Gerência de Interface Homem-Computador para Sistemas de Informação Empresariais: uma abordagem baseada em modelos

Wilane Carlos da Silva[1], Juliano Lopes de Oliveira[1]

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)
Caixa Postal 131 – 74001-970 – Goiânia – GO – Brasil

wilanecarlos@gmail.com, juliano@inf.ufg.br

Abstract. *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. This contributes to the software usability, as it ensures consistency and homogeneity of the GUIs; 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.*

Resumo. *Construir e manter a Interface Gráfica com Usuário (GUI) para Sistemas de Informação Empresariais normalmente demanda muito tempo e esforço da equipe de Engenharia de Software. Este artigo descreve uma abordagem dirigida por modelos para criar e gerenciar essas interfaces. Nessa abordagem, o Engenheiro de Software projeta modelos conceituais do sistema de informação usando meta-modelos orientados a objetos. Um conjunto de regras de mapeamento pré-definidas é aplicado para refinar e transformar os modelos conceituais, gerando, em tempo de execução, a interface gráfica com sua aparência e comportamentos específicos. Esse mecanismo contribui para a usabilidade do software, visto que assegura a consistência e homogeneidade das GUIs; aumenta a produtividade da equipe de Engenharia de Software; e simplifica sua manutenção. Em nossos experimentos foi observada uma média de produtividade quinze vezes maior do que os métodos tradicionais de desenvolvimento de GUIs.*

1. Introdução

A Engenharia de Software tem como um dos desafios solucionar ou amenizar a ineficiência no que diz respeito ao desenvolvimento de softwares. O objetivo é fazer com que esse processo seja mais produtivo e, ao mesmo tempo, tenha maior qualidade. A importância desse desafio é justificada pelo pouco tempo disponível para o desenvolvimento e manutenção de sistemas cada vez mais complexos e que sofrem freqüentes mudanças durante seu ciclo de vida.

Interfaces Gráficas com Usuário (GUIs) contribuem amplamente para a baixa produtividade no desenvolvimento de software, principalmente porque:

i. as modificações nas GUIs são mais frequentes em relação a outros componentes, como funções de aplicação ou procedimentos armazenados no Banco de Dados [Shneiderman and Plaisant 2004];

ii. gera grande volume de código-fonte. Em média 48% do código de todo o sistema é relacionado a interfaces gráficas [Myers 1995];

iii. demanda grande esforço. De fato, como demonstra [Myers 1995], 50% do tempo gasto para o desenvolvimento de um sistema é dedicado para a criação e configuração de interfaces.

iv. sua complexidade vem aumentando nos últimos anos e tende a crescer cada vez mais [Galitz 2002];

v. depende de visuais e comportamentos padrões para fornecer boa usabilidade ao usuário final [Galitz 2002, Shneiderman and Plaisant 2004]; e

vi. necessita de maior cuidado durante o projeto para garantir melhor manutenibilidade, visto que a fase de manutenção do software costuma ser mais duradoura que a fase de desenvolvimento [Peters and Pedrycz 2001].

Devido a estas dificuldades, a construção e manutenção de GUIs requer sofisticados recursos técnicos da equipe de Engenharia de Software para especificar uma grande quantidade de aspectos de projeto, incluindo tamanhos, localizações, organizações e comportamentos gráficos [Collignon 2008].

Outros fatores motivaram o desenvolvimento de uma abordagem dirigida por modelos voltada para a criação de GUIs de Sistemas Empresariais. Neste tipo de sistemas, há a presença de um número extenso de telas de cadastro, com aparência e comportamentos semelhantes. Isso gera um volume considerável de código-fonte repetível voltado à interface gráfica.

Com isso, a modificação de algum requisito visual ou comportamental de uma janela implicaria na manutenção de todos os códigos-fontes das demais janelas. Para um número pequeno de janelas essa manutenção não seria trabalhosa. Porém, para grandes sistemas com, por exemplo, 150 telas de cadastro, um esforço considerável seria despendido.

Visando a redução de recursos e esforços dedicados à definição, construção e gerência de GUIs de Sistemas Empresariais, foi desenvolvida uma abordagem dirigida por modelos para esse fim. Para sua concretização, foi construído o Gerador Dinâmico de Interfaces Gráficas (GDIG). Trata-se de uma ferramenta que, a partir de modelos conceituais de Sistemas de Informação, constrói interfaces gráficas com usuário.

Modelos conceituais voltados ao negócio do sistema de informação e independentes de tecnologia, denominados de **Modelos de Negócio**, são criados e, através da aplicação de regras de mapeamento pré-definidas, são progressivamente transformados e refinados, gerando modelos mais específicos para a interface gráfica (**Modelos de Apresentação**). Estes últimos contêm conceitos e dados voltados para aparência, restrições, organização, usabilidade, relacionamentos e ações de GUIs. Os

Modelos de Apresentação passam por um último mecanismo de mapeamento onde são criadas as janelas do sistema propriamente ditas, denominadas de **Interfaces Concretas**.

