuário. O modelo de interação é o conjunto de protocolos que permite ao usuário interagir com a aplicação. Estes protocolos muitas vezes são determinados por uma linguagem, e por isso também são chamados de linguagens de interação. O modelo de interação determina as atividades mentais

e físicas que o usuário deve desempenhar, bem como os processos computacionais que o software da interface deve ter para interpretar os comandos e dados do sistema [35].

Um modelo de interação pode ser caracterizado por um padrão e por um estilo de interação. O estilo diz respeito ao tipo de interação adotada que, normalmente, pode ser linguagem de comando, menus, linguagem natural, e WIMP [27]. O WIMP (*Windows, Icons, Menus and Pointers*) é classificado como a terceira geração de interações entre o ser humano e o computador [35]. É um tipo de GUI, uma especialização, orientada a janela, interface de apontar e arrastar e traz o conceito de mesa de trabalho (*desktop*). Toda interface WIMP é uma GUI, mas nem toda GUI é WIMP.

1.2 Proposta deste trabalho

As dificuldades associadas à construção de GUIs motivaram o desenvolvimento de uma abordagem dirigida por modelo, voltada para a criação de GUIs para Sistemas de Informação. Neste tipo de sistema, ocorre sempre a presença de um número extenso de telas de manutenção de cadastro, também conhecidas como CRUD (*Create, Retrieve, Update, Delete*), com aparência e comportamentos semelhantes. Essas telas geram um volume considerável de código-fonte com muitas características comuns voltado à interface gráfica.

Além disso, dentre os vários componentes de um SI, a interface gráfica tem um papel de destaque na maioria das aplicações, já que ela está em contato direto com o usuário. Para este, a interface gráfica é o próprio sistema, ou seja, a percepção que o usuário tem do software é aquela oferecida pela interface. Por isso, a qualidade do projeto da interface do usuário é um fator crítico para o sucesso do software [24, 30, 16]. Uma interface do baixa qualidade pode colocar em risco todo um sistema bem elaborado. Por outro lado, um sistema com arquitetura deficiente, por exemplo, pode ter sucesso caso ofereça uma interface gráfica agradável.

Sob o aspecto de gerência e manutenibilidade de software, a presente proposta tem o objetivo de minimizar o esforço e o tempo empregados para implementar as modificações na GUI de um SI ao longo do seu ciclo de vida. Essa preocupação é relevante, pois a fase de manutenção de um software de sucesso é mais duradoura que a sua fase de desenvolvimento.

Um dos principais problemas das abordagens tradicionais de geração de interfaces (abordagens não dirigidas por modelo) concentra-se justamente nos gastos excessivos de recursos da equipe de Engenharia de Software para especificar uma grande quantidade de informações relacionadas aos aspectos gráficos da interface, inclusive sobre localizações, formatos e comportamentos [10, 40].

A redução de recursos e esforços dedicados à definição, construção e gerência de GUIs de Sistemas de Informação proposta neste trabalho é baseada na geração automática de GUIs utilizando uma abordagem dirigida por modelo. A proposta define a arquitetura de um software, Gerador Dinâmico de Interfaces Gráficas (GDIG), capaz de criar, a partir de modelos conceituais de alto nível de abstração, o código que implementa a GUI de um SI. O GDIG se baseia em regras de mapeamento predefinidas para progressivamente transformar e refinar o modelo abstrato, gerando modelos mais específicos. Estes, por sua vez, contêm dados e conceitos relevantes à interface gráfica, definindo sua aparência, restrições, organização, comportamento, relacionamentos e ações.

Na abordagem proposta nessa dissertação, a geração das interfaces ocorre em tempo de execução, sem a necessidade do armazenamento persistente do código-fonte. A implementação do GDIG, descrita neste trabalho, permite gerar interfaces *Desktop* do tipo CRUD, com componentes gráficos baseados na tecnologia *Swing* da linguagem Java. O mecanismo possibilita a definição de apresentações e comportamentos complexos, tornando-se útil para uma extensa gama de aplicações de SI.

Dentre as contribuições obtidas com o desenvolvimento do presente trabalho, enfatiza-se um significativo aumento na produtividade do desenvolvimento de GUIs, chegando a uma diferença de quinze vezes se comparado à geração de código manual. Adicionalmente, houve uma redução considerável no volume de código-fonte do SI resultante, bem como uma completa padronização do *look and feel* da GUI, aumentando os níveis de usabilidade e consistência do SI subjacente. Adicionalmente, implementa-se uma técnica automática de geração de interfaces cujo futuro tende a ser promissor [7].

1.3 Organização da Dissertação

O restante deste documento descreve os detalhes da abordagem proposta para criação e gerenciamento de GUIs em SI e é organizado da seguinte maneira: o Capítulo 2 traz uma explanação geral sobre a teoria em torno do Desenvolvimento Dirigido por Modelos, seus propósitos, principais conceitos, suas vantagens e desvantagens, aplicações gerais dessa abordagem e seu uso em interfaces de usuário.

O Capítulo 3 relata a arquitetura do GDIG (Gerador Dinâmico de Interfaces Gráficas com Usuário), descrevendo: o macro contexto no qual a ferramenta encontra-se inserida, a motivação para geração dinâmica de GUI, sua visão geral e o funcionamento do tráfego de metadados e dados de negócio dentro da arquitetura.

O Capítulo 4 mostra os modelos definidos neste trabalho que servem de base para a geração de GUIs. Detalha o Modelo de Negócio e o Modelo de Apresentação, descreve os elementos internos de cada modelo, seus objetivos e os relacionamentos entre esses elementos.

O Capítulo 5 discute como funcionam as regras para, a partir dos modelos de Negócio e Apresentação, ser gerada a GUI. Traz o mecanismo de transformação entre os modelos e de geração da Interface Concreta a partir do Modelo de Apresentação.

O Capítulo 6 descreve a metodologia seguida para o estudo experimental sobre a produtividade na construção de GUIs, com sua coleta de dados, métricas e análise de resultados.

O Capítulo 7 conclui o trabalho, trazendo as características gerais das GUIs criadas, os trabalhos correlatos, contribuições e limitações da abordagem proposta, considerações finais e indicação de direções para extensões e trabalhos futuros.

CAPÍTULO 2

Desenvolvimento Dirigido por Modelo

Um processo de Engenharia de Software abrange um conjunto de atividades técnicas e gerenciais que devem ser realizadas para o desenvolvimento ou a manutenção de software com qualidade. Existem inúmeras variações de processos de Engenharia de Software e um problema comum a todas elas é a dificuldade para modelar, documentar, construir e evoluir o software [1].

A emergência em reparar aplicações em uso, integrá-las com outros sistemas, mudar sua infra-estrutura, alterar seus requisitos de acordo com as necessidades do negócio e, principalmente, a freqüente evolução, seja por motivos de mercado, tecnologias, ou legislações locais, levaram ao estabelecimento de abordagens de desenvolvimento de software dirigidas por modelos. Tais abordagens têm como objetivo geral abstrair os detalhes de implementação e de plataforma, focando assim na modelagem das regras do negócio.

Com esse propósito, surgiu o MDD (*Model-Driven Development*), uma abordagem de desenvolvimento de software baseada em modelos [31]. Essa abordagem consiste em separar a especificação das operações de um sistema dos detalhes de como esse sistema usa as potencialidades de sua plataforma.

No MDD as arquiteturas são modeladas independentemente de plataforma, e progressivamente transformadas e refinadas de modo que se transformem em modelos mais específicos que venham a atender a uma determinada necessidade [14].

Abordagens para desenvolvimento de software nas quais modelos abstratos são criados e sistematicamente transformados em implementações concretas são nomeadas de várias formas, como: Desenvolvimento Dirigido por Modelo (MDD - *Model-Driven Development*) [31] ou Desenvolvimento Baseado em Modelo (MBD - *Model-Based Development*) [37]. Neste trabalho essa abordagem é referenciada como Desenvolvimento Dirigido por Modelo (MDD).

Esse capítulo reúne os principais conceitos sobre o Desenvolvimento Dirigido por Modelo (MDD). A Seção 2.1 mostra os objetivos e benefícios do MDD; a Seção 2.2 traz os principais conceitos dessa abordagem; a Seção 2.3 mostra suas vantagens e desvantagens; e as Seções 2.4 e 2.5 trazem exemplos de implementações que utilizam o MDD.

2.1 Objetivos e Benefícios de MDD

Em geral, projetos convencionais de Engenharia de Software apresentam características indesejáveis como [16, 51, 35, 39]:

- **Baixa produtividade.** O processo de desenvolvimento de software é dividido em várias fases e, em cada uma delas, é produzida grande quantidade de documentação (figuras, diagramas) que, muitas vezes, não são utilizadas como deveriam. Isso faz com que a equipe de Engenharia de Software perca tempo gerando artefatos que não são utilizados ou são subutilizados. Além disso, na maioria dos sistemas a parte que demanda mais tempo para se chegar a um resultado satisfatório em relação ao usuário final é a interface gráfica. Isso devido à falta de conhecimento técnico dos profissionais e às constantes mudanças para que o usuário fique contente com o sistema e este se alinhe às necessidades de seus negócios.
- **Inconsistência entre documentação e código-fonte.** A documentação produzida perde, rapidamente, seu valor assim que a fase de codificação tem início. Ao invés de ser uma exata especificação do código, os diagramas geralmente tornam-se apenas figuras relacionadas ao código. Isso acontece porque, frequentemente, as modificações são feitas apenas no código-fonte, pois atualizar toda a documentação exige tempo que, na maioria das vezes, não está disponível.
- **Necessidade constante de mudança de tecnologias.** Muitas empresas precisam utilizar novas tecnologias por exigência dos clientes, pela obsolescência das tecnologias adotadas, ou para resolver novos problemas. Esses fatores fazem com que as empresas migrem para novas tecnologias, aumentando consideravelmente o custo e o tempo para desenvolvimento do software.
- **Os sistemas são desenvolvidos de maneira isolada.** Um software deve ser desenvolvido com o planejamento de que, no futuro, ele possa ser utilizado ou reutilizado por um outro sistema como uma funcionalidade externa. Porém, na prática esse pensamento não é seguido devido ao aumento de trabalho para projetar e criar o sistema, haja vista a preocupação com aspectos adicionais como modelos e padrões de projeto corporativos.
- **Falta de manutenção e documentação adequadas.** A documentação de sistemas geralmente não é de alta qualidade. Isso porque aqueles que estão envolvidos com atividade de desenvolvimento do sistema, principalmente a equipe de implementação, preocupam-se apenas com as tarefas que os ajudam diretamente na codificação, sem considerar futuras necessidades de manutenção do sistema. Esse problema multiplica-se na interface gráfica devido à sua importância, pois é o ponto de contato do usuário com o sistema. Isso faz com que o usuário veja a interface como

o próprio sistema e modifique-a constantemente para que atenda aos requisitos de negócio, gerando um número maior de mudanças no decorrer do desenvolvimento do software.

- **Aumento da quantidade de código-fonte semelhante:** A interface é o componente do software que produz maior volume de código-fonte, devido à grande quantidade de elementos gráficos (*widgets*) envolvidos. Além disso, muitas janelas são semelhantes, principalmente em sistemas grandes, tornando o código-fonte repetitivo, dificultando o gerenciamento e a evolução do SI.
- **Dificuldade para se manter o *look and feel* padrão:** Uma das características elementares de uma boa interface gráfica com o usuário é a padronização do comportamento e da aparência. Quanto mais janelas, mais código-fonte e, conseqüentemente, mais difícil sua gerência. Caso haja alguma mudança nos requisitos de comportamento e aparência de alguma janela, todas as outras terão que ser ajustadas também.

O MDD é uma abordagem para solucionar ou reduzir o impacto desses problemas no contexto de desenvolvimento e manutenção de software, tendo como base a utilização de modelos para representar os sistemas.

As principais vantagens esperadas de uma aplicação bem sucedida dos princípios de MDD são [1, 31]:

- **Alta produtividade.** O ganho de produtividade é obtido com a utilização de ferramentas que realizam a transformação dos modelos abstratos para os específicos. Os desenvolvedores de modelos abstratos não precisam se preocupar com os detalhes específicos de plataformas. Esses detalhes já são tratados pela abordagem MDD na definição da transformação do modelo mais abstrato para o mais específico. No nível do modelo específico, existe muito menos informação a ser definida, pois muitas já foram geradas a partir da transformação do modelo abstrato. Os desenvolvedores gastam menos tempo realizando codificação e utilizam o tempo para se preocuparem mais com a modelagem do negócio.
- **Portabilidade.** A utilização de modelos abstratos no desenvolvimento aumenta a portabilidade do sistema, visto que esses modelos podem ser transformados para vários modelos específicos voltados para plataformas variadas. Para cada nova tecnologia são desenvolvidas novas definições de transformações de modelo abstrato, considerando as especificidades da nova tecnologia. No entanto, o modelo abstrato já existente não necessita sofrer qualquer modificação.
- **Reutilizabilidade.** Um modelo abstrato pode ser diretamente reutilizado para a construção de outros sistemas ou subsistemas. O mesmo ocorre com os mecanismos de transformação destes modelos. Dessa forma, a reutilização das informações está auto-contida na abordagem MDD.

- **Manutenção e documentação.** Com o MDD, o modelo abstrato representa a documentação em mais alto nível de um sistema. Toda modificação que é feita no sistema deve ser primeiramente feita no modelo abstrato. Depois disso, o modelo específico é gerado novamente, assim como o código da aplicação. A documentação do sistema sempre será consistente com o seu código gerado.

Todos os benefícios de MDD são obtidos através da separação entre os conceitos do negócio (ou especificação) de um sistema da forma como ele será implementado. Em outras palavras, os conceitos da realidade alvo de um sistema devem ser especificados durante a modelagem inicial, sem focar em informações sobre plataforma alvo ou tecnologias específicas. Essas informações são inseridas posteriormente, durante o processo de transformação destes modelos conceituais para modelos específicos de cada plataforma.

2.2 Principais Conceitos da Abordagem MDD

MDD baseia-se em mecanismos que possibilitam a especificação de um sistema independente de plataformas, a escolha de uma plataforma particular para implementar o sistema, e a transformação da especificação do sistema para essa plataforma particular. A Figura 2.1 mostra uma visão geral da abordagem MDD.

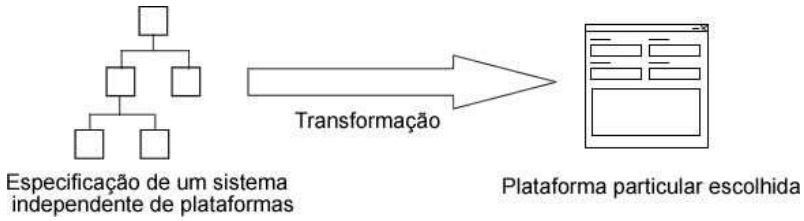

Figura 2.1: *Abordagem geral do MDD*

A modelagem de conceitos é o ponto de foco do MDD. Os modelos que são levados em conta no MDD são mais do que modelos de software propriamente ditos, pois informações que não são relacionadas diretamente aos aspectos do sistema a ser desenvolvido podem ser úteis para sua geração. Por exemplo, restrições legais ou informações sobre a cultura organizacional da empresa são importantes para definir o sistema.

No contexto do MDD, um modelo deve ser capaz de ajudar a especificar uma transformação de um modelo para um outro até ser gerada a aplicação alvo. Segundo [1]:

- Um modelo é sempre uma abstração de algum conceito que existe na realidade
- Um modelo pode ser usado para produzir algum conceito que existe na realidade

O MDD promove uma abordagem na qual a especificação do sistema é feita de forma independente de plataforma e, para cada uma das plataformas de interesse, tal especificação pode ser automaticamente transformada em uma implementação correspondente, como mostra a Figura 2.2. Isso é feito através da manipulação e transformação de modelos independentes de plataforma em modelos de uma plataforma específica.

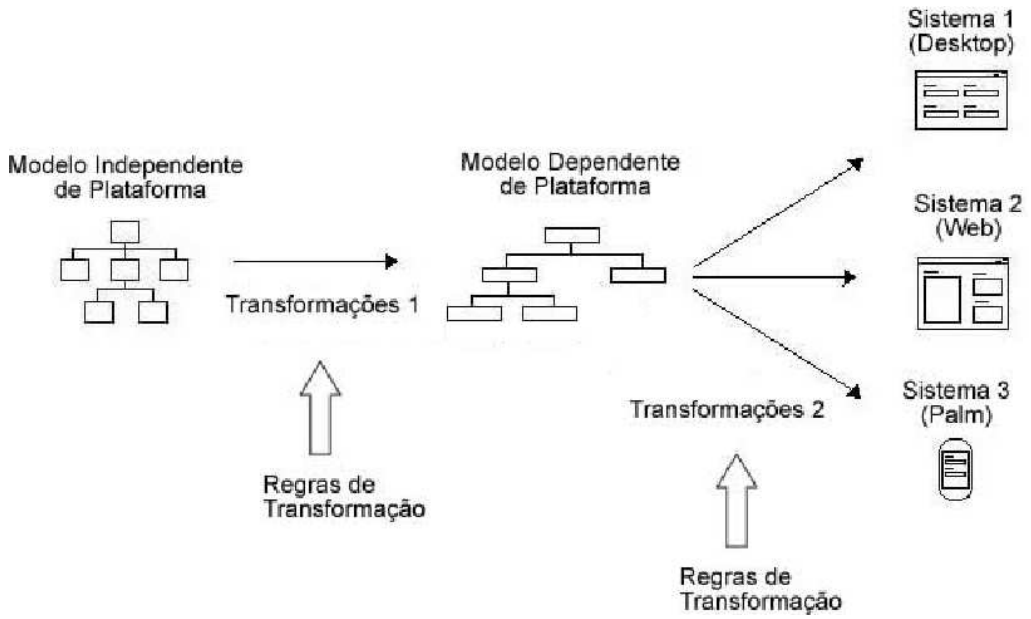

Figura 2.2: *Transformação para várias plataformas*

Esses modelos são conhecidos por Modelo Independente de Plataforma (ou PIM - *Plataform Independent Model*) e Modelo Dependente de Plataforma (ou PSM - *Plataform Specific Model*), respectivamente [1]. Aquele é um modelo com alto nível de abstração e que não tem qualquer referência ou ligação a qualquer tipo de tecnologia. Sua função é a de descrever a lógica do negócio. Já o PSM define um sistema de acordo com uma tecnologia específica, sendo o resultado do mapeamento do Modelo Independente de Plataforma para uma plataforma específica [34, 53].

A aplicação e uso de modelos vêm tornando-se uma prática cada vez mais adotada no processo de desenvolvimento de software. Com o MDD, os modelos passaram a ser peças fundamentais e amplamente usados como componentes de construção do sistema.

Para realizar o mapeamento entre os modelos é necessária uma ferramenta de transformação que utiliza a definição da transformação como entrada. Essa definição corresponde a um conjunto de regras de transformação que determinam como os elementos de uma determinada linguagem podem ser mapeados em elementos da linguagem alvo da transformação.

Com o Modelo Dependente de Plataforma definido, o próximo passo no processo de desenvolvimento é realizar a transformação desse modelo em código-fonte.

2.3 Vantagens e desvantagens do MDD

A diferença essencial entre o processo de desenvolvimento proposto pelo MDD e os processos tradicionais de desenvolvimento é que no desenvolvimento dirigido por modelo a transformação entre os modelos é feita de forma automática por uma ferramenta de transformação, enquanto nos processos tradicionais isso é feito manualmente pela equipe técnica de implementação ou de manutenção do software. No MDD, o desenvolvedor cria a ferramenta de transformação e não o programa específico como é feito normalmente.

MDD aparece como uma alternativa para enfrentar problemas comuns no desenvolvimento de software, como baixas portabilidade, produtividade da equipe de desenvolvimento e manutenibilidade. Como o Modelo Dependente de Plataforma e o código estão bem próximos, o esforço maior é gasto na criação da ferramenta de transformação entre os modelos, que é uma operação bastante complexa.

2.3.1 Vantagens do MDD

A produtividade da equipe de desenvolvimento que implementa a abordagem MDD é maior, pois os desenvolvedores do Modelo Independente de Plataforma têm menos trabalho para criar um modelo do sistema, visto que os detalhes referentes à tecnologia de implementação podem ser abstraídos deste modelo. Esses detalhes mais específicos são levados em consideração no momento da definição da transformação entre os modelos, pelo mecanismos de transformação, dentro de suas regras internas. Além disso, as ferramentas de transformação e geração de código permitem a redução da quantidade de código que deve ser escrita manualmente por um programador.

Com relação à portabilidade, a utilização do MDD permite que um mesmo Modelo Independente de Plataforma possa ser transformado em Modelos Dependentes voltados para plataformas variadas. Para isso, são utilizados mecanismos de transformação específicos para cada linguagem alvo correspondente.

Em relação à manutenção, sempre que alguma alteração for necessária no sistema, basta modificar o Modelo Independente de Plataforma e executar novamente a transformação para o Modelo Dependente de Plataforma. Portanto, nota-se que o processo de manutenção é simplificado, visto que se resume em reestruturar o Modelo Independente de Plataforma e aplicar a transformação de modelos.

Como explica [53], usando o conceito de modelos, o MDD pode reduzir potencialmente o ciclo de vida do desenvolvimento de software e permitir a reutilização de

módulos de sistemas, não sendo necessário, portanto, projetar um novo sistema completo com funcionalidades similares quando uma nova tecnologia de implementação torna-se disponível. Em outras palavras, um Modelo Independente de Plataforma, que representa o comportamento e aparência conceitual de um sistema, pode ser reutilizado em vários ambientes de software, já que não se restringe a nenhuma tecnologia específica em nível de projeto ou de desenvolvimento do software.

Aumentando-se o nível de abstração em relação aos detalhes de implementação, aumenta-se significativamente a facilidade de entendimento do sistema e, ao invés de se analisar complexos códigos-fonte, os projetistas trabalham com conceitos de mais alto nível. Isso diminui consideravelmente o tempo de geração e manutenção do sistema.

2.3.2 Desvantagens do MDD

Embora tenha vantagens claras, o desenvolvimento dirigido por modelo apresenta desvantagens em sua implementação. As principais são a dificuldade de se gerar sistemas que contêm funcionalidades que não se repetem constantemente e a dificuldade de obter-se a personificação do software.

As aplicações geralmente contêm, em seu escopo de funcionalidades, aquelas que apresentam grandes similaridades, tais como as operações CRUD. Essas aplicações obtêm maior êxito na abordagem MDD devido ao fato de otimizarem sua implementação, visto que pode ser utilizado o mesmo mecanismo para criar várias funcionalidades semelhantes. Já as aplicações que contêm pouca repetição funcional têm sua eficiência prejudicada, já que não há aproveitamento de um mesmo mecanismo para várias funções.

Além disso, existe o custo de projetar e implementar mecanismos complexos de transformação. O desenvolvimento desses mecanismos requer mais tempo que processos tradicionais (que não utilizam a abordagem MDD). Implementada a abordagem MDD, a geração de SI com base nessa abordagem é rápida, mas o custo para se chegar ao final dessa implementação é alto.

Outra desvantagem é a dificuldade de gerar uma aparência ou comportamento personificado de acordo com o perfil do usuário. Como no MDD uma aplicação é criada a partir de um mesmo mecanismo de geração, as aparências e comportamentos são padronizados para todas as aplicações. Com isso, caso o usuário defina, como requisito, uma característica que não esteja entre aquelas contempladas pelo gerador, ele, o usuário, pode não ter plena satisfação no uso do sistema. Para tentar solucionar esse problema, a ferramenta de geração deve prever o maior número possível de configurações, a fim de que os usuários possam adaptar as telas de acordo com os seus padrões e suas necessidades. Porém, isso tem um custo muito elevado.

2.4 Aplicações Gerais de MDD

Apesar de ser uma abordagem recente para o desenvolvimento de software, o MDD tem recebido grande atenção da comunidade de Engenharia de Software, tanto em pesquisas acadêmicas quanto em trabalhos da indústria de software. Vários são os temas: segurança de sistemas [2]; desenvolvimento de sistemas para controle de processos [13]; ferramentas de automatização de abordagens MDD [5]; desenvolvimento e reconfiguração de sistemas de software baseados em componentes [22, 38]; assistência em consultoria e serviços de Tecnologia da Informação [23]; projeto de sistemas de tempo real [32]; ambientes de projetos conceituais [44]; geração automática de WSDL (*Web Service Definition Language*) e descrição de serviços a partir de modelos [50, 52].

Outro exemplo de aplicação da abordagem MDD é *Ruby on Rails* [11]. Trata-se de um *meta-framework* gratuito cujo propósito é aumentar a produtividade no desenvolvimento de aplicações *Web* orientadas a banco de dados (*database-driven web sites*). Escrito na linguagem de programação *Ruby*, suas aplicações utilizam o *framework Rails* e são desenvolvidas com base no padrão de projeto MVC (*Model-View-Controller*).

A construção de sistemas em *Ruby on Rails* baseia-se no método de metaprogramação denominado *Scaffolding*. Nesta técnica o programador escreve especificações que descrevem como a aplicação de banco de dados deve ser usada. O compilador utiliza essas especificações para gerar o código que a aplicação usa para executar as funcionalidades CRUD.

O *Scaffolding* é muito útil na geração de controladores básicos, para dar apoio à prototipação e ao início do desenvolvimento, principalmente para funções de cadastro. Ele dá suporte à implementação inicial de uma aplicação, devendo esta, posteriormente, ser gradualmente melhorada e ajustada de acordo com adições de código-fonte específicos. Obtém-se, portanto, uma aplicação funcional correta desde o início. Pode-se, posteriormente, mudar a aparência e adicionar novas características funcionais até que a aplicação esteja pronta para a implantação.

Adicionalmente, o *Scaffolding* tem como principais vantagens a implantação de um mecanismo para rápida geração de interfaces gráficas, apoiando o desenvolvimento do software e agilizando tarefas repetitivas. Entretanto, possui as seguintes desvantagens: as ações geradas pelo *framework* são limitadas e consistem em apoio à prototipação. Para que seja criada uma aplicação mais completa e complexa, deve ser inserido código-fonte personalizado.

Os *Scaffoldings* são uma visão limitada de uma aplicação, não oferecendo suporte para sua geração completa, e não levam em conta a possibilidade de gerar interfaces alternativas, de melhor usabilidade.

2.5 MDD em Interfaces de Usuário

No campo de Interfaces Gráficas com Usuário, a abordagem MDD vem sendo usada de várias formas e para vários propósitos como: comportamentos e eventos de componentes gráficos [4]; leiaute e aparências [8, 49]; projeto e geração de GUIs em geral [12, 21, 28, 25, 26, 41]; entre outros [15].

Pode-se aplicar o MDD para especificar, por exemplo, conceitos vinculados a aparência, comportamento, organização ou leiaute de seus componentes gráficos, relacionamento entre esses componentes, restrições de validação, princípios de usabilidade e individualização desses aspectos de acordo com necessidades ou preferências dos usuários finais. Todos esses fatores podem ser especificados utilizando linguagens de modelagem de alto nível de abstração e independentes de plataforma. Na abordagem convencional isto é feito com uma linguagem de programação, ou usando-se um *toolkit* para construção de interfaces.

Pesquisadores têm desenvolvido ferramentas de geração de interfaces gráficas com usuário dirigidas por modelo de várias formas. Segundo [10], existem 4 categorias de ferramentas:

- **baseadas em linguagem:** o projetista especifica a interface gráfica em uma linguagem de propósito específico, que pode ser de várias formas, incluindo gramáticas livres de contexto, diagramas de estado, linguagens declarativas, máquinas de estado finito e linguagens de evento.
- **frameworks de aplicações:** dão suporte a partes importantes de uma aplicação, como a janela principal e os comandos. O desenvolvedor especializa essas classes para fornecer detalhes específicos de aplicação. Contém um problema em potencial: o desenvolvedor precisa especificar muitos detalhes sobre localização, formato e projeto em geral das IU.
- **geradores automáticos:** um ou vários modelos são explorados para, automaticamente, gerar o código da interface gráfica. Ajuda a solucionar o problema dos *frameworks*, pois requer pouca ou nenhuma especificação por parte do usuário.
- **ferramentas interativas:** permite ao desenvolvedor selecionar componentes gráficos a partir de uma biblioteca predefinida e inseri-los na tela, gerando caixas de diálogo, menus e janelas. Alguns geram uma descrição da IU em uma linguagem que pode ser lida em tempo de execução como, por exemplo, em UsiXML [48].

Os tipos mais implementados são os geradores [3, 21, 26] e os frameworks [34, 29, 46, 20, 45, 18, 19, 47]. Uma diferença importante entre geradores e *frameworks* reside no fato de os primeiros funcionarem como uma caixa-preta, encontrando-se prontos para serem utilizados sem a necessidade de inclusão ou modificação de código. Enquanto

isso, os *frameworks* fornecem uma estrutura geral para que o desenvolvedor insira sua implementação, estendendo e particularizando seu comportamento. Outras diferenças importantes entre geradores e *frameworks* de interface são [39, 16]:

- Os *frameworks* fornecem uma estrutura geral para que o desenvolvedor insira sua implementação, estendendo e particularizando seu comportamento. Já o gerador dinâmico está pronto e a sua utilização típica não envolve a inclusão ou a modificação de código.
- Um *framework* exige que o seu usuário (o programador) conheça a sua estrutura interna (caixa-branca) enquanto um gerador dinâmico encapsula a sua forma de trabalho (caixa-preta).
- Os geradores dinâmicos são menos flexíveis, porém mais eficientes. Os *frameworks* possibilitam construir interfaces mais complexas, mas exigem um grande esforço do desenvolvedor.
- Os *frameworks* "chamam" funcionalidades do sistema (inversão de controle), enquanto os geradores dinâmicos são "chamados" por alguma parte do sistema de informação subjacente.

O mecanismo proposto nesta dissertação (GIDG) pode ser classificado como fazendo parte da categoria de geradores automáticos baseados em modelos, haja vista que recebe modelos como entrada e gera a GUI como saída de forma automática. Além disso, GDIG não necessita que o projetista da interface gráfica conheça seu mecanismo interno, suas classes ou seu código, uma vez que não demanda nenhum tipo de especialização.

Sua utilização típica não envolve a inclusão ou a modificação de código e encapsula sua forma de trabalho. São, portanto, menos flexíveis porém mais eficientes que os *frameworks*. Isso reflete as decisões de projeto de atender a um dos principais objetivos da abordagem: a eficiência na geração de interfaces gráficas.

CAPÍTULO 3

Arquitetura do GDIG - Gerador Dinâmico de Interfaces Gráficas com Usuário

3.1 Arquitetura do *framework*

O gerador de interfaces gráficas com usuário dirigido por modelos proposto neste trabalho (GDIG) é um componente funcional que faz parte de uma arquitetura maior. Essa macro-arquitetura representa um *framework* dirigido por modelo que contém vários outros módulos, cada um com sua finalidade específica, tais como criação e manutenção de repositórios de dados, e gerência de um conjunto de regras de negócio. O objetivo do *framework* é apoiar o desenvolvimento de SI complexos.

Esta seção descreve uma visão geral da arquitetura do *framework*, suas camadas, as formas de comunicações entre seus componentes, e situa o GDIG nessa estrutura, definindo seu papel dentro desse contexto.

A arquitetura do *framework* segue o padrão MVC (*Model-View-Controller*). O GDIG, juntamente com todo seu mecanismo de geração de GUIs e conexão com outras camadas, forma a camada de visão dentro desta arquitetura.

O *framework* foi inicialmente concebido com uma arquitetura de 4 camadas (Interface, Interface Aplicação, Aplicação e Persistência). Com o aumento de sua complexidade e o surgimento de novas funcionalidades, os componentes foram redistribuídos, novos componentes foram inseridos e a arquitetura foi modelada e implementada, segundo padrões UML, como mostra a Figura 3.1.

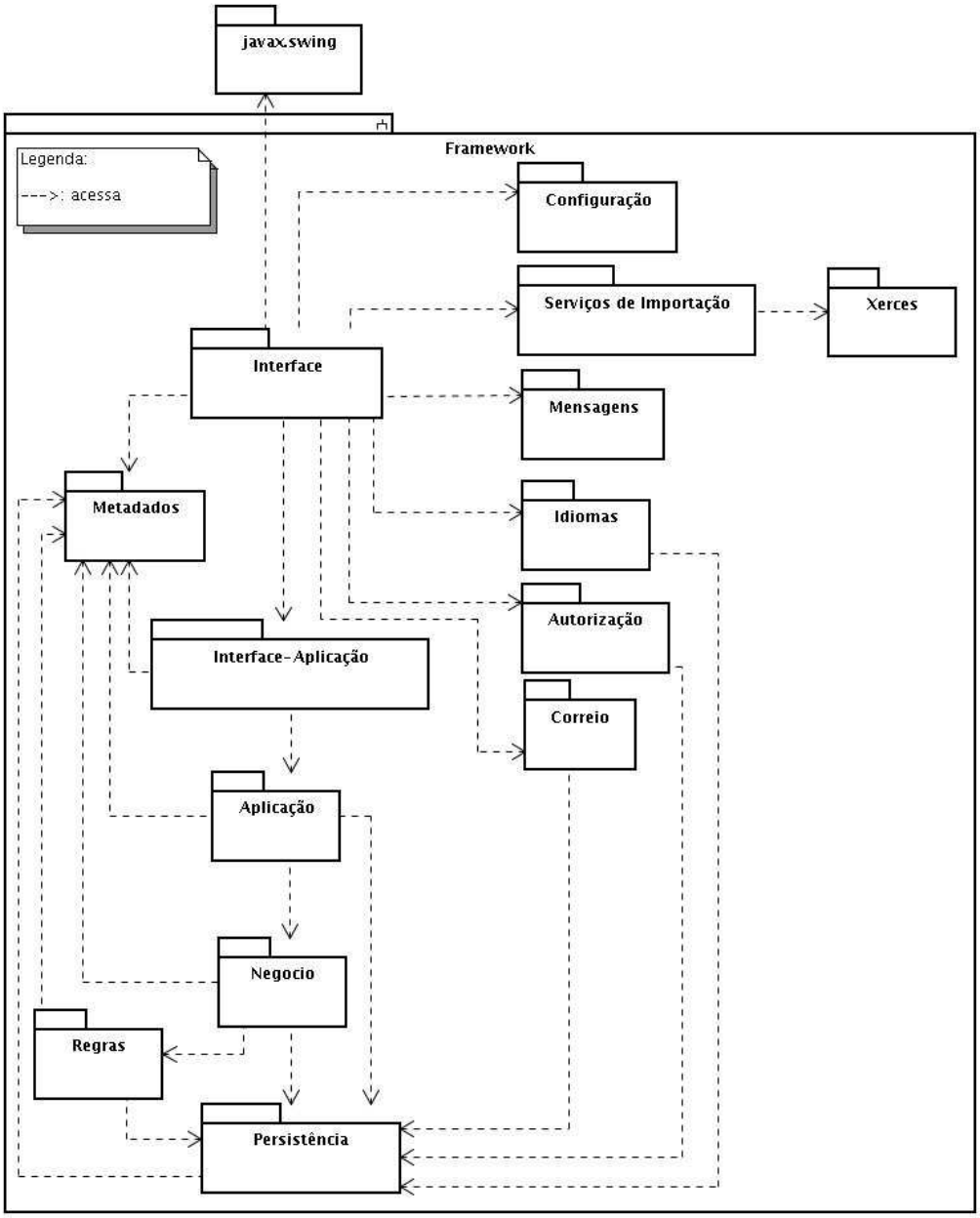

Figura 3.1: *Arquitetura do framework*

As principais camadas e componentes da arquitetura são descritos a seguir.

A **Camada Interface** é responsável por gerar e mostrar ao usuário final as interfaces gráficas do SI gerado com o *framework*. Isso é feito baseado nas informações que são obtidas do modelo conceitual do SI. Ela executa o papel "*View*" dentro do padrão MVC. É nesta camada que o GDIG localiza-se.

Suas principais funções são: (i) criar e gerenciar as interfaces mostradas ao usuário final; (ii) receber as informações nos campos de dados; (iii) executar algumas validações dos dados inseridos pelo usuário; e (iv) enviar os dados às camadas inferiores.

Por exemplo, quando um usuário insere um dado em um campo da interface para

que seja armazenado no Banco de Dados, a Camada de Interface envia os dados para a Camada de Interface Aplicação. Esta funciona como um Controlador, direcionando a chamada para a Aplicação específica.

A lógica de tratamento dos dados das camadas inferiores não é conhecida pela Camada Interface, ou seja, esta camada não contém código que executa lógica, processamento da aplicação ou regras de negócio, sendo uma camada fina (*thin*) [9].

A Camada Interface executa algumas verificações ou validações como, por exemplo, tipos ou cardinalidade de dados.

Para a geração dos componentes gráficos das GUIs, é utilizada a tecnologia *Swing* do Java. Para isso é necessário o acesso ao pacote *javax.swing*.

A **Camada Interface Aplicação** tem a tarefa de servir como fachada entre a Camada Interface e a Aplicação. Seu objetivo principal é repassar mensagens da interface para a aplicação e preparar dados vindos da camada de aplicação para serem enviados para a interface. Esta camada funciona como o *Controller* do padrão MVC.

A Camada Interface Aplicação não executa qualquer regra de negócio ou lógica da aplicação. Sua função é despachar as chamadas para as camadas inferiores responsáveis por executá-las. Dessa forma, ela torna as camadas de Interface e Aplicação pouco acopladas, diminuindo a dependência entre elas.

A responsabilidade do pacote **Aplicação** do *framework* é realizar as operações CRUD, sendo apoiado pelas operações disponíveis nos pacotes Persistência e Negócio. Operações como inserir, alterar, remover, desativar e obter dados são executadas pelo pacote Aplicação.

Outro tipo de operação realizada por esse componente é o tratamento de dados. Portanto, os dados vindos da Camada Interface podem sofrer transformações para que sejam reconhecidos e tratados pelos componentes inferiores, como Persistência.

O objetivo do pacote **Negócio** é verificar se os dados vindos das camadas superiores estão de acordo com determinadas regras pré-estabelecidas (regras de negócio). Ele executa o controle para que informações inseridas no Banco de Dados sejam válidas para o negócio subjacente ao SI.

Através das informações obtidas nos metadados, são tratados aspectos como cardinalidade máxima e mínima dos dados, tipos de dados, valores máximos e mínimos, regras de composição, consistência, verificação de regras, dentre outros.

O pacote **Regras** contém o mecanismo de gerência das regras dentro do sistema. Regras de validação são executadas antes de os dados serem persistidos no Banco de Dados. Regras de derivação são executadas para obter-se informações calculadas a partir de outras informações.

As regras podem ser inseridas ou removidas de forma independente de outras partes do *framework*. Ou seja, o mecanismo que trata as regras tem um acoplamento fraco

com as demais partes da arquitetura.

O pacote **Persistência** cuida de toda a lógica para a persistência dos dados: inserção, exclusão, modificação e obtenção dos dados do Banco de Dados. Ao receber a chamada de serviços, Persistência processa os dados (executa verificações, validações, transformações, busca, dentre outros), implementa o serviço indicado e devolve a resposta.

Além de gerenciar os dados da aplicação, como aqueles inseridos pelo usuário durante a operação do sistema, Persistência gerencia também as informações dos modelos (isto é, metadados). Para exemplificar, quando o usuário quer abrir uma janela de cadastro de Pessoa Física, primeiramente Persistência fornecerá as informações de metadados para que a GUI seja criada. Depois de criada a janela, o usuário digita dados da Pessoa Física e executa a funcionalidade para que eles sejam inseridos no sistema. Persistência receberá esses dados da aplicação e irá inserí-los no Banco de Dados. Ou seja, os dois tipos de informação - metadados e dados - são tratados pelo mesmo pacote Persistência.

O pacote **Metadados** é um componente essencial para a abordagem MDD. Quase todos os componentes (Interface, Aplicação, Persistência) utilizam serviços do pacote Metadados. Nele estão localizadas todas as informações para o tratamento e manipulação dos metadados que são usados desde a persistência até a geração de interfaces gráficas com o usuário. Por isso, o pacote Metadados não se encontra em uma estrutura hierárquica, mas sim funcionando paralelamente aos demais componentes dentro da arquitetura. Ele pode (e deve) ser acessado desde as camadas mais baixas até as mais altas da arquitetura do *framework*.

A função do pacote **Configuração** é armazenar arquivos que contêm dados sobre atributos configuráveis do *framework*. Por exemplo, o número máximo de elementos que podem ser mostrados na janela de cadastro de uma determinada entidade. Isso porque pode haver um número muito grande de elementos em algumas entidades, o que inviabiliza a obtenção de todos os dados devido à falta de espaço na memória, ou à sobrecarga da rede de comunicação.

Outras configurações como Ativar/Desativar (*On/Off*) alguns comportamentos da interface, e obtenção de dados do Banco de Dados estão disponíveis nesse arquivo de configuração.

Esses dados de configuração podem ser modificados através de uma interface gráfica no próprio sistema, que é acessível somente a pessoas deviamente autorizadas. Essa funcionalidade traz várias vantagens, como a customização de comportamentos e aparências, e a facilidade para testes, já que caso alguns comportamentos não estejam funcionando corretamente, basta desligá-los (*Off*) e, assim, o sistema como um todo funcionará normalmente.

Um sistema geralmente não trabalha de forma isolada. Podem existir atores

externos como dispositivos eletrônicos, aparelhos de leitura de códigos-de-barra e bases de dados de outros sistemas. Portanto, pode haver a necessidade da importação de dados desses atores externos como, por exemplo, quando algum usuário necessita migrar os dados de seu sistema atual para algum outro.

O pacote Serviços de Importação contém funcionalidades que permitem que dados de outros sistemas sejam importados para o sistema construído com o *framework*, sem violar nenhum tipo de restrição descrita na base de dados alvo. Essa funcionalidade importa dados com base em arquivos XML e utiliza o pacote *xerces* [36] de Java para manipular esses arquivos.

Com o intuito de facilitar a gerência dos dados manipulados pelos SI e aumentar a usabilidade da interface gráfica, o *framework* comporta um pacote **Mensagens** responsável pela gerência de mensagens aos usuários, de modo a orientá-los na execução de suas atividades ou alertá-los sobre algum evento. Mensagens do tipo *feedback*, erro e *warning* estão contempladas.

O *framework* foi preparado para gerar sistemas de informação baseados não somente no idioma português, mas também em espanhol, inglês e francês. O componente Interface é preparado para adaptar textos e figuras à linguagem escolhida pelo usuário. Essa funcionalidade é apoiada pelo pacote Idiomas e acrescenta qualidades essenciais a softwares que querem romper barreiras geográficas, sendo um requisito muito importante para a qualidade do software [16].

Outra característica do *framework* é fornecer suporte para o trabalho em modo multiusuário e sua personalização, de acordo com o tipo de perfil do usuário.