A geração das interfaces ocorre em tempo de execução, sem a necessidade do armazenamento do código-fonte em disco. São interfaces *Desktop* do tipo CRUD (*Create, Read, Update, Delete*), ou seja, interfaces de formulário. Os componentes gráficos são baseados na tecnologia *Swing* da linguagem Java. O mecanismo possibilita a criação de aparências e comportamentos complexos, tornando-se útil para uma gama extensa de aplicações empresariais.

Dentre as contribuições, enfatiza-se: (i) significativo aumento na produtividade do desenvolvimento de GUIs de quinze vezes comparado à geração de código manual, devido ao fato de os projetistas trabalharem com modelos (informações de mais fácil entendimento); (ii) redução considerável no código fonte resultante, visto que o GDIG constrói as interfaces em tempo de execução, não criando nenhum código além daquele do próprio mecanismo, ou seja, o fato de aumentar o número de GUIs geradas não aumenta o código-fonte interno ao gerador; e (iii) padronização do *look and feel*, melhorando a usabilidade e consistência do Sistema de Informação, já que todas as GUIs são geradas a partir do mesmo mecanismo.

O restante deste documento descreve os detalhes da abordagem da criação e gerenciamento de GUIs, sendo organizado da seguinte maneira. A Seção 2 descreve os detalhes do mecanismo de geração de GUI apresentado nesse trabalho. A Seção 3 discute os experimentos utilizados para avaliar o aumento de desempenho proporcionado pelo mecanismo de geração de GUIs. A Seção 4 conclui o documento e indica direções para trabalhos futuros.

O restante deste documento descreve os detalhes de nossa abordagem do gerenciamento de GUIs e é organizado da seguinte maneira. A Seção 2 compara a abordagem a trabalhos relacionados sobre a geração dirigida a modelos de interfaces gráficas com usuário. A Seção 3 descreve os detalhes do mecanismo de geração de GUI apresentado nesse trabalho. A Seção 4 discute os experimentos utilizados para avaliar a produtividade desse mecanismo para geração de GUIs. A Seção 5 conclui o documento e indica direções para trabalhos futuros.

2. Ferramentas para geração de interfaces gráficas com usuário

Várias abordagens para geração de GUIs foram propostas na literatura. De acordo com [Collignon 2008], existem quatro categorias de ferramentas: as baseadas em linguagens, as ferramentas interativas, os *frameworks* de aplicações e os geradores automáticos baseados em modelos.

No contexto das ferramentas baseadas em linguagens, o projetista especifica a interface gráfica em uma linguagem de propósito específico. Essa especificação pode ser feita de várias formas, incluindo gramáticas livres de contexto, diagramas de estado, linguagens declarativas, máquinas de estado finito e linguagens de evento.

As ferramentas interativas permitem ao desenvolvedor selecionar componentes gráficos a partir de uma biblioteca pré-definida e inseri-los na tela, gerando, por exemplo, caixas de diálogos, menus e janelas.

Os *frameworks* são os tipos mais implementados [Pereira 2007, Mrack 2008]. Eles funcionam como uma solução "caixa-branca" onde o desenvolvedor da aplicação precisa inserir informações e códigos que estendam e adaptem seu comportamento para que funcione corretamente. Com isso, há muito trabalho a ser feito para implementar uma GUI usando esse tipo de ferramenta, visto que ela requer o projeto de novo código, demandando da equipe técnica a compreensão do projeto do *framework* e a especialização em relação à sua tecnologia.

Uma outra categoria de ferramentas existente, e que foi implementada nesse trabalho, é a dos geradores automáticos baseados em modelos. Nesta, um ou vários modelos são explorados para, automaticamente, gerar a interface gráfica. Ela ajuda a solucionar o problema dos *frameworks*, pois requer pouca ou nenhuma especificação por parte do usuário.

A maioria das ferramentas geram interfaces focando somente uma plataforma alvo. As mais comuns são para *Desktop* [Sbfb 2008; Jmatter 2008] e *Web* [Openxava 2008; Roma 2008]. A maioria das abordagens e ferramentas existentes geram telas de cadastro [Trais 2008], baseando-se, principalmente, no esquema do Banco de Dados da aplicação.

O controle das operações e da validação dos dados nessas telas pode ser feito através de regras de negócio. As ferramentas apresentadas em [Roma 2008; Trais 2008] inserem essas regras no próprio código-fonte, acoplando-as à linguagem de programação e fazendo com que qualquer mudança nas regras implique na necessidade de nova compilação do código.

A ferramenta GDIG apresentada nesse documento, implementada como gerador de GUIs dirigido a modelos, proporciona uma mecanismo mais simples que os *frameworks* para geração de interfaces gráficas. Sua utilização típica não envolve a inclusão ou a modificação de código, visto que funciona como uma "caixa-preta", encapsulando sua forma de trabalho.

São, portanto, menos flexíveis. Porém, apresentam-se 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.