Quando um usuário se registra no sistema, a interface é capaz de ajustar a barra de ferramentas e a estrutura de menus de acordo com as ações que este usuário está autorizado a executar. Isso traz o conforto de ser mostrado ao usuário somente as funcionalidades que ele realmente pode realizar, evitando uma interface com um visual carregado e de difícil compreensão.

A interface obtém essas ações através do pacote **Autorização**. Por exemplo, o usuário João pode executar o Cadastro de Pessoa Física, então aparecerá o menu para que ele possa cadastrar uma pessoa física no sistema. Quando João executa o Cadastro de Pessoa Física, a interface obtém do pacote Autorização as informações indicando o que ele pode ou não executar dentro desse cadastro, ou seja, se ele pode alterar, excluir, inserir ou consultar dados de pessoas físicas. Esse mecanismo melhora a confidencialidade, integridade e disponibilidade no manuseio dos dados.

Outro componente funcional que o *framework* oferece é o pacote **Correio**, responsável pela gerência das informações referentes ao sistema. Essa gerência inclui avisos, lembretes e mensagens enviadas por outros usuários.

Esses avisos agendados são mostrados junto com uma janela de dicas, que é

aberta assim que o usuário entra com seu login e senha.

O sistema também proporciona um ambiente de correio interno, onde o usuário pode mandar mensagens para outros dentro do sistema.

Percebe-se, portanto, que esse conjunto de pacotes e camadas formam um *framework* para rápida criação e evolução de sistemas, com um extenso conjunto de funcionalidades. O GDIG é responsável pela criação das janelas gráficas do sistema.

3.2 Motivação para geração dinâmica de GUI

Aplicações que trabalham com manipulação e gerência de dados em SI, necessariamente executam operações CRUD (*Create, Read, Update, Delete*). As interfaces gráficas com usuário para essas operações são semelhantes tanto na aparência quanto no comportamento. Em relação ao visual, elas contêm campos para a edição de dados pelo usuário, geralmente distribuídos de forma organizada e seguindo uma sequência coerente. Analisando o lado comportamental, a maior parte dos SI precisam de funcionalidades de consulta, inserção, exclusão e alteração dos dados, além de outras operações comuns como *feedback* de ajuda ao usuário.

O desenvolvimento de um SI de grande porte envolve diversas entidades de negócio e uma extensa base de dados que deve ser gerenciada. Assim, um volume considerável de aplicações e interfaces cuida do cadastro de dados básicos do negócio. Como as GUIs devem ser semelhantes na aparência e comportamento, visando a melhor usabilidade para o usuário final, ocorre uma dificuldade de padronização e manutenção dessas interfaces.

Com o método tradicional de desenvolvimento de interfaces gráficas, onde cada janela é criada de forma estática, a dificuldade para padronizar e manter um grande número de GUIs é maior. Uma criação de forma estática significa que, para cada janela, existe um ou mais arquivos com seu código-fonte correspondente já predefinido.

Com um sistema onde existam dezenas de diferentes itens que podem ser cadastrados, deverá existir uma GUI para cada item, cada uma com seus campos específicos. Neste caso, haverá um ou mais arquivos de código-fonte para cada GUI.

Suponhamos que todas as interfaces estejam prontas e padronizadas, e que um componente da Equipe de Engenharia de Software tenha uma idéia de mudança da disposição dos campos para melhorar a usabilidade das interfaces. Neste caso, os desenvolvedores teriam que manter cada arquivo referente a cada GUI, ou seja, passar por dezenas de arquivos, modificando a posição dos componentes gráficos.

A interface gráfica é a parte do software que, durante seu desenvolvimento e uso, sofre o maior número de modificações, e que ocupa cerca de 50% de todo o código-fonte e do tempo gasto para o desenvolvimento do software [30]. Portanto, esse mecanismo

de manutenção não se mostra eficiente e atrasa o desenvolvimento e a manutenção do SI como um todo.

O componente GDIG do *framework* MDD se propõe a otimizar o processo de desenvolvimento e manutenção de interfaces gráficas. Através da geração dinâmica dirigida por modelos, as GUIs são criadas automaticamente e em tempo de execução. Ou seja, não existe um código-fonte pré-compilado para cada interface gráfica.

Com isso, obtém-se um processo de criação e manutenção mais eficiente, já que, caso seja necessário modificar algum aspecto das interfaces, basta modificar o mecanismo de geração e todas as GUIs serão atualizadas automaticamente. Ocorre também um alto grau de padronização visual e comportamental, já que todas as interfaces gráficas são criadas a partir de um mesmo gerador.

3.3 Arquitetura do GDIG

O GDIG faz o papel da Camada Interface, dentro do padrão MVC. Os componentes desta Camada são apresentados na 3.2.

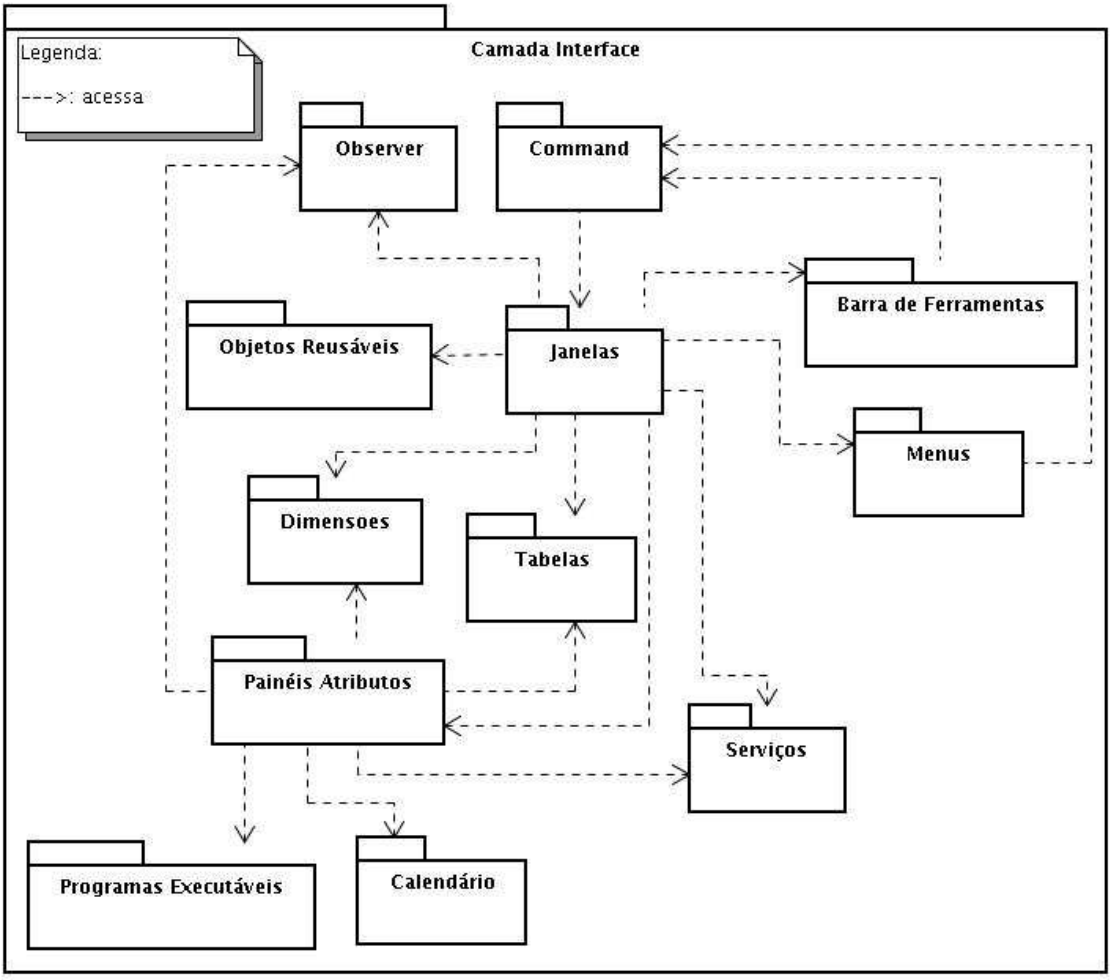

Figura 3.2: *Arquitetura do GDIG*

A arquitetura da camada Interface utiliza vários padrões de projeto [17]. Um desses padrões é o ***Observer***. Este padrão é o responsável pelo ajuste de comportamentos de componentes da interface de acordo com a mudança de valor de outros componentes. Nos metadados é definido que um componente gráfico Y terá um comportamento Z caso um outro componente W mude seu valor. Essa funcionalidade amplia o poder de expressão comportamental da interface. O pacote Observer implementa esta funcionalidade.

Outro padrão importante para tornar a interface mais elegante, reutilizável e manutenível é ***Command***. Esse padrão é encontrado nos menus, na barra de ferramentas e nos botões, desacoplando a funcionalidade de um comando do *widget* que o dispara. O pacote Command implementa esse comportamento.

O pacote **Objetos Reusáveis** implementa uma das características marcantes na arquitetura proposta, que é a reutilização dos componentes. Ao se iniciar projeto de um componente, pensa-se principalmente em como ele poderá ser reutilizado no futuro. Todos os componentes com alto potencial de reutilização são colocados no pacote Objetos Reusaveis.

O GDIG gera um conjunto limitado de tipos de janelas, mas cada tipo pode ser gerado infinitas vezes. Por exemplo, há apenas um tipo de janela de Cadastro, mas existem cadastros específicos de várias entidades. Então a interface pode gerar a janela de cadastro de qualquer entidade que esteja modelada no sistema. Existem outros tipos de janela como: Manejo Coletivo, Janela Principal, Correio, *Login*, Dicas, entre outras. Todas as classes que representam essas janelas ficam no pacote **Janelas**.

Visto que a interface manipula grande quantidade de dados, tabelas são úteis no sentido de facilitar o agrupamento, visualização, organização, ordenação e busca desses dados. Assim, quase todas as janelas contêm tabelas. Por isso a arquitetura de interface prevê uma estrutura de hierarquia coesa e pouco acoplada de tabelas. Todas elas estão no pacote **Tabelas**.

A barra de ferramentas é dinâmica, ou seja, ela muda de elementos e funcionalidades de acordo com o usuário que está utilizando o sistema. Todo o mecanismo de adaptação e de estruturação localiza-se no pacote **Barra de Ferramentas**.

Assim como a barra de ferramentas, os menus são criados dinamicamente pelo GDIG. Sua estruturação e mecanismo de criação localizam-se no pacote **Menus**. O GDIG permite que os desenvolvedores cadastrem menus e itens de menu. Um menu pode ter um outro menu como pai, formando um menu dentro de outro menu (submenu). Um item de menu, ao ser inserido no sistema, deve obrigatoriamente informar o seu menu pai. Desse modo o GDIG armazena o conjunto de menus e itens de menus em uma estrutura de árvores. Ao ser gerada a GUI, o GDIG acessa essa estrutura e monta os menus, submenus e itens de menu.

A criação dos componentes gráficos do sistema é feita dinamicamente. Logo, não é possível saber de antemão quais elementos uma janela irá conter, nem quais suas dimensões. Tudo isso depende da modelagem da entidade que será trabalhada pelo usuário.

Quando um usuário seleciona uma funcionalidade, por exemplo, cadastro de Pessoa Física, a interface obtém os metadados dessa entidade e cria os componentes gráficos correspondentes. As dimensões são estabelecidas, primeiramente, de acordo com a resolução de tela usada pelo usuário (800x600, 1200x768, etc.) e depois de acordo com o volume de dados que cada *widget* contém. Os espaços entre *widgets*, altura e largura de botões, etc, também são configuráveis, refletindo em toda a interface. Todo esse mecanismo de ajuste de dimensões encontra-se no pacote **Dimensões**.

Existem vários tipos de elementos gráficos para que o usuário possa interagir e entrar com os dados. Esses elementos são predefinidos e agrupados no pacote denominado **Paineis Atributos**. Este pacote contém o código-fonte que cria esses elementos, cada um com sua aparência e comportamentos já projetados e implementados. O GDIG recebe o Modelo de Apresentação e, de acordo com as informações nele contidas, define qual

elemento gráfico buscar no pacote Paineis Atributos. À medida que os elementos vão sendo obtidos a GUI vai sendo montada de acordo com o leiaute definido.

O pacote **Serviços** contém funções como consistência dos dados inseridos pelo usuário, formatação dos dados para enviar às camadas inferiores, obtenção dos dados das camadas inferiores, e formatação para obter uma boa visualização. A inteligência desses serviços encontra-se no pacote Serviços.

Na arquitetura proposta, o pacote **Programas Executáveis** contém as classes ".java" que são referenciadas pela interface em tempo de execução. Essa funcionalidade permite que o desenvolvedor defina nos metadados classes que ele deseja que sejam executadas em algum momento no sistema. Essas classes ".java" são armazenadas no pacote Programas Executáveis.

Informações do tipo data são constantemente utilizadas em SI, principalmente para manipulação de dados históricos. O pacote **Calendário** contém um conjunto de funcionalidades para o tratamento desse tipo de informação. Verificações, validações e transformações do formato da data entre diferentes idiomas são exemplos dessas funcionalidades.

3.4 Visão geral do mecanismo implementado

O mecanismo para geração da GUI foi implementado de acordo com a arquitetura apresentada na seção 3.3 utiliza modelos para representar e construir mecanismo GUI para SI. Deste modo, optou-se por uma abordagem não convencional em que os modelos passam a ser a preocupação central da equipe de Engenharia de Software, sendo o alicerce para a construção e adaptação de interfaces gráficas. Isto é diferente da abordagem convencional ou tradicional, onde o produto principal é o código-fonte [6, 31].

Existem dois tipos de modelos na abordagem proposta: o **Modelo de Negócio**, um modelo independente de plataforma contendo informações de negócio; e o **Modelo de Apresentação**, contendo informações relevantes sobre comportamento e aparência da interface a ser gerada.

Para que a interface gráfica seja gerada, os modelos passam por interpretações e transformações em tempo de execução. Primeiro, o Modelo de Negócio é transformado no Modelo de Apresentação e depois esse é transformado na Interface Concreta. Uma visão geral das transformações e dos modelos criados é mostrada na Figura 3.3.

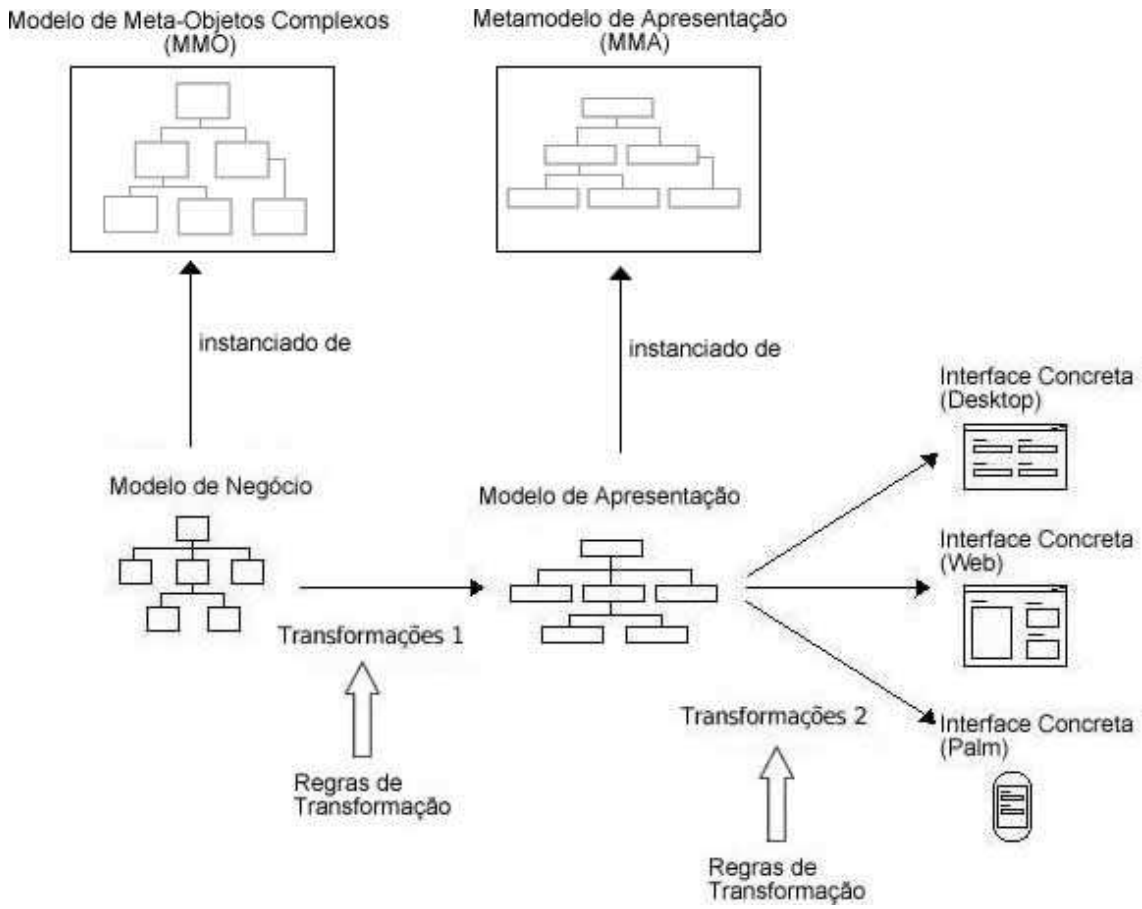

Figura 3.3: *Transformações entre modelos até a criação da interface*

O último passo da geração é a transformação do modelo específico (Modelo de Apresentação) para a interface propriamente dita. Nesta fase o gerador analisa os dados contidos no modelo específico e define, em tempo de execução, quais componentes gráficos serão inseridos na janela que está sendo construída.

3.5 Tráfego de Metadados e Dados de Negócio

A abordagem de geração dinâmica de GUIs proposta neste trabalho divide-se em duas fases distintas: **projeto** e **execução** de interface. A Figura 3.4 mostra as relações entre essas duas fases.

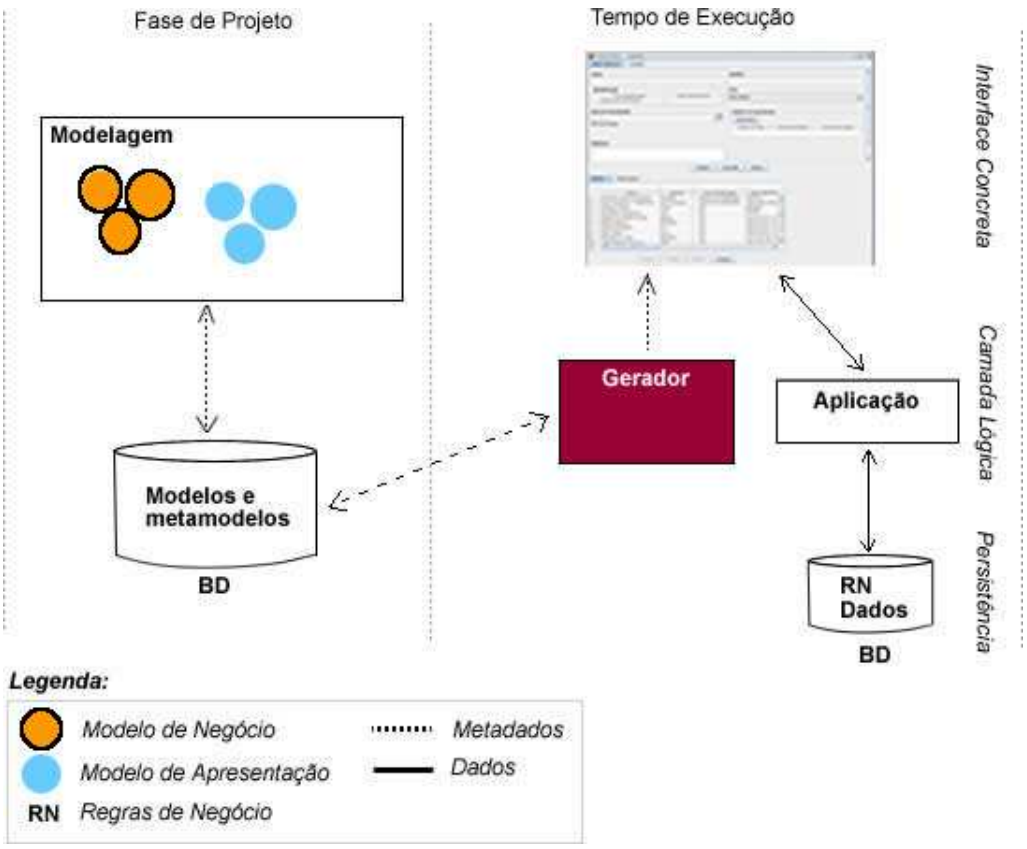

Figura 3.4: *Fases da abordagem MDD para gerência de interfaces*

Na fase de **projeto**, o projetista da interface gráfica cria o Modelo de Negócio (que é transformado no Modelo de Apresentação), insere informações específicas no Modelo de Apresentação e armazena-os no Banco de Dados (a Figura 3.3 mostra uma visão desse esquema). Nesta fase o GDIG trabalha somente com metadados, ou seja, informações sobre os modelos.

O GDIG disponibiliza um editor de modelos para que o usuário crie e ajuste as informações dos modelos que estão sendo criados. A Figura 3.5 mostra o editor de modelos. Este editor manipula, atualmente, os modelos de Negócio e Apresentação ao mesmo tempo. Esses modelos são conceitualmente separados, mas editados juntos e, para um trabalho futuro, pretende-se separar também sua edição. No campo “Atributos” são definidos os atributos do tipo de entidade que, neste caso, é “Pessoa Jurídica”, ou seja, o que é editado é o conceito de Pessoa Jurídica no SI (isto é, um conceito de negócio) e não uma instância desse conceito.

O GDIG aproveita seu próprio mecanismo de geração dirigida por modelos para gerar a Janela de Edição, que permite criar os modelos. Trata-se de um mecanismo recursivo: a Janela de Edição usada para criar os modelos que são usados para a geração das janelas CRUD é gerada dinamicamente a partir desses mesmos modelos. Ou seja, existe um Modelo de Negócio e um Modelo de Apresentação, criados dentro da Janela de

Edição, que descrevem a própria Janela de Edição. A Figura 3.6 mostra esse esquema.

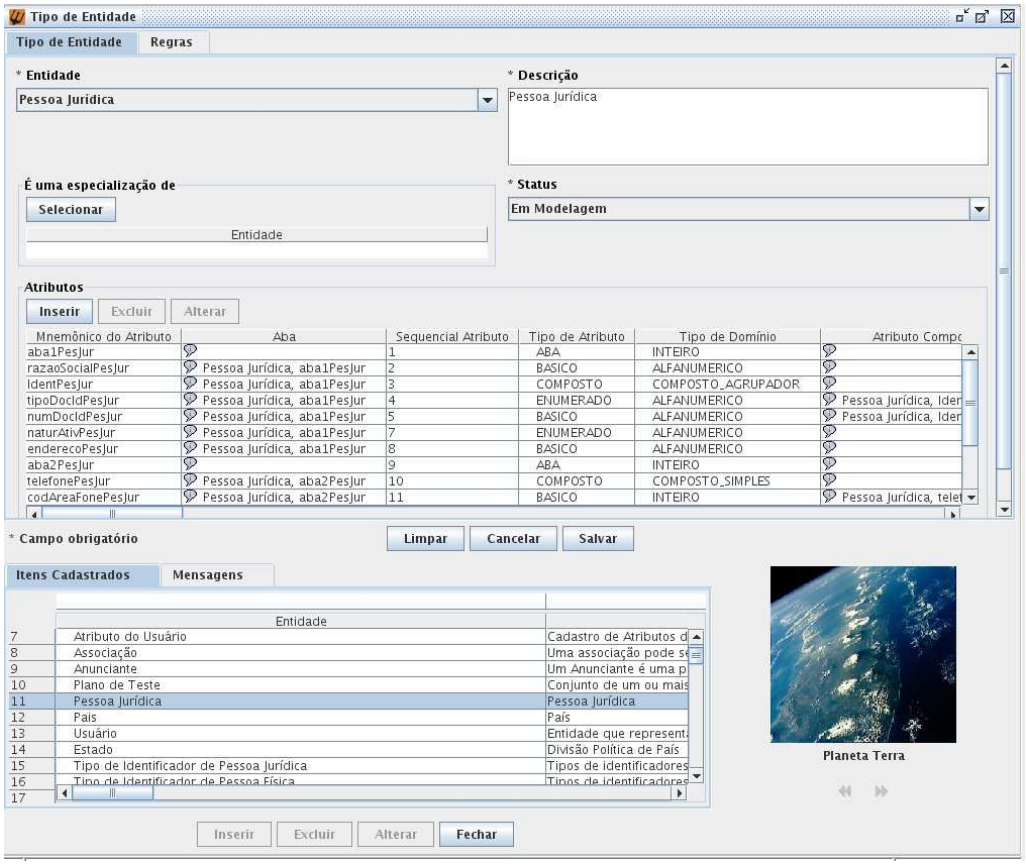

Figura 3.5: *Editor de modelos do GDIG*

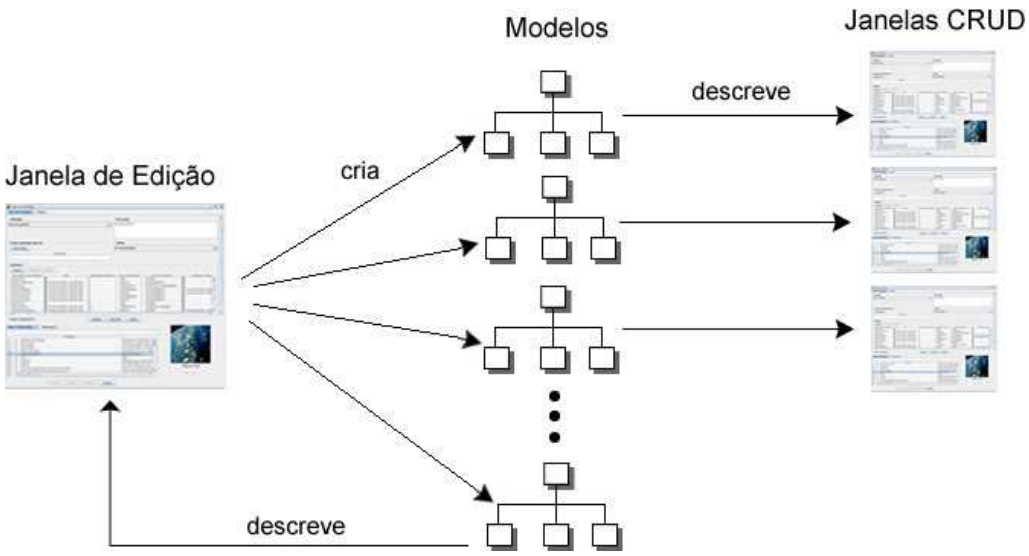

Figura 3.6: *Geração da Janela de Edição dirigida por modelos*

Antes de o usuário executar a abertura da GUI é necessário que esta esteja disponível para geração. Para isso, o projetista deve criar um item de menu, dentro de

algum menu do SI, que executará a criação dessa GUI, e autorizar essa funcionalidade ao usuário.

Uma vez disponibilizada a criação da GUI para o usuário, a fase de **execução** pode ser iniciada. Quando o usuário final requisita a abertura da janela clicando no item de menu, o GDIG recebe essa requisição, obtém do Banco de Dados os modelos referentes à janela (entidade), a monta e mostra ao usuário. Todo esse processo, desde o clique no item de menu até a apresentação da GUI, é feito em tempo de execução.

Até a geração da janela, houve somente o tráfego de metadados dentro do *framework* do qual o GDIG faz parte. A partir do momento em que a janela foi criada e acoplada à Janela Principal, o GDIG irá gerenciar o tráfego dos dados referentes ao negócio, ou seja, dados gerados, alterados ou removidos através das funcionalidades CRUD.

Para o tráfego de metadados e de dados de negócio, projetou-se uma estrutura onde as informações são encapsuladas em objetos e enviadas ou obtidas da camadas inferiores em canais específicos de comunicação. Preferiu-se utilizar a tecnologia Java ao invés de outras como XMI (*XML Metadata Interchange*) [33] por questão de desempenho.

O *workflow* de construção dos modelos e geração automática da GUI diferencia-se das demais abordagens propostas na literatura em dois aspectos principais. Primeiramente, os modelos são armazenados em um Banco de Dados, ao invés de outras técnicas de armazenamento como *Annotations* [20] ou arquivos XML [46]. Com isso, obtém-se independência referente a linguagens de programação. O armazenamento em *Annotations*, por exemplo, torna os metadados dependentes da linguagem de programação utilizada.

Em segundo lugar, o código que implementa a interface não é armazenado em disco. Não são armazenados código-fontes para uma janela gerada pelo GDIG. Todo o processo de leitura do modelo, criação e organização dos componentes gráficos é feito em tempo de execução e, após o fechamento da GUI, não ocorre qualquer tipo de registro de sua estrutura visual.

As únicas informações utilizadas para a construção das GUIs são aquelas dos modelos, diferentemente de outras abordagens que criam o código-fonte para posteriormente executá-lo [21]. A criação do código-fonte, juntamente com a permissão de sua modificação, pode dificultar um posterior reconhecimento do código modificado pela própria ferramenta.

O próximo capítulo descreve os detalhes da abordagem proposta para o gerenciamento de GUIs, analisando os modelos, descrevendo o mecanismo de geração e apresentando a interface gerada.

CAPÍTULO 4

Modelos para Geração de Interface

4.1 O Modelo de Negócio

O Modelo de Negócio é construído com base no metamodelo chamado de Modelo de Meta-Objetos Complexos (MMO), um modelo genérico que representa e descreve objetos complexos de sistemas de informação.

O MMO é capaz de representar diversos domínios de negócio. A generalidade do MMO vem da capacidade de representação de conhecimento do Paradigma OO e a utilidade vem das necessidades comuns aos diversos tipos de Sistemas de Informação (SI).

O objetivo do MMO é prover primitivas de modelagem representação do Modelo de Negócio de um SI, que consiste em entidades de negócio, atributos e associações de objetos. A Figura 4.1 apresenta a visão conceitual do MMO. Para facilitar a visualização, não foram mostrados os atributos das entidades. A seção 4.1.1 traz essas informações de forma detalhada.

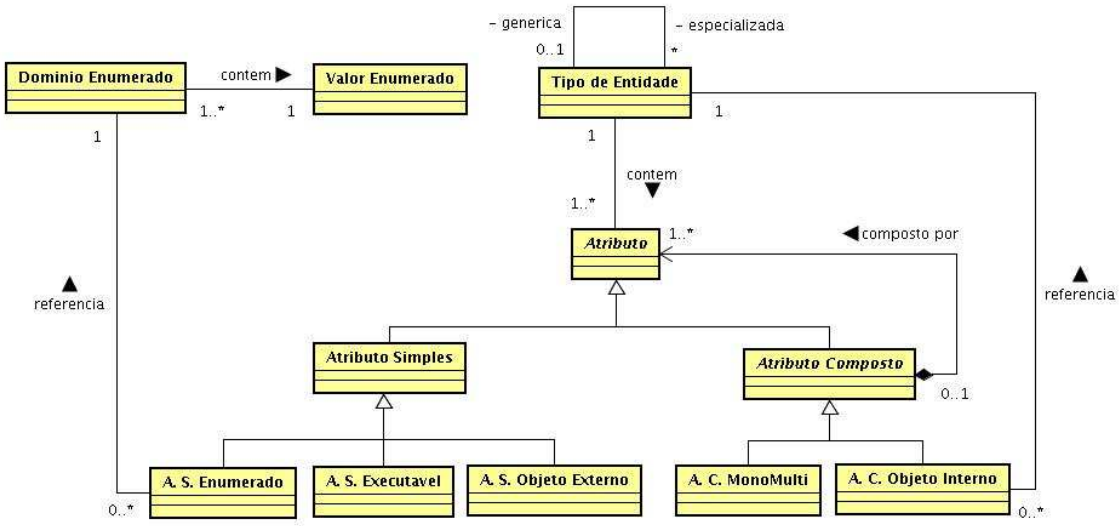

Figura 4.1: *Visão conceitual do MMO*

Um Tipo de Entidade contém as informações que servem de base para a geração da interface gráfica. Ele contém um mnemônico único para diferenciá-lo dos demais tipos, pode especializar ou generalizar outro tipo de entidade e é composto de um ou mais atributos. Cada atributo pode ser do tipo simples ou composto.

Os Atributos Simples podem ser do tipo Enumerado, Executável ou Objeto Externo. Os Enumerados são atributos cujo valor só pode ser um daqueles contidos no domínio enumerado predefinido. Os atributos do tipo Executável permitem executar programas dentro do sistema. Os atributos do tipo Objeto Externo contêm informações sobre arquivos ditos como externos ao sistema, como arquivos de texto e mídia (imagem, áudio e vídeo).

Os Atributos Compostos são formados de um ou mais atributos. O Composto do tipo Objeto Interno contém uma referência para uma outra entidade, ou seja, seu valor refere-se a dados de outras entidades já cadastradas no sistema.

Utilizando o MMO como linguagem, a tarefa do projetista consiste na criação de um Modelo de Negócio que representa as entidades do negócio com suas diversas características e associações. Posteriormente, cada entidade é transformada em uma interface CRUD pelo mecanismo interno do GDIG.

4.1.1 Descrição dos Atributos

Figura 4.2: *Visão detalhada do Atributo*

Um tipo de entidade dentro de um SI é composto por um ou mais Atributos, como mostra a Figura 4.2. Cada Atributo, no mecanismo interno do GDIG, corresponderá a um campo de formulário, onde o usuário poderá editar dados.

Um Atributo é uma entidade abstrata, ou seja, não pode ser instanciada. Seu objetivo é reunir informações e funcionalidades comuns para os demais Atributos especializados na hierarquia. São características de um Atributo e comuns a todos os demais atributos que o herdam:

String rotuloAgrupador Indica o agrupador do qual esse atributo faz parte. Um agrupador é um conceito dentro dos metadados que é usado pelo GDIG para agrupar os componentes gráficos visualmente como, por exemplo, numa mesma aba. Neste caso, para cada agrupador será criada uma aba na janela da GUI.

String mnemonico Representa o identificador do atributo. Esse mnemônico é *case sensitive* e único dentro de todo o sistema.

TipoAtributo tipo Define o tipo de atributo descrito. De acordo com o tipo de atributo, o GDIG sabe qual componente gráfico deve ser criado na geração da interface. Os tipos de atributos disponíveis são:

- **BASICO:** o atributo do tipo básico indica que é um atributo simples, podendo ser alfanumérico, decimal e data.
- **ENUMERADO:** indica que o atributo é Simples Enumerado.
- **OBJETO_EXTERNO:** indica que o atributo é do tipo Simples Objeto Externo.
- **ABA:** indica que é um atributo do tipo aba. Um atributo aba referencia um grupo de atributos que será usado pelo GDIG para agrupá-los visualmente.
- **EXECUTAVEL:** indica que o atributo é do tipo Executável.
- **OBJETO_INTERNO:** indica que o atributo é do tipo Composto Objeto Interno.
- **COMPOSTO:** indica que o atributo é do tipo Composto MonoMulti

TipoDominio dominio O domínio refere-se ao tipo de valor que este atributo armazena. Por exemplo, caso seja um atributo do tipo básico, pode ter o tipo de domínio como alfanumérico ou data. Essas informações orientam o GDIG na criação da GUI.

São tipos de domínio:

- **INTEIRO:** indica que o atributo descreve um dado numérico do tipo inteiro.
- **ALFANUMERICO:** indica que o atributo descreve um dado alfanúmérico, ou seja, uma string.
- **DATA:** indica que o atributo descreve um dado do tipo data.

- **MOEDA:** indica que o atributo descreve um dado do tipo moeda como, por exemplo, Real ou Dólar.
- **DECIMAL:** indica que o atributo descreve um dado numérico decimal.
- **COMPOSTO_SIMPLES:** indica que o atributo descreve um dado composto simples. Isso significa que a interface deverá tratá-lo diferentemente dos demais tipos de compostos. A Seção 5.2.3 mostra o que é criado pelo GDIG quando encontra um composto deste tipo.
- **COMPOSTO_COMPLEXO:** indica que o atributo descreve um dado composto complexo. A Seção 5.2.3 descreve o que é criado pelo GDIG quando encontra um composto deste tipo.
- **COMPOSTO_LISTA:** indica que o atributo descreve um dado composto do tipo lista. A Seção 5.2.3 define o que significa um atributo pertencer a esse domínio.
- **COMPOSTO_AGRUPADOR:** indica que o atributo descreve um dado composto agrupador. A Seção 5.2.3 define o que significa um atributo pertencer a esse domínio.
- **BOOLEANO:** indica que o atributo descreve um dado do tipo booleano, que aceita os valores verdadeiro ou falso.

int cardMin Define a cardinalidade mínima do atributo, ou seja, qual o número mínimo de valores que o atributo pode ter.

int cardMax Define a cardinalidade máxima do atributo, ou seja, qual o número máximo de valores que o atributo pode ter. Determina se o atributo é mono ou multivalorado.

boolean ehParteChave Esta característica indica se o atributo faz parte de alguma chave candidata dentro do Banco de Dados. É útil para que o sistema de persistência possa executar suas validações e para que a interface possa inserir um visual diferenciado para esses campos, ou para que execute algum tipo de verificação.

boolean ehUnico Determina se o atributo é único. Desta forma, não poderá haver dois valores iguais no Banco de Dados.

boolean ehAtribLigacao O *framework* está capacitado para gerenciar relacionamentos N para N do Banco de Dados. Para que possa, automaticamente, executar operações de junção, devem ser definidos quais atributos, dentro de uma tabela, podem fazer parte da junção. Esse é o objetivo do atributo de ligação. Caso seja ajustado como verdadeiro (*true*), isso significa que o mecanismo de persistência irá usá-lo para realizar a junção.

Atributo Simples

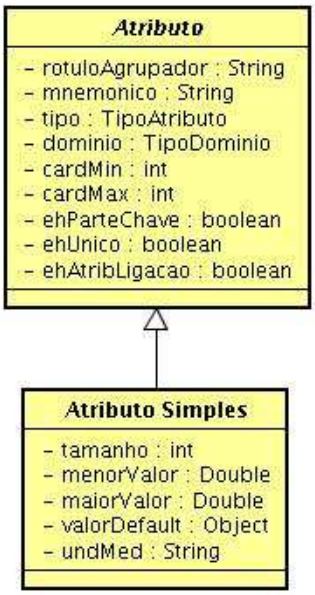

Figura 4.3: *Visão detalhada do Atributo Simples*

O Atributo Simples representa um dado atômico dentro da entidade da qual ele faz parte. Ou seja, essa informação não pode ser dividida em sub-partes. A Sub-seção 4.1.1 detalha um Atributo Composto, ideologicamente o oposto do Atributo Simples.

Como mostra a Figura 4.3, além das características herdadas de Atributo, o Atributo Simples contém as seguintes.

int tamanho Essa característica limita o tamanho máximo em caracteres que o Atributo Simples poderá ter. É usado somente no caso de valores do tipo alfanumérico, já que outros tipos como valores numéricos contêm os controles *menorValor* e *maiorValor*.

Double menorValor Caso o Atributo Simples contenha um Tipo de Domínio numérico, a característica *menorValor* determinará o valor mínimo permitido para este atributo.

Double maiorValor Caso o Atributo Simples contenha um Tipo de Domínio numérico, a característica *maiorValor* determinará o valor máximo permitido para este atributo.

Object valorDefault Como o Atributo Simples representa um campo de uma entidade que receberá informações vindas do usuário final, o modelador do sistema poderá determinar que o referido campo terá algum valor como padrão (*default*). Ou seja, o sistema mostrará de início esse valor ao usuário, que poderá ou não modificá-lo.

String undMed Um Atributo Simples pode descrever uma informação que contém uma unidade de medida como, por exemplo, de comprimento. Neste caso deverá haver a unidade a que o atributo se refere. Esta informação está presente na característica *undMed*.

Atributo Simples Enumerado

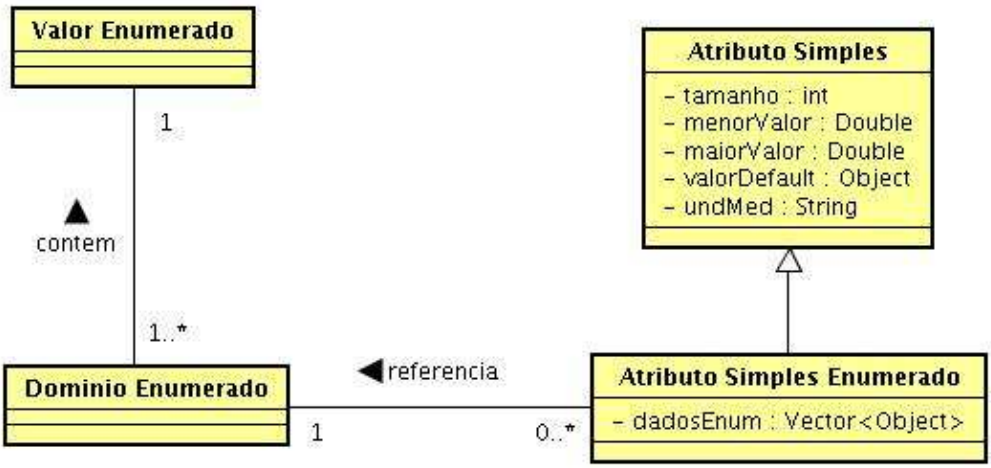

Figura 4.4: *Visão detalhada do Atributo Simples Enumerado*

Grande parte dos sistemas que gerenciam informações contém campos que armazenam valores enumerados. O Atributo Simples Enumerado é aquele que, além de ser simples, descreve dados oriundos de um conjunto de valores pré-determinados, isto é, uma enumeração. Um exemplo de enumeração é o conjunto de dados *masculino, feminino*.

Além das características herdadas de Atributo Simples, como mostra a Figura 4.4, o Atributo Simples Enumerado contém as seguintes.

String dominioEnum O *dominioEnum* define o conjunto onde estão localizados determinados valores. O domínio é necessário porque alguns valores podem se repetir, por isso devem ser separados pelo domínio. Por exemplo, podem existir os valores *branco, marrom* para cor de pêlo de animal e para cor de uniforme de empregado. Neste caso, serão necessárias as criações dos domínios *corPelo* e *corUniforme*, cada um com seus valores enumerados.

Vector dadosEnum Esta característica referencia o conjunto de valores enumerados.

Atributo Simples Executável

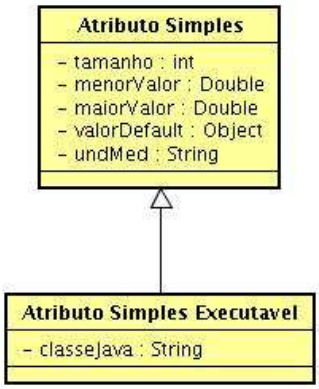

Figura 4.5: *Visão detalhada do Atributo Simples Executável*

Ações como abrir uma GUI de um aplicativo ou módulo interno ao sistema são descritas no Atributo Simples Executável.

Além das características herdadas de Atributo Simples, o Atributo Simples Executável contém as seguintes, como detalha a Figura 4.5:

String classeJava Essa característica referencia um arquivo *.class* que é instanciado em tempo de execução. Esse arquivo pré-compilado poderá executar ações diversas dentro do sistema.

Atributo Simples Objeto Externo

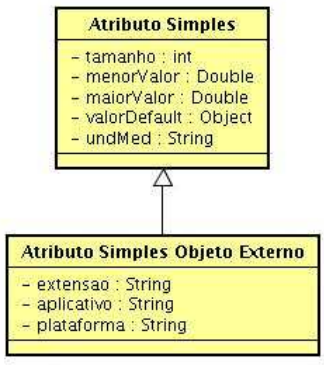

Figura 4.6: *Visão detalhada do Atributo Simples Objeto Externo*

Existem dois tipos de objetos dentro do sistema, os denominados Objetos Externos e os Objetos Internos. Os Atributos Objeto Externo contêm informações sobre entidades ditas como externas ao sistema como, por exemplo, arquivos de texto e imagem. São entidades que não são obtidas do Banco de Dados.

Ações como abrir uma imagem, acionar um *player* de áudio ou vídeo, ou visualizar uma página *web* através de um *browser* são comuns em grandes sistemas. Um Atributo Simples Objeto Externo descreve informações acerca dessas ações a serem executadas.

A Figura 4.6 traz os detalhes do Atributo Simples Objeto Externo que, além das características herdadas de Atributo Simples, contém as seguintes.

String extensao Descreve a extensão do objeto externo como, por exemplo, *.doc*, *.html*, *.jpg*.

String aplicativo Esta característica indica qual aplicativo deve ser usado para abrir o objeto externo.

String plataforma Determina qual plataforma é necessária para que o objeto externo seja aberto.

Atributo Composto

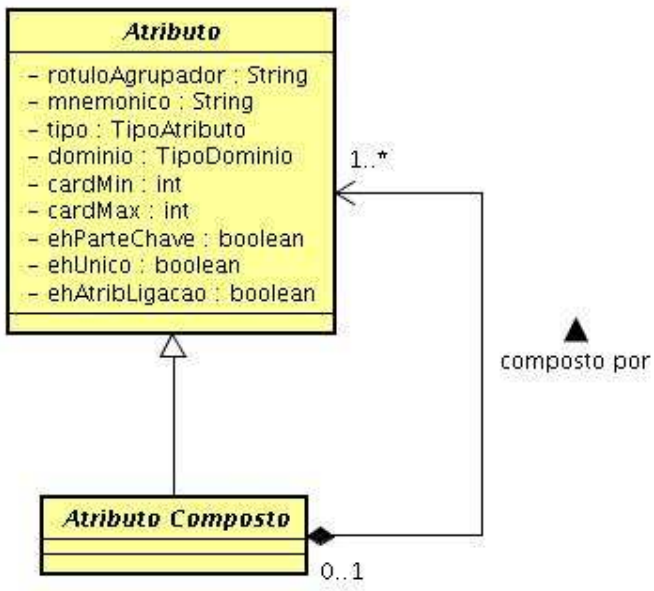

Figura 4.7: *Visão detalhada do Atributo Composto*

Ao contrário do Atributo Simples, o Atributo Composto é formado por um ou mais atributos. Os atributos compositores, por sua vez, podem também ser do tipo Composto, ou seja, serem formados por outros atributos, gerando, dessa forma, uma hierarquia de composição.

O Atributo Composto é abstrato e por isso não pode ser instanciado, como mostra a Figura 4.7. Além das características herdadas de Atributo, o Atributo Composto contém as seguintes:

Vector<Atributo> subAtributos Esta característica referencia o conjunto de atributos que compõem o Atributo Composto. Neste contexto, são chamados de subatributos.

Atributo Composto MonoMulti

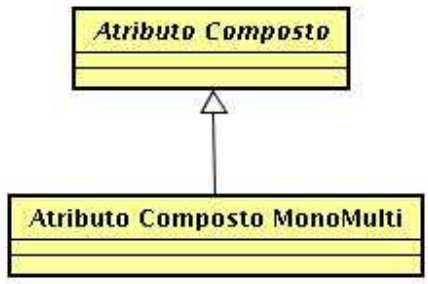

Figura 4.8: *Visão detalhada do Atributo Composto MonoMulti*

O objetivo do Atributo Composto MonoMulti é semelhante ao do Atributo Composto, ou seja, servir como um agrupador de sub-atributos.

Como mostra a Figura 4.8, o Atributo Composto MonoMulti herda todas as características do Atributo Composto.

Atributo Composto Objeto Interno

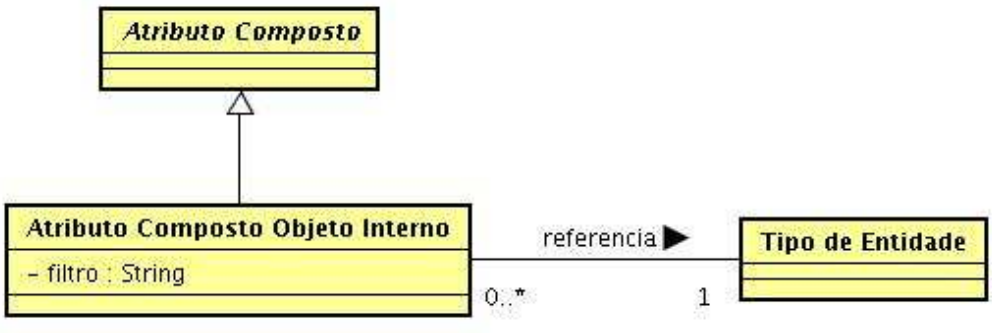

Figura 4.9: *Visão detalhada do Atributo Composto Objeto Interno*

Outro tipo de Atributo Composto é o Atributo Composto Objeto Interno (Figura 4.9). Um objeto interno é aquele que referencia uma entidade interna ao sistema. Esse atributo contém suas próprias descrições e, ao mesmo tempo, as descrições da entidade referenciada. Por isso foi denominado como “interno”, porque a entidade referenciada encontra-se armazenada dentro do Banco de Dados.

Além das características herdadas de Atributo Composto, o Atributo Composto Objeto Interno contém as seguintes:

String entReferencia A *entReferencia* contém o mnemônico da entidade que o Atributo Composto Objeto Interno referencia.

String filtro Como o Atributo Composto Objeto Interno referencia dados de uma outra entidade, a escolha desses dados é feita na GUI que lista as informações para que o usuário as selecione. O *filtro* restringe essa lista de informações de acordo com a String referenciada. Caso não haja o filtro, o aplicativo trará todas as informações ao usuário.

4.2 O Modelo de Apresentação

Ao contrário do Modelo de Negócio, que contém informações que podem ser utilizadas por todo o *framework* (pelo GDIG, pelo sistema de persistência e pelo sistema de regras de negócio), o Modelo de Apresentação é um modelo específico para a geração das interfaces gráficas, ou seja, contém informações que são utilizadas somente pelo GDIG. O Modelo de Apresentação é criado como resultado da aplicação de regras de transformação sobre o Modelo de Negócio e baseado no Metamodelo de Apresentação (MMA).

O MMA contém todos os elementos necessários para a criação do Modelo de Apresentação. A Figura 4.10 ilustra o MMA. Para facilitar a visualização, não foram mostradas as características (atributos) das entidades na figura. A Seção 4.2.1 traz essas informações de forma detalhada.

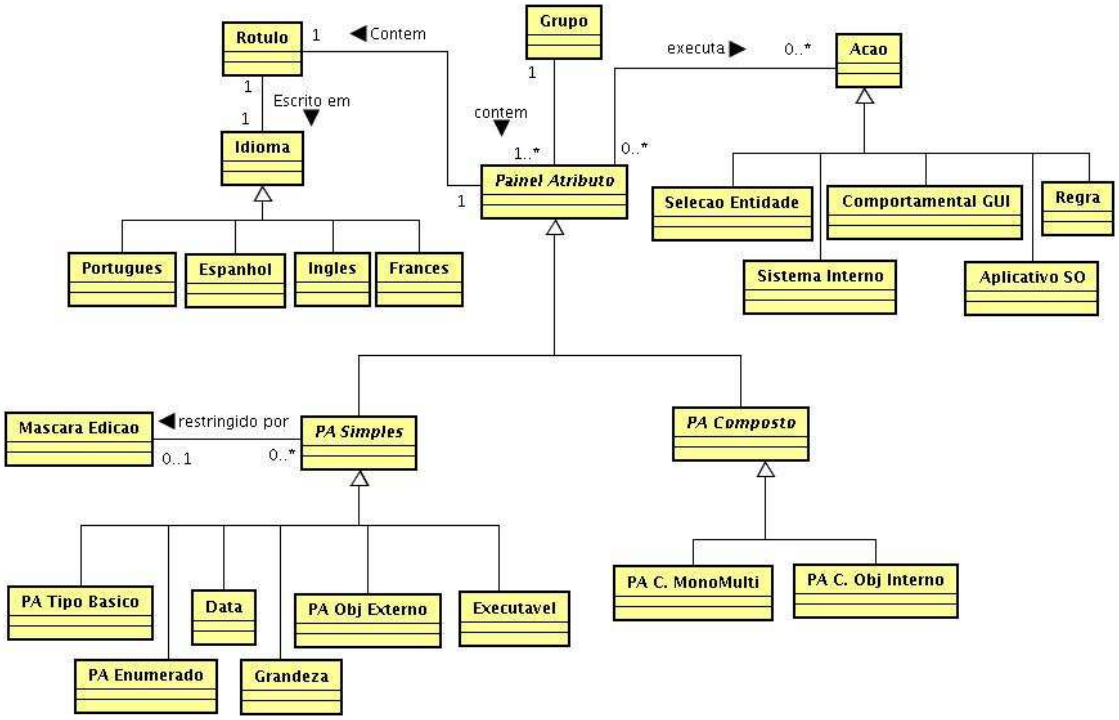

Figura 4.10: *Visão conceitual do MMA*

O metamodelo ilustrado na Figura 4.10 mostra os conceitos utilizados para a geração da interface gráfica, contendo informações específicas para GUIs. Cada campo para entrada de dados na interface gráfica é definido como um Painel Atributo. Ele contém um rótulo que pode ser definido em quatro linguagens: português, espanhol, inglês ou francês.

Cada Painel Atributo descreve um campo de formulário dentro da interface CRUD para que o usuário entre com informações a serem gerenciadas. Para que o GDIG possa ter uma referência de agrupamento desses painéis atributos, cada Painel Atributo deve fazer parte de um Grupo.

Existem vários tipos de painéis atributos específicos, cada um responsável por referenciar diferentes tipos de componentes gráficos e diferentes tipos de ações.

4.2.1 Descrição dos Painéis Atributos

Cada Painel Atributo contém informações necessárias para que o GDIG crie os componentes gráficos das GUIs.

As sub-seções seguintes trazem descrições detalhadas da hierarquia de painéis atributos da Figura 4.10, composta por: Painel Atributo, Painel Atributo Simples e suas especificações (Tipo Basico, Enumerado, Data, Grandeza, Objeto Externo, Texto Longo e Executável), Painel Atributo Composto e suas especificações (Painel Atributo Composto MonoMulti e Painel Atributo Composto Objeto Interno).

Painel Atributo

Dentro de uma interface CRUD, os componentes gráficos são sempre agrupados, seja por motivos de organização ou por serem elementos de uma informação composta. No MMA, esse agrupamento é indicado pelo atributo “Grupo”. Cada Painel Atributo deve, obrigatoriamente, localizar-se dentro de um determinado Grupo. Posteriormente, na fase de criação da GUI, cada grupo corresponderá a uma aba que conterá os componentes gráficos de cada Painel Atributo. A Seção 5.2 traz os detalhes da criação da GUI.

Um Painel Atributo é uma entidade abstrata, ou seja, não pode ser criada isoladamente. Seu objetivo é reunir informações comuns para os demais Painéis Atributos específicos na hierarquia. A Figura F traz as associações e características do um Painel Atributo.

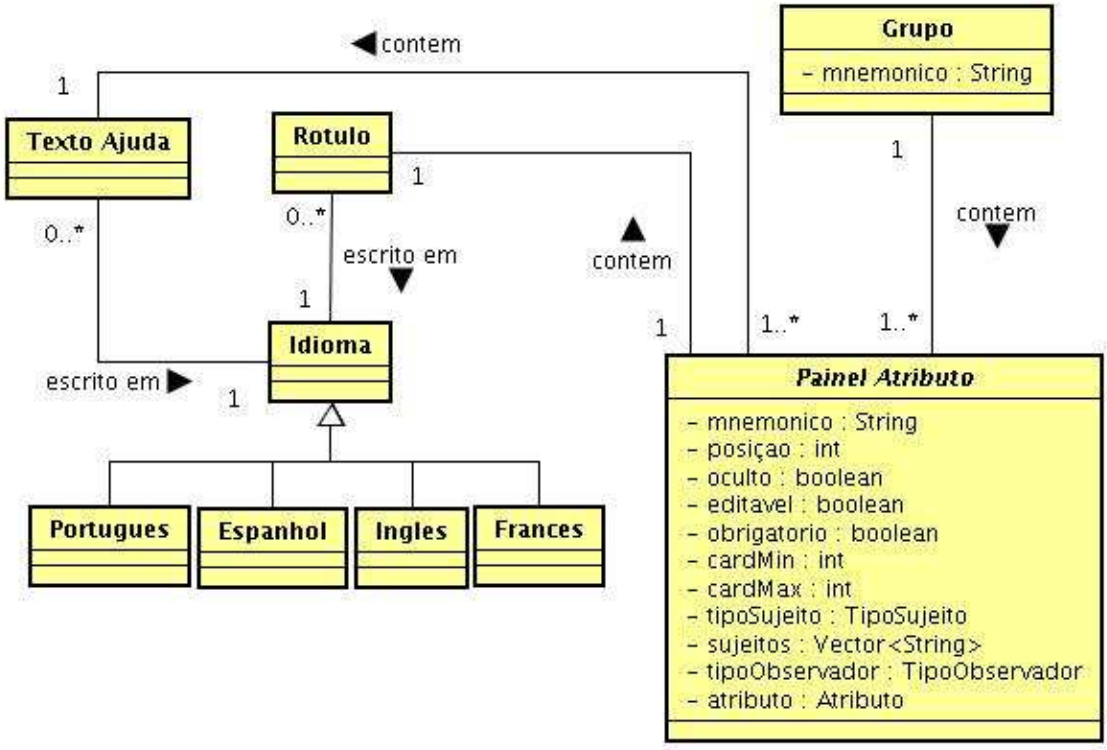

Figura 4.11: *Visão detalhada do Painel Atributo*

Cada Painel Atributo faz parte de um agrupador, contém um rótulo representando o texto visível ao usuário e contém um texto de ajuda ao usuário com intuito de trazer uma explicação sobre aquele campo dentro do formulário. São características de um Painel Atributo:

String mnemonico Esta característica é obtida do Atributo sem nenhuma modificação. Representa um identificador único do Painel Atributo.

int posicao Esta característica define a posição que o Painel Atributo irá ocupar dentro do grupo do qual faz parte.

boolean oculto Define se o campo que o Painel Atributo representa será ou não visível ao usuário. Caso contenha o valor *verdadeiro*, o Painel Atributo estará presente no Modelo Apresentação mas não será visível ao usuário.

Este caso é útil quando se deseja manter algum campo que não precisa ser editado pelo usuário como, por exemplo, uma chave ou identificador definido pelo sistema.

boolean editavel Determina se o campo que o Painel Atributo representa terá ou não permissão de edição pelo usuário. Caso contenha o valor *verdadeiro*, o usuário poderá ajustar o valor do campo na janela. Caso contrário, o campo terá sua edição bloqueada.

Este caso é útil quando, por exemplo, o campo contiver um dado derivado, ou seja, cujo valor é obtido por algum cálculo dentro do sistema, não necessitando, deste modo, de intervenção do usuário final.

boolean obrigatorio Este atributo contém informação acerca da obrigatoriedade de conter ou não algum valor. Caso seja *verdadeiro*, o campo deverá conter algum valor, inserido pelo usuário ou obtido da característica *valorDefault*.

int cardMin Define a cardinalidade mínima do Painel Atributo, ou seja, o número mínimo de valores que o campo representado por esse Painel Atributo pode ter.

int cardMax Define a cardinalidade máxima do Painel Atributo, ou seja, o número máximo de valores que o campo representado por esse Painel Atributo pode ter. Determina se o atributo é mono ou multivalorado.

TipoSujeito tipoSujeito O GDIG contém um mecanismo de comportamento de seus componentes gráficos utilizando o padrão *Observer* [17]. Este mecanismo faz com que um componente gráfico execute uma ação caso algum outro componente tenha seu valor modificado. Por exemplo, o componente X torna-se não editável caso o componente Y modifique seu valor. Neste caso, o componente X observa as modificações ocorridas no componente Y, ou seja, assim que o componente Y modifica seu valor, ele avisa a todos seus observadores e, desse modo, cada observador executa sua ação específica. Nesse esquema, o componente X é chamado de Observador (*Observer*) e o Y de Sujeito (*Subject*).

Portanto, a característica *tipoSubject* determina se esse atributo é um sujeito do padrão *Observer*.

Vector<String> sujeitos Esta característica traz os sujeitos que este atributo observa. Ela está vinculada à característica *tipoObserver*, ou seja, caso o atributo seja de algum tipo de observador, necessariamente terá que observar algum sujeito. O contrário também é valido, caso não seja de nenhum tipo de observador, não terá nenhum sujeito a observar.

TipoObservador tipoObservador Seguindo o raciocínio da característica *tipoSubject*, o *tipoObserver* define o tipo de observador que esse atributo implementa. Caso não implemente nenhum, seu valor é definido como *null*.

Os tipos de observadores diferem-se pelo comportamento que os mesmos implementam como, por exemplo, tornar-se não editáveis, modificar seus valores, iniciar um programa, dentre outros.

Atributo atributo Esta característica mantém uma referência para o Atributo respectivo ao Painel Atributo. Essa ligação é necessária para o fato de o Painel Atributo necessitar de acessar alguma informação do Atributo.

Cada Painel Atributo específico irá conter a referência para seu atributo específico. Por exemplo, o Painel Atributo Simples referenciará o atributo do tipo Atributo Simples, ao invés de referenciar o atributo do tipo Atributo.

Painel Atributo Simples

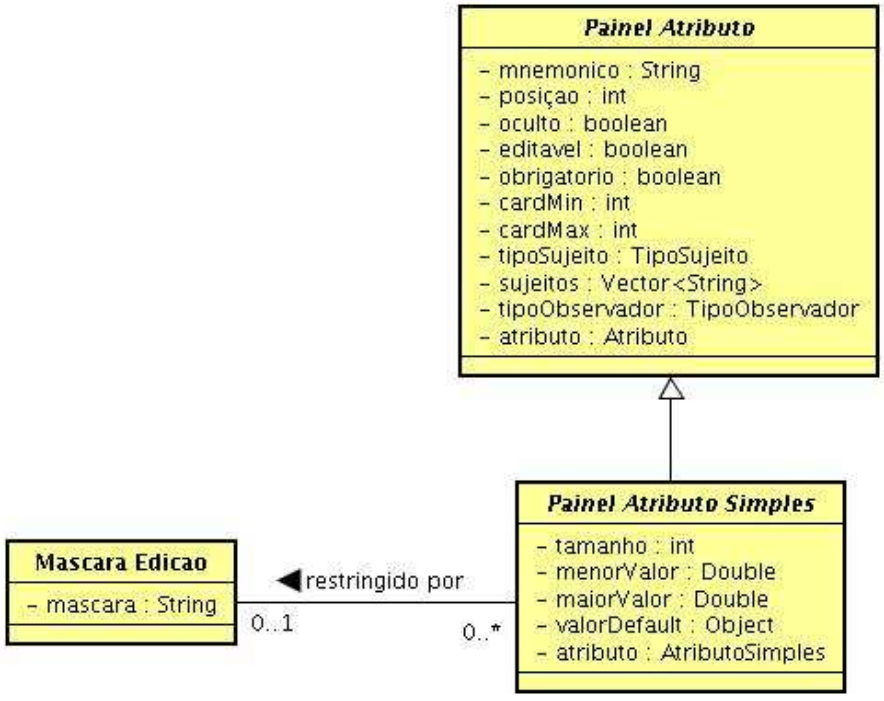

Figura 4.12: *Visão detalhada do Painel Atributo Simples*

O Painel Atributo Simples é indivisível, atômico e não-composto. Define uma máscara de edição onde o usuário insere um valor no campo de acordo com sua formatação. Um exemplo de máscara é a de data: *dd/mm/aaaa*. Através da máscara, o sistema poderá validar as informações e verificar se elas estão de acordo com a máscara de edição.

Como mostra a Figura 4.12, além das características herdadas do Painel Atributo, contém as seguintes:

int tamanho Limita o tamanho máximo, em caracteres, que o valor do campo representado pelo Painel Atributo Simples poderá ter. É usado somente no caso de valores do tipo alfanumérico, já que outros tipos, como valores numéricos, contém os controles *menorValor* e *maiorValor*.

Double menorValor A característica *menorValor* determina o valor mínimo permitido para o campo cujo Painel Atributo representa.

Double maiorValor A característica *maiorValor* determina o valor máximo permitido para o campo cujo Painel Atributo representa.

Object valorDefault O modelador do sistema poderá determinar que o campo descrito pelo Painel Atributo Simples terá algum valor como padrão (*default*). Ou seja, o sistema mostrará de início esse valor ao usuário, que poderá ou não modificá-lo.

AtributoSimples atributo Esta característica mantém uma referência para o Atributo Simples respectivo ao Painel Atributo Simples com objetivo de, caso necessário, ter acesso a informações do Atributo Simples do Modelo de Negócio.

Painel Atributo Tipo Basico

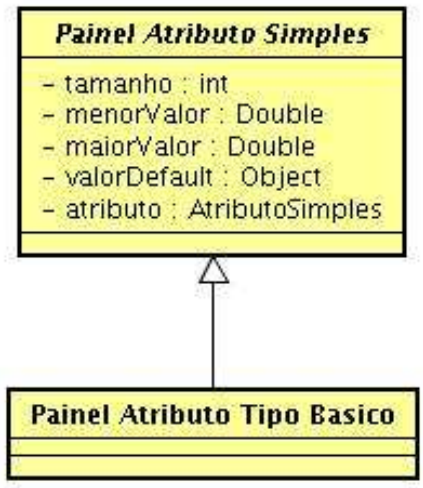

Figura 4.13: *Visão detalhada do Painel Atributo Tipo Basico*

O Painel Atributo Tipo Basico representa um campo básico dentro do formulário da janela. Um campo básico é aquele onde o usuário insere os dados em um único campo de texto.

Este painel herda todas as características do Painel Atributo Simples, como mostra a Figura 4.13.

Painel Atributo Enumerado

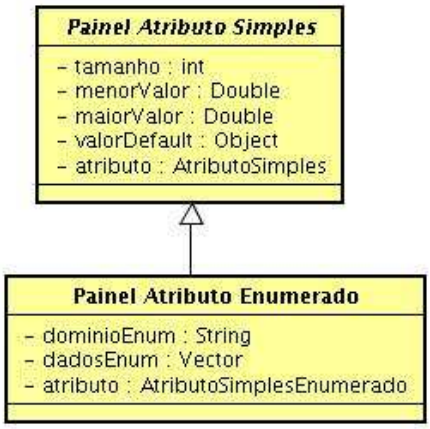

Figura 4.14: *Visão detalhada do Painel Atributo Enumerado*

O Painel Atributo Enumerado representa um campo da janela onde o usuário irá inserir um valor escolhido a partir de um conjunto de valores predefinidos.

Como mostra a Figura 4.14, além das características herdadas do Painel Atributo Simples, o Painel Atributo Enumerado contém as seguintes propriedades específicas:

String dominioEnum O *dominioEnum* define o conjunto pré-determinado de valores de onde o usuário escolherá o(s) valor(es) para o campo no formulário.

Vector dadosEnum Esta característica referencia o conjunto de valores enumerados que estão dentro do domínio enumerado.

AtributoSimplesEnumerado atributo O Painel Atributo Enumerado referencia um atributo do tipo Atributo Simples Enumerado do Modelo de Negócio.

Painel Atributo Data

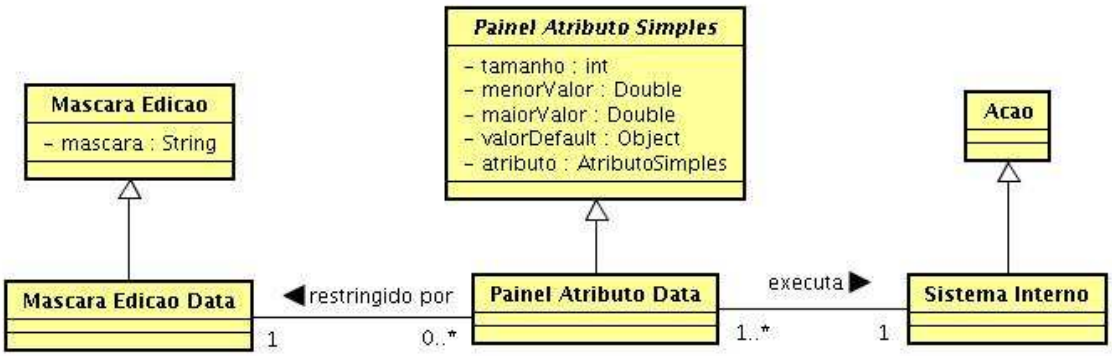

Figura 4.15: *Visão detalhada do Painel Atributo Data*

O Painel Atributo Data herda todas as características do Painel Atributo Simples. Além dessas, como mostra a Figura 4.15, contém outras específicas: (i) é restringido por uma máscara de edição de data (*dd/mm/aaaa*) e (ii) executa um mini-aplicativo de calendário para que o usuário possa escolher a data selecionando-a com o *mouse*, ao invés de inseri-la manualmente.

Por ter essas características específicas, decidiu-se criar um painel atributo restrito para informações do tipo data.

Painel Atributo Grandeza

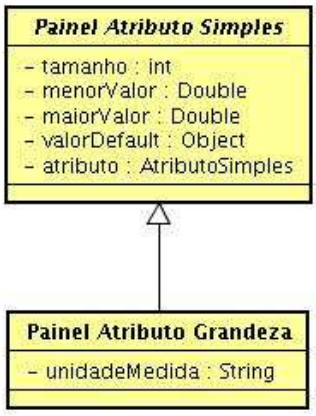

Figura 4.16: *Visão detalhada do Painel Atributo Grandeza*

O Painel Atributo Grandeza representa um campo que contém valores referentes a uma grandeza medida como, por exemplo, comprimento, peso ou volume. A Figura 4.16 traz o detalhamento do Painel Atributo Grandeza que, além de herdar as características do Painel Atributo Simples, contém:

String unidadeMedida Representa a unidade de medida referente à grandeza em questão. Exemplo: *m* para metro, *Km* para quilômetro, *Kg* para quilo.

Painel Atributo Objeto Externo

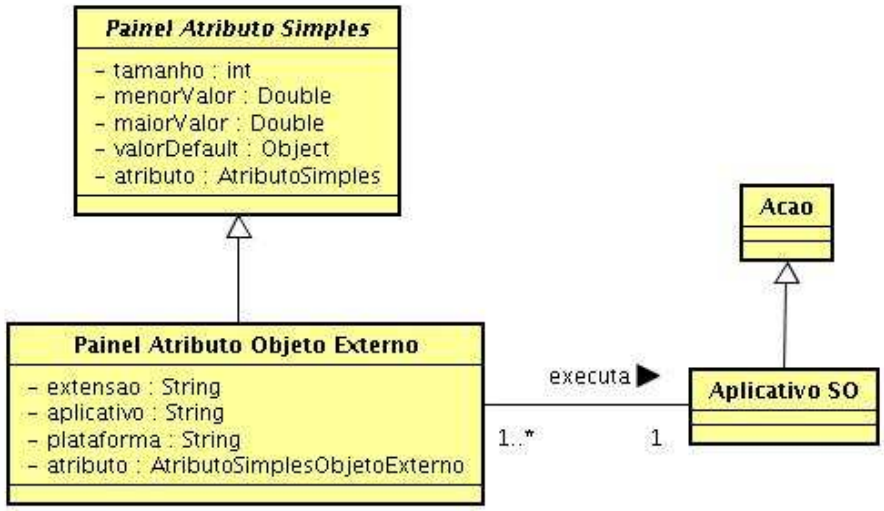

Figura 4.17: *Visão detalhada do Painel Atributo Objeto Externo*

A função do Painel Atributo Objeto Externo é representar um campo cujo valor refere-se a um dado denominado como Objeto Externo. O conceito de Objeto Externo foi descrito na Seção 4.1.1.

Esse painel executa a ação do tipo Aplicativo SO. Essa ação consiste em abrir um aplicativo que manipule o objeto externo em questão de acordo com a extensão, aplicativo e plataforma definidos nas características correspondentes.

Como mostra a Figura 4.17, além das características herdadas de Painel Atributo Simples, o Painel Atributo Objeto Externo contém as seguintes características específicas:

String extensao Descreve a extensão do objeto externo como, por exemplo, *doc*, *html*, *jpg*.

String aplicativo Esta característica indica qual aplicativo tem a preferência para ser reconhecido para abrir o objeto externo. Caso esse aplicativo não seja encontrado, poderá ser usado algum outro. Ou seja, primeiro o GDIG vai verificar se existe esse aplicativo e, caso não encontre, procura algum outro que execute o objeto externo.

String plataforma Determina qual plataforma é necessária para que o objeto externo seja aberto. Essa informação é útil pois alguns aplicativos têm a restrição de serem executados em algumas plataformas específicas.

AtributoSimplesObjetoExterno atributo O Painel Atributo Objeto Externo referencia um Atributo Simples Objeto Externo do Modelo de Negócio.

Painel Atributo Executável

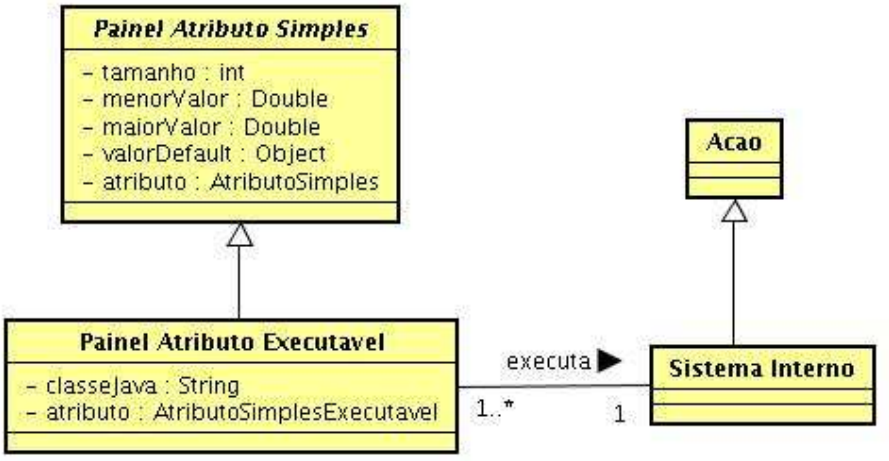

Figura 4.18: *Visão detalhada do Painel Atributo Executavel*

O Painel Atributo Executável representa um campo cujo objetivo é executar alguma funcionalidade interna do *framework*, abrindo uma GUI do sistema.

Além das características herdadas pelo Painel Atributo Simples, como detalha a Figura 4.18, o Painel Atributo Executável contém as seguintes características específicas:

String classeJava Essa característica referencia um arquivo *.class* que é instanciado em tempo de execução. Esse arquivo pré-compilado poderá executar ações diversas dentro do sistema.

AtributoSimplesExecutavel atributo O Painel Atributo Executável referencia um Atributo Simples Executavel do Modelo de Negócio para obter informações do Atributo e controlar suas aparências e comportamentos (do Painel Atributo Executável).

Painel Atributo Composto

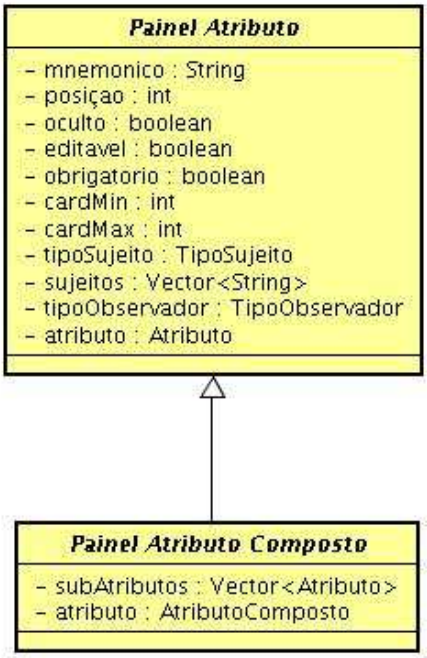

Figura 4.19: *Visão detalhada do Painel Atributo Composto*

O Painel Atributo Composto (Figura 4.19) é abstrato e por isso não pode ser criado isoladamente. Além das características herdadas de Painel Atributo, o Atributo Composto contém as seguintes.

Vector<Atributo> subAtributos Esta característica referencia o conjunto de atributos que compõem o Painel Atributo Composto, seus atributos compositores. Neste contexto, são chamados de subatributos.

AtributoComposto atributo O Painel Atributo Composto sobrescreve o atributo herdado de Painel Atributo, referenciando um atributo do tipo *AtributoComposto* do Modelo de Negócio, para obter informações do Atributo e controlar suas aparências e comportamentos (do Painel Atributo Composto).

Painel Atributo Composto MonoMulti

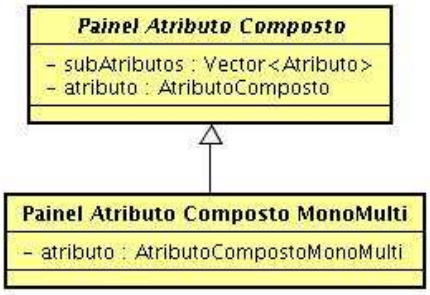

Figura 4.20: *Visão detalhada do Painel Atributo Composto Mono-Multi*

O Painel Atributo Composto MonoMulti herda todas características do Painel Atributo Composto, exceto o atributo, que é sobrescrito de modo a referenciar seu tipo específico *AtributoCompostoMonoMulti*. Essas características podem ser visualizadas na Figura 4.20.

Painel Atributo Composto Objeto Interno

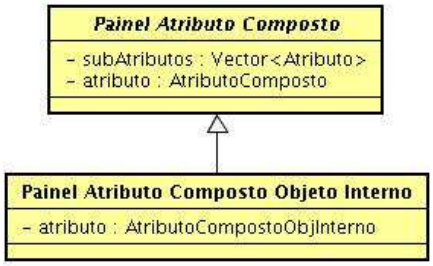

Figura 4.21: *Visão detalhada do Painel Atributo Composto Objeto Interno*

O Painel Atributo Composto Objeto Interno descreve um campo do formulário cujos dados são oriundos de outra entidade já cadastrada no sistema. Por isso o nome "Objeto Interno". A Seção 4.1.1 traz uma explicação detalhada deste conceito.

Como mostra a Figura 4.21, este painel herda todas as características do Painel Atributo Composto e sobrescreve o atributo, sendo este do tipo *AtributoCompostoObjInterno* do Modelo de Negócio.

CAPÍTULO 5

Geração de Interface Baseada em Modelo

5.1 Mapeando o Modelo de Negócio para o Modelo de Apresentação

Descritos os modelos de Negócio e Apresentação, este capítulo discute como funcionam as regras para a transformação de um Modelo de Negócio em um Modelo de Apresentação.

Como mostra a Seção 3.3, as regras de transformação são aplicadas, mapeando o Modelo de Negócio para o Modelo de Apresentação.

O Modelo de Negócio é formado por um conjunto de Atributos que contêm informações acerca de uma entidade dentro do sistema. O Modelo de Apresentação é composto por um conjunto de Painéis Atributos que contêm informações sobre a GUI a ser gerada.

Para transformar o conjunto de Atributos no conjunto de Painéis Atributos, o GDIG segue o algoritmo 5.1:

Algoritmo 5.1: transformarMNemMA(Vetor<Atributo> MN)

Entrada: vetor $MN[A1..An]$, $A1$ a An são Atributos do Modelo de Negócio.

Saída: vetor $MA[PA1..PAn]$, $PA1$ a PAn são Painéis Atributos do Modelo de Apresentação.

```
1 Vetor<PainelAtributo> MA
2 PainelAtributo painelAtributo
3 para cada elemento de MN faça
4 |   painelAtributo = aplicarRegra(MN[i])
5 |   MA.adicionar(painelAtributo)
6 fim
7 retornar MA
```

O algoritmo 5.1 percorre o conjunto de Atributos, transforma cada Atributo em um Painel Atributo e insere-o em um conjunto de Painel Atributo, formando o Modelo de Apresentação.

A parte mais importante desse algoritmo consiste no método *aplicarRegra*. Este método acessa os dados do Atributo, examina-os e, de acordo com as informações obtidas, gera o Painel Atributo correspondente.

As Figuras 5.1 a 5.8 trazem todas as possibilidades de mapeamento de um Atributo para um Painel Atributo. O texto na parte superior de cada figura mostra as condições para que o mapeamento ocorra. Nas figuras não foram mostrados os detalhes dos Atributos e dos Painéis Atributos, que podem ser obtidos nas seções 4.1 e 4.2, respectivamente.

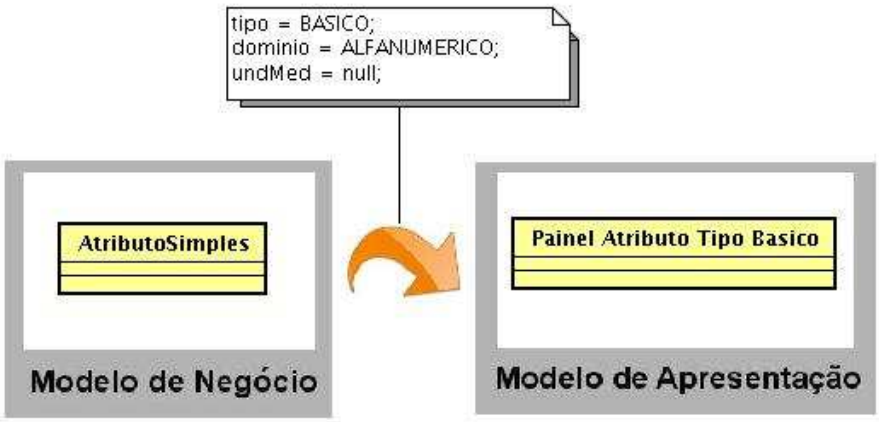

Figura 5.1: *Mapeamento do Atributo Simples no Painel Atributo Tipo Basico*

O mapeamento de um Atributo Simples para um Painel Atributo Tipo Basico (Figura 5.1) ocorre se o tipo do atributo for *BASICO*, seu domínio for *ALFANUMERICO* e não envolver unidade de medida (*undMed* = *null*).

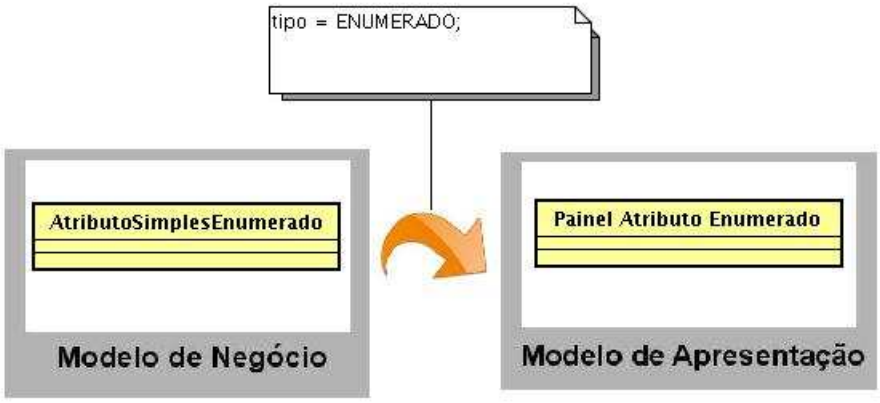

Figura 5.2: *Mapeamento do Atributo Simples Enumerado no Painel Atributo Enumerado*

A transformação de um Atributo Simples Enumerado em um Painel Atributo Enumerado é direta (Figura 5.2). É observado somente se o Atributo é do tipo ENUMERADO.

A princípio, o Atributo Simples Enumerado pode parecer redundante, já que bastaria colocar o *tipo = ENUMERADO* dentro de um Atributo Simples. Porém, caso o MMO (Figura 4.1) fosse projetado dessa forma, um Atributo Simples deveria fazer uma referência para um Domínio Enumerado, visto que caso o tipo fosse ENUMERADO essa relação deveria existir. Desse modo o projeto ficaria menos coeso, contendo relacionamentos opcionais. Por isso criou-se o Atributo Simples Enumerado.

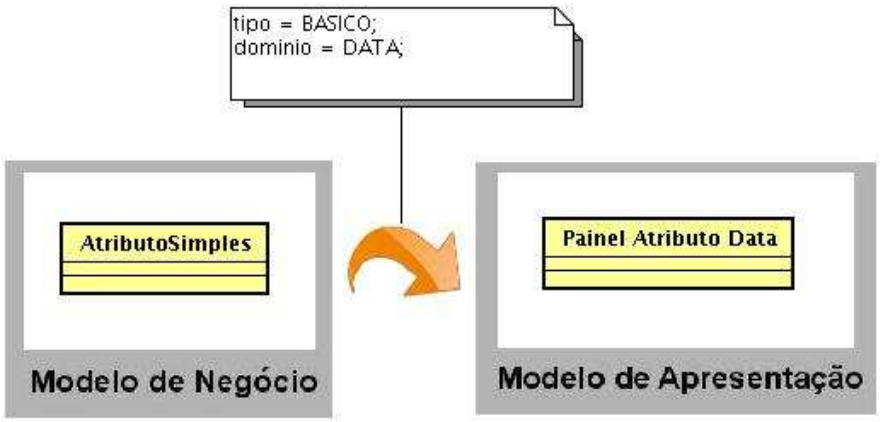

Figura 5.3: *Mapeamento do Atributo Simples no Painel Atributo Data*

Caso o Atributo Simples contenha *tipo = BASICO* e *dominio = DATA*, o GDIG criará um Painel Atributo Data a partir dessas informações (Figura 5.3).