Outra característica que foi requisito básico de nosso projeto é a portabilidade. Nossa abordagem foi planejada e estruturada para suportar a construção de interfaces gráficas para diferentes plataformas, incluindo *Desktop*, *Web* e dispositivos móveis, apesar de a versão atual do GDIG implementar somente os mapeamentos para *Desktop*.

Além das telas de cadastro, o GDIG contempla a geração de telas de edição coletiva. Elas consistem em permitir que o usuário execute as operações CRUD de vários elementos numa única transação de negócio. Outras funcionalidades baseadas em modelos como l*ogon*, gerenciamento de usuários, controle de acesso, calendário e geração de menus também estão presentes na ferramenta.

Diferentes abordagens são usadas para permitir a edição dos modelos de GUIs nas ferramentas dirigidas a modelos. A mais comum consiste na edição de arquivos de texto, ou arquivos XML, através de editores textuais [Naked 2008; Lehtonen 2002; Bendsen 2004]. 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.

Nossa abordagem permite também que regras de negócio sejam acopladas aos modelos. Com isso obtém-se uma independência em relação ao código-fonte, permitindo que regras possam ser inseridas ou retiradas da aplicação dinamicamente e sem necessidade de recompilação.

O projetista cria uma instância desse metamodelo e insere-o no Modelo de Negócio que depois será usado pelo gerador da interface gráfica. Ao executar as funcionalidades da aplicação, o GDIG verifica a existência de tal regra e, caso exista, executa automaticamente o que está especificado no Modelo de Regras.

O metamodelo de Regras é mostrado na Figura 1. Neste metamodelo, um conceito de negócio dá origem a uma regra, que pode ser do tipo validação, derivação ou de importação. Cada uma dessas regras é executada por uma operação que ocorre antes ou depois da ação que originou a regra. [Alvarenga 2007] fornece os detalhes sobre o projeto e funcionamento das regras utilizadas pelo GDIG.

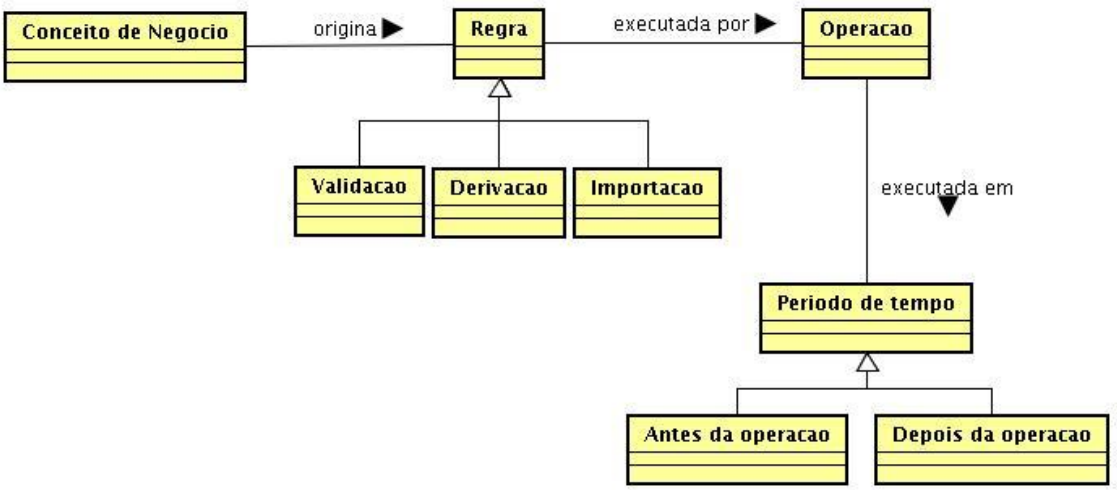

Figura 1. Metamodelo simplificado de Regras de Negócio.

3. O Gerador Dinâmico de Interfaces Gráficas com Usuário

O GDIG utiliza modelos para construir e gerenciar sistemas. Deste modo, optou-se por uma abordagem não convencional, onde os modelos passam a ser a preocupação central da equipe de Engenharia de Software, tornando-se 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 [Bordin 2007; Obrenovic 2005].

A visão geral dessa abordagem é mostrada na Figura 2. Existem dois tipos de modelos: o **Modelo de Negócio (MN)**, um modelo independente de tecnologia que contém informações de negócio; **e Modelo de Apresentação (MA)**, que contém informações relevantes sobre comportamento e aparência da interface a ser gerada**.**

Para a geração de uma GUI, cria-se primeiramente o MN correspondente, ou seja, cada GUI será representada por um MN. Ele é criado a partir do Modelo de Meta-

Objetos Complexos (MMO), que é um meta-modelo utilizado para criação de MNs. Em seguida, são aplicadas regras de transformação ao MN, originando o MA. Este baseia-se no meta-modelo Metamodelo de Apresentação (MMA), usado para a criação de MAs. O último passo para a criação da GUI consiste na transformação do MA para a interface propriamente dita, chamada de **Interface Concreta**. Nesta fase o GDIG analisa as informações contidas no MA e define quais componentes gráficos serão inseridos na janela que está sendo construída.