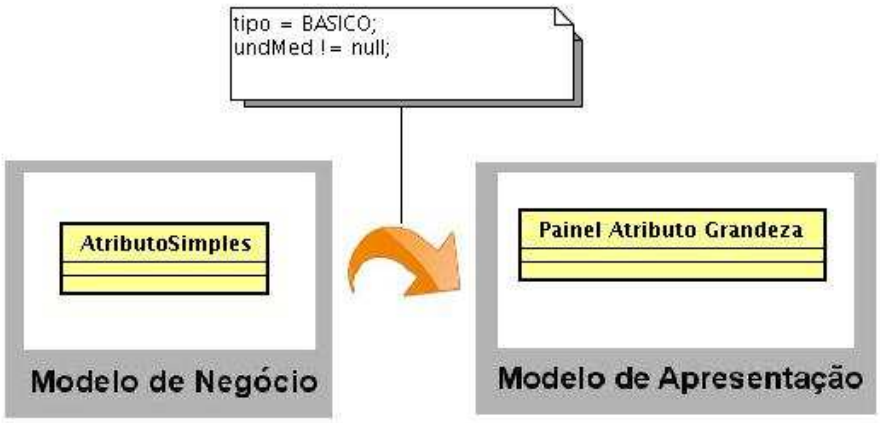

Figura 5.4: *Mapeamento do Atributo Simples no Painel Atributo Tipo Grandeza*

O mapeamento de um Atributo Simples para um Painel Atributo Grandeza, como mostra a Figura 5.4, ocorre caso *undMed != null*, ou seja, caso contenha alguma unidade de medida em sua característica *undMed*.

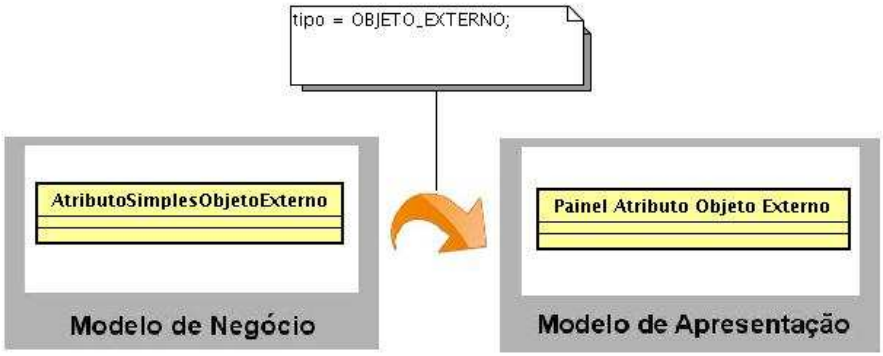

Figura 5.5: *Mapeamento do Atributo Simples Objeto Externo no Painel Atributo Objeto Externo*

A criação de um Painel Atributo Objeto Externo ocorre caso o Atributo contenha *tipo = OBJETO_EXTERNO*, como demonstra a Figura 5.5.

Assim como na criação do Painel Atributo Enumerado, o Atributo Simples Objeto Externo pode parecer redundante. A mesma explicação cabe neste caso. Foi necessária a criação do Atributo Simples Objeto Externo pelo fato dele conter informações específicas, como discute a Seção 4.1.1.

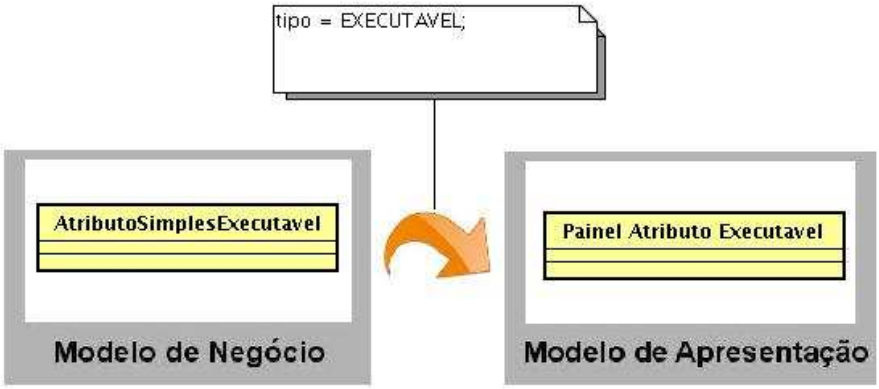

Figura 5.6: *Mapeamento do Atributo Simples Executavel no Painel Atributo Executavel*

O Painel Atributo Executavel é criado quando o GDIG encontra um atributo do tipo EXECUTAVEL (Figura 5.6).

A questão da coesão das informações segue a mesma linha de raciocínio do Atributo Simples Enumerado.

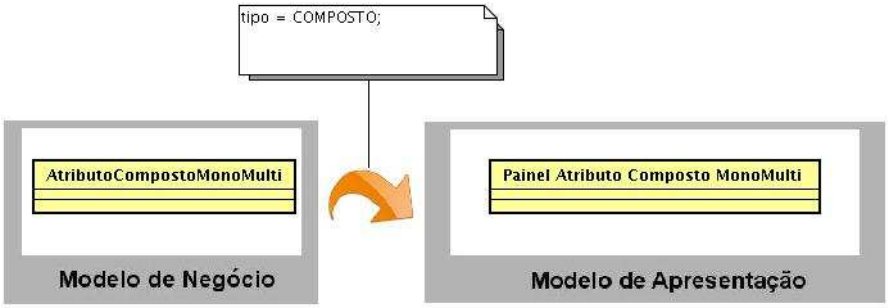

Figura 5.7: *Mapeamento do Atributo Composto MonoMulti no Painel Atributo Composto MonoMulti*

Caso o tipo do atributo seja *COMPOSTO*, o GDIG criará um Painel Atributo Composto MonoMulti (Figura 5.7). Neste caso o atributo deve ser *AtributoCompostoMonoMulti*.

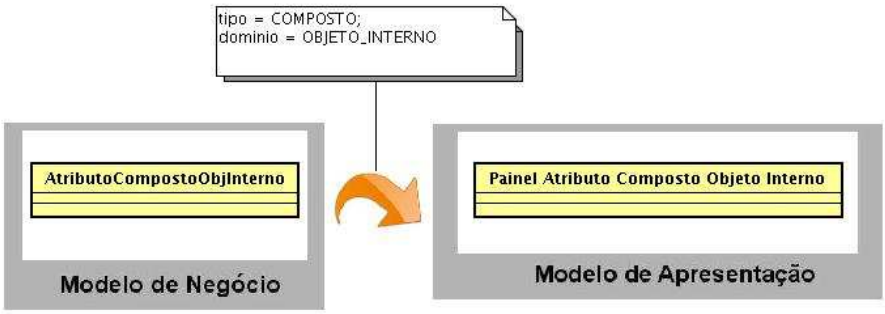

Figura 5.8: *Mapeamento do Atributo Composto Objeto Interno no Painel Atributo Composto Objeto Interno*

Como mostra a Figura 5.8, a criação do Painel Atributo Composto Objeto Interno ocorre quando tem-se, no Modelo de Negócio, um Atributo Composto Objeto Interno, do tipo *COMPOSTO* e domínio *OBJETO_INTERNO*.

Na criação de todos os Painéis Atributos Compostos verifica-se a questão da aparente redundância de informações. A justificativa é a mesma dada na descrição do Painel Atributo Simples Enumerado (Figura 5.2).

5.2 Criação da Interface Concreta a partir do Modelo de Apresentação

O último passo do processo de geração automática de GUIs é a criação da interface propriamente dita. O GDIG usa o Modelo de Apresentação para criar interfaces na linguagem Java, utilizando a tecnologia *Swing* [42] para a geração dos componentes gráficos.

O GDIG cria GUI do tipo WIMP. Além de construir a interface CRUD dinamicamente, o GDIG tem responsabilidade de criar outras janelas para o SI, algumas delas de forma estática. Uma dessas janelas é a Janela Principal, responsável pela apresentação de todo o SI ao usuário final e por fornecer as funcionalidades disponíveis em forma de menus e barra de ferramentas. As telas criadas dinamicamente ocupam um lugar predefinido dentro da Janela Principal, o espaço da *Área Funcional*, como mostra a Figura 5.9.

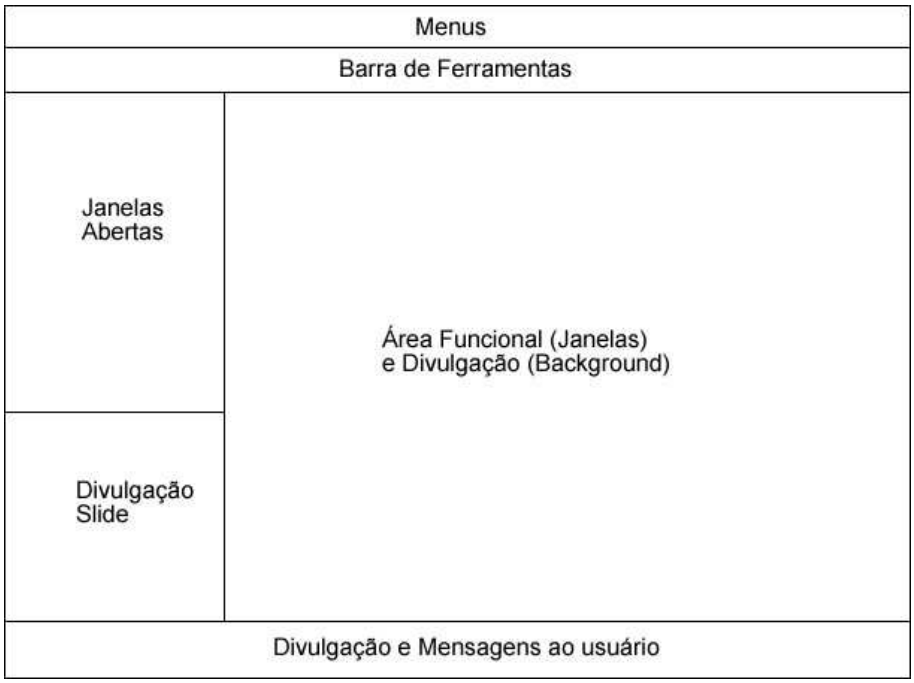

Figura 5.9: *Estrutura visual da Janela Principal*

A Janela Principal foi projetada de modo a disponibilizar o maior número possível de funções ao usuário sem se tornar visualmente carregada. Para isso, a janela é dividida nas seguintes partes:

Menus: disponibilizam as funções do sistema ao usuário através de opções agrupadas de acordo com a funcionalidade prevista no SI.

Barra de Ferramentas: juntamente com os menus, são as únicas partes da Janela Principal através das quais o usuário pode ter acesso a uma determinada funcionalidade do sistema.

Janelas Abertas: espaço que contém um atalho para cada janela aberta do sistema. As janelas abertas são organizadas em uma estrutura de árvore, pois uma janela pode gerar (ser "pai" de) outra. Como em sistemas de informação em geral executam-se várias funções ao mesmo tempo (como, por exemplo, vários cadastros) é normal que várias janelas fiquem abertas simultaneamente. Surge, então, a necessidade de um espaço onde o usuário tenha controle do que está sendo executado em um determinado momento. Essa funcionalidade é útil também para ativar diretamente, através de um clique duplo, uma janela que está escondida atrás de outras.

Divulgação *slide*: a Janela Principal contém três áreas para divulgação de informações do sistema ou do negócio: *Divulgação slide*, *Divulgação Background* e *Divulgação Texto* (parte inferior). O espaço de divulgação tipo *slide* permite que, em um mesmo

espaço, várias mensagens possam ser divulgadas, através de um mecanismo que, de tempo em tempo, mostra uma determinada mensagem, simulando uma sequência de *slides*.

Divulgação e Mensagens ao usuário: essa área situa-se no rodapé da janela principal e tem duas finalidades principais:

- mostrar mensagens curtas, tipo "slogans", a cada janela que é aberta no sistema ou de tempo em tempo.
- mostrar mensagens de erro, *feedback* e aviso ao usuário. Por exemplo: "Operação de inserção efetuada com sucesso".

Área Funcional (Janelas) e Divulgação (*Background*): esse é o principal espaço da Janela Principal, e tem dupla função. Todas as janelas geradas dinamicamente pelo GDIG localizam-se nessa área. Quando não existe janela aberta, esse espaço comporta-se como uma área de divulgação, com o pano de fundo dividido em 9 espaços para que mensagens distintas possam ser divulgadas através de imagens, e os usuários possam acessar *sites*, *emails* e outros meios de comunicação através de mecanismos de *hiperlink*.

A execução do GDIG passa por uma série de passos durante seu ciclo de vida, conforme ilustra a Figura 5.10.

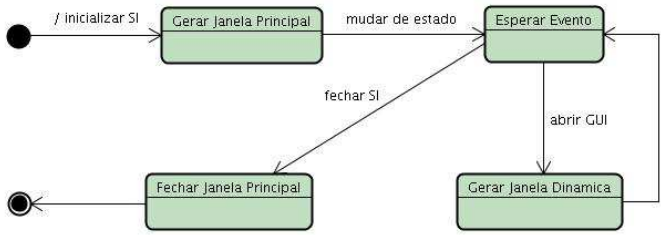

Figura 5.10: *Ciclo de vida do GDIG*

Quando o GDIG é inicializado ele cria a Janela Principal, com seus menus, barra de ferramentas e áreas de trabalho. Em seguida, o GDIG espera por eventos vindos do usuário, como cliques em botões, menus ou ícones. Um evento poderá abrir uma janela dinâmica ou fechar a Janela Principal. Uma janela dinâmica também espera por um evento e pode abrir outra, formando um mecanismo recursivo.

Após o término do trabalho ser sinalizado pelo usuário, o GDIG fechará a Janela Principal e se auto finaliza, terminando seu ciclo de vida. O estado "Esperar Evento" não executa uma tarefa propriamente dita, mas sim espera por alguma informação.

5.2.1 Criação da Janela Principal

A Janela Principal é criada somente uma vez no início de cada execução do *framework*. O diagrama de colaboração da Figura 5.11 apresenta as tarefas para a inicialização do componente GDIG.

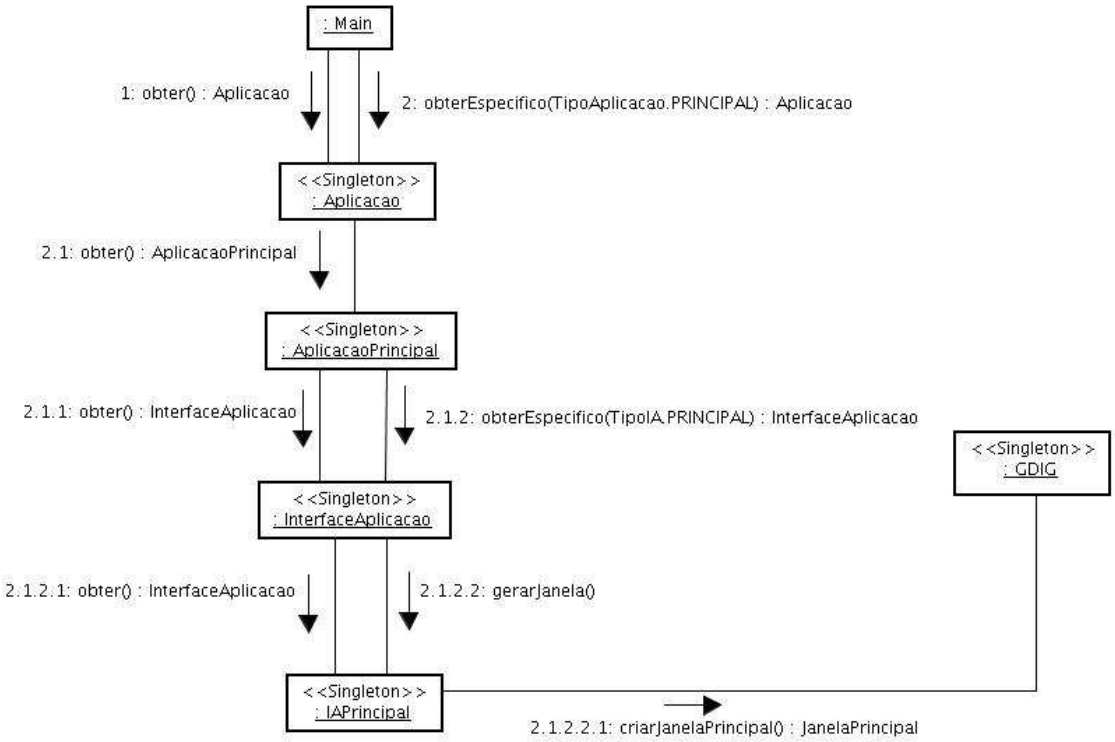

Figura 5.11: *Conexão inicial entre as camadas e criação da Janela Principal*

Quando o *framework* é inicializado para executar um determinado SI, são criadas e estabelecidas comunicações entre os principais componentes da arquitetura do *framework*.

Primeiramente é criada a Camada Aplicação, que implementa o padrão *Singleton* [17], tendo somente uma instância em todo o tempo de execução do sistema. Logo após é instanciado um objeto do tipo Aplicação Principal, que é do tipo Aplicação e controla as funcionalidades da aplicação principal que está sendo executada.

Em seguida a Aplicação Principal conecta-se com a Interface Aplicação (camada que executa o papel de *Controller* entre as camadas Interface e Aplicação) e obtém a Interface Aplicação específica (Interface Aplicação Principal). Esta, por sua vez, invoca o GDIG para que este crie a Janela Principal.

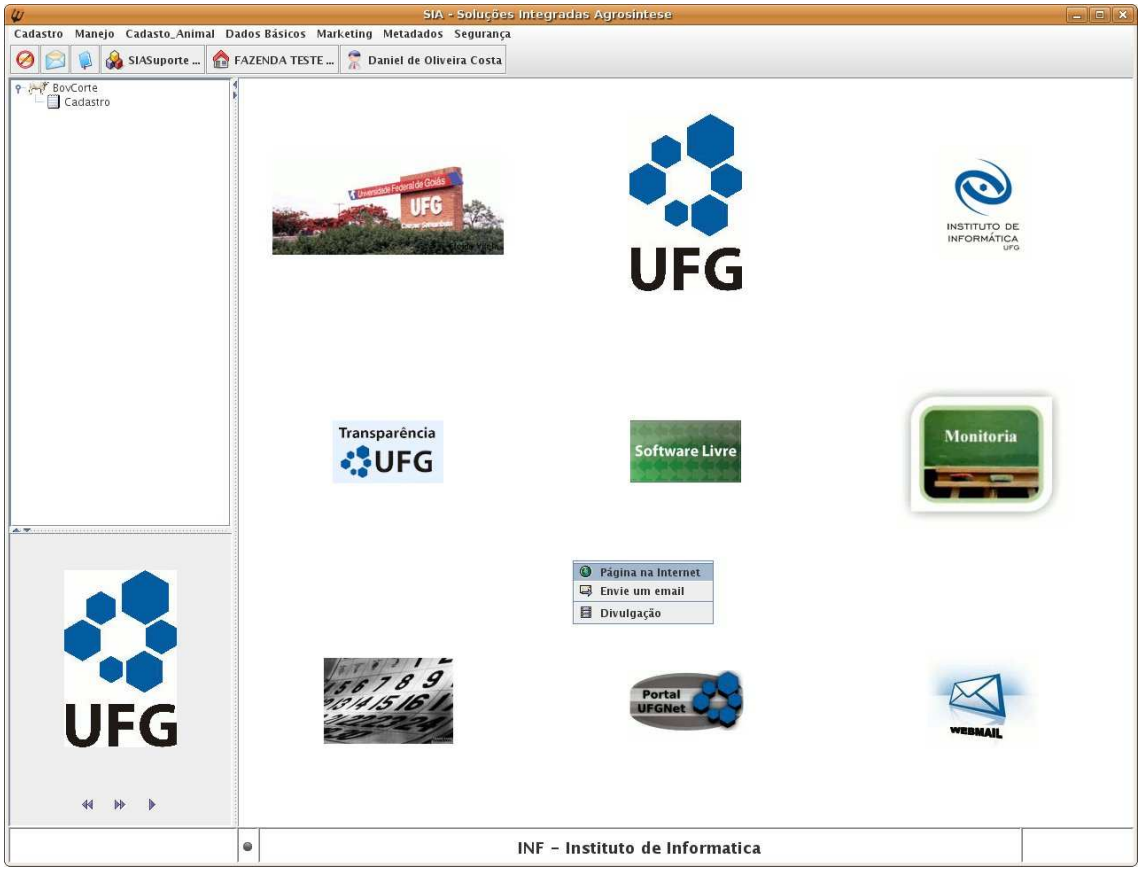

Figura 5.12: *Exemplo de GUI Swing da Janela Principal de um SI*

Gerada a Janela Principal ilustrada na Figura 5.12, o GDIG passa a atuar como um observador, aguardando alguma ação oriunda do usuário, para que possa criar dinamicamente uma GUI para a função do SI solicitada pelo usuário.

5.2.2 Criação da Janela de Cadastro

Para ser inicializada a geração da Janela de Cadastro, o usuário pode, por exemplo, selecionar um item de menu com a finalidade de abrir a janela de cadastro de alguma entidade no SI. Neste instante, a sequência de ações ilustrada no diagrama de colaboração da Figura 5.13 será executada.

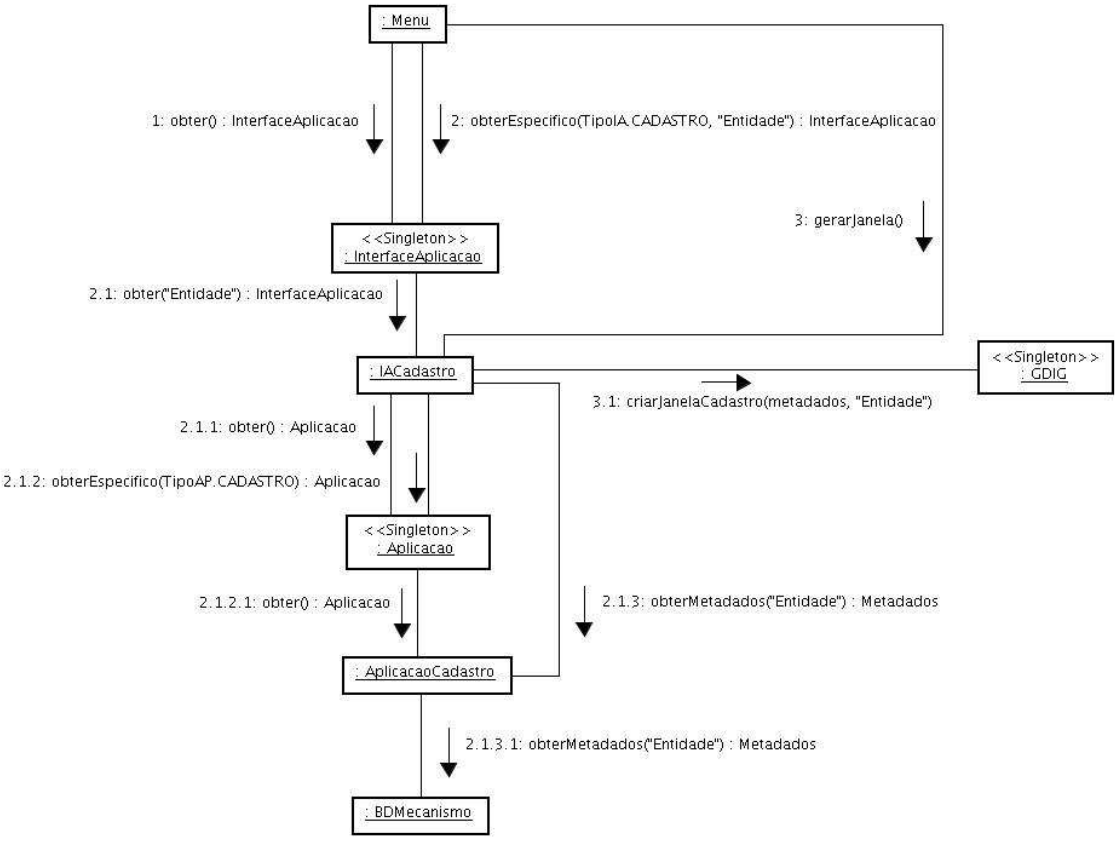

Figura 5.13: *Criação dinâmica de GUI dirigida por modelos*

Quando o usuário seleciona um item de menu, o GDIG observa a ação (pelo padrão *Observer*) e faz uma conexão com a Camada Interface Aplicação (*Controller*). Depois disso, o GDIG analisa a ação, identificando a aplicação "cadastro", e cria a Interface Aplicação Cadastro, específica para cadastros.

Esta, por sua vez, conecta-se com a Camada Aplicação e obtém a Aplicação Cadastro, também específica. No momento seguinte, a Aplicação Cadastro obtém da Camada BDMecanismo (responsável pela execução de vários serviços de persistência e obtenção de dados junto à base de dados) os metadados relativos à entidade referenciada pelo item de menu selecionado. Neste diagrama, esta informação é referenciada como "Entidade".

Todo esse mecanismo é realizado de forma síncrona, ou seja, o GDIG espera que os metadados sejam obtidos para continuar sua execução. Criadas as conexões entre as camadas e obtidos os metadados, o GDIG gera a janela de formulário, usando a abordagem dirigida por modelos.

A Janela de Cadastro, em seu ciclo de vida, pelos passos descritos na Figura 5.14.

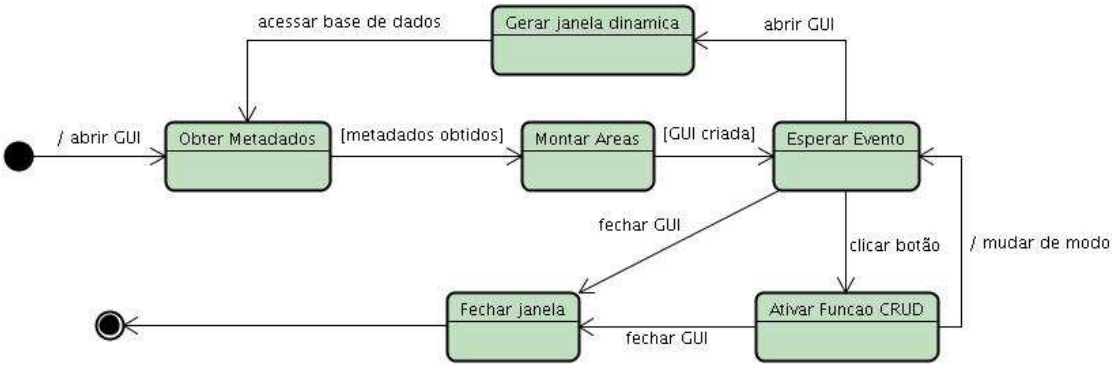

Figura 5.14: *Ciclo de vida da Janela de Cadastro*

Uma vez criada, a Janela de Cadastro obtém os metadados das camadas inferiores dentro da arquitetura, monta as áreas mostradas na Figura 5.15 e entra no estado de espera por algum evento. Este evento pode ser um fechamento da própria janela, finalizando seu ciclo de vida.

Outra possibilidade de evento é a ativação de uma função CRUD. Dentro da funcionalidade selecionada como, por exemplo, uma inserção, pode ocorrer de o usuário ativar um evento de gerar uma outra Janela de Cadastro (Janela Dinâmica), fazendo com que, para essa nova janela, o ciclo recomece.

A Janela de Cadastro (formulário CRUD) é gerada dinamicamente e em tempo de execução. Como mostra a Figura 5.15, ela é dividida nas seguintes áreas: (i) aquela composta pelos *widgets* do formulário, onde o usuário insere os dados; (ii) a que contém as ações referentes aos dados do formulário; (iii) a que contém os elementos já cadastrados; e (iv) a que contém as ações referentes à Janela de Cadastro.

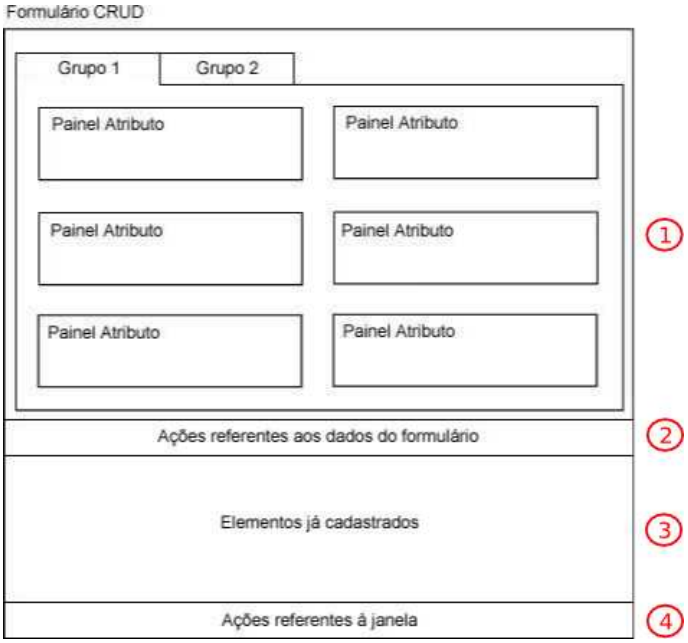

Figura 5.15: *Estrutura do formulário CRUD, seus grupos e Painéis Atributos*

Para formar a Janela de Cadastro, o GDIG segue os seguintes passos:

1. Cria a área de campos do formulário;
2. Cria a área de ações referentes ao formulário;
3. Cria a área de elementos já cadastrados do formulário;
4. Cria a área de ações CRUD e a janela; e,
5. Acopla a janela criada dentro da Janela Principal, na Área Funcional (Figura 5.9).

5.2.3 Criação da área de campos do formulário

O mecanismo de criação da área de campos do formulário consiste no mapeamento do Modelo de Apresentação para a Interface Concreta. Esta seção, detalha como ocorre esse mapeamento e como funciona sua organização visual e comportamental.

Cada Painel Atributo do Modelo de Apresentação é mapeado para um campo do formulário, nomeado como Componente Gráfico. Cada Componente Gráfico é composto por um Rótulo seguido de um ou mais *widgets* para que o usuário gerencie os dados do cadastro.

Existem 18 tipos de componentes gráficos, como mostra a Tabela 5.1. Juntos, eles fornecem uma cobertura extensa para que o usuário possa trabalhar com qualquer tipo de dado dentro das janelas.

Tabela 5.1: *Componentes gráficos existentes*

Nome	Descrição
Tipo Básico Monovalorado	Manipula informações simples e monovaloradas.
Tipo Básico Multivalorado	Manipula informações simples e multivaloradas.
Tipo Básico Texto Longo	Trabalha com informações longas.
Tipo Enumerado Monovalorado	Armazena somente um valor e este encontra-se dentro de um conjunto predefinido de dados.
Tipo Enumerado Multivalorado	Pode armazenar mais de um valor dentro de um conjunto predefinido de dados.
Tipo Data Monovalorado	Armazena um valor do tipo data.
Tipo Data Multivalorado	Pode armazenar vários valores do tipo data.
Tipo Grandeza Monovalorado	Manipula um valor representando uma medida física.
Tipo Grandeza Multivalorado	Manipula um ou mais valores representando uma medida física.
Tipo Objeto Externo Monovalorado	Contém um valor que representa um objeto externo ao sistema (arquivo de vídeo, áudio, imagem, dentre outros).
Tipo Objeto Externo Multivalorado	Contém um ou mais valores que representam objetos externos ao sistema.
Executável	Executa algum aplicativo dentro do SI.
Composto MonoMulti Simples	Manipula informações compostas onde todas são editadas no próprio componente.
Composto MonoMulti Complexo	Manipula informações compostas onde nem todas são editadas no próprio componente.
Composto MonoMulti Agrupador	Agrupa outros componentes.
Composto MonoMulti Lista	Lista informações, não permitindo edição das mesmas.
Composto Objeto Interno Simples	Armazena informações sobre entidades persistidas na base de dados do SI e permite sua edição.
Composto Objeto Interno Complexo	Armazena informações sobre entidades persistidas na base de dados do SI, mas não permite sua edição.

As seções seguintes mostram a criação de cada Componente Gráfico a partir dos Painéis Atributos. Para facilitar o entendimento, as características visuais e comportamentais de cada Componente Gráfico são explicadas separadamente.

Componente Gráfico Tipo Básico

O componente gráfico mais simples que o GDIG gera é o Tipo Básico. Pode ser Monovalorado, Multivalorado ou Texto Longo.

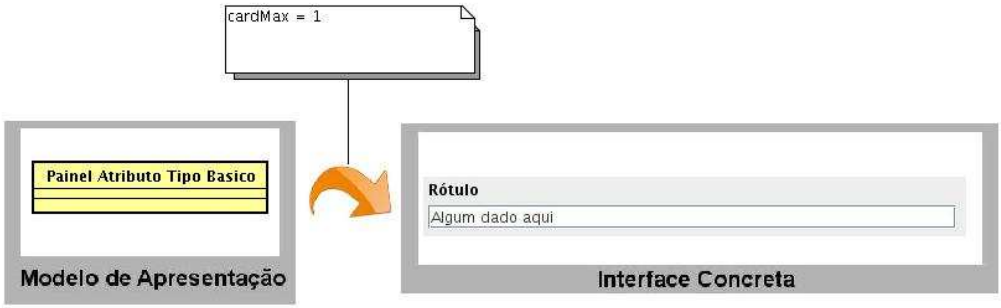

Figura 5.16: *Mapeamento do Painel Atributo Tipo Básico no Componente Gráfico Tipo Básico Monovalorado*

Monovalorado O Componente Gráfico Tipo Básico Monovalorado é gerado a partir de um Painel Atributo Tipo Básico (Figura 5.16) caso a cardinalidade máxima seja igual a 1 (um). Isso significa que o campo, no formulário, conterá no máximo um elemento.

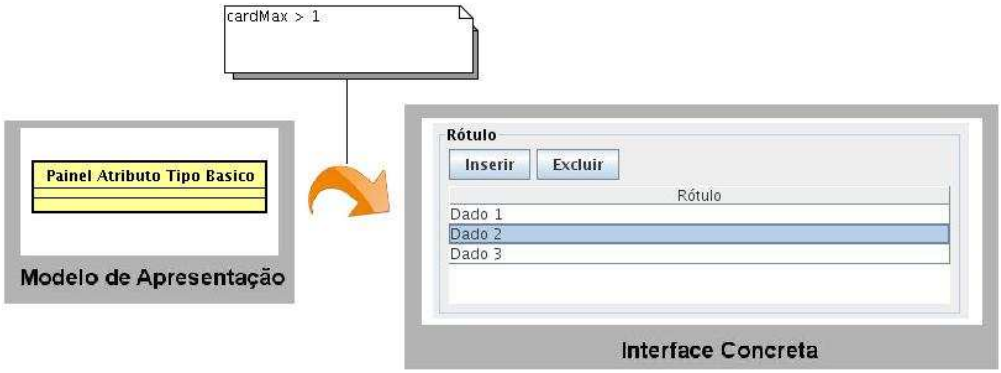

Figura 5.17: *Mapeamento do Painel Atributo Tipo Básico no Componente Gráfico Tipo Básico Multivalorado*

Multivalorado Caso o projetista do sistema deseje que o campo do formulário possa conter mais de um valor, deverá indicar a cardinalidade máxima maior que um (> 1) na definição do atributo. Neste caso o GDIG gerará um Componente Gráfico Tipo Básico Multivalorado (Figura 5.17).

Este componente é composto por, um rótulo, dois botões (*Inserir* e *Excluir*) e uma tabela que mostra os dados já inseridos.

O botão *Inserir* sempre fica habilitado e indica ao usuário que ele pode inserir um novo valor no campo. Quando selecionado, adiciona uma nova linha na tabela onde o usuário pode inserir um valor.

O botão *Excluir* exclui um valor da tabela. Fica habilitado caso alguma linha na tabela esteja selecionada. Quando ativado, exclui a linha selecionada.

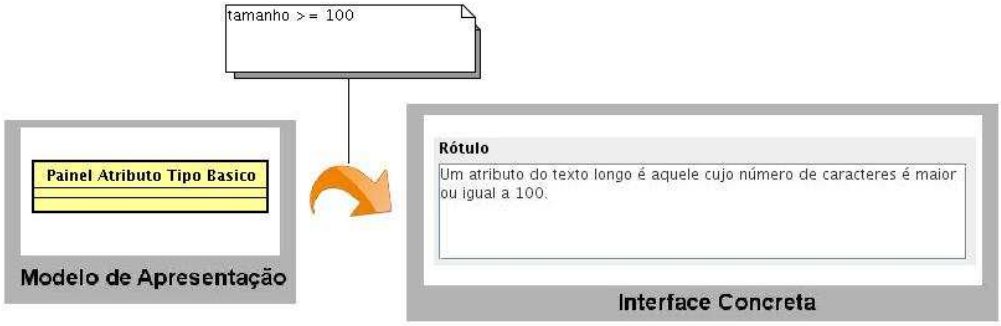

Figura 5.18: *Mapeamento do Painel Atributo Tipo Básico no Componente Gráfico Tipo Básico Texto Longo*

Texto Longo O projetista pode querer proporcionar ao usuário final um campo para que seja inserido um texto com número de caracteres maior do que o usual. Determinou-se que, caso o atributo *tamanho* do modelo de apresentação seja maior ou igual a cem (>= 100), o GDIG gerará o Componente Gráfico Tipo Básico Texto Longo (Figura 5.18), permitido ao usuário inserir um texto maior. Este limite de cem caracteres pode ser aumentado ou diminuído a partir dos parâmetros de inicialização do GDIG.

Valor obrigatório

Caso o Painel Atributo tenha a característica *obrigatorio* igual a verdadeiro (*obrigatorio = true*), significa que o campo não pode ser vazio. Ao se deparar com essa informação, o GDIG insere no rótulo o símbolo de asterisco (*), indicando essa obrigatoriedade ao usuário, como mostra a Figura 5.19.

Figura 5.19: *Componente Gráfico com valor obrigatório*

Esse mecanismo funciona da mesma forma para qualquer tipo de Componente Gráfico.

Componente Gráfico Enumerado

O tipo de componente gráfico Enumerado, como explica a Seção 4.1.1, armazena valores de um conjunto predefinido de dados, de onde o usuário poderá escolher um ou mais valores.

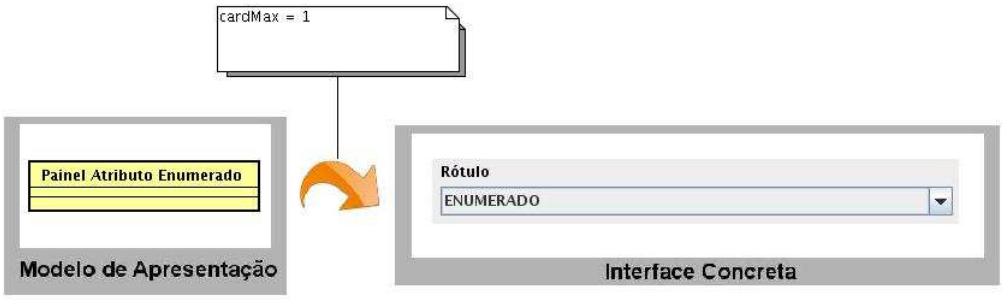

Figura 5.20: *Mapeamento do Painel Atributo Enumerado no Componente Gráfico Enumerado Monovalorado*

Monovalorado Como mostra a Figura 5.20, caso a cardinalidade máxima do Painel Atributo Enumerado seja igual a um, este será mapeado para um Componente Gráfico Enumerado Monovalorado.

Este componente contém um Combo, que apresenta os valores da enumeração para que o usuário escolha somente um.

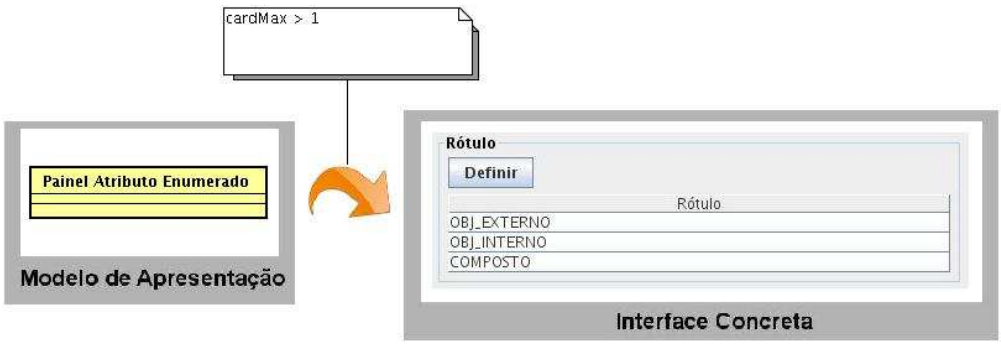

Figura 5.21: *Mapeamento do Painel Atributo Enumerado no Componente Gráfico Enumerado Multivalorado*

Multivalorado Será criado um Componente Gráfico Enumerado Multivalorado caso a cardinalidade máxima seja maior que um (>1), como mostra a Figura 5.21. Este componente contém o botão *Definir* e uma tabela contendo os valores escolhidos pelo usuário.

O botão *Definir* abre uma janela de *Seleção* onde o usuário pode gerenciar os valores escolhidos dentro da enumeração, como mostra a Figura 5.22.

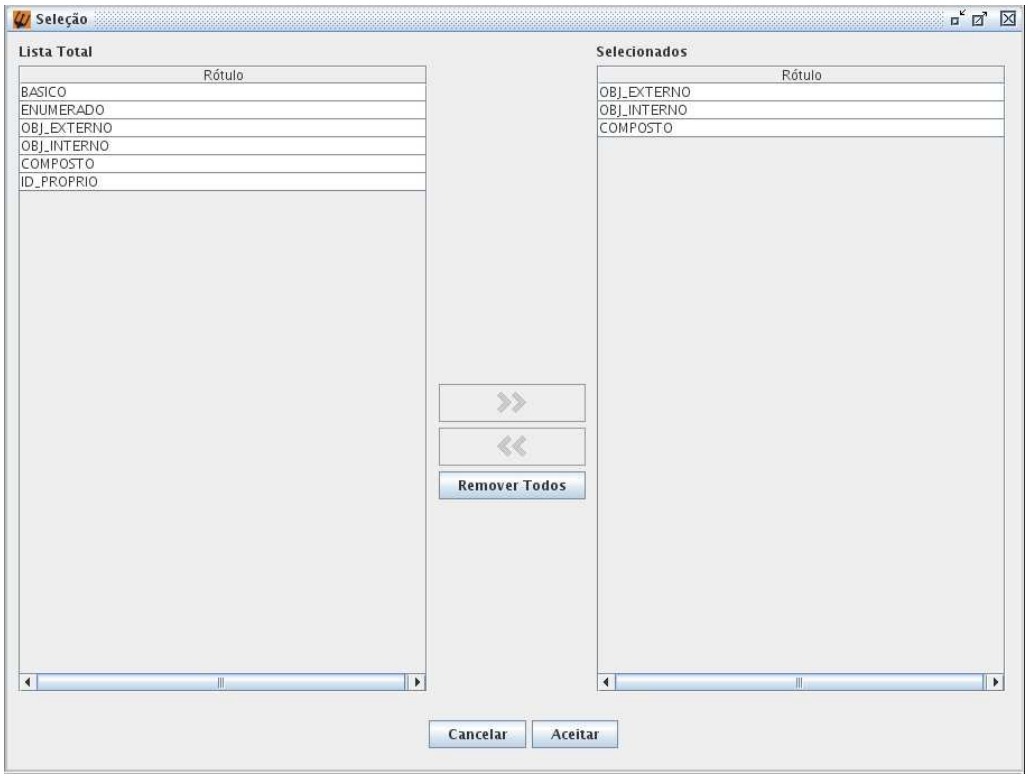

Figura 5.22: *Janela de Seleção do Componente Gráfico Enumerado*

Dentro da janela de *Seleção*, a tabela *Lista Total* mostra todos os valores predefinidos na enumeração e a tabela *Selecionados* mostra os valores escolhidos pelo usuário dentro da *Lista Total*.

Clicando nos botões "»" e "«" o usuário pode escolher valores (passando da tabela *Lista Total* para a *Selecionados*) ou remover valores (passando da tabela *Selecionados* para *Lista Total*). O botão *Remover* Todos exclui todos os valores da tabela *Selecionados*.

Ao ativar o botão *Aceitar*, os valores selecionados pelo usuário são inseridos na tabela do componente gráfico.

Componente Gráfico Data

O Componente Gráfico Data representa um campo do formulário onde o usuário informa dados de data no formato dia/mês/ano (*dd/mm/AAAA*). O campo pode ser mono ou multivalorado.

Monovalorado É criado o Componente Gráfico Data Monovalorado caso o projetista estabeleça a cardinalidade máxima igual a um do Painel Atributo Data, como mostra a Figura *Seleção* onde o usuário pode gerenciar os valores escolhidos dentro da enumeração, como mostra a Figura 5.23.

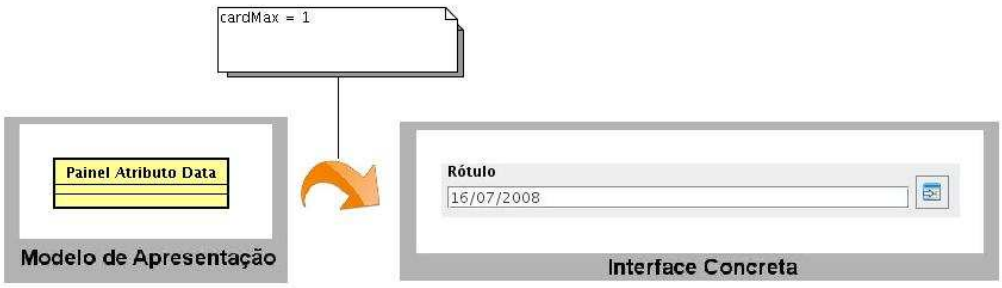

Figura 5.23: *Mapeamento do Painel Atributo Data no Componente Gráfico Data Monovalorado*

Através deste componente o usuário pode definir a data manualmente, digitando no campo de texto ou escolhendo através de um mini-aplicativo de calendário, representado pelo botão.

Ao clicar no botão de calendário, é mostrada a janela de calendário (Figura 5.24), onde o usuário pode navegar rapidamente pelos anos, meses e dias.

Figura 5.24: *Janela de Calendário*

Multivalorado O Componente Gráfico Enumerado Multivalorado é criado caso a cardinalidade máxima do Painel Atributo Data seja definida como maior que um (> 1), como traz a Figura 5.25.

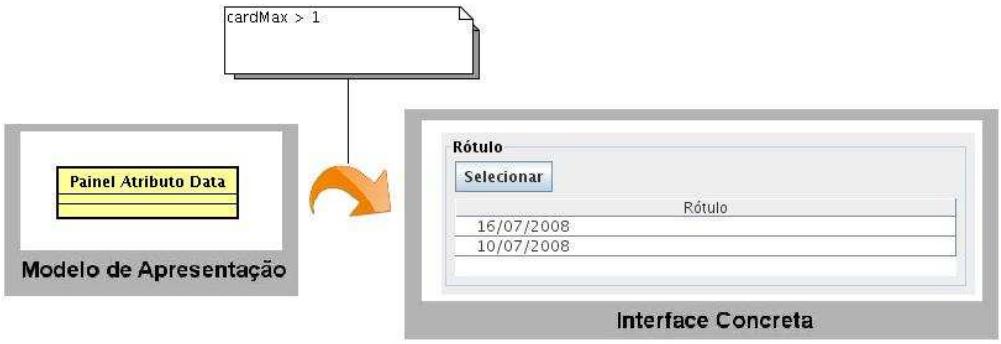

Figura 5.25: *Mapeamento do Painel Atributo Data no Componente Gráfico Data Multivalorado*

Este componente é composto pelo botão *Selecionar* e por uma tabela que contém os valores inseridos pelo usuário.

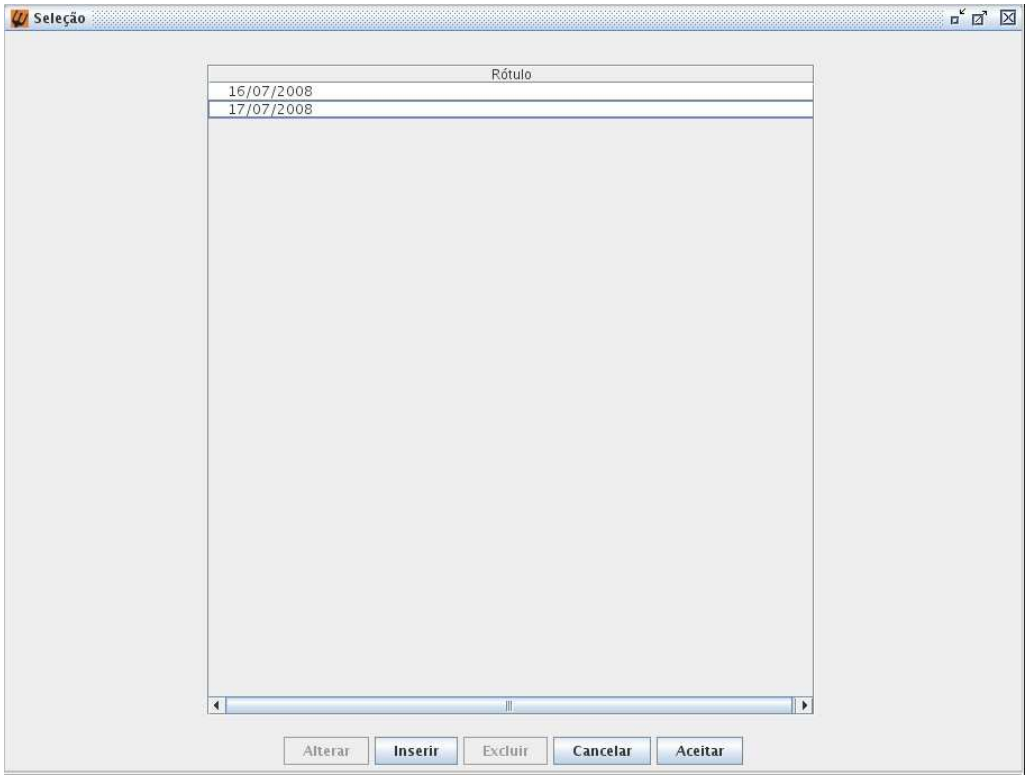

Figura 5.26: *Janela de Seleção do Componente Gráfico Data*

Ao ativar o botão *Selecionar*, é aberta uma janela (Figura 5.26) onde, através dela, o usuário pode alterar uma data já inserida (botão *Alterar*), inserir uma nova data (botão *Inserir*), remover uma data selecionada (botão *Remover*), cancelar a seleção (botão *Cancelar*) ou salvar as alterações (botão *Aceitar*).

Ao ativar o botão *Aceitar*, os valores ajustados pelo usuário são retornados para o componente gráfico.

Componente Gráfico Grandeza

Outro tipo de componente gráfico é Grandeza. Este componente é utilizado quando o usuário deseja inserir uma informação sobre alguma medida física. Pode ser mono ou multivalorado.

Monovalorado Caso o projetista defina a cardinalidade mínima como um, no Painel Atributo Grandeza, o GDIG mapeará para um Componente Gráfico Grandeza Monovalorado (Figura 5.27).

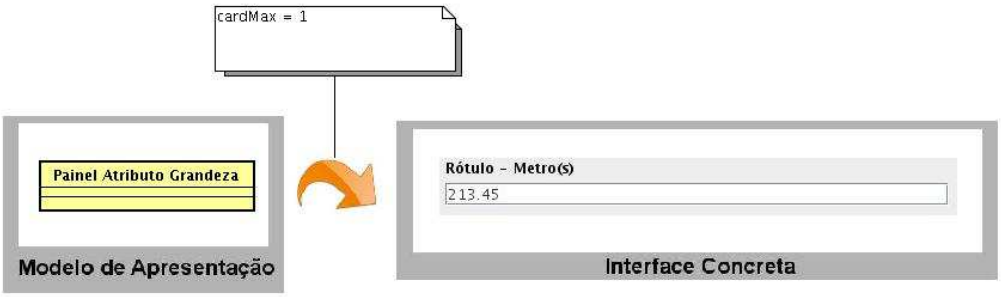

Figura 5.27: *Mapeamento do Painel Atributo Grandeza no Componente Gráfico Grandeza Monovalorado*

Multivalorado O projetista pode também definir a cardinalidade máxima com sendo maior que um (> 1). Neste caso, é criado o Componente Gráfico Grandeza Multivalorado, como mostra a Figura 5.28.

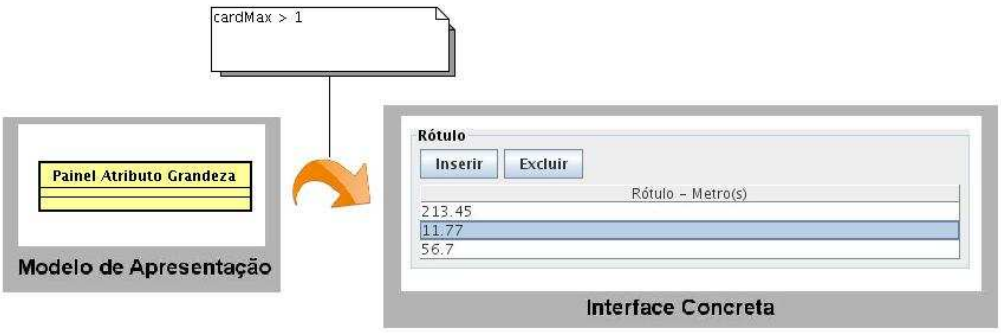

Figura 5.28: *Mapeamento do Painel Atributo Grandeza no Componente Gráfico Grandeza Multivalorado*

Este componente gráfico contém dois botões (*Inserir* e *Excluir*) e uma tabela que mostra os valores já inseridos pelo usuário.

O botão *Inserir* sempre encontra-se habilitado e indica ao usuário que ele pode inserir um novo valor no campo. Quando selecionado, adiciona uma nova linha na tabela onde o usuário pode inserir um valor.

O botão *Excluir* exclui um valor da tabela. Fica habilitado caso alguma linha na tabela esteja selecionada. Quando ativado, exclui a linha selecionada.

Componente Gráfico Objeto Externo

Como descreve a Seção 4.1.1, um objeto externo refere-se a um valor que não provém internamente no Banco de Dados do sistema. O Componente Gráfico Objeto Externo armazena valores referentes a um objeto externo e pode ser mono ou multivalorado.

Monovalorado O Componente Gráfico Objeto Externo Monovalorado é criado pelo GDIG caso o Painel Atributo Objeto Externo contenha a cardinalidade mínima igual a um (Figura 5.29).

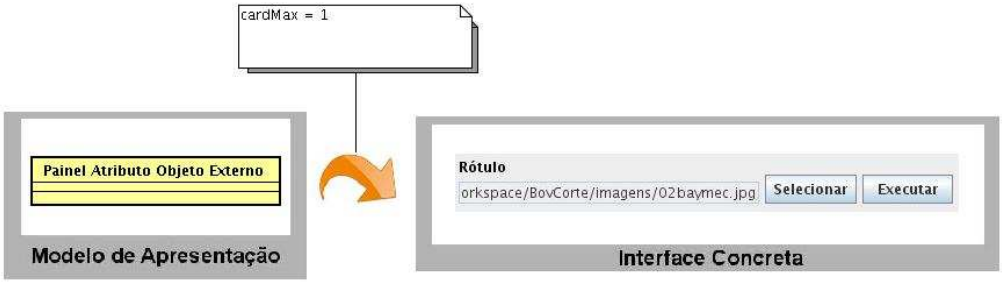

Figura 5.29: *Mapeamento do Painel Atributo Objeto Externo no Componente Gráfico Objeto Externo Monovalorado*

Este componente gráfico contém um campo de texto, o botão *Selecionar* e o botão *Executar*.

O campo de texto encontra-se sempre no estado não editável, obrigando o usuário a escolher o arquivo pelo botão *Selecionar*. Evitando-se, deste modo, erros na digitação do campo.

O botão *Selecionar* abre a janela para seleção de arquivo mostrada na Figura 5.30. O caminho do arquivo escolhido é inserido no campo de texto.

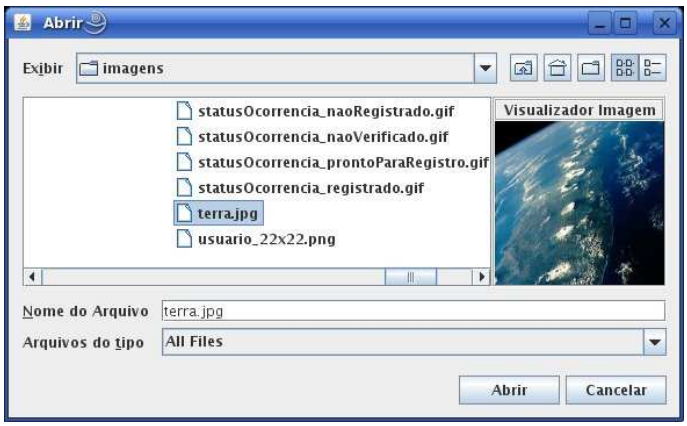

Figura 5.30: *Janela para seleção de arquivo*

O botão *Executar* chama o programa apropriado no Sistema Operacional para executar o arquivo cujo caminho encontra-se armazenado no campo de texto.

Multivalorado Caso a cardinalidade máxima seja definida como maior que um (> 1), o GDIG mapeia o Painel Atributo Objeto Externo para o Componente Gráfico Enumerado Multivalorado, como mostra a Figura 5.31.

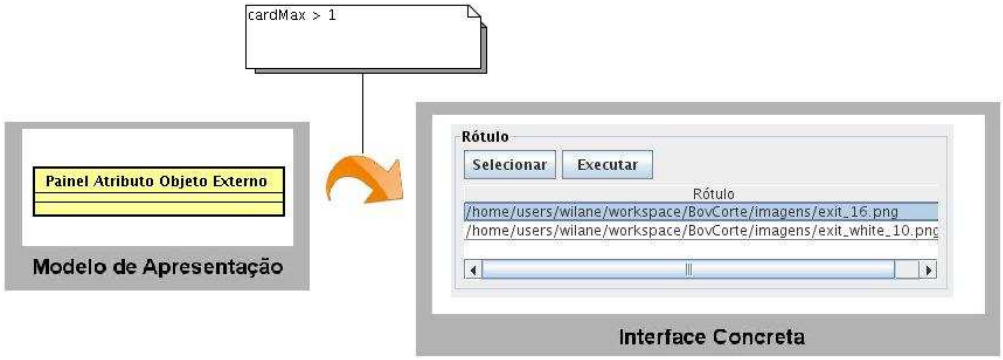

Figura 5.31: *Mapeamento do Painel Atributo Objeto Externo no Componente Gráfico Objeto Externo Multivalorado*

Este componente gráfico contém os botões *Selecionar* e *Executar* e uma tabela contendo os valores já definidos pelo usuário.

O botão *Selecionar* abre a janela de seleção da Figura 5.32.

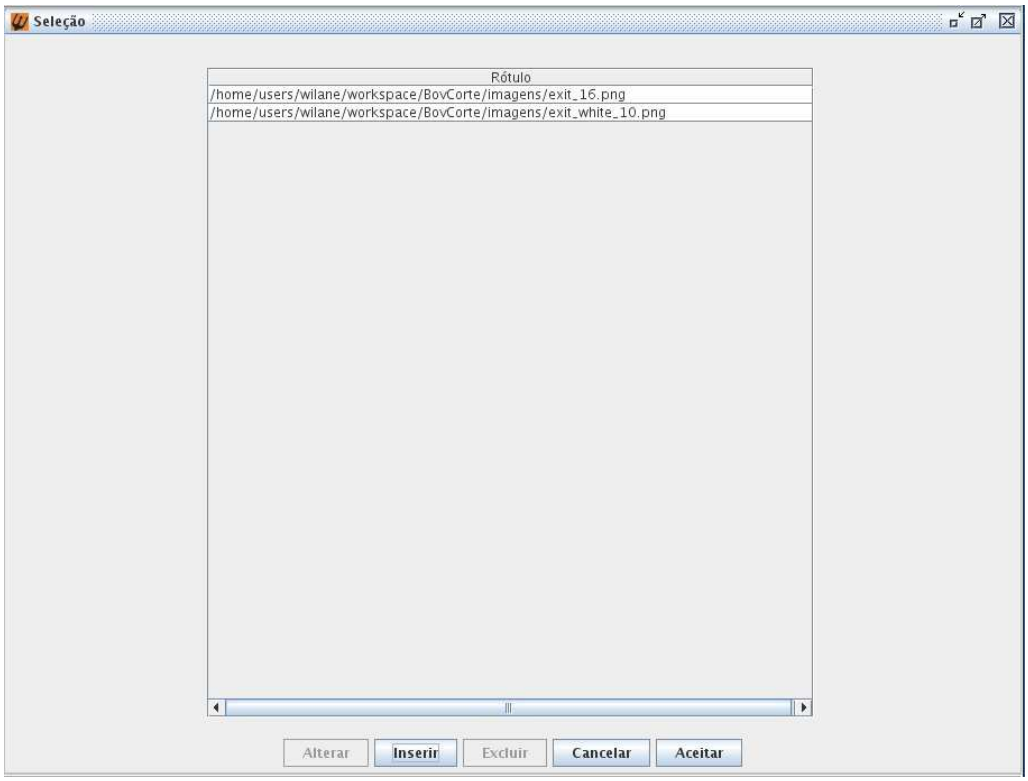

Figura 5.32: *Janela de Seleção de Objeto Externo Multivalorado*

Nesta janela o usuário pode alterar um dado já inserido (botão *Alterar*), inserir um novo dado (botão *Inserir*), remover um dado selecionado (botão *Excluir*), cancelar a Aceitar tarefa (botão *Cancelar*) ou salvar as modificações realizadas (botão *Aceitar*).

Ao clicar em *Aceitar*, os dados são salvos na tabela do componente.

O botão *Executar* do componente gráfico chama o programa apropriado no Sistema Operacional para executar o arquivo cujo caminho encontra-se armazenado na linha selecionada da tabela.

Componente Gráfico Executável

Não é necessário avaliar informações para a criação do Componente Gráfico Executável. O GDIG cria-o diretamente a partir do Painel Atributo Executável.

Não foi encontrado nenhum caso onde esse tipo de campo exigisse multi valores. Por isso, foi definido somente como monovalorado, como mostra a Figura 5.33.

Figura 5.33: *Mapeamento do Painel Atributo Executável no Componente Gráfico Executável*

Componente Gráfico Composto MonoMulti

O Componente Gráfico do tipo composto representa graficamente uma composição de atributos que, no caso, são sub-atributos do atributo compositor. Esse tipo de componente representa seus sub-atributos através de tabelas. Cada coluna da tabela representa um atributo e, caso o composto seja multivalorado, cada linha representa uma instância desses sub-atributos e cada célula um valor para um sub-atributo.

Um componente gráfico composto pode ser de quatro tipos: Simples, Complexo, Agrupador ou Lista.

Simples Este componente é gerado quando o GDIG encontra um Painel Atributo Composto MonoMulti com o domínio *COMPOSTO_SIMPLES* (Figura 5.34).

O Componente Gráfico Composto MonoMulti Simples difere-se do complexo pelo fato de permitir a edição dos valores da célula na própria tabela. Essa diferenciação existe porque o projetista pode querer não permitir a edição dos valores nas células da tabela, sendo permitida somente através de uma outra janela. Neste caso o tipo de domínio deve ser *COMPOSTO_COMPLEXO*.

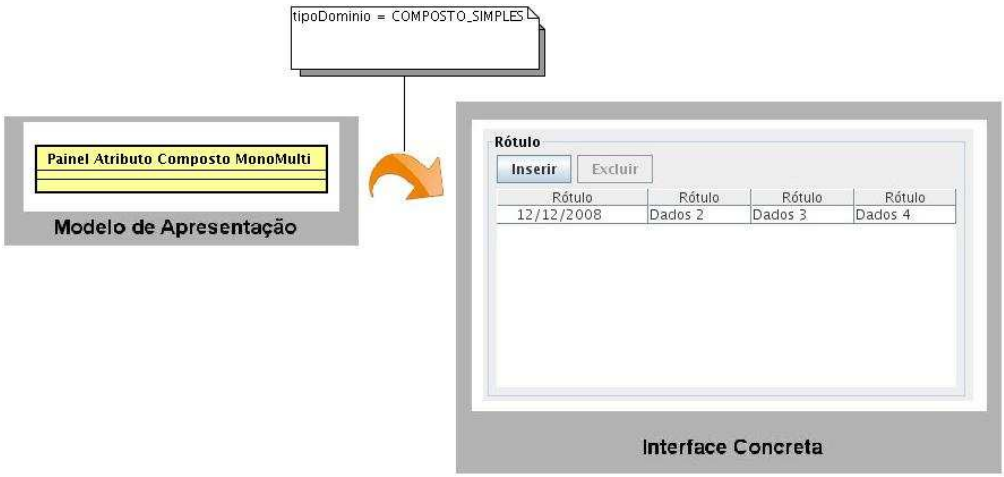

Figura 5.34: *Mapeamento do Painel Atributo Composto MonoMulti no Componente Gráfico Composto MonoMulti Simples*

Este componente gráfico é composto pelos botões *Inserir*, *Excluir* e por uma tabela de dados. Como dito anteriormente, cada célula representa um atributo que faz parte do compositor.

O botão *Inserir* permanece sempre editável e, ao ser acionado, é adicionada uma linha à tabela onde o usuário pode inserir os valores de acordo com cada sub-atributo na própria célula.

O botão *Excluir* torna-se editável quando uma linha da tabela é selecionada. Quando clicado, remove uma linha selecionada da tabela.

A tabela permite a edição dos valores nas células. Cada célula comporta-se diferenciadamente de acordo com o tipo de atributo que ela representa. Por exemplo, caso o primeiro sub-atributo seja do tipo *DATA*, a célula conterá uma máscara de edição do tipo *dd/mm/AAAA*. Ou seja, cada célula aproxima-se ao máximo dos componentes gráficos de seus respectivos atributos.

Complexo O Componente Gráfico Composto MonoMulti Complexo é gerado quando o GDIG encontra um Painel Atributo Composto MonoMulti com o domínio *COMPOSTO_COMPLEXO* (Figura 5.35).

Este componente difere do *COMPOSTO_SIMPLES* pelo fato de poder haver a edição de valores, porém esta pode ser feita nas próprias células da tabela.

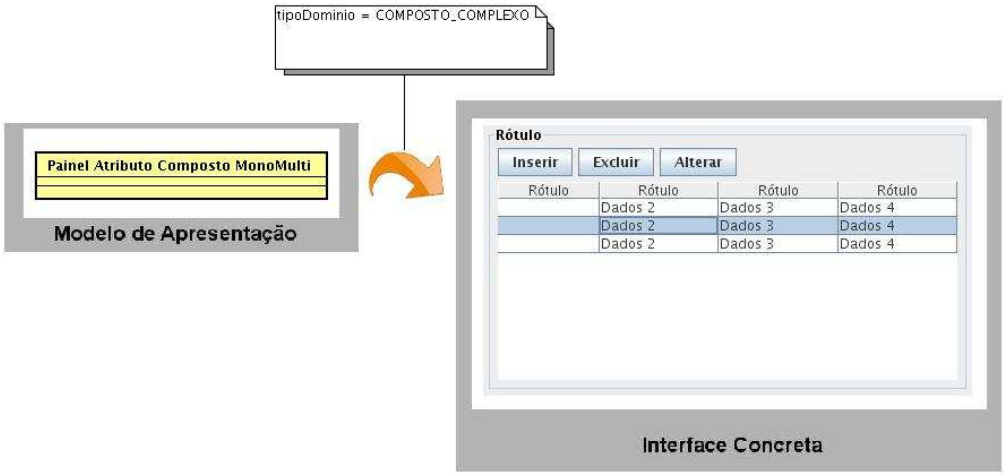

Figura 5.35: *Mapeamento do Painel Atributo Composto MonoMulti no Componente Gráfico Composto MonoMulti Complexo*

Este componente gráfico é formado pelos botões *Inserir*, *Excluir*, *Alterar* e pela tabela de dados.

O botão *Inserir* adiciona uma linha à tabela de dados.

O botão *Excluir* remove uma linha selecionada da tabela.

O botão *Alterar* executa a ação de edição de dados. Como os dados não podem ser modificados na própria tabela, o usuário deve clicar no botão *Alterar* que abre uma janela de formulário (mesmo estilo da janela de cadastro) contendo os campos no formulário referentes a cada sub-atributo. Deste modo, o usuário pode modificar os dados no formulário e estes serem salvos e trazidos novamente até a tabela.

Agrupador Este componente é gerado quando o GDIG encontra um Painel Atributo Composto MonoMulti com o domínio *COMPOSTO_AGRUPADOR* (Figura 5.36).

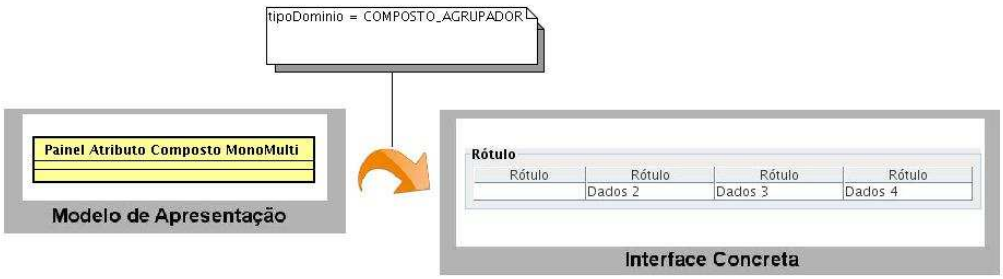

Figura 5.36: *Mapeamento do Painel Atributo Composto MonoMulti no Componente Gráfico Composto MonoMulti Agrupador*

A função deste componente é a de literalmente agrupar outros componentes. Os motivos deste agrupamento podem ser vários como, por exemplo, a economia de espaço ou colocar os componentes de um mesmo assunto juntos.

O Componente Gráfico Composto MonoMulti Agrupador é formado por uma tabela de uma linha. Cada célula representa um componente gráfico que estaria desagrupado mas que agora está dentro do agrupador. Como no formulário da janela de cadastro é obrigatória a presença de um e somente um componente gráfico (exceto se eles forem invisíveis), a tabela do Componente Gráfico Composto MonoMulti Agrupador contém, obrigatoriamente, uma célula de cada componente agrupado.

As Figuras 5.37 e 5.38 demonstram essa idéia. Na Figura 5.37 temos os componentes gráficos 1, 2, 3 e 4 desagrupados. Caso o projetista deseje agrupá-los por, por exemplo, falta de espaço, a Janela de Cadastro ficaria com o formulário mostrado na Figura 5.38.

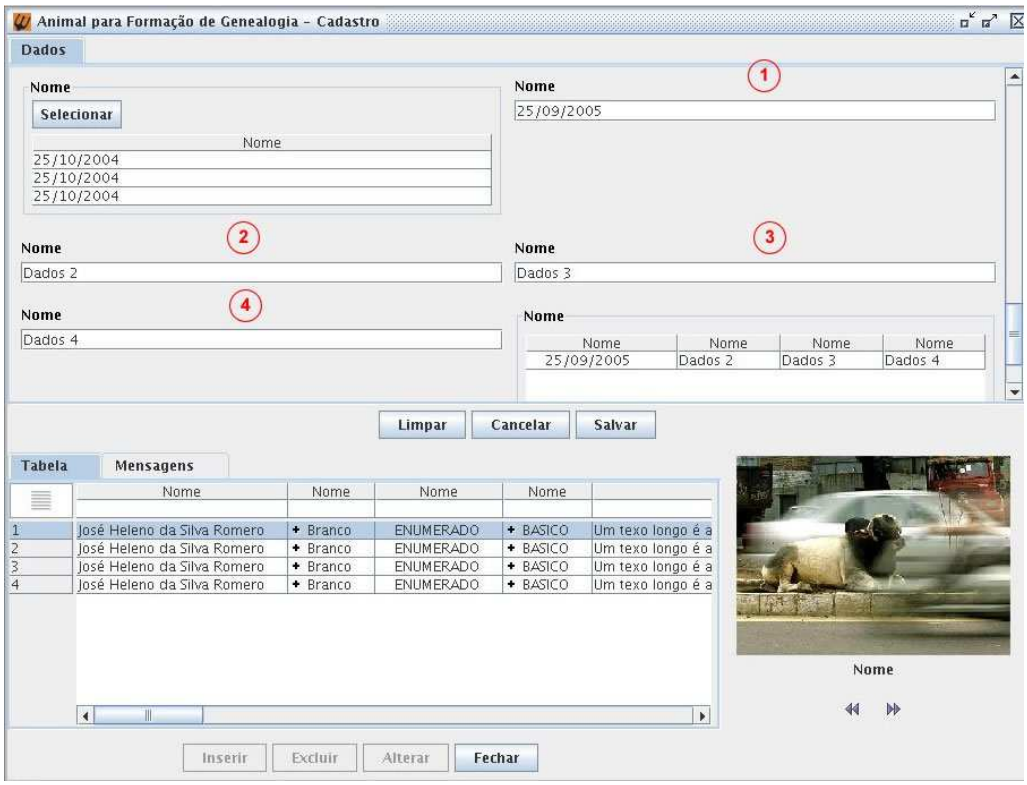

Figura 5.37: *Componentes desagrupados*

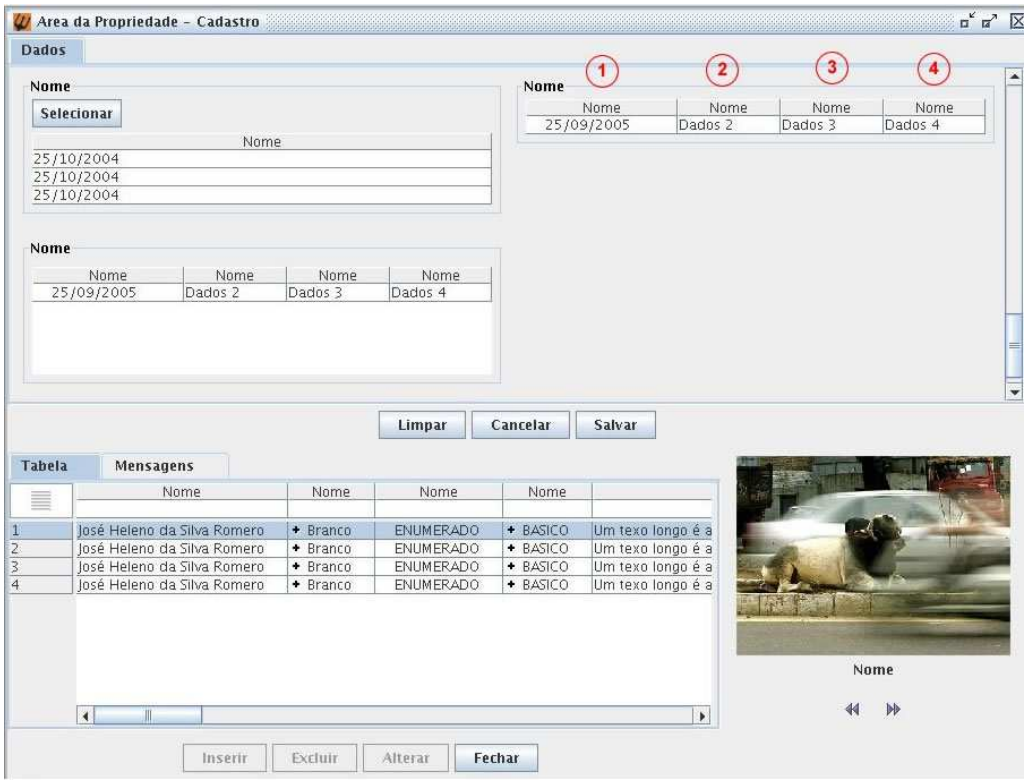

Figura 5.38: *Componentes agrupados*

Lista O Componente Gráfico Composto MonoMulti Lista é gerado quando o GDIG encontra um Painel Atributo Composto MonoMulti com o domínio *COMPOSTO_LISTA* (Figura 5.39).

Tem como objetivo listar informações, não permitindo nenhum tipo de edição e não contendo nenhum tipo de ação.

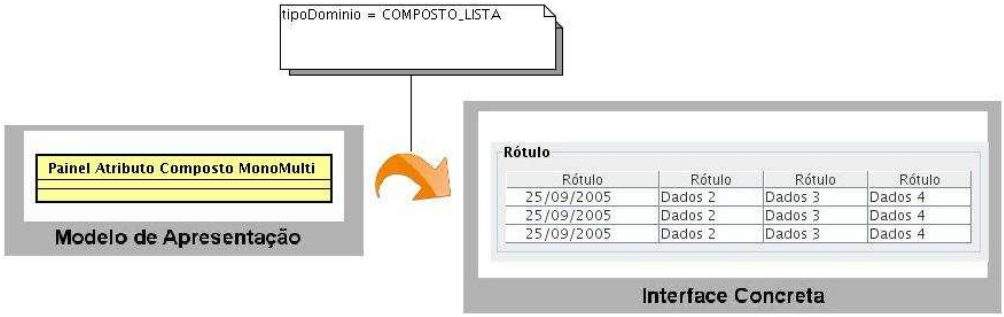

Figura 5.39: *Mapeamento do Painel Atributo Composto MonoMulti no Componente Gráfico Composto MonoMulti Lista*

Este componente gráfico contém uma tabela onde são listadas as informações ao usuário. Suas células não são editáveis.

Componente Gráfico Composto Objeto Interno

O objetivo deste componente é referenciar informações de outras entidades. Por exemplo, considere que tenhamos duas entidades já cadastradas no sistema: Pessoa Física e Animal. Vamos supor também que estamos dentro da tela de Cadastro de Animal e que um dos campos refere-se ao dono do animal. Neste caso este campo deve conter dados sobre uma pessoa física que seria dono do animal que está sendo cadastrado. Definimos então este campo como Objeto Interno e o ajustaríamos para referenciar a entidade Pessoa Física com os campos Nome e CPF.

Neste caso, quando o usuário fosse escolher uma pessoa como dona do animal, esta já deveria estar cadastrada no sistema. Caso contrário o usuário deve primeiro cadastrar a pessoa para depois ajustá-la como dona do animal.

Existem dois tipos de Componente Gráfico Composto Objeto Interno: Simples e Complexo.

Simples Este componente é gerado quando o GDIG encontra um Painel Atributo Composto Objeto Interno com o domínio *COMPOSTO_SIMPLES* (Figura 5.40).

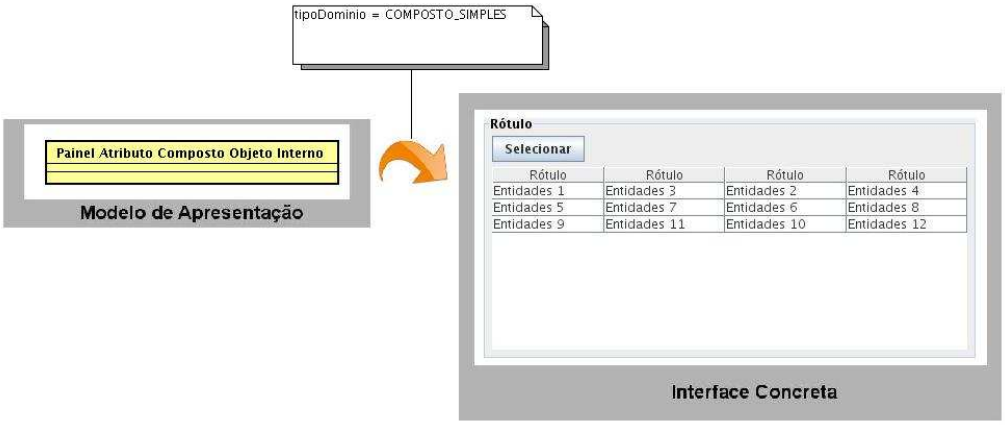

Figura 5.40: *Mapeamento do Painel Atributo Composto Objeto Interno no Componente Gráfico Composto Objeto Interno Simples*

O Componente Gráfico Composto Objeto Interno Simples é formado pelo botão *Selecionar* e pela tabela de dados.

O botão *Selecionar* abre a janela mostrada na Figura 5.41. Essa janela contém dados já cadastrados da entidade que o objeto interno referencia.

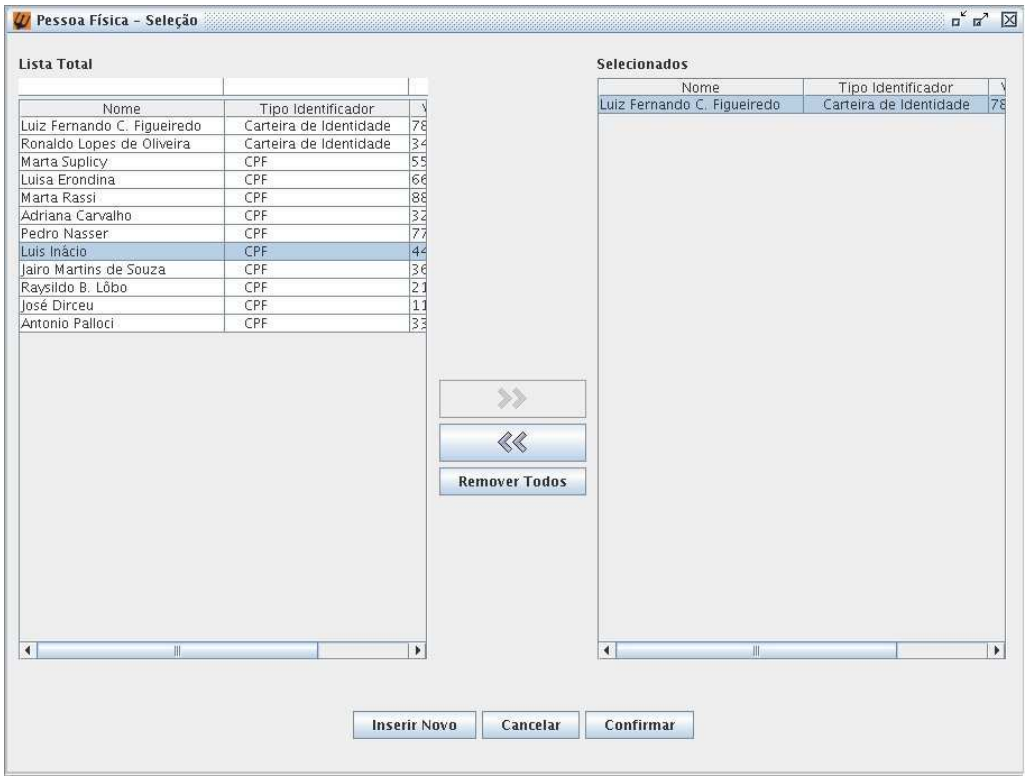

Figura 5.41: *Janela de Seleção do Objeto Interno*

Na janela de seleção, a tabela da esquerda (*Lista Total*) contém todos os elementos já cadastrados da entidade referenciada. A tabela *Selecionados* contém os elementos que o usuário já selecionou.

O botão "»"seleciona novos dados, passando as linhas selecionadas da tabela *Lista Total* para a tabela *Selecionados*.

O botão "«"remove os dados selecionados da tabela *Selecionados*.

O botão *Remover Todos* remove todos os elementos da tabela *Selecionados*.

Ao selecionar o botão *Inserir Novo*, é aberta uma nova janela de cadastro para que o usuário insira um elemento que não consta na tabela *Lista Total*. Essa janela é semelhante à *Janela de Cadastro*.

O botão *Cancelar* volta ao componente gráfico, cancelando a tarefa atual.

O botão *Confirmar* salva as mudanças feitas e insere os valores na tabela do Componente Gráfico Composto Objeto Interno Simples.

Além do botão Selecionar, este componente gráfico contém a tabela que contém os dados. Essa tabela é formada por dois tipos de colunas: aquelas referentes à entidade referenciada e aquelas da própria tabela, que não se referem a nenhuma entidade. São comparadas às colunas de uma tabela de relacionamento de um Banco de Dados: existem as colunas com chaves estrangeiras (colunas das entidades referenciadas) e aquelas do próprio relacionamento (da própria tabela).

Quando obtêm-se dados da entidade referenciada, este são inseridos nas células das colunas referentes à essa entidade e as células das outras colunas ficam vazias. A diferença do Objeto Interno Simples para o Composto é que, no Simples, essas células que não referenciam uma entidade podem ser editadas na própria tabela, enquanto no Composto isso não é permitido.

Complexo O Componente Gráfico Composto Objeto Interno Complexo é gerado quando o GDIG encontra um Painel Atributo Composto Objeto Interno com o domínio *COMPOSTO_COMPLEXO* (Figura 5.42).

A diferença deste para o *SIMPLES* situa-se na edição dos dados da tabela. Neste componente não é permitida a edição na própria tabela.

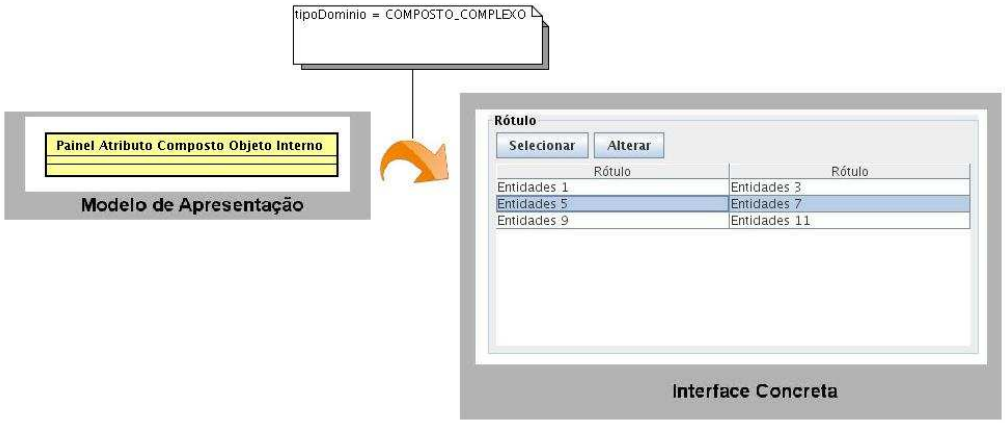

Figura 5.42: *Mapeamento do Painel Atributo Composto Objeto Interno no Componente Gráfico Composto Objeto Interno Complexo*

Este componente é formado pelos botões *Selecionar*, *Alterar* e pela tabela de dados.

O botão *Selecionar* aciona a janela da Figura 5.41.

O botão *Alterar* executa a tarefa de edição dos valores de uma linha selecionada. Ao clicar nesse botão, será aberta uma Janela de Cadastro contendo os campos referentes aos sub-atributos deste componente composto, ou seja, referente às colunas da tabela. A edição poderá ser executada somente nesta janela de cadastro.

5.2.4 Criação da área de ações referentes ao formulário

A área de ações referentes ao formulário contém os botões *Limpar*, *Cancelar* e *Salvar*, como mostra a Figura 5.51.

A ação *Limpar* consiste em limpar o valor de cada um dos campos de todo o formulário. Caso o Componente Gráfico tenha o valor *default* diferente de *null* (a Seção 4.1.1 trata sobre o valor *default*), o valor do campo será ajustado com o seu valor *default*. Caso contrário, será ajustado como vazio. A Figura 5.43 traz um diagrama de colaboração que mostra a sequência de tarefas executadas pelo GDIG quando o botão *Limpar* é acionado.

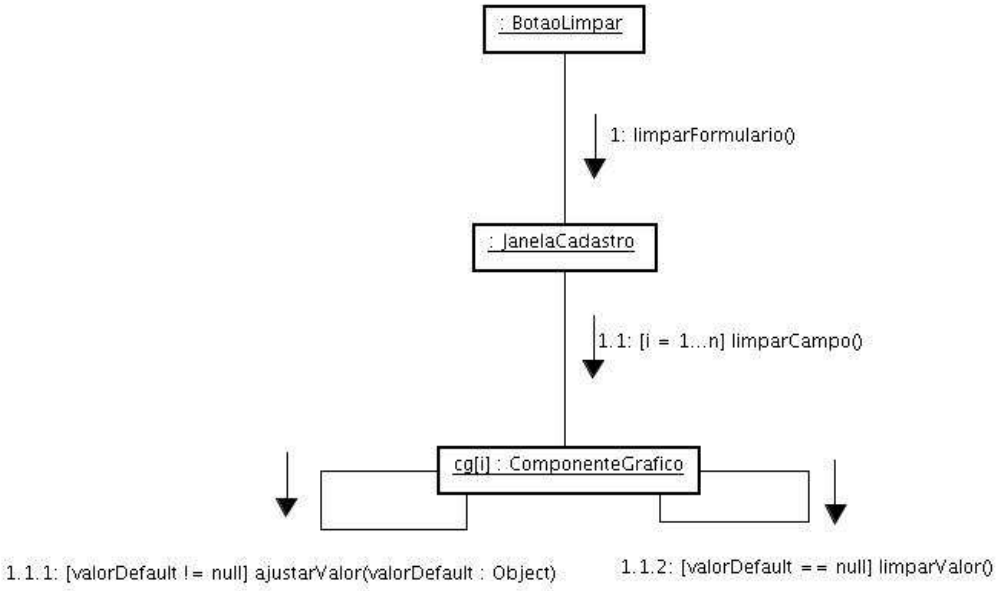

Figura 5.43: *Diagrama de Colaboração da funcionalidade Limpar*

Quando acionado, o botão *Limpar* informa à Janela de Cadastro que os campos do seu formulário devem esvaziar seus valores. A Janela Cadastro percorre todos os Componentes Gráficos e, para cada um, insere o valor *default* ou então limpa o valor.

A ação *Cancelar* executa o cancelamento da ação CRUD que está sendo executada no momento. Por exemplo, caso o usuário esteja alterando dados de um elemento já cadastrado no sistema, ao clicar em *Cancelar*, todas as alterações já realizadas até o momento são canceladas. A Figura 5.44 traz um diagrama de colaboração que mostra a sequência de tarefas executadas pelo GDIG quando o botão *Cancelar* é acionado.

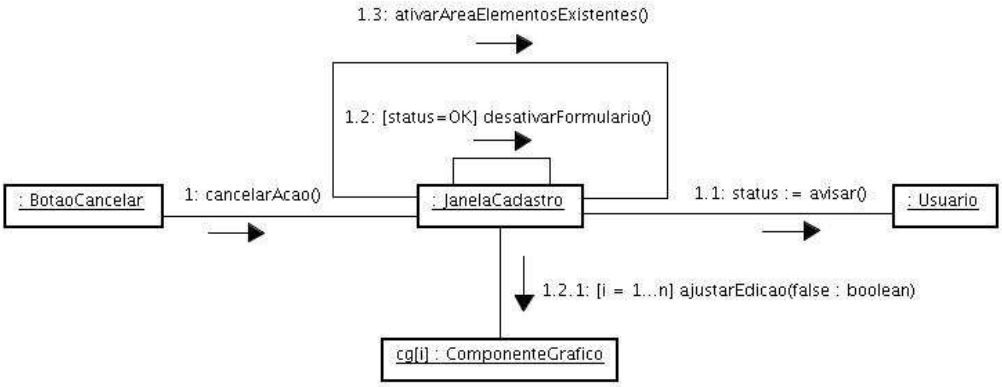

Figura 5.44: *Diagrama de Colaboração da funcionalidade Cancelar*

Ao ser acionado o botão *Cancelar*, este envia uma mensagem para que a Janela Cadastro desative a área de formulário. Em seguida, é enviada uma mensagem de aviso

ao usuário que, caso confirmada, faz com que cada Componente Gráfico tenha sua edição ajustada para falso, desativando toda a área de formulário. Feito isso, a área que contém a tabela com os elementos já cadastradas no sistema será ativada, sendo permitida a interação com seus *widgets*.

A ação *Salvar* executa tarefas que visam à persistência, no Banco de Dados, das modificações executadas no formulário. A Figura 5.45 traz um diagrama de colaboração que mostra a sequência de tarefas executadas pelo GDIG quando o botão *Salvar* é acionado.

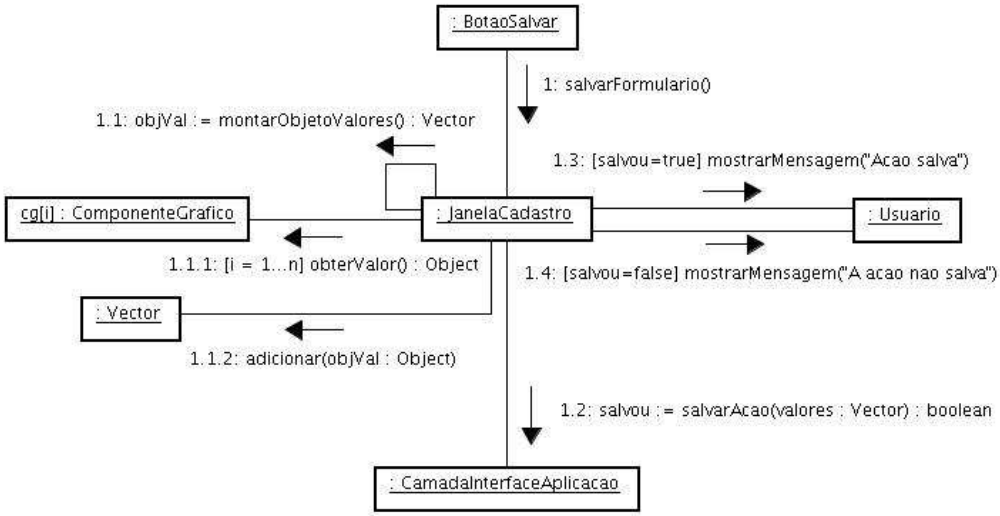

Figura 5.45: *Diagrama de Colaboração da funcionalidade Salvar*

Quando acionado, o botão *Salvar* envia uma mensagem à Janela Cadastro para que esta salve os dados do formulário. Neste momento, a Janela Cadastro cria um vetor e adiciona o valor de cada campo no mesmo. Logo após, enviará esse vetor às camadas inferiores dentro da arquitetura para que os dados sejam persistidos. Caso não ocorra erro, será retornado o valor booleano *true*, o usuário será informado que os dados foram salvos com sucesso e a Janela Cadastro passará o controle para a área de elementos já cadastrados para que o usuário possa acionar alguma outra funcionalidade CRUD. Caso haja algum erro e as camadas inferiores retornem falso, a Janela Cadastro avisará o usuário do erro ocorrido para que este corrija-o, mantendo os campos do formulário editáveis para que o usuário possa ainda manipular seus dados.

As ações de *Cancelar* e *Salvar* são utilizadas também para confirmar ou cancelar as operações CRUD de *Inserir*, *Alterar* e *Excluir*. Esse aproveitamento de botões é vantajoso para a usabilidade do sistema, uma vez que, além de aproveitar um botão já existente, diminui a dificuldade de aprendizagem do usuário e evita as janelas *pop-ups*.

A Seção 5.3 descreve o comportamento visual da Janela de Cadastro quando o usuário clica nos botões *Limpar*, *Cancelar* e *Salvar*.

5.2.5 Criação da área de elementos já cadastrados

Esta área é composta, dentre outros elementos, por uma tabela que contém os elementos já inseridos no sistema referentes a algum Tipo de Entidade. Por exemplo, suponha que um usuário queira inserir mais uma pessoa física no sistema. Ele seleciona no menu Cadastro de Pessoa Física e o GDIG abre a janela de cadastro do tipo de entidade *Pessoa Física*. Nessa janela, esta área contém todas as pessoas físicas já cadastradas no sistema. A Figura 5.46 traz sua estrutura.

Figura 5.46: *Estrutura da Área de elementos já cadastrados*

Essa área é composta de seis partes:

1. **Filtro:** permite que o usuário filtre cada coluna da tabela. À medida que vão sendo digitados os caracteres, as linhas vão sendo filtradas automaticamente.

2. **Cabeçalho:** contém os rótulos das colunas. Os rótulos nos cabeçalhos são os mesmos rótulos dos Componentes Gráficos dentro do formulário. O cabeçalho também fornece a funcionalidade de ordenação. Cada clique no cabeçalho de uma coluna faz com que os elementos dessa coluna sejam ordenados de forma crescente, decrescente e não ordenados.

3. **Elementos existentes:** a tabela propriamente dita que contém os dados dos elementos já cadastrados no sistema.

4. **Scroll vertical:** move as linhas verticalmente. Quando ocorre a rolagem, são movidas a área de elementos existentes e de número de linhas. As demais não sofrem movimentação.

5. **Scroll horizontal:** move as linhas horizontalmente. Quando ocorre a rolagem, são movidas as áreas de filtro, cabeçalho e elementos existentes. As demais não são movimentadas.

6. **Número de linhas:** mostra o número das linhas da tabela. Tem como objetivo mostrar a quantidade de elementos existentes ao usuário e facilitar a busca de algum destes elementos.

Assim como o formulário, a criação da tabela é dirigida por modelos e ocorre dinamicamente em tempo de execução. O GDIG lê o Modelo de Apresentação referente ao Tipo de Entidade e, para cada Painel Atributo, cria uma coluna na tabela. Cada coluna cria um mesmo tipo de componente gráfico para todas as suas células para que os dados possam ser mostrados.

Cada coluna na Área de elementos já cadastrados refere-se a um Componente Gráfico na área de formulário. A Figura 5.47 ilustra essa correspondência.

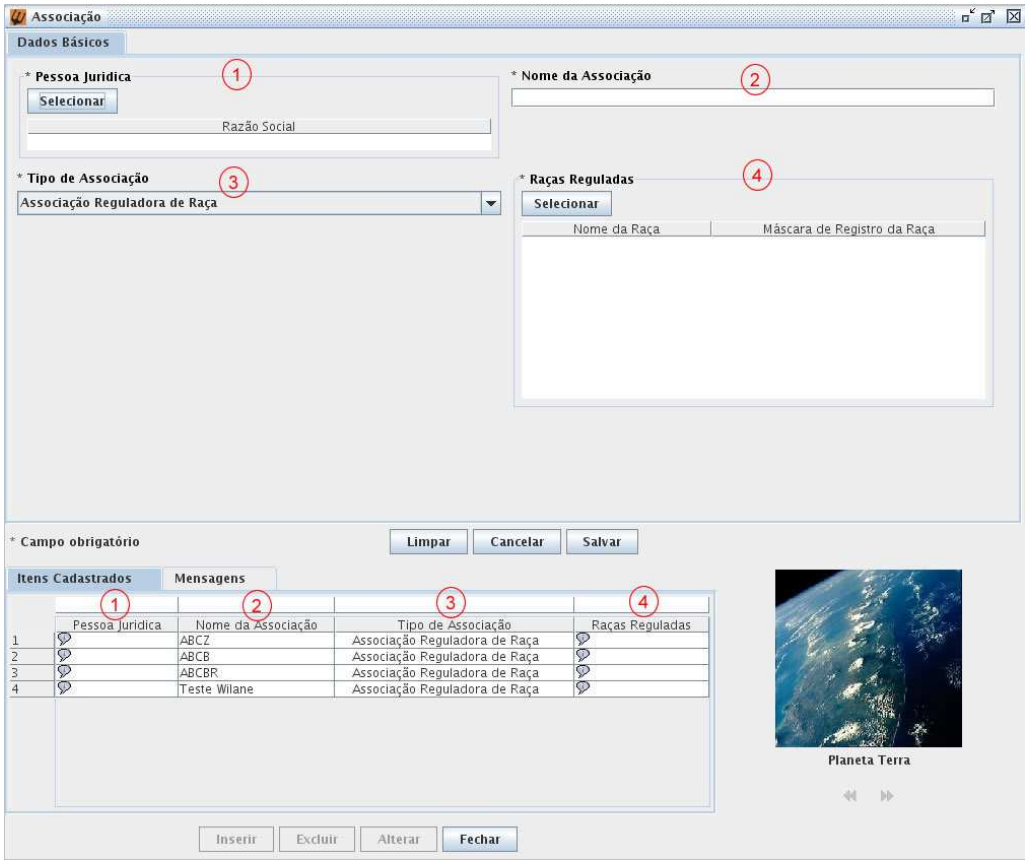

Figura 5.47: *Relação entre Componentes Gráficos e colunas das tabelas*

A tabela sempre encontra-se em modo não-editável, ou seja, o usuário pode somente ver os dados nas linhas. A edição das informações deve ocorrer somente no formulário, pela função *Alterar*.

Na tabela, é permitida a seleção de somente uma linha. Quando essa seleção ocorre, o GDIG envia as informações da linha selecionada para os campos do formulário,

fazendo com que o usuário tenha mais facilidade de observá-las, ao contrário do que se tivesse que lê-las pelas células da tabela.

A Seção 5.3 (comportamento modal) descreve o comportamento visual da Janela de Cadastro quando o usuário seleciona ou cancela a seleção de uma linha da tabela de elementos já cadastrados.

5.2.6 Criação da área de ações CRUD

Esta área compõe-se pelos botões *Inserir* (***C**reate*), *Excluir* (***D**elete*), *Alterar* (***U**pdate*) e *Fechar*, como mostra a Figura 5.51. A única função CRUD que não encontra-se disponibilizada nessa área é a de leitura (***R**ead*). Isso porque ela encontra-se implícita na área de elementos já cadastrados (Seção 5.2.5), onde o usuário pode ler as informações pela tabela.

A ação *Inserir* pode ser executada de duas maneiras. Caso a tabela não tenha linha selecionada, o GDIG ativar a área de formulário com os campos vazios ou com seus respectivos valores *default*, caso haja.

Do outro modo, se alguma linha da tabela apresentar-se selecionada, o GDIG ativar o formulário contendo as respectivas informações. Essa característica é interessante em um caso onde o usuário deseja inserir um novo elemento cujas informações são, em sua grande parte, iguais às de um elemento já cadastrado. Desse modo, o usuário seleciona o elemento já cadastrado e aproveita suas informações, não precisando inseri-las novamente no novo elemento que está sendo criado.

A Figura 5.48 traz o diagrama de colaboração referente à ação *Inserir*.

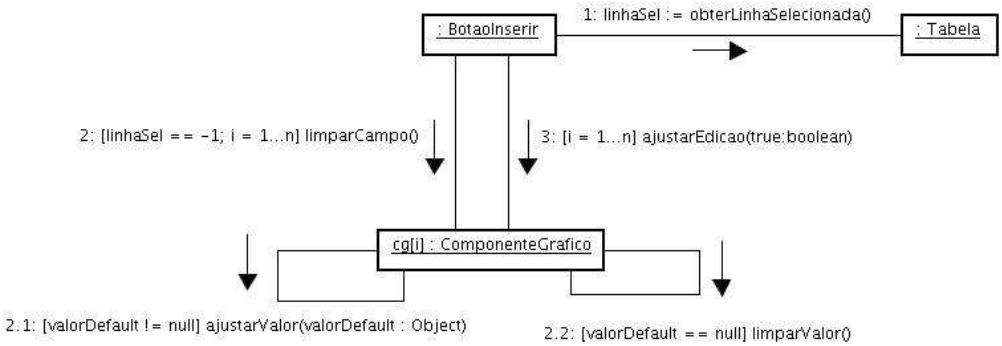

Figura 5.48: *Diagrama de Colaboração da ação Inserir*

Após a ativação do botão *Inserir*, o GDIG verifica se existe alguma linha selecionada na tabela. Caso não exista (*linhaSel == -1*), todos os campos do formulário são limpos ou ajustados com o valor *default* e os campos são ajustados como editáveis. Se houver alguma linha selecionada na tabela, os valores dos campos referentes à linha selecionada já estão presentes no formulário, visto que quando o usuário seleciona uma

linha o GDIG passa os valores da linha para o formulário. Neste caso, basta ajustar os campos como editáveis.

A ação *Excluir* remove um elemento selecionado pelo usuário. A Figura 5.49 traz a sequência de passos para essa ação.

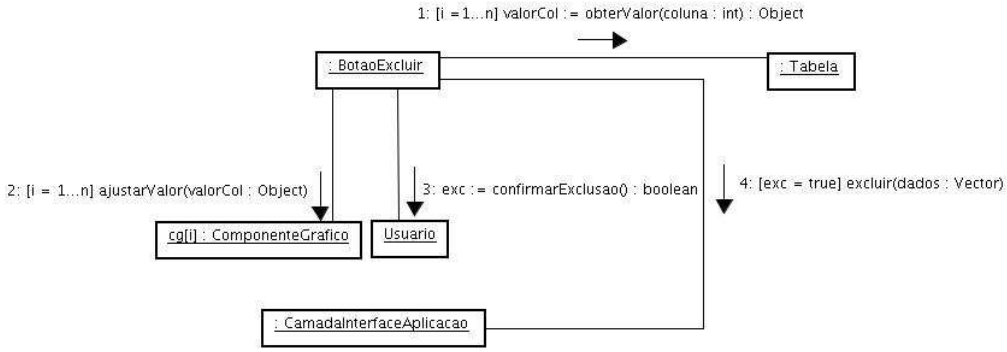

Figura 5.49: *Diagrama de Colaboração da ação Excluir*

Ao clicar no botão *Excluir*, o GDIG mostra os dados da linha selecionada na parte do formulário para que o usuário tenha certeza da remoção. Neste caso, todos os campos encontram-se no modo não-editável. Em seguida, o usuário deve clicar em *Cancelar* ou *Salvar* da área de ações referentes ao formulário (Seção 5.2.4). Caso confirme, o GIDG passa os dados para as camadas inferiores para que a ação seja realizada.

A ação *Alterar* permite que o usuário modifique os dados do elemento selecionado na tabela. A Figura 5.50 mostra como essa ação é realizada.

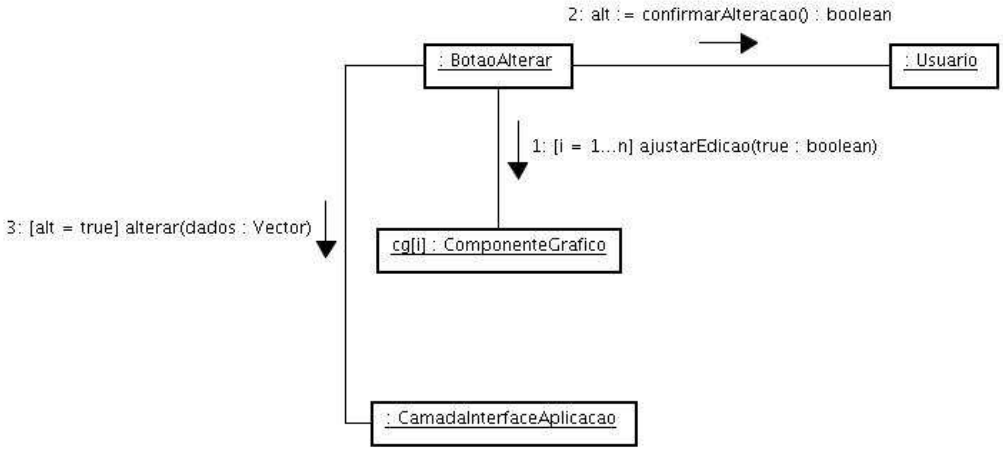

Figura 5.50: *Diagrama de Colaboração da ação Alterar*

Quando o usuário clica em *Alterar*, o GDIG ativa o formulário contendo as informações da linha selecionada e com seus campos editáveis, caso a edição do campo

esteja permitida. Após realizar a alteração, para confirmá-la o usuário deve clicar em *Salvar* e, para cancelá-la, em *Cancelar*, ambos na área de ações referentes ao formulário. Caso a ação seja confirmada, os dados para alteração são enviados para as camadas inferiores dentro da arquitetura.

Ao ser acionada a ação *Fechar*, caso alguma operação CRUD esteja em curso, o GDIG avisará ao usuário que a mesma será perdida. Caso o usuário confirme, o GDIG fecha a Janela de Cadastro. O mesmo ocorre se o usuário clicar no botão fechar na parte superior-direita da janela.

5.2.7 Acoplamento da janela criada à Janela Principal

A Janela Principal tem como objetivo gerenciar o conjunto de janelas abertas dentro do sistema. Todas as janelas específicas geradas dinamicamente são inseridas dentro da Área Funcional da Figura 5.9. A Figura 5.51 traz um exemplo de uma interface CRUD acoplada à Janela Principal.

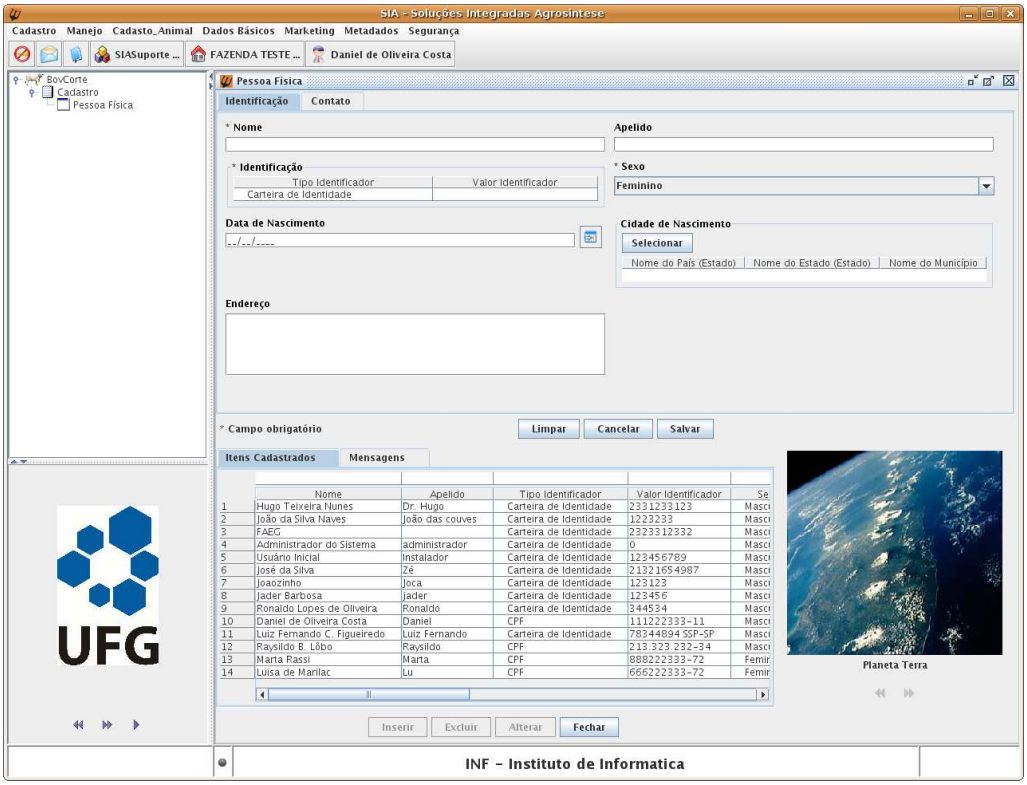

Figura 5.51: *Exemplo de interface gerada dinamicamente acoplada à Janela Principal*

À medida que as janelas são abertas, elas são inseridas na Área Funcional, que funciona em forma de pilha, ou seja, a primeira janela fica no fundo da pilha e a última localiza-se no início da pilha, sendo visível ao usuário.

Cada janela pode ser minimizada. Ao ser executada essa ação, o GDIG transforma a janela em um ícone dentro da Área Funcional. Caso outra janela esteja aberta, esse ícone ficará ao fundo e não será visível. Contudo, se todas as janelas encontrem-se minimizadas, o usuário pode ver os ícones das janelas, como mostra a Figura 5.52.

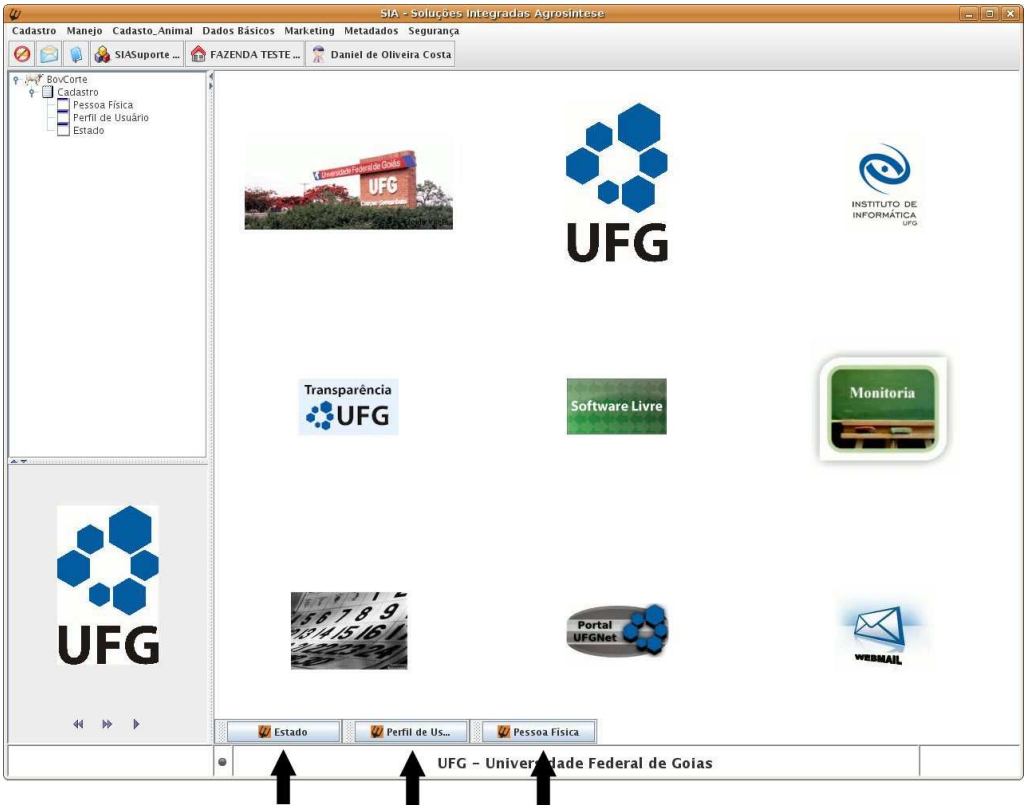

Figura 5.52: *Janelas minimizadas na área funcional*

Como a Área Funcional pode conter várias janelas, e o usuário não consegue visualizar a mais alta no topo da pilha, é preciso disponibilizar um meio rápido para o usuário ter acesso à janela no fundo da pilha. Essa é a função da Área de Janelas Abertas. Para cada janela criada na Área Funcional, o GDIG mantém um *link* na Área de Janelas Abertas, também mostrada na Figura 5.9. Essa área contém uma estrutura de árvore mostrando todas as janelas abertas no momento, onde o usuário pode, através de uma simples seleção, abrir qualquer janela da pilha de janelas abertas. A Figura 5.53 mostra a Área de Janelas Abertas relativa às janelas da Figura 5.52.

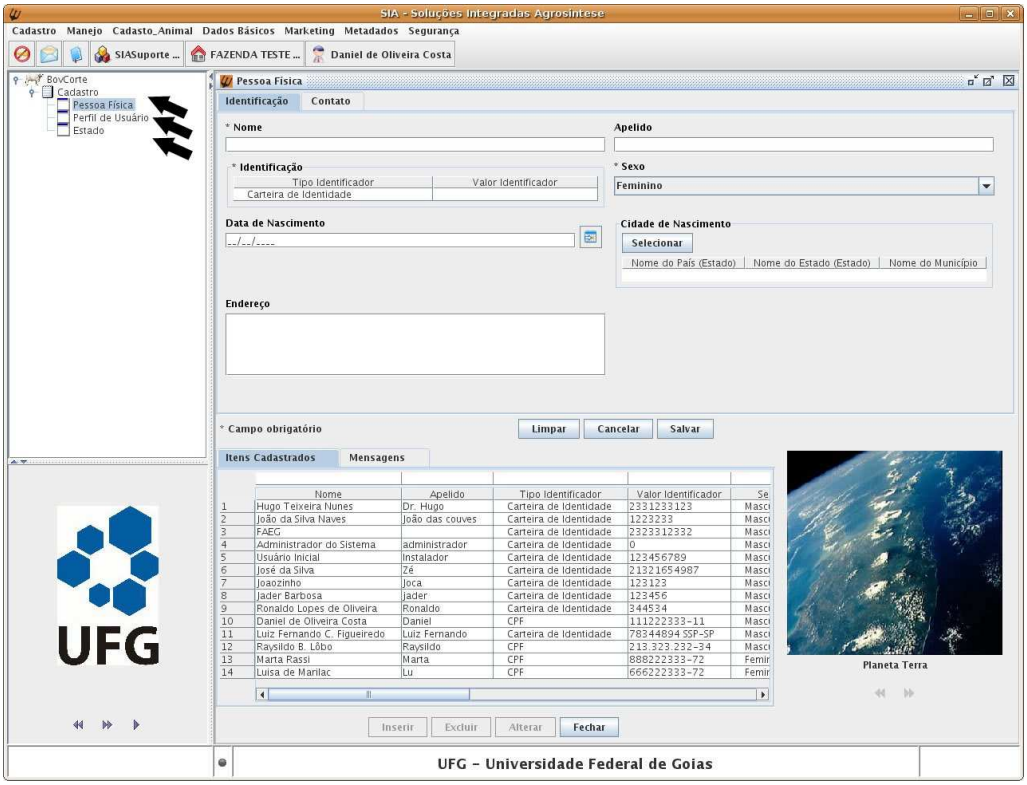

Figura 5.53: *Área de Janelas Abertas*

5.3 Comportamento modal da Janela de Cadastro

A Janela de Cadastro proporciona ao usuário dois modos de trabalho bem claros e específicos:

1. **Modo conjunto:** Visualização do conjunto de instâncias de um conceito. Neste modo o usuário tem uma visão do conjunto de elementos pertinentes a um certo conceito do SI, podendo filtrá-los, selecioná-los e escolher alguma funcionalidade CRUD a ser executada sobre o elemento selecionado (ou, no caso da Inserção, criar um novo elemento). Este modo é visualizado pelas áreas 3 e 4 da Figura 5.15.

2. **Modo elemento:** Manipulação de uma instância de um conceito. Neste modo o usuário edita as informações de um elemento dentro dos campos do formulário, podendo limpar todo o conteúdo do formulário (botão Limpar), cancelar a operação de edição (botão Cancelar), ou concluir a edição (botão Salvar). Este modo é visualizado pelas áreas 1 e 2 da Figura 5.15.

O mecanismo modal ocorre da seguinte forma. Quando uma Janela de Cadastro é aberta, ela encontra-se no modo 1, como mostra a Figura 5.54. Neste modo, os componentes gráficos do modo 2 estão desabilitados, ou seja, encontram-se não editáveis.

Quando o usuário passa a trabalhar no modo 2, através da ativação de uma operação sobre um elemento selecionado, os componentes gráficos do modo 1 tornam-se não editáveis, enquanto os do modo 2, tornam-se disponíveis para o usuário.

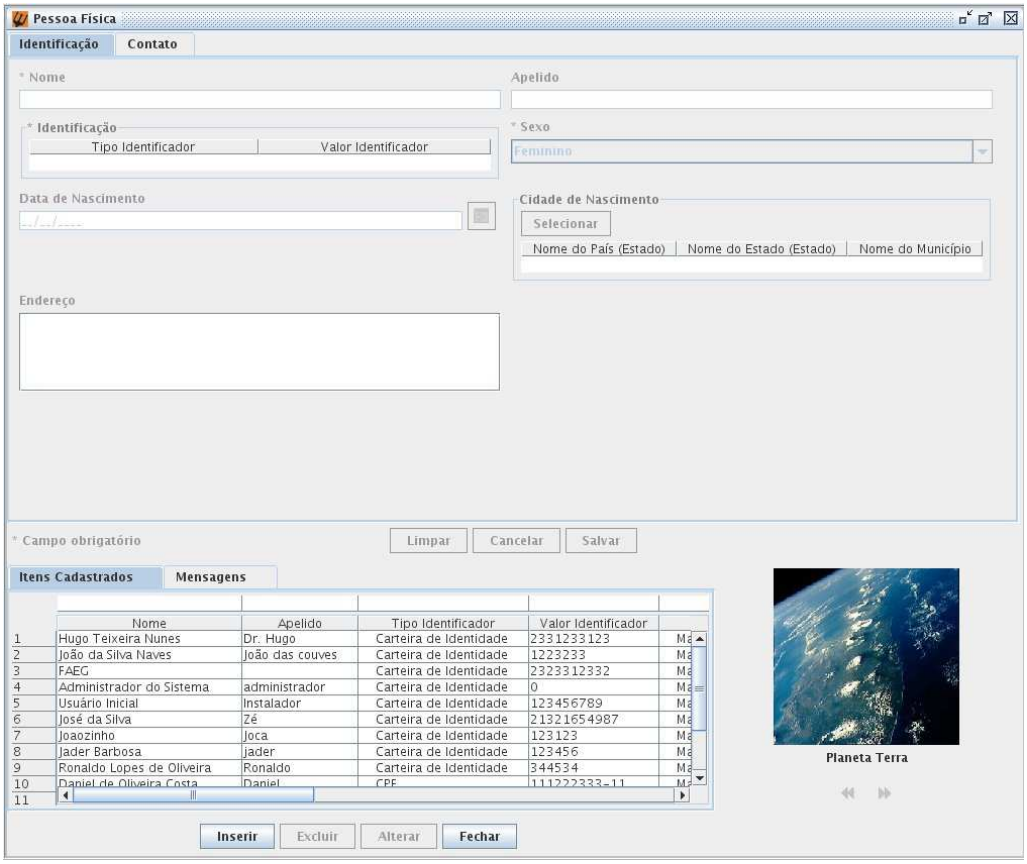

Figura 5.54: *Janela de Cadastro no Modo Conjunto*

Quando uma GUI é criada, ela é apresentada inicialmente no modo conjunto e traz, na tabela inferior, os elementos daquele tipo de GUI já cadastrados no sistema. Por exemplo, se o usuário abrir uma janela de cadastro de Pessoa Física, a tabela inferior mostrará todas as Pessoas Físicas já cadastradas no SI.

No modo conjunto, todos os campos do formulário encontram-se desabilitados, assim como os botões *Limpar*, *Cancelar* e *Salvar*. O usuário pode interagir somente com a área de elementos já cadastrados e com os botões *Inserir*, *Excluir*, *Alterar* e *Fechar*. Se nenhuma linha da tabela estiver selecionada, os botões *Excluir* e *Alterar* ficam desabilitados. Porém, como mostra a Figura 5.55, quando o usuário seleciona alguma linha esses botões são habilitados, permitindo sua execução.

É importante observar que, o administrador do SI pode retirar a autorização de um usuário, impedindo-o executar alguma das operações de inserção, exclusão ou alteração. Neste caso, o botão referente a essa função não autorizada encontrar-se-á sempre desabilitado, mesmo quando o usuário selecionar alguma linha da tabela. Desta

forma, o mecanismo de interface implementa funcionalidades de controle de acesso, de acordo com as regras definidas no SI.

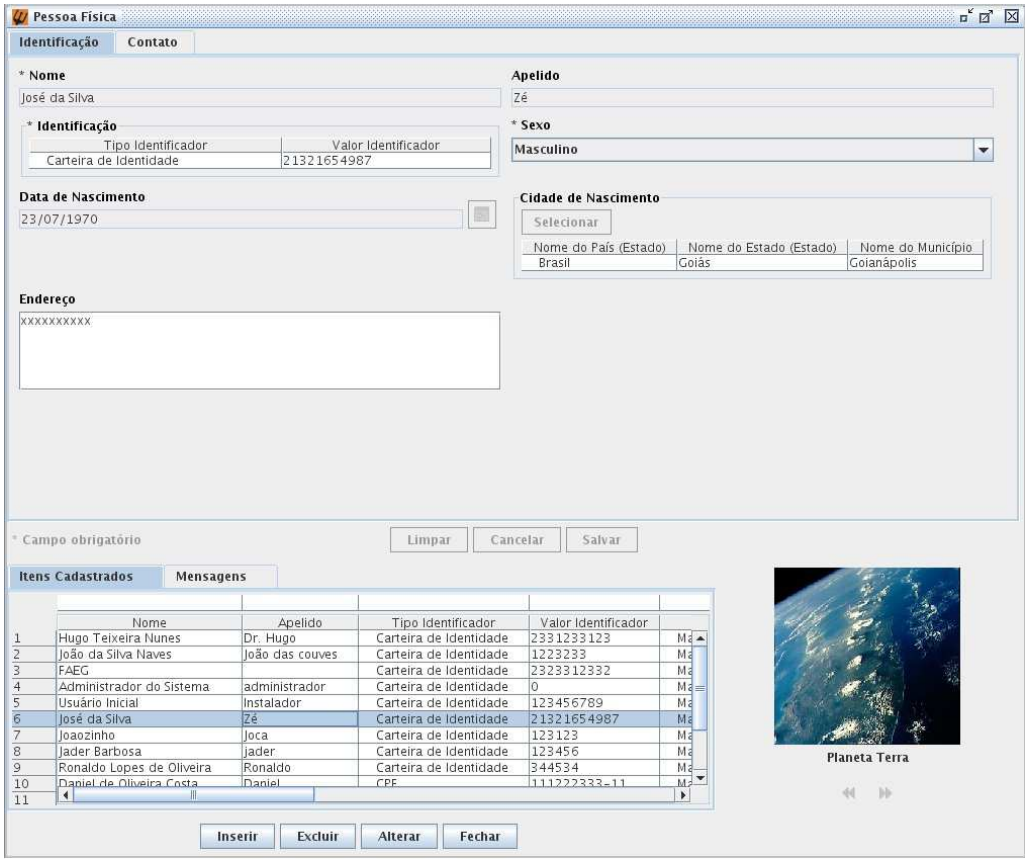

Figura 5.55: *Janela de Cadastro no Modo Conjunto com linha selecionada*

Observa-se que, com a linha selecionada, além de os botões tornarem-se habilitados para execução, de acordo com as permissões de acesso do usuário, o GDIG encarrega-se de repassar os dados da linha escolhida para os campos do formulário. Com isso, o usuário tem um meio de consultar todos os dados de um elemento. Porém, os campos do formulário continuam não-editáveis.

As abas, que agrupam e organizam os componentes gráficos, ficam habilitadas para que o usuário possa consultar as informações de todos os campos do formulário.

A Janela Cadastro passa para o Modo Elemento assim que o usuário ativa um dos botões: *Inserir*, *Excluir* ou *Alterar*. A Figura 5.56 mostra o visual deste modo.

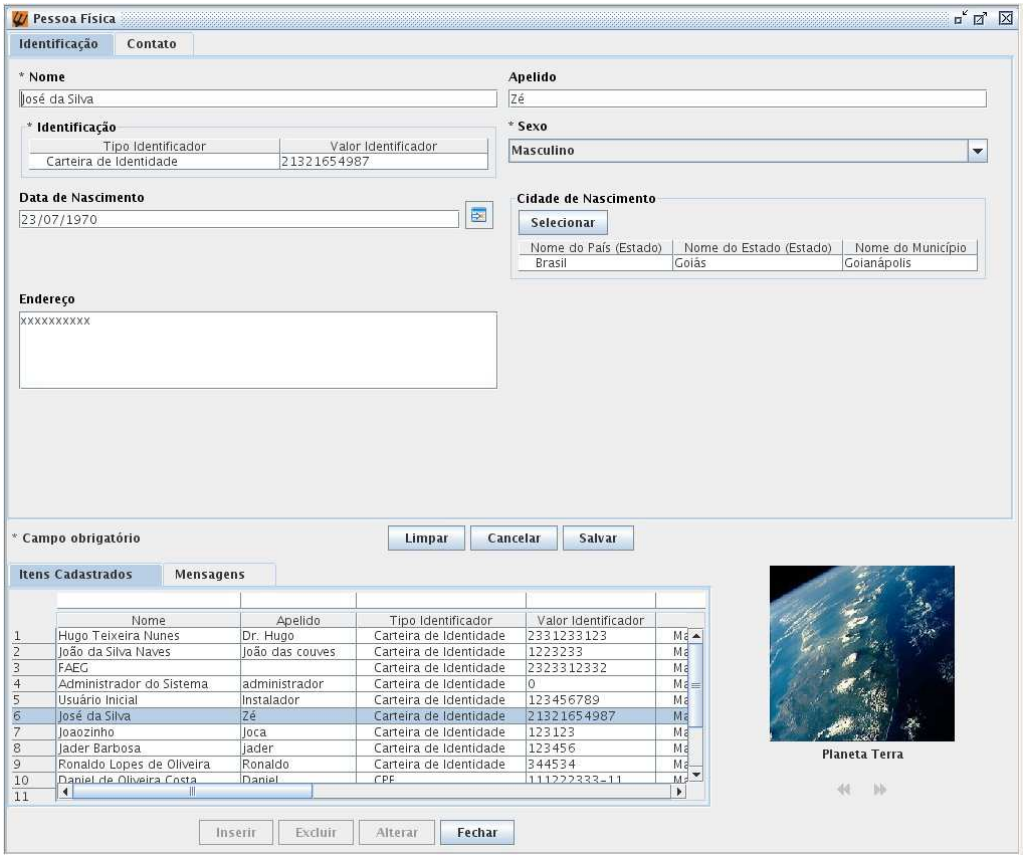

Figura 5.56: *Janela de Cadastro no Modo Elemento*

No Modo Elemento, os campos do formulário encontram-se habilitados (editáveis), assim como os botões *Limpar*, *Cancelar* e *Salvar*. Por outro lado, os *widgets* das áreas 3 e 4 apresentam-se não-editáveis, exceto a tabela de elementos, que não é completamente desabilitada. Apesar disso, a tabela tem um comportamento diferenciado. O usuário pode executar rolagens horizontal e vertical para verificar os dados dos demais elementos do conjunto. Porém, a seleção de outro elemento não é permitida, já que a linha selecionada indica o elemento está sendo utilizado para a execução da funcionalidade CRUD.

Quando o usuário ativa o botão *Cancelar* ou *Salvar*, a interface retorna ao Modo Conjunto, na forma que encontrava-se anteriormente, ou seja, com ou sem a linha de elemento selecionada.

Interfaces modais têm sérias críticas devido ao maior trabalho de interação, já que o usuário precisa mudar de modo para executar as funcionalidades. Apesar disso, decidiu-se utilizar o comportamento modal para privilegiar a segurança, principalmente para informações críticas, em detrimento da rapidez na interação. Para executar, por exemplo, uma ação de remoção, o usuário deve indicar essa intenção, a interface então passa para o modo de remoção (protegendo e habilitando componentes gráficos pertinentes) e o usuário executa efetivamente a deleção. Com isso, reduz-se a possibilidade de o usuário

deletar um elemento por acaso.

CAPÍTULO 6

Estudo Comparativo Experimental sobre a Produtividade na Construção de GUI

Medir a complexidade de um software, ou de parte dele, é tarefa fundamental para se avaliar o esforço necessário para sua produção. A partir desse esforço é possível estimar custo, prazo e recursos para o desenvolvimento do software.

Esse capítulo avalia a produtividade no desenvolvimento de GUIs obtido pelo uso do GDIG, em comparação com outras abordagens, através da medição da complexidade das interfaces gráficas geradas. A Seção 6.1 descreve a metodologia seguida nesse experimento. A Seção 6.2 mostra como ocorreu a coleta de dados, e a Seção 6.3 traz as conclusões a partir dos dados obtidos.

6.1 Metodologia e Métricas

Para avaliar a produtividade com o uso do GDIG no desenvolvimento das GUIs, a seguinte metodologia foi adotada. Primeiramente, foram escolhidas três GUI CRUD para serem geradas, uma de menor, uma de média e outra de maior complexidade, definidas como tela 1, tela 2 e tela 3, respectivamente. As figuras 6.1, 6.2 e 6.3 a 6.9 mostram essas telas.

Em seguida, calculou-se a complexidade de cada tela (subseções 6.1.1, 6.1.2 e 6.1.3). Essa medida foi definida com base no número de *widgets* (componentes gráficos) de cada GUI e no comportamento associado aos *widgets*. Para definir a métrica de complexidade foram tomadas as seguintes decisões:

1. Cada *widget*, como botão, *combo box*, caixa de texto ou rótulo, tem o mesmo valor. Somente a tabela contém valores diferenciados de acordo com o número de colunas, pois cada coluna corresponde a um *widget*.

2. Em um primeiro momento, escolheu-se a Análise por Ponto de Função (APF) para a contagem da complexidade da interface. Porém verificou-se, posteriormente, que a APF não é adequada para medir GUIs pois baseia-se principalmente nas

funcionalidades do ponto de vista do usuário, não considerando questões que impactam no custo de geração de GUIs (como número de componentes gráficos e dificuldade para a implementação de suas aparências e comportamentos).

Executou-se, então, uma revisão sistemática descrita no Apêndice A em busca de uma metodologia para contagem ou estimativa da complexidade de interfaces gráficas. Como não obteve-se êxito, decidiu-se pela metodologia descrita nesse capítulo.

3. Foram computados os *widgets* somente da tela CRUD, não sendo levados em conta os da Janela Principal, pois o objetivo do experimento é verificar a produtividade de telas dirigidas por modelos, o que não é o caso da Janela Principal.

4. Os *widgets* computados são aqueles localizados em todas as áreas da Figura 5.15. Desse modo, haverá diferença no número de *widgets* somente na área de campos do formulário e no número de colunas da tabela de elementos já cadastrados, visto que esses dependem do número de atributos nos modelos. As demais áreas terão o mesmo número de *widgets* para todas as telas.

Uma vez definida a métrica, mediu-se o tempo de geração das GUIs. Cada tela foi gerada de três formas: (i) por meio de codificação manual utilizando o NetBeans IDE 6.1 como editor; (ii) através de editor visual, onde o programador somente escolhe e arrasta os componentes gráficos para formar a tela, utilizando também a ferramenta NetBeans IDE 6.1; e (iii) através da ferramenta GDIG, onde o programador determina as informações dos modelos para que a ferramenta gere a GUI.

Cada tipo de geração foi executado por quatro desenvolvedores (D1, D2, D3 e D4), todos atuantes na área de desenvolvimento de software, com nível semelhante de conhecimento e experiência na linguagem Java e no pacote *Swing*.Em cada geração da tela foi medido o tempo, em horas, para se determinar a produtividade no desenvolvimento da interface gráfica.

Após as medições dos tempos, calculou-se e comparou-se a produtividade através da métrica tempo/*widget*. Ou seja, quanto tempo cada tipo de geração levou para construir cada *widget*.

Para a obtenção do tempo de geração de cada tela foram implementadas somente suas aparências e alguns comportamentos. As aparências e comportamentos localizam-se todos dentro da camada de apresentação, ou seja, dentro do escopo da interface gráfica, não envolvendo regras de negócio. Os comportamentos implementados foram do tipo:

1. **ação de acionamento de botão:** quando um botão é acionado e abre outra janela. As ações que envolvem o envio de mensagens para camadas inferiores do *framework* que não se referem a questões de interface não foram implementadas.

2. **ação de alteração de valor de *widget*:** quando o valor de um *widget* é modificado esse tipo de ação é executada para modificar comportamentos ou aparências de um outro *widget*. Por exemplo, quando o usuário escolhe um "Estado" em uma lista de estados, uma outra lista altera seus valores de modo que contenha as cidades do estado escolhido.

3. **ações de manipulação de tabela:** são aquelas que ocorrem quando uma linha da tabela é selecionada ou quando o usuário quer filtrar ou ordenar seus dados.

4. **ações de estados da interface:** são aquelas onde determinados componentes gráficos são habilitados ou desabilitados como, por exemplo, desativar botões quando outros são ativados, ou controlar os modos de uma interface modal.

5. **tratamento de validações:** algumas validações ocorrem na camada de interface, antes de os dados serem enviados para as camadas de aplicação ou persistência. São validações de pequeno custo, cuja execução é indispensável para a continuação do processamento. Como exemplo desse tipo de validação tem-se a validação de máscara de edição, como no caso de datas e validação de tipos de dados na entrada dos campos.

A produtividade analisada não se refere, portanto, somente à geração da aparência da interface, mas também de todo o mecanismo comportamental que a envolve e que com ela se relaciona. Dessa forma consegue-se uma aproximação maior da aplicação gerada pelo GDIG.

6.1.1 Tela de menor complexidade

A tela de menor complexidade (tela 1) é mostrada na Figura 6.1. Essa tela tem como objetivo cadastrar itens de interface com seus respectivos rótulos (*labels*) nos diferentes idiomas suportados pelo GDIG.

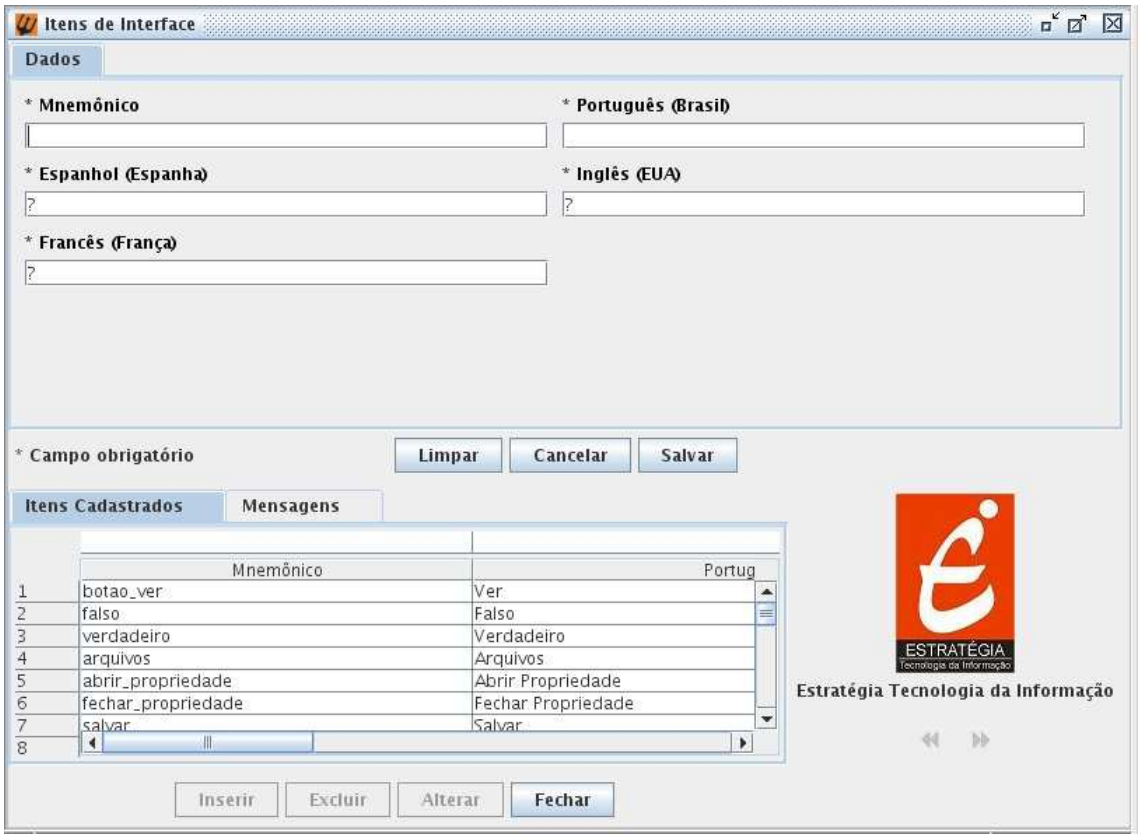

Figura 6.1: *Tela de menor complexidade (tela 1)*

A tela 1 pode ser descrita estruturalmente, em relação a visuais e comportamentos, como mostra a Tabela 6.1. Como todos os campos do formulário contêm rótulos, estes não foram indicados na tabela, mas entram na contagem do número de *widgets*. Com isso, tela 1 tem um tamanho de 36 *widgets*.

Tabela 6.1: *Descrição da tela de menor complexidade (tela 1)*

Área da janela	Componente(s) gráfico(s)	Comportamento(s)	Tipo(s) de comportamento
Campos			
1	Campo de texto	Não há	
2	Campo de texto	Não há	
3	Campo de texto	Não há	
4	Campo de texto	Não há	
5	Campo de texto	Não há	
Ações			
	Rótulo	Não há	
	Botão Limpar	Limpar campos do formulário	2, 4

Continua na próxima página...

Tabela 6.1 – Continuação...

Área da janela	Componente(s) gráfico(s)	Comportamento(s)	Tipo(s) de comportamento
	Botão Cancelar	Cancelar operação CRUD	4
	Botão Salvar	Salvar operação CRUD	4, 5
Tabela			
	5 campos de filtro	Filtrar os dados da tabela	3
	5 Cabeçalhos da tabela (um para cada coluna)	Executar a ordenação dos dados da coluna	3
	5 Campos da tabela (um para cada coluna)	Não há	
	Número de linhas da tabela	Seleciona a linha correspondente na tabela e ativa/desativa botões de ações CRUD	3, 4
	Scroll vertical	Rola a tabela verticalmente	3
	Scroll horizontal	Rola a tabela horizontalmente	3
Ações CRUD			
	Botão Inserir	Ativa os widgets do formulário e ativa os botões de ação do formulário	2, 4, 5
	Botão Excluir	Envia os dados para o formulário e ativa os botões de ação do formulário	2, 4
	Botão Alterar	Idem ao botão Inserir	
	Botão Fechar	Fecha a janela	1

6.1.2 Tela de média complexidade

A tela de complexidade média (tela 2) é mostrada na Figura 6.2.

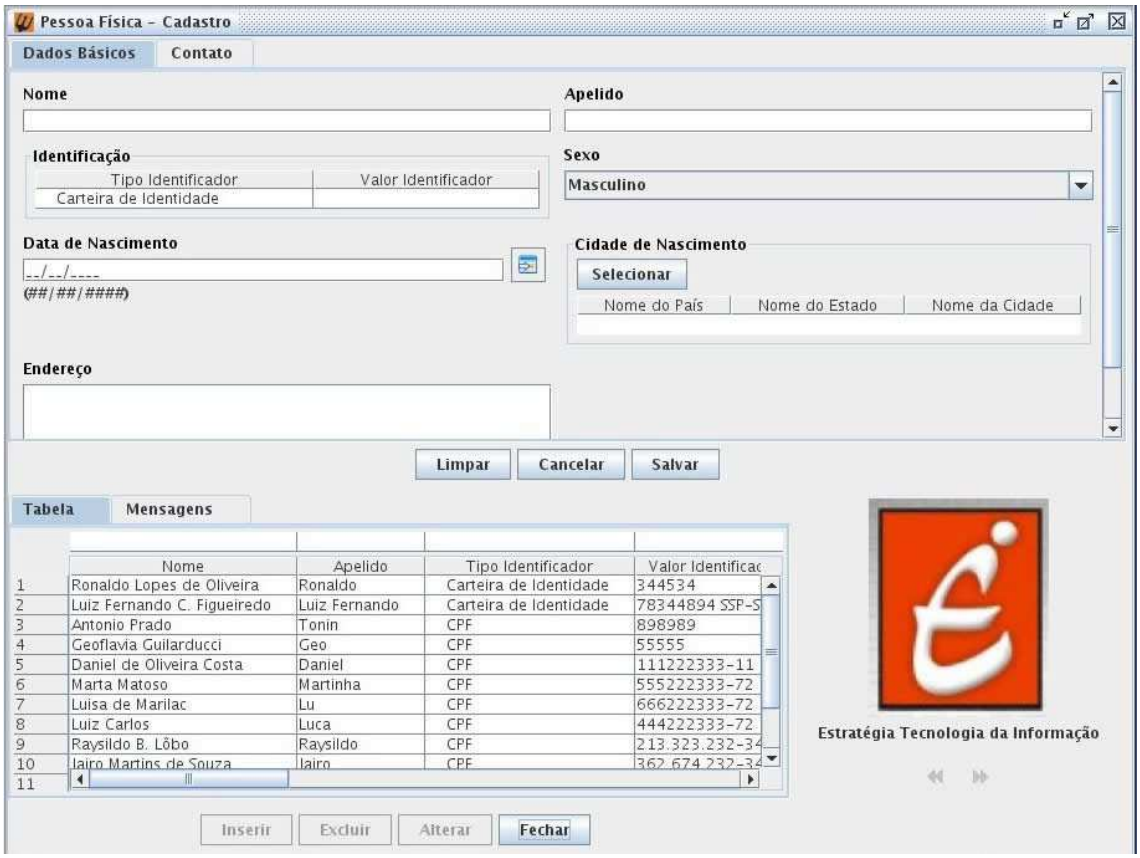

Figura 6.2: *Tela de média complexidade*

Esta tela tem como objetivo cadastrar e gerenciar entidades do tipo Pessoa Física. Contém, portanto, dados sobre uma pessoa (como Data de Nascimento, Nome, Cidade de Nascimento, dentre outros). Diferentemente da tela 1, a tela 2 contém um número maior de campos e campos mais complexos tanto no visual quanto no comportamento.

A tela 2 pode ser descrita estruturalmente, em relação a visuais e comportamentos, como mostra a Tabela 6.2. Como todos os campos do formulário contêm rótulos, estes não foram indicados na tabela, mas entram na contagem dos *widgets*.

Tabela 6.2: *Descrição da tela de tamanho médio (tela 2)*

Área da janela	**Componente(s) gráfico(s)**	**Comportamento(s)**	**Tipo(s) de comportamento**
Campos			
1	Campo de texto	Não há	
2	Campo de texto	Não há	
3	Tabela contendo os seguintes *widgets*:	Os dados do Combo box são obtidos do BD.	2, 3

Continua na próxima página...

Tabela 6.2 – Continuação...

Área da janela	Componente(s) gráfico(s)	Comportamento(s)	Tipo(s) de comportamento
	Combo box; Campo de texto;		
4	Combo box	Os dados do Combo box são obtidos do BD.	2
5	Campo de texto; Botão;	O campo de texto analisa se a informação inserida pelo usuário está de acordo com a máscara de edição. O botão abre um mini-aplicativo de calendário para que o usuário escolha a data.	2, 5 1
6	Botão; Tabela contendo os seguintes campos: Campo de texto; Campo de texto; Campo de texto;	O botão abre uma janela para que o usuário selecione os dados que serão inseridos nos campos de texto.	1
7	Campo de texto	Não há	
8	Botão Botão Tabela contendo os seguintes campos: Campo de texto; Campo de texto;	O botão “Inserir” adiciona uma linha na tabela. O botão “Excluir” remove uma linha na tabela. O Combo box contém dados que são obtidos do BD. A tabela controla a quantidade máxima e mínima de linhas permitidas, de acordo com as informações contidas no Modelo de Apresentação.	2, 3 2, 3

Continua na próxima página...

Tabela 6.2 – Continuação...

Área da janela	**Componente(s) gráfico(s)**	**Comportamento(s)**	**Tipo(s) de comportamento**
9	Combo box;		2, 5
	Tabela contendo os seguintes campos: Campo de texto; Campo de texto;	Não há	
Ações			
	Rótulo	Não há	
	Botão Limpar	Limpar campos do formulário	2, 4
	Botão Cancelar	Cancelar operação CRUD	4
	Botão Salvar	Salvar operação CRUD	4, 5
Tabela			
	9 campos de filtro	Filtrar os dados da tabela	3
	9 Cabeçalhos da tabela (um para cada coluna)	Executar a ordenação dos dados da coluna	3
	9 Campos da tabela (um para cada coluna)	Não há	
	Número de linhas da tabela	Seleciona a linha correspondente na tabela e ativa/desativa botões de ações CRUD	3, 4
	Scroll vertical	Rola a tabela verticalmente	3
	Scroll horizontal	Rola a tabela horizontalmente	3
Ações CRUD			
	Botão Inserir	Ativa os widgets do formulário e ativa os botões de ação do formulário	2, 4, 5
	Botão Excluir	Envia os dados para o formulário e ativa os botões de ação do formulário	2, 4
	Botão Alterar	Idem ao botão Inserir	
	Botão Fechar	Fecha a janela	1

A diferença entre as telas 1 e 2 pode ser obtida comparando-se os dados das tabelas 6.1 e 6.2. A tela 2 contém 66 *widgets* e a tela 1 tem 36 *widgets*. Percebe-se que o tamanho da tela 2 é praticamente o dobro do tamanho da tela 1.

6.1.3 Tela de maior complexidade

A tela de maior complexidade (tela 3) possui, na área de formulário, 24 campos para entrada de dados, divididos em 7 abas, como demonstram as Figuras 6.3 a 6.9.

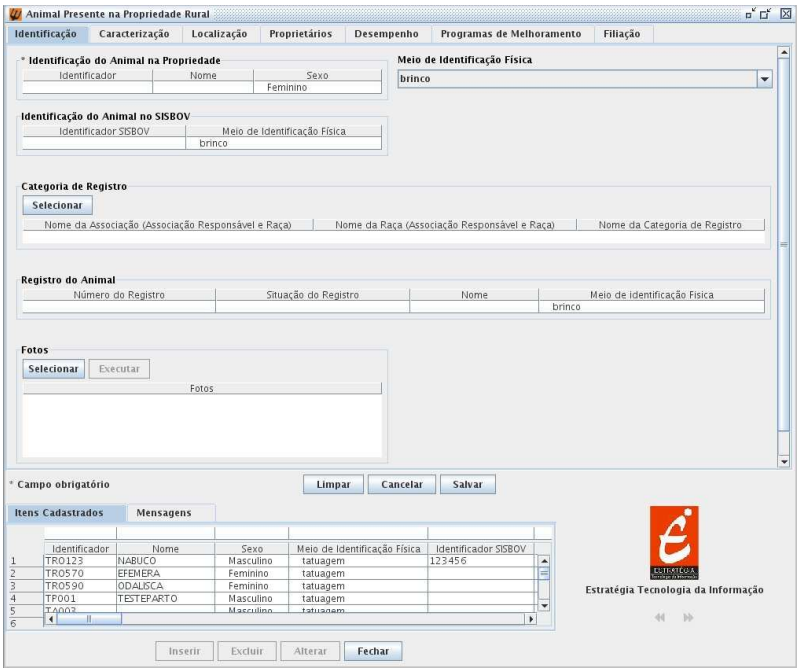

Figura 6.3: *Tela 3, aba 1*

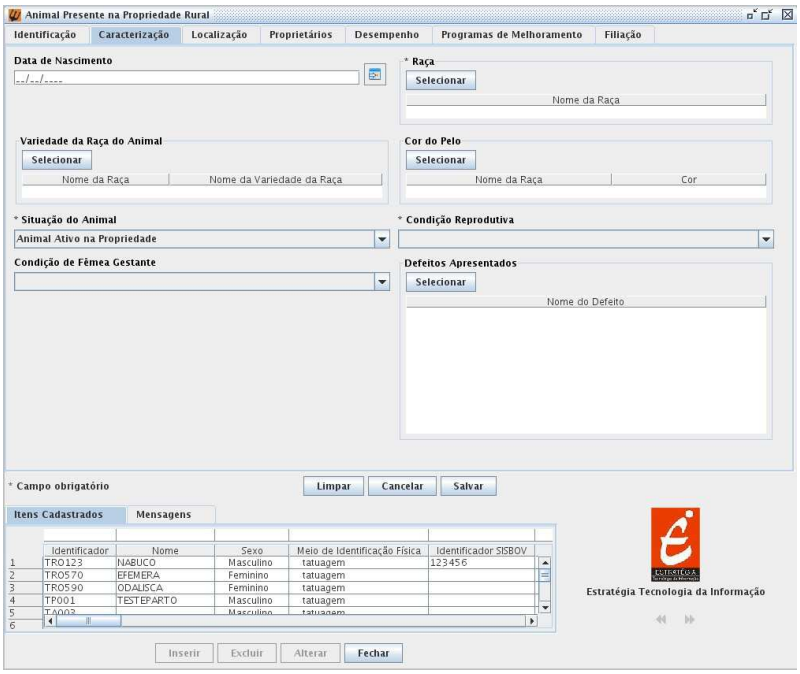

Figura 6.4: *Tela 3, aba 2*

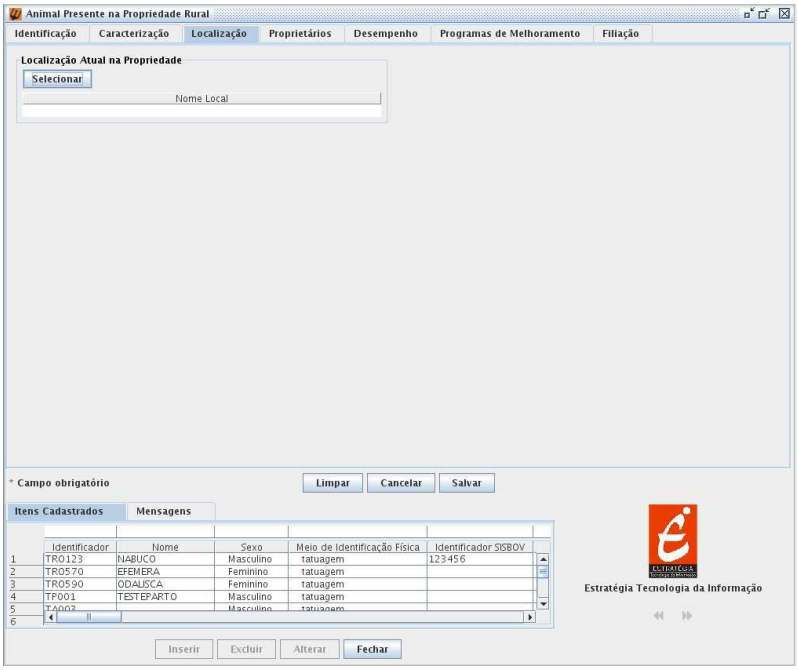

Figura 6.5: *Tela 3, aba 3*

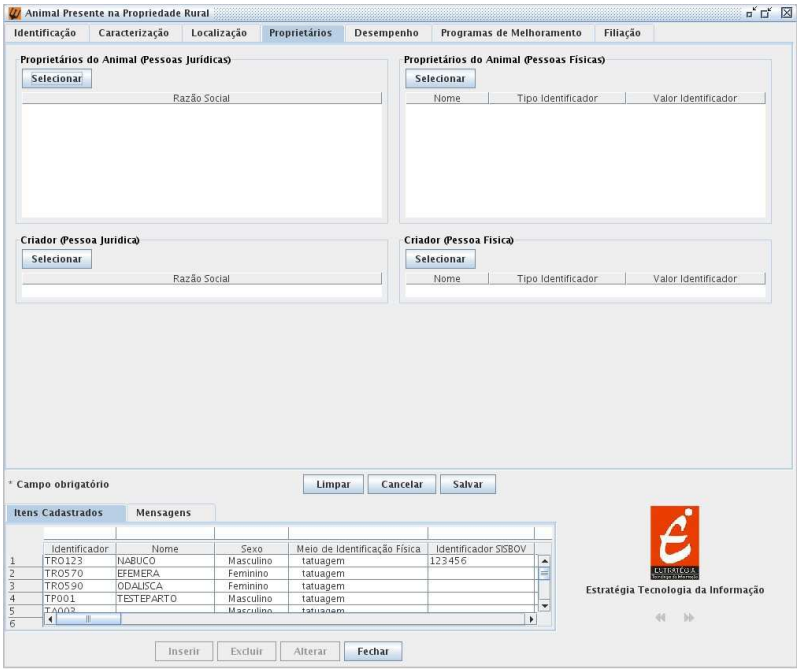

Figura 6.6: *Tela 3, aba 4*

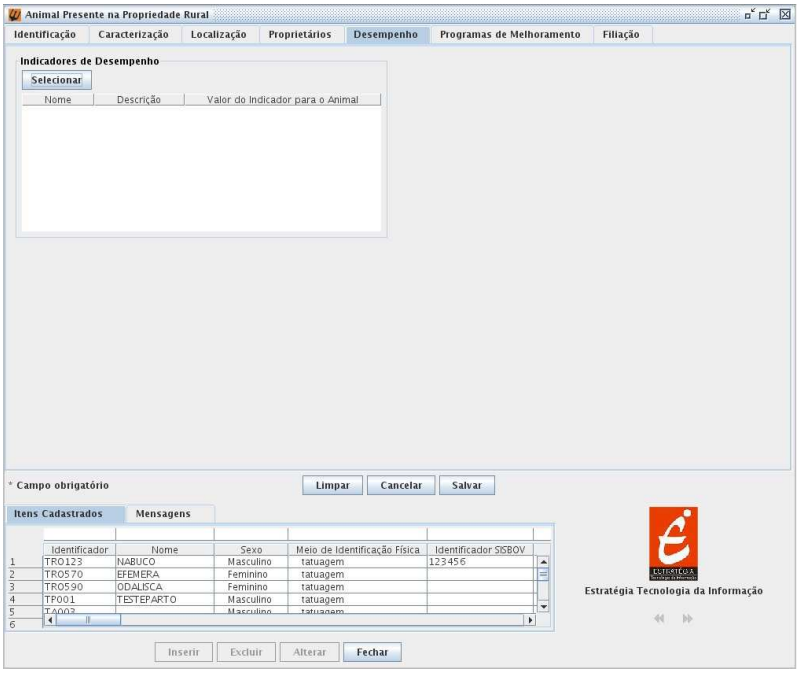

Figura 6.7: *Tela 3, aba 5*

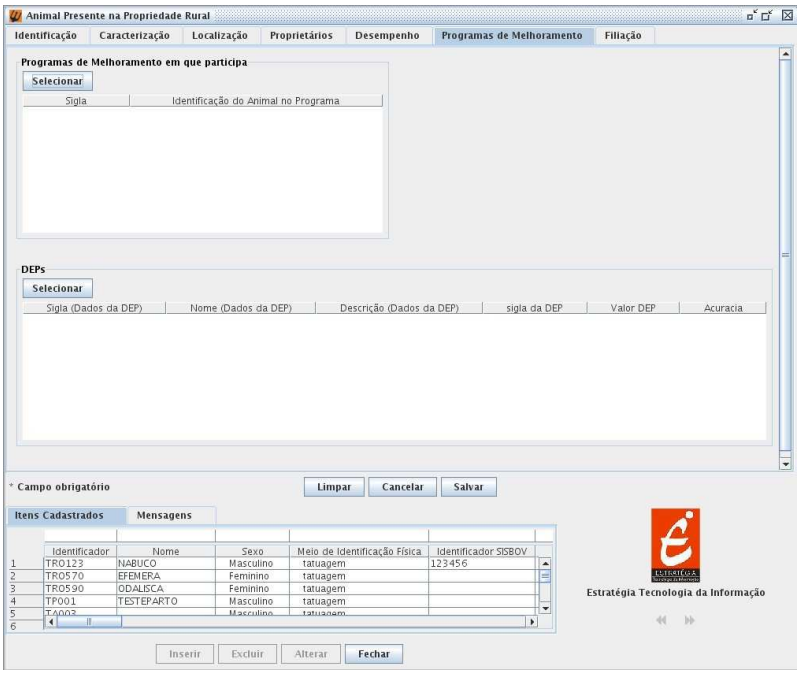

Figura 6.8: *Tela 3, aba 6*

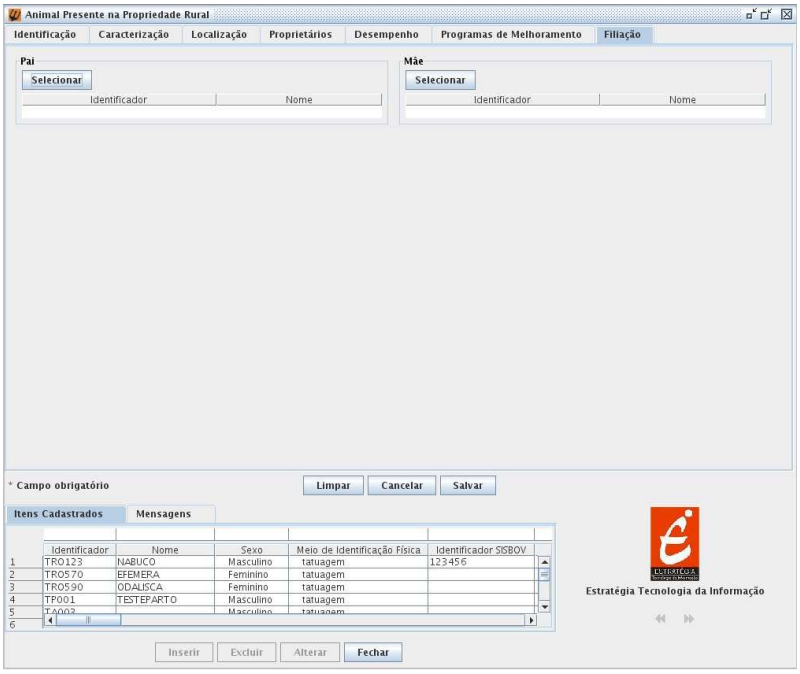

Figura 6.9: *Tela 3, aba 7*

Esta tela tem como objetivo cadastrar e gerenciar animais presentes em uma propriedade rural. Contém, portanto, dados sobre esses animais, como sua identificação única na propriedade, meio de registro físico e fotos. A tela 3, em comparação com as telas 1 e 2, contém um número maior de campos e componentes visuais e comportamentais mais complexos.

A tela 3 pode ser descrita estruturalmente, em relação a visuais e comportamentos, como mostra a Tabela 6.3. Nesta tabela foi inserida a coluna "Aba" para facilitar a localização dos componentes gráficos. Como todos os campos do formulário contêm rótulos, estes não foram indicados na tabela, mas entram na contagem do número de *widgets*.

Tabela 6.3: *Descrição da tela de maior complexidade (tela 3)*

Área da janela	**Aba**	**Componente(s) gráfico(s)**	**Comportamento(s)**	**Tipo(s) de comportamento**
Campos				
1	1	Tabela contendo os seguintes campos: Campo de texto; Campo de texto; Combo Box;	Os dados do Combo box são obtidos do BD.	2, 3, 5
2	1	Combo Box	Os dados do Combo box são obtidos do BD.	2

Continua na próxima página...

Tabela 6.3 – Continuação...

Área da janela	Aba	Componente(s) gráfico(s)	Comportamento(s)	Tipo(s) de comportamento
3	1	Tabela contendo os seguintes campos: Campo de texto; Combo Box;	Os dados do Combo box são obtidos do BD.	2
4	1	Botão Tabela contendo os seguintes campos: Combo Box; Combo Box; Combo Box;	O botão abre uma janela para que o usuário selecione os dados que serão inseridos nos Combo box.	1, 2, 3, 5
5	1	Tabela contendo os seguintes campos: Campo de texto; Campo de texto; Campo de texto; Combo Box;	Os dados do Combo box são obtidos do BD.	2
6	1	Botão Botão; Tabela contendo os seguintes campos: Campo de texto;	O botão “Selecionar” abre uma janela para que o usuário insira ou exclua arquivos representando fotos do animal. O botão “Executar” abre um aplicativo que mostra a foto do animal.	1, 2, 3, 5 1

Continua na próxima página...

Tabela 6.3 – Continuação...

Área da janela	**Aba**	**Componente(s) gráfico(s)**	**Comportamento(s)**	**Tipo(s) de comportamento**
7	2	Campo de texto Botão;	O campo de texto analisa se a informação inserida pelo usuário está de acordo com a máscara de edição. O botão abre um mini-aplicativo de calendário para que o usuário escolha a data.	5 1, 2
8	2	Botão Tabela contendo os seguintes campos: Combo box;	O botão abre uma janela para que o usuário selecione os dados que serão inseridos no Combo box.	1, 2, 3
9	2	Botão Tabela contendo os seguintes campos: Combo box; Combo box;	O botão abre uma janela para que o usuário selecione os dados que serão inseridos nos Combo box.	1, 2, 3
10	2	Botão Tabela contendo os seguintes campos: Combo box; Combo box;	O botão abre uma janela para que o usuário selecione os dados que serão inseridos nos Combo box.	1, 2, 3
11	2	Combo box	Os dados do Combo box são obtidos do BD.	2
12	2	Combo box	Os dados do Combo box são obtidos do BD.	2

Continua na próxima página...

Tabela 6.3 – Continuação...

Área da janela	**Aba**	**Componente(s) gráfico(s)**	**Comportamento(s)**	**Tipo(s) de comportamento**
13	2	Combo box	Os dados do Combo box são obtidos do BD.	2
14	2	Botão Tabela contendo os seguintes campos: Campo de texto;	O botão abre uma janela para que o usuário selecione os dados que serão inseridos no campo de texto.	1, 2, 3
15	3	Botão Tabela contendo os seguintes campos: Campo de texto;	O botão abre uma janela para que o usuário selecione os dados que serão inseridos no campo de texto.	1, 2, 3
16	4	Botão Tabela contendo os seguintes campos: Combo box;	O botão abre uma janela para que o usuário selecione os dados que serão inseridos no Combo box.	1, 2, 3
17	4	Botão Tabela contendo os seguintes campos: Campo de texto; Combo box; Campo de texto;	O botão abre uma janela para que o usuário selecione os dados que serão inseridos nos campos de texto e no combo box.	1, 2, 3
18	4	Botão	O botão abre uma janela para que o usuário selecione os dados que serão inseridos no Combo box.	1, 2, 3

Continua na próxima página...

Tabela 6.3 – Continuação...

Área da janela	Aba	Componente(s) gráfico(s)	Comportamento(s)	Tipo(s) de comportamento
		Tabela contendo os seguintes campos: Combo box;		
19	4	Botão Tabela contendo os seguintes campos: Campo de texto; Combo box; Campo de texto;	O botão abre uma janela para que o usuário selecione os dados que serão inseridos nos campos de texto e no combo box.	1, 2, 3
20	5	Botão Tabela contendo os seguintes campos: Campo de texto; Campo de texto; Campo de texto;	O botão abre uma janela para que o usuário selecione os dados que serão inseridos nos campos de texto.	1, 2, 3
21	6	Botão Tabela contendo os seguintes campos: Combo box; Campo de texto;	O botão abre uma janela para que o usuário selecione os dados que serão inseridos no campo de texto e no combo box.	1, 2, 3
22	6	Botão	O botão abre uma janela para que o usuário selecione os dados que serão inseridos nos campos de texto.	1, 2, 3

Continua na próxima página...

Tabela 6.3 – Continuação...

Área da janela	**Aba**	**Componente(s) gráfico(s)**	**Comportamento(s)**	**Tipo(s) de comportamento**
		Tabela contendo os seguintes campos: Campo de texto; Campo de texto; Campo de texto; Campo de texto; Campo de texto; Campo de texto;		
23	7	Botão Tabela contendo os seguintes campos: Campo de texto; Campo de texto;	O botão abre uma janela para que o usuário selecione os dados que serão inseridos nos campos de texto.	1, 2, 3
24	7	Botão Tabela contendo os seguintes campos: Campo de texto; Campo de texto;	O botão abre uma janela para que o usuário selecione os dados que serão inseridos nos campos de texto.	1, 2, 3
Ações				
		Rótulo	Não há	
		Botão Limpar	Limpar campos do formulário	2, 4
		Botão Cancelar	Cancelar operação CRUD	4
		Botão Salvar	Salvar operação CRUD	4, 5
Tabela				
		24 campos de filtro	Filtrar os dados da tabela	3
		24 Cabeçalhos da tabela (um para cada coluna)	Executar a ordenação dos dados da coluna	3

Continua na próxima página...

Tabela 6.3 – Continuação...

Área da janela	**Aba**	**Componente(s) gráfico(s)**	**Comportamento(s)**	**Tipo(s) de comportamento**
		24 Campos da tabela (um para cada coluna)	Não há	
		Número de linhas da tabela	Seleciona a linha correspondente na tabela e ativa/desativa botões de ações CRUD	3, 4
		Scroll vertical	Rola a tabela verticalmente	3
		Scroll horizontal	Rola a tabela horizontalmente	3
Ações CRUD				
		Botão Inserir	Ativa os widgets do formulário e ativa os botões de ação do formulário	2, 4, 5
		Botão Excluir	Envia os dados para o formulário e ativa os botões de ação do formulário	2, 4
		Botão Alterar	Idem ao botão Inserir	
		Botão Fechar	Fecha a janela	1

A diferença de tamanho entre as telas 1, 2 e 3 pode ser obtida comparando-se os dados das tabelas 6.1, 6.2 e 6.3. A tela 3 contém 173 *widgets*, sendo quase 5 vezes maior que a da tela 1 e quase 3 vezes maior que a tela 2.

6.2 Coleta de Dados

Os objetivos principais da coleta de dados, que durou em torno de 160 dias, foram medir o tempo de desenvolvimento e verificar a diferença de produtividade entre três tipos de geração de interfaces gráficas: manual (digitação do código-fonte), visual (com auxílio de editor de *widgets*) e através da ferramenta proposta neste trabalho, o GDIG.

Após obter o tempo de geração para cada desenvolvedor, calculou-se uma média de tempo para cada forma de geração. O tempo considerado foi aquele em que o desenvolvedor esteve efetivamente programando ou realizando algum estudo sobre a

forma de programar a tela. Nos intervalos em que essas atividades não eram realizadas o cronômetro era paralisado, como os intervalos para descanso, por exemplo.

A Tabela 6.4 mostra os dados coletados para cada tipo de geração executada pelos desenvolvedores (D1 a D4).

Tabela 6.4: *Coleta dos tempos de desenvolvimento da tela de menor complexidade*

Tipo de Geração	Tempo (horas)				Média	Produtividade
	D1	D2	D3	D4		
Manual	15,5 h	15,2 h	15 h	15,1 h	15,2 h	0,42 h/*widget*
Editor Visual	7,5 h	7,4 h	7,1 h	7,2 h	7,3 h	0,20 h/*widget*
GDIG	1,3 h	1,2 h	1,1 h	1,2 h	1,2 h	0,03 h/*widget*

Para calcular a produtividade de cada tipo de geração, definiu-se primeiramente a média aritmética de cada um, conforme mostra a Tabela 6.4. Calculou-se então a produtividade média no desenvolvimento em horas/*widget*:

Nas execuções da geração manual e com auxílio do editor visual, os passos em sequência foram agrupados de acordo com as diferentes áreas da GUI (Figura 5.15):

1. Janela
   (a) Criação da janela;
   (b) Ajuste do tamanho da janela;
   (c) Ajuste do tamanho e leiaute do painel da janela;
2. Área de campos do formulário
   (a) Criação da aba de formulário;
   (b) Ajuste do tamanho da aba de formulário;
   (c) Criação dos painéis contendo (rótulo com os respectivos componentes gráficos);
   (d) Ajuste do tamanho dos painéis;
   (e) Criação dos componentes gráficos dos painéis;
   (f) Ajuste do tamanho dos componentes gráficos dos painéis;
   (g) Inserção dos componentes gráficos nos painéis;
   (h) Inserção dos painéis no painel da aba;
3. Área de ações referentes ao formulário
   (a) Criação do painel central de botões

(b) Ajuste do seu tamanho;

(c) Criação de seus componentes gráficos;

(d) Inserção de seus componentes gráficos;

4. Área de elementos já cadastrados

(a) Criação da aba inferior de tabelas;

(b) Ajuste do tamanho da aba inferior das tabelas;

(c) Criação do painel das tabelas;

(d) Ajuste do tamanho do painel das tabelas;

(e) Criação da tabela de filtro;

(f) Ajuste do tamanho da tabela de filtro;

(g) Criação do cabeçalho da tabela de dados;

(h) Ajuste do tamanho do cabeçalho da tabela de dados;

(i) Criação da tabela de linhas;

(j) Ajuste do tamanho da tabela de linhas;

(k) Criação da tabela de dados;

(l) Ajuste do tamanho da tabela de dados;

(m) Criação da barra de rolagem vertical;

(n) Ajuste do tamanho da barra de rolagem vertical;

(o) Criação da barra de rolagem horizontal;

(p) Ajuste do tamanho da barra de rolagem horizontal;

5. Área de ações CRUD e referentes à janela (ação fechar)

(a) Criação do painel de botões inferior;

(b) Ajuste do seu tamanho;

(c) Criação de seus componentes gráficos;

(d) Inserção de seus componentes gráficos;

Na geração da GUI pelo GDIG, foram seguidos os seguintes passos em sequência:

1. Definição do meta-modelo

(a) Cadastro de uma nova entidade em Enumerado de Tipo de Entidade;

(b) Criação de um novo tipo de entidade em Tipo de Entidade;

(c) Criação de aba de dados;

(d) Criação de cada campo, com seu tipo específico;

2. Implantação no SI, incluindo o controle de acesso

(a) Criação do Caso de Uso referente à entidade;

(b) Criação do item de menu referente ao caso de uso;

(c) Autorização do caso de uso para os usuários;

(d) Autorização do caso de uso ao módulo específico;

Para cada tela (1, 2, e 3), esses mesmos passos foram seguidos. Nota-se, nesse momento, um dos motivos da diferença do tempo consumido em cada tipo de geração. A quantidade de tarefas a serem realizadas na codificação manual e com auxílio de editor visual é consideravelmente maior do que na geração com o GDIG.

A Tabela 6.5 traz os dados coletados para cada tipo de geração de tela de complexidade média.

Tabela 6.5: *Coleta dos tempos de desenvolvimento da tela de média complexidade*

Tipo de Geração	Tempo (horas)				Média	Produtividade
	D1	D2	D3	D4		
Manual	31,5 h	32,3 h	33,9 h	33,1 h	32,7 h	0,5 h/*widget*
Editor Visual	14,1 h	15 h	14,7 h	14,2 h	14,5 h	0,22 h/*widget*
GDIG	1,1 h	1,04 h	1,08 h	1,02 h	1,06 h	0,02 h/*widget*

Para calcular a produtividade de cada tipo de geração, definiu-se primeiramente a média aritmética de cada um. Calculou-se então a produtividade média no desenvolvimento em horas/*widget*.

A Tabela 6.6 traz os dados coletados para cada tipo de geração executada para a tela complexa.

Tabela 6.6: *Coleta dos tempos de desenvolvimento da tela de maior complexidade*

Tipo de Geração	Tempo (horas)				Média	Produtividade
	D1	D2	D3	D4		
Manual	70,1 h	72,5 h	70,8 h	71,4 h	71,2 h	0,41 h/*widget*
Editor Visual	33 h	35,1 h	32,5 h	33 h	33,4 h	0,19 h/*widget*
GDIG	1,9 h	1,5 h	1,6 h	1,8 h	1,7 h	0,01 h/*widget*

Para calcular a produtividade de cada tipo de geração, definiu-se primeiramente a média aritmética de cada um. Calculou-se então a produtividade média no desenvolvimento em horas/*widget*.

6.3 Análise dos resultados

A tabela 6.7 resume e compara os dados coletados. Pelos dados obtidos observa-se que, em relação ao **tempo de desenvolvimento**, a geração da interface gráfica pela ferramenta GDIG foi quase 13 vezes mais rápida na tela simples (tela 1); 30 vezes mais rápida na tela média (tela 2); e quase 42 vezes mais rápida na tela complexa (tela 3). Ou seja, quanto maior o tamanho e complexidade da tela gerada, maior o ganho de tempo para o desenvolvimento com o GDIG. Isso porque, à medida que cresce a complexidade da GUI, o trabalho do projetista resume-se em ter mais atributos para criar e ajustar. O aumento do número de atributos não influencia o trabalho de criação, que é praticamente o mesmo, não importando se um campo é mais complexo que outro.

Tabela 6.7: *Resumo dos dados coletados*

	Tamanho		
	Baixa (1)	**Média (2)**	**Alta (3)**
Widgets	36	66	173
Tempo Manual	15,2 h	32,7 h	71,2 h
Tempo Editor Visual	7,3 h	14,5 h	33,4 h
Tempo GDIG	1,2 h	1,06 h	1,7 h
Prod. Manual	0,42 h/*widget*	0,50 h/*widget*	0,41 h/*widget*
Prod. Editor Visual	0,20 h/*widget*	0,22 h/*widget*	0,19 h/*widget*
Prod. GDIG	0,03 h/*widget*	0,02 h/*widget*	0,01 h/*widget*

Pelo lado da **produtividade**, o GDIG demonstrou uma produtividade 14 vezes maior na tela 1; 25 vezes maior na tela 2 e 41 vezes maior na tela 3. O GDIG encapsula o trabalho de criação dos componentes gráficos. Isso não acontece nos outros dois tipos de desenvolvimento onde, quanto mais complexa a interface, mais trabalho terá o projetista para criar e organizar os *widgets* dentro da janela.

Nesta experiência, o número de telas geradas e o número de desenvolvedores envolvidos é pequeno para a obtenção de um conjunto de dados suficientes para uma conclusão definitiva. Devido à limitação de tempo e recursos humanos, não pôde-se obter uma amostragem maior. Porém, os dados coletados permitem uma comparação realística da melhoria que o GDIG pode proporcionar frente aos outros métodos de geração de GUIs.

CAPÍTULO 7

Conclusões

Este capítulo apresenta as conclusões finais sobre o trabalho realizado no presente projeto. A Seção 7.4 destaca as considerações finais, a Seção 7.2 analisa trabalhos correlatos ao tema abordado, a Seção 7.3 discute as contribuições e limitações da abordagem proposta, e a Seção 7.5 mostra algumas oportunidades para extensões desta proposta.

7.1 Características das GUIs geradas

As GUIs geradas pelo GDIG têm como principais características:

- **Usabilidade:** uma das principais características das janelas geradas pelo GDIG é que elas têm visuais e comportamentos padronizados. Como todas as janelas são criadas a partir de um mesmo gerador, o mecanismo de geração insere as mesmas características em todas as janelas. Esse fator diminui o tempo de aprendizagem de uso do sistema pelo usuário final. Além disso, deixa esse mesmo usuário mais confiante e torna seu desempenho para utilizar o sistema melhor, visto que a próxima janela com a qual ele irá interagir tem o mesmo padrão de qualquer janela que ele tenha utilizado anteriormente. Essa consistência de uso é uma das principais características de qualidade em uma interface gráfica com usuário [35, 9, 16, 39].
- **Manutenibilidade:** uma vez que o projetista trabalha com modelos ao invés de código-fonte, a modificação de qualquer característica das janelas mostra-se mais simples e eficiente, visto que os modelos trazem informações mais abstratas e de mais fácil entendimento. Um exemplo simples seria o fato de o projetista desejar alinhar o título de todas as janelas para a direita ao invés de centralizá-lo. Dessa forma basta ajustar o gerador para que todas as janelas geradas já contenham seu título alinhado corretamente.
- **Portabilidade:** como as janelas são descritas através de modelos abstratos e independentes de tecnologia, estes podem ser aproveitados em outros SI ou para gerar GUIs em outras plataformas, como dispositivos móveis ou *Web*. Além disso, o pró-

prio GDIG pode ser executado em vários tipos de Sistemas Operacionais, como Windows e Linux, pois foi implementado em Java.

- **Confiabilidade:** outra característica relevante do mecanismo de GUI proposto é o sistema de controle de acesso. Quando um modelo referente a uma interface é criada, ela não aparece automaticamente para o usuário no menu do *framework*. Antes disso o projetista deve informar os papéis dos usuários do sistema que podem acessar a interface e, dentro de cada janela, as operações CRUD permitidas.
- **Internacionalização:** o mecanismo de geração de GUI foi preparado para gerar janelas que podem ser apresentadas em quatro idiomas: português, inglês, francês e espanhol. Desta forma consegue-se superar as limitações de idioma em SI que são utilizados em vários países.
- **Ajuda sensível ao contexto:** quando o usuário tenta salvar alguma ação CRUD, como Inserção ou Alteração, e ocorre um erro de validação como, por exemplo, um campo contém um valor inválido, o sistema avisa esse erro ao usuário e disponibiliza um *link* para que, quando clicado, o GDIG posicione o foco no campo que gerou o erro. A Figura 7.1 mostra o que acontece quando o usuário tenta salvar uma *Pessoa Física* e não insere dados nos campos obrigatórios *Nome* e *Identificação*. O GIDG informa o erro ao usuário na aba *Mensagens* e modifica para vermelho a cor do(s) rótulo(s) do(s) campo(s) em que falta a informação. Além disso, o GIDG disponibiliza, logo após a mensagem de erro, um *link* que, quando clicado, modifica a cor para laranja do campo correspondente, servindo de atalho para o acesso ao mesmo, mesmo que ele encontre-se em outra aba.

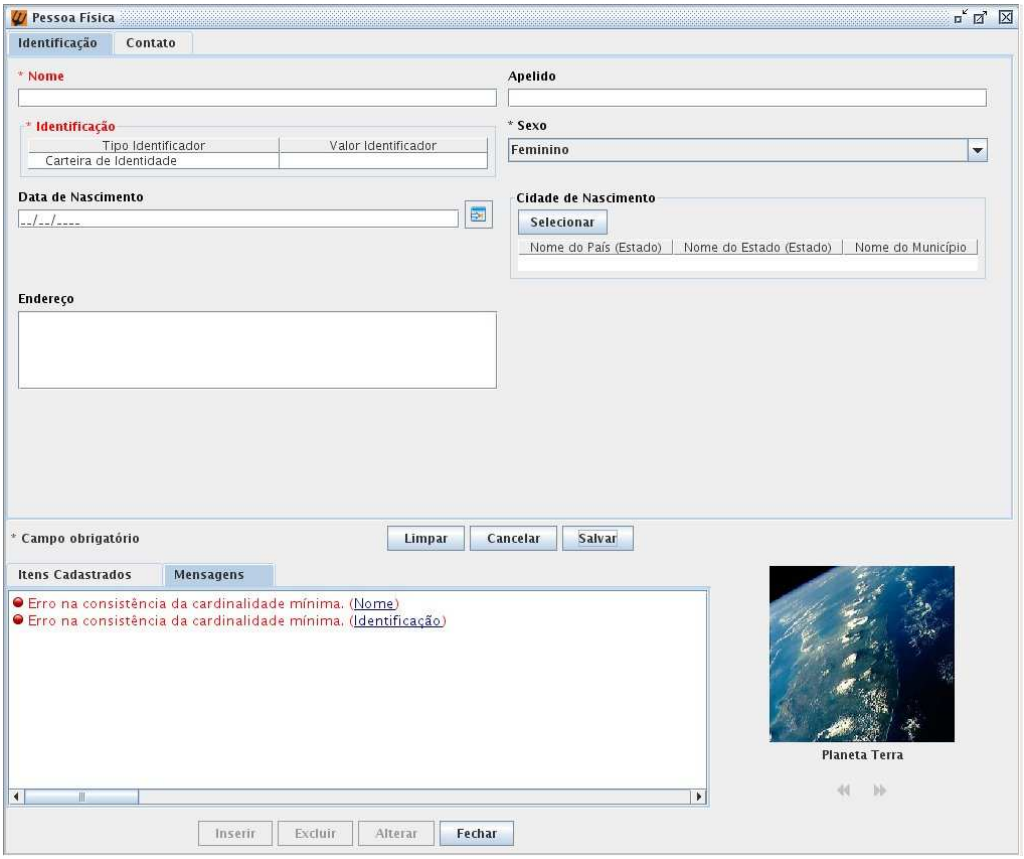

Figura 7.1: *Helper sensível ao contexto*

- **Consistência de dados:** quando o usuário salva alguma operação de inserção ou alteração, antes de as informações serem enviadas para persistência na base de dados, são feitas algumas verificações de consistência dos dados. As primeiras delas são realizadas pela própria GUI. O GDIG percorre cada componente gráfico do formulário e faz com que este execute a verificação de suas próprias informações. Por exemplo, um componente cujo domínio seja *INTEIRO* não poderá conter Strings ou decimais em seu campo de valor. Outras verificações são executadas, como de cardinalidade mínima e máxima.

  Caso alguma inconsistência seja encontrada nessa fase de verificação, o GDIG mostra uma mensagem ao usuário, informa qual o componente gráfico deve ter seu valor corrigido e não permite que as informações sejam repassadas às camadas inferiores da arquitetura até que todas encontrem-se devidamente consistentes.

  Após as verificações de consistência na camada de interface, as informações passam por regras de validação nas camadas inferiores da arquitetura, antes de serem persistidas na base de dados. O mecanismo dessas regras de validação estão fora do escopo deste trabalho, mas caso alguma informação não esteja de acordo com alguma dessas regras, o GDIG recebe um aviso de erro de validação vindo do mecanismo de regras, mostra uma mensagem que oriente o usuário a corrigi-lo

e informa o campo onde esta inconsistência está acontecendo.

- **Valor *default*:** cada campo do formulário pode ter um valor padrão (*default*) definido. Este valor será mostrado quando a GUI for apresentada inicialmente ao usuário, que poderá ou não modificá-lo. Isso permite que campos cujos valores sejam frequentemente os mesmos possam ser ajustados com um valor *default* que raramente será modificado, poupando trabalho ao usuário.
- **Organização da interface:** o leiaute dos componentes gráficos dentro do formulário é padronizado. São primeiramente organizados em abas para agrupar informações de acordo com os conceitos do negócio. Dentro de cada aba, são ajustados em N linhas e 2 colunas. Caso a altura do formulário ultrapasse o limite máximo, é criada uma barra de rolagem vertical.
  Essa padronização na organização e aparência dos componentes gráficos aumenta a facilidade do usuário em trabalhar com GUIs diferentes, já que todas têm aparências semelhantes. Isso melhora sensivelmente a usabilidade do SI.
- **Áreas de Mensagens:** as telas geradas pelo GDIG são principalmente para uso em Sistemas de Informação, onde a divulgação de mensagens, incluindo produtos, e serviços, é fortemente explorada. Nesse sentido, a Janela Principal contém três áreas que podem ser usadas para *marketing*, como mostra a Figura 7.2.

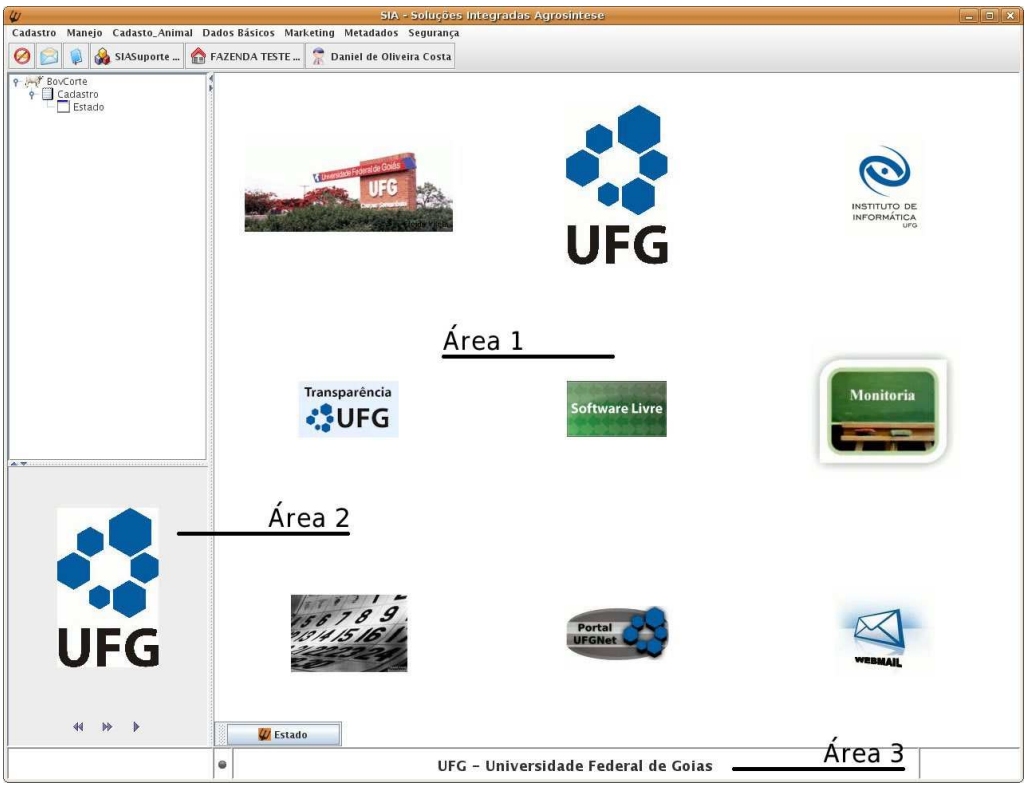

Figura 7.2: *Áreas para mensagem usadas em marketing*

A Área 1 é o pano de fundo da Área Funcional. Quando não existe janela aberta, ou se todas as janelas encontram-se minimizadas, o usuário visualiza essa área que é divida em uma matriz NxN, contendo a figura da mensagem. Ao clicar com o botão direito, o usuário pode ter acesso a outros meios de comunicação, como site e vídeo, como exemplifica a Figura 7.3. A Área 2 contém um *slide show* de imagens, onde são mostradas mensagens em sequência. Ao mostrar a última, volta-se novamente à primeira, simulando uma lista circular. Assim como na Área 1, caso o usuário clique com o botão direito terá acesso a outros meios de interação. A Área 3 traz mensagens de rodapé, em forma de texto curto.

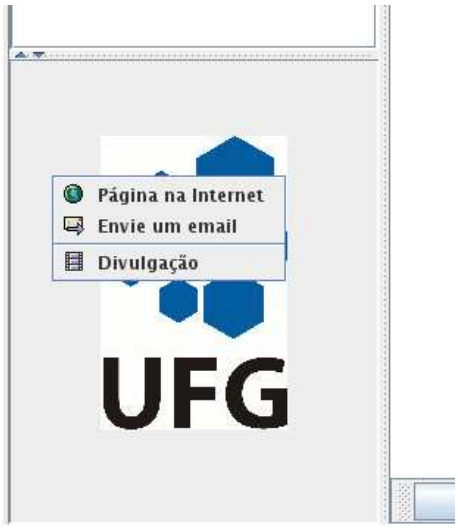

Figura 7.3: *Acesso a outros meios de divulgação*

- **Padrão *Observer*:** além das aparências, o GDIG implementa comportamentos que são padrões para as GUIs criadas. O padrão de projeto *Observer* executa um mecanismo que ajusta o comportamento de componentes gráficos de acordo com a mudança de valor de outros componentes.
  Por exemplo, o campo de texto *Telefone* torna-se não-editável caso o usuário indique em um *Check Box* que não contém número de telefone para contato, ou torna-se editável caso indique o contrário. Neste caso, o campo de texto *Telefone* observa as modificações ocorridas no *Check Box*. Assim que o *Check Box* modifica seu valor, ele avisa a todos os seus observadores (ele pode ter mais de um Observador) e, desse modo, cada Observador executa sua ação específica.
  Nesse exemplo, o campo de texto *Telefone* é o Observador (*Observer*) e o *Check Box* é o Sujeito (*Subject*). Mas esse mesmo campo de texto pode ser sujeito de outro(s) observador(es) ou o *Check Box* pode ser Observador de outro Sujeito. Dentro de uma GUI um componente gráfico pode ser Observador e Sujeito ao mesmo tempo, não há restrições quanto a isso.

Os tipos de observadores diferem-se pelo comportamento que os mesmos implementam como, por exemplo, tornar-se não editáveis, modificar seus valores, iniciar um programa, dentre outros.

7.2 Trabalhos Correlatos

Várias abordagens para geração de GUIs foram propostas na literatura. Discutiremos nessa seção os aspectos mais relevantes dessas abordagens e suas diferenças com a proposta deste trabalho.

As ferramentas de geração de GUIs podem ser divididas, de acordo com [10], em quatro categorias: as baseadas em linguagens, as ferramentas interativas, os *frameworks* de aplicações e os geradores automáticos baseados em modelos (a Seção 2.5 traz mais detalhes). Nesse aspecto, a abordagem deste trabalho foi implementada como um gerador, assim como [3, 21, 26] e diferente de [34, 29, 46, 20, 45, 18, 19, 47] que foram implementados como *frameworks*.

Além da geração da interface, as ferramentas disponibilizam meios de edição dos modelos. O mais comum é o meio textual [20, 45, 18, 19, 47, 26, 21, 3], onde arquivos de texto ou arquivos XML, com sintaxe bem definida, são usados para gerenciar as informações dos modelos. Existem também ferramentas que utilizam editores gráficos [46], ou formulários [34]. O GDIG utiliza, para o gerenciamento dos modelos, editores do tipo formulário. Isso torna a tarefa de gerenciar os metamodelos de GUIs mais simples e agradável em relação aos editores textuais.

Como as interfaces geradas do tipo cadastro visam à execução de operações em Banco de Dados (inserção, alteração, exclusão e consulta), uma característica importante é a cardinalidade das associações entre entidades dos modelos que as ferramentas suportam. Das ferramentas analisadas, a maioria não informa a cardinalidade dessas associações, exceto as [46, 45, 19], que suportam associações com cardinalidade 1 para 1, 1 para N e N para N. O GDIG, assim como essas ferramentas, suporta os três tipos de associações. Dessa forma, trabalha com modelos mais complexos e tem condições de fornecer interfaces mais completas na aparência e no comportamento.

7.3 Contribuições e Limitações

A abordagem proposta neste trabalho contribui para reduzir um problema chave da construção de GUIs: a ineficiência, pois a equipe de Engenharia de Software passa a trabalhar com modelos, não se preocupando com código-fonte ou tecnologias relacionadas ao *look and feel* da interface.

O fator da eficiência é citado em [19], indicando que a principal vantagem de se desenvolver aplicações usando mecanismos dinâmicos de geração de interface é a grande redução no tempo de desenvolvimento, que pode alcançar a casa de redução em uma ordem de grandeza, se comparado aos métodos tradicionais de desenvolvimento.

A abordagem proposta mostrou-se 42 vezes mais eficiente em relação à codificação manual de interfaces e quase 20 vezes em relação à geração auxiliada por editores visuais (veja o Capítulo 6 para mais detalhes), corroborando a afirmação de [19].

Além da eficiência no desenvolvimento de software, outras contribuições são verificadas como:

1. redução do volume de código-fonte.
2. diminuição da complexidade na geração das interfaces e, como consequência, do SI como um todo.
3. maior facilidade para manutenção das interfaces.
4. geração de visuais e comportamentos padronizados, fazendo com que a interface tenha melhor usabilidade.
5. maior influência das partes interessadas no decorrer do projeto da interface, pois as mudanças têm um menor custo.
6. reutilização dos modelos em outros SI ou em módulos do mesmo sistema.

Pode-se também citar como contribuições a revisão sistemática descrita no Apêndice A e a metodologia de medição da complexidade de GUIs detalhada no Capítulo 6.

A abordagem de geração automática de GUI baseada em modelos proposta neste trabalho contém também algumas limitações conhecidas:

- **Dificuldade de gerar aparência ou comportamento para um usuário específico:** a criação das telas contempla a geração de visuais e comportamentos suportados pelo GDIG, ou seja, somente aqueles que o gerador é capaz de criar são implementados. Deste modo, caso algum usuário queira personalizar algum requisito de uma ou mais telas, o GDIG não será capaz de atendê-lo.
  Por exemplo, pode ocorrer uma situação onde o usuário queira modificar a aparência de todas as telas, onde ele queira ver as telas em uma organização diferente. Neste caso o GDIG não estaria preparado para atendê-lo de imediato. Seriam executadas mudanças para implementar esse requisito, e isso consumiria um tempo maior do que se a geração estivesse sendo feita de forma estática (sem o uso de modelos).

Podemos ter também o caso onde o usuário queira modificar somente uma ou poucas telas em um conjunto de várias. Neste caso, como as mudanças seriam pontuais (somente em algumas telas), deveriam ser criadas telas específicas que contemplassem essas mudanças, enquanto as demais telas continuariam sendo geradas pelo mecanismo baseado em modelos.

- **Perda de desempenho na criação dinâmica da GUI:** teoricamente, a partir do momento em que o usuário aciona um menu para a criação da tela, o GDIG demora mais para gerá-la, se comparado à geração estática, onde o código da tela encontra-se pré-compilado. Isso porque o GDIG busca os metadados, verifica suas informações, escolhe os componentes gráficos e organiza-os dentro da tela de cadastro. Entretanto, na prática, essa diferença é imperceptível, tanto para telas pequenas quanto para aquelas que contêm um número grande de campos.

7.4 Considerações Finais

Este trabalho apresenta uma abordagem dirigida por modelos para criação e gerência de Interface Homem-Computador para Sistemas de Informação. Para tanto, foi projetado e implementado um gerador dinâmico de GUI denominado GDIG (Gerador Dinâmico de Interfaces Gráficas).

A abordagem proposta nesse trabalho consiste, primeiramente, na criação de um modelo independente de plataformas ou tecnologias, denominado neste trabalho como Modelo de Negócio (MN). Através de regras de mapeamento predefinidas dentro do mecanismo do GDIG, o MN é transformado em um Modelo de Apresentação (MA). Este, por sua vez, contém informações específicas e direcionadas para a interface gráfica. MN e MA são armazenados em um Banco de Dados gerenciado pelo GDIG. Cada MA refere-se a uma GUI dentro do SI.

Quando o usuário informa ao sistema que deseja abrir alguma janela como, por exemplo, selecionando em um item de menu ou um botão, o GDIG recebe a solicitação, busca os metadados referentes à janela solicitada, aplica ao MA mais um conjunto de regras predefinidas, define os componentes gráficos a serem criados e monta a GUI. Nenhum código-fonte referente à GUI é armazenado. As únicas informações permanentes são aquelas contidas no MA.

As interfaces construídas são baseadas no paradigma *Desktop*, do tipo CRUD. O mecanismo foi implementado na linguagem Java, utilizando componentes gráficos do pacote *Swing*.

Como característica diferenciada das demais abordagens dirigidas por modelos, o GDIG proporciona o fato de os metadados não dependerem de qualquer tecnologia ou linguagem, como no caso de aplicações que utilizam *Annotations* [43] para esse fim.

Outra diferenciação do GDIG é a capacidade de geração e de gerenciamento dos modelos. Essas atividades ocorrem no mesmo ambiente da aplicação, e constitui em um processo recursivo, já que os próprios modelos são utilizados para gerar as telas que permitem criar e gerenciar os modelos.

Como especificidade adicional da abordagem aqui proposta, tem-se a geração das GUIs em tempo de execução e sem criação de código-fonte. As abordagens já existentes geram o código-fonte a partir de modelos, compilam-no e armazenam o código para uma futura criação da GUI.

Porém, essa característica tem um grande problema para a abordagem dirigida por modelos: a modificação do código gerado impede que o mecanismo de geração e execução o reconheça novamente. Ou seja, quando o usuário modifica o código criado, os padrões de codificação são desfeitos e o mecanismo não mais o reconhece.

Para evitar esse problema, o GDIG trabalha de forma diferente. Não há criação de código-fonte e as GUIs são geradas a partir dos modelos quando solicitadas, em tempo de execução. Não há, portanto, risco de incompatibilidade entre modelos e código.

7.5 Trabalhos Futuros

Apesar dos avanços e resultados obtidos, a proposta aqui apresentada permite várias extensões e melhorias. Pode-se implementar, por exemplo, a geração automática de GUI para outras plataformas alvo como *Web* e *Palm*, estendendo os mecanismos de transformação do GDIG.

Outra melhoria que pode ser implementada é a definição de meios de comunicação e intercâmbio de dados com outras ferramentas de modelagem de SI para importação e exportação de modelos.

Pode-se também gerar mecanismos inteligentes de organização dos componentes gráficos de acordo com o perfil do usuário. Ou seja, o leiaute dos componentes gráficos deixaria de ser fixo para se tornar dinâmico, ajustando-se de acordo com necessidades ou preferências definidas pelo usuário.

Os modelos de Negócio e Apresentação, apesar de conceitualmente separados, são editados na mesma GUI. Uma mudança proposta seria a separação da edição destes modelos para que o mecanismo da abordagem de criação de GUIs torne-se mais claro.

Outro trabalho futuro proposto é a melhoria da coleta de dados e medição de interfaces gráficas, de modo a transformar o trabalho dos experimentos em métricas padronizadas, confiáveis, formais e mais tecnicamente embasadas para a definição da complexidade de interfaces gráficas.

Referências Bibliográficas

[1] ANNEKE KLEPPE, JOS WARMER, W. B. **MDA Explained: The Model Driven Architecture: Practice and Promise**. Addison Wesley, 1 edition, April 2003.

[2] BASIN, D; DOSER, J; LODDERSTEDT, T. **Model driven security: From uml models to access control infrastructures**. ACM Trans. Softw. Eng. Methodol., 15(1):39–91, 2006.

[3] BENDSEN, P. **Model driven business ui based on maps**. In: SIGMOD '04: PROCEEDINGS OF THE 2004 ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, p. 887–891, New York, NY, USA, 2004. ACM.

[4] BERSTEL, J; REGHIZZI, S. C; ROUSSEL, G; PIETRO, P. S. **A scalable formal method for design and automatic checking of user interfaces**. ACM Trans. Softw. Eng. Methodol., 14(2):124–167, 2005.

[5] BETTIN, J; VAN EMDE BOAS, G; WILLINK, E. **Generative model transformer: an open source mda tool initiative**. In: OOPSLA '03: COMPANION OF THE 18TH ANNUAL ACM SIGPLAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, SYSTEMS, LANGUAGES, AND APPLICATIONS, p. 294–295, New York, NY, USA, 2003. ACM.

[6] BORDIN, M; VARDANEGA, T. **Real-time java from an automated code generation perspective**. In: JTRES '07: PROCEEDINGS OF THE 5TH INTERNATIONAL WORKSHOP ON JAVA TECHNOLOGIES FOR REAL-TIME AND EMBEDDED SYSTEMS, p. 63–72, New York, NY, USA, 2007. ACM.

[7] BRAD MYERS, SCOTT E. HUDSON, R. P. **Past, present, and future of user interface software tools**. ACM Transactions on Computer-Human Interaction, 7:3–28, March 2000.

[8] BULL, R. I. **Integrating dynamic views using model driven development**. In: CASCON '06: PROCEEDINGS OF THE 2006 CONFERENCE OF THE CENTER FOR ADVANCED STUDIES ON COLLABORATIVE RESEARCH, p. 17, New York, NY, USA, 2006. ACM.

[9] BUTOW, E. **User Interface Design for Mere Mortals**. Addison-Wesley Professional, 1 edition, May 2007.

[10] COLLIGNON, B; VANDERDONCKT, J; CALVARY, G. **An intelligent editor for multi-presentation user interfaces**. In: SAC '08: PROCEEDINGS OF THE 2008 ACM SYMPOSIUM ON APPLIED COMPUTING, p. 1634–1641, New York, NY, USA, 2008. ACM.

[11] CORE TEAM, R. **Rails - web development that doesn't hurt**. http://rubyonrails.org/ , Acesso em: 05 de Junho de 2009, 2009.

[12] CRYSTAL, A. **Interface design for metadata creation**. In: CHI '03: CHI '03 EXTENDED ABSTRACTS ON HUMAN FACTORS IN COMPUTING SYSTEMS, p. 1038–1039, New York, NY, USA, 2003. ACM.

[13] DEANTONI, J; BABAU, J. **A mda approach for systems dedicated to process control**. Embedded and Real-Time Computing Systems and Applications, 2005. Proceedings. 11th IEEE International Conference on, p. 567–570, Aug. 2005.

[14] DEARLE, A. **Software deployment, past, present and future**. In: FOSE '07: 2007 FUTURE OF SOFTWARE ENGINEERING, p. 269–284, Washington, DC, USA, 2007. IEEE Computer Society.

[15] DRAHEIM, D; LUTTEROTH, C; WEBER, G. **Graphical user interfaces as documents**. In: CHINZ '06: PROCEEDINGS OF THE 7TH ACM SIGCHI NEW ZEALAND CHAPTER'S INTERNATIONAL CONFERENCE ON COMPUTER-HUMAN INTERACTION, p. 67–74, New York, NY, USA, 2006. ACM.

[16] GALITZ, W. O. **The Essential Guide to User Interface Design-An Introduction to GUI Design Principles and Techniques**. John Wiley and Sons, 2002.

[17] GAMMA, E; HELM, R; JOHNSON, R; VLISSIDES, J. M. **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison-Wesley, 1998.

[18] GARULLI, L. **Roma framework**. http://www.romaframework.org/ , Acesso em: 15 de Outubro de 2008.

[19] GROUP, J. **Jmatter**. http://jmatter.org/ , Acesso em: 10 de Outubro de 2008.

[20] GROUP, N. O. **Naked objects**. http://www.nakedobjects.org/home/index.shtml , Acesso em: 20 de Outubro de 2008.

[21] KAVALDJIAN, S. **A model-driven approach to generating user interfaces**. In: ESEC-FSE '07: PROCEEDINGS OF THE THE 6TH JOINT MEETING OF THE EUROPEAN SOFTWARE ENGINEERING CONFERENCE AND THE ACM SIGSOFT SYMPOSIUM ON THE FOUNDATIONS OF SOFTWARE ENGINEERING, p. 603–606, New York, NY, USA, 2007. ACM.

[22] KETFI, A; BELKHATIR, N. **Model-driven framework for dynamic deployment and reconfiguration of component-based software systems**. In: MIS '05: PROCEEDINGS OF THE 2005 SYMPOSIA ON METAINFORMATICS, p. 8, New York, NY, USA, 2005. ACM.

[23] KULKARNI, V; REDDY, S. **Introducing mda in a large it consultancy organization**. Software Engineering Conference, 2006. APSEC 2006. 13th Asia Pacific, p. 419–426, Dec. 2006.

[24] LARMAN, C. **Utilizando UML e Padrões**. Bookman, 3 edition, 2007.

[25] LECOLINET, E. **A molecular architecture for creating advanced guis**. In: UIST '03: PROCEEDINGS OF THE 16TH ANNUAL ACM SYMPOSIUM ON USER INTERFACE SOFTWARE AND TECHNOLOGY, p. 135–144, New York, NY, USA, 2003. ACM.

[26] LEHTONEN, M; PETIT, R; HEINONEN, O; LINDÉN, G. **A dynamic user interface for document assembly**. In: DOCENG '02: PROCEEDINGS OF THE 2002 ACM SYMPOSIUM ON DOCUMENT ENGINEERING, p. 134–141, New York, NY, USA, 2002. ACM.

[27] LEITE, J. C. **Projeto de interfaces de usuário**. Master's thesis, Universidade Federal do Rio Grande do Norte. Departamento de Informática e Matemática Aplicada. Mestrado em Sistemas e Computação., 2001.

[28] LIU, N; HOSKING, J; GRUNDY, J. **A visual language and environment for specifying user interface event handling in design tools**. In: AUIC '07: PROCEEDINGS OF THE EIGHT AUSTRALASIAN CONFERENCE ON USER INTERFACE, p. 87–94, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.

[29] MRACK, M. **Geração automática e assistida de interfaces de usuário**. Master's thesis, Universidade Federal do Rio Grande do Sul - Instituto de Informática, Porto Alegre, BR - RS, Fevereiro 2008.

[30] MYERS, B. A. **User interface software tools**. ACM Trans. Comput.-Hum. Interact., 2(1):64–103, 1995.

[31] OBRENOVIC, Z; STARCEVIC, D. **Model-driven development of user interfaces: Promises and challenges**. Computer as a Tool, 2005. EUROCON 2005.The International Conference on, 2:1259–1262, 2005.

[32] OLIVER, I. **Applying uml and mda to real systems design**. p. 70–71 Vol. 1, March 2005.

[33] (OMG), T. O. M. G. **Xml metadata interchange (xmi)**. http://www.omg.org/technology/documents/formal/xmi.htm , Acesso em: 11 de Março de 2008.

[34] PEREIRA, D. A. **Infraguibuilder - framework para geraçao dinâmica e customizaçao de interfaces gráficas de usuário**. Novo Hamburgo, Instituto de Ciencias Exatas e Tecnologicas, Curso de Ciencia da Computaçao, Novembro 2007.

[35] PRESSMAN, R. S. **Engenharia de Software**. Mc Graw Hill, 2006.

[36] PROJECT, T. A. X. **Xerces java parser readme**. http://xerces.apache.org/xerces-j/ , Acesso em: 22 de Setembro de 2007.

[37] PUERTA, A; MICHELETTI, M; MAK, A. **The ui pilot: a model-based tool to guide early interface design**. In: IUI '05: PROCEEDINGS OF THE 10TH INTERNATIONAL CONFERENCE ON INTELLIGENT USER INTERFACES, p. 215–222, New York, NY, USA, 2005. ACM.

[38] RIBA, N; CERVANTES, H. **A mda tool for the development of service-oriented component-based applications**. Current Trends in Computer Science, 2007. ENC 2007. Eighth Mexican International Conference on, p. 149–156, Sept. 2007.

[39] SHNEIDERMAN, B; PLAISANT, C. **Designing the User Interface: Strategies for Effective Human-Computer Interaction**. Addison Wesley, 4 edition, 2004.

[40] SOUSA, K; MENDONÇA, H; VANDERDONCKT, J; ROGIER, E; VANDERMEULEN, J. **User interface derivation from business processes: a model-driven approach for organizational engineering**. In: SAC '08: PROCEEDINGS OF THE 2008 ACM SYMPOSIUM ON APPLIED COMPUTING, p. 553–560, New York, NY, USA, 2008. ACM.

[41] STUART-MOORE, J; EVANS, M; JACOBS, P. **Interface design for browsing faceted metadata**. In: JCDL '06: PROCEEDINGS OF THE 6TH ACM/IEEE-CS JOINT CONFERENCE ON DIGITAL LIBRARIES, p. 349–349, New York, NY, USA, 2006. ACM.

[42] SUN. **Trail: Graphical user interfaces**. http://java.sun.com/docs/books/tutorial/ui/index.html Acesso em: 16 de Abril de 2008.

[43] SUN MICROSYSTEMS, I. **Annotations**. http://java.sun.com/docs/books/tutorial/java/java00/a Acesso em: 20 de Setembro de 2008.

[44] TAN, W; MA, L; LI, J; XIAO, Z. **Application mda in a conception design environment**. p. 702–704, 2006.

[45] TEAM, O. **Open xava**. http://www.gestion400.com/web/guest/home , Acesso em: 18 de Outubro de 2008.

[46] TEAM, S. **Sbfb - swing bean form builder**. https://sbfb.dev.java.net/ , Acesso em: 20 de Outubro de 2008.

[47] TEAM, T. F. P. **Trails framework**. http://www.trailsframework.org , Acesso em: 21 de Outubro de 2008.

[48] TEAM, U. **Usixml - user interface extensible markup language**. http://www.usixml.org/ , Acesso em: 10 de Fevereiro de 2008.

[49] TEEVAN, J. B. **Displaying dynamic information**. In: CHI '01: CHI '01 EXTENDED ABSTRACTS ON HUMAN FACTORS IN COMPUTING SYSTEMS, p. 417–418, New York, NY, USA, 2001. ACM.

[50] VARA, J; DE CASTRO, V; MARCOS, E. **Wsdl automatic generation from uml models in a mda framework**. Next Generation Web Services Practices, 2005. NWeSP 2005. International Conference on, p. 6 pp.–, Aug. 2005.

[51] WALRATH, K. **The JFC Swing Tutorial: A Guide to Constructing GUIs**. Addison-Wesley, 2004.

[52] YU, B; ZHANG, C; ZHAO, Y. **Transform from models to service description based on mda**. Services Computing, 2006. APSCC '06. IEEE Asia-Pacific Conference on, p. 605–608, Dec. 2006.

[53] ZHAO, C; ZHANG, K. **Transformational approaches to model driven architecture - a review**. In: SEW '07: PROCEEDINGS OF THE 31ST IEEE SOFTWARE ENGINEERING WORKSHOP, p. 67–74, Washington, DC, USA, 2007. IEEE Computer Society.

APÊNDICE A

Revisão Sistemática - Busca de uma metodologia para contagem de complexidade de GUIs

Este apêndice descreve a realização da revisão sistemática visando encontrar uma metodologia de contagem ou estimativa da complexidade ou tamanho de interfaces gráficas.

O foco principal da pesquisa era obter artigos científicos, porém foram pesquisados também revistas, livros ou qualquer documento que nos orientasse sobre o assunto. Em cada local de pesquisa (site de busca e bibliotecas digitais) foram inseridos alguns termos de busca de modo a obter o resultado esperado.

Os locais de pesquisa e os termos de busca utilizados foram:

Local de busca: Google

Termos de pesquisa:

1. calculate complexity size + "graphical user interface"
2. measuring complexity graphical user interface
3. measuring + "graphical user interface"
4. gui measurement
5. user interface measurement
6. function point measurement GUI
7. function point measurement user interface
8. measure "user interface"
9. measure "user interfaces"

Local de busca: ACM e IEEE

Termos de pesquisa:

1. FP GUI
2. FP hci
3. FP UI
4. FP user interface
5. FPA GUI
6. FPA hci
7. FPA UI
8. FPA user interface
9. function point GUI
10. function point HCI
11. function point UI
12. function point user interface
13. metric GUI
14. metric hci
15. metric UI
16. metric user interface
17. measurement GUI
18. size measurement GUI
19. size GUI
20. size hci
21. size UI
22. size user interface
23. size graphical user interface
24. cost GUI
25. cost hci

26. cost UI
27. cost user interface
28. cost graphical user interface
29. complexity GUI
30. complexity hci
31. complexity UI
32. complexity user interface
33. complexity graphical user interface

Após a pesquisa, não foi obtida nenhuma metodologia de contagem ou estimativa de complexidade ou tamanho de interfaces gráficas. Porém, isso não significa que tal metodologia não exista, somente que esta pesquisa detalhada não foi capaz de encontrá-la.

Livros Grátis

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

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo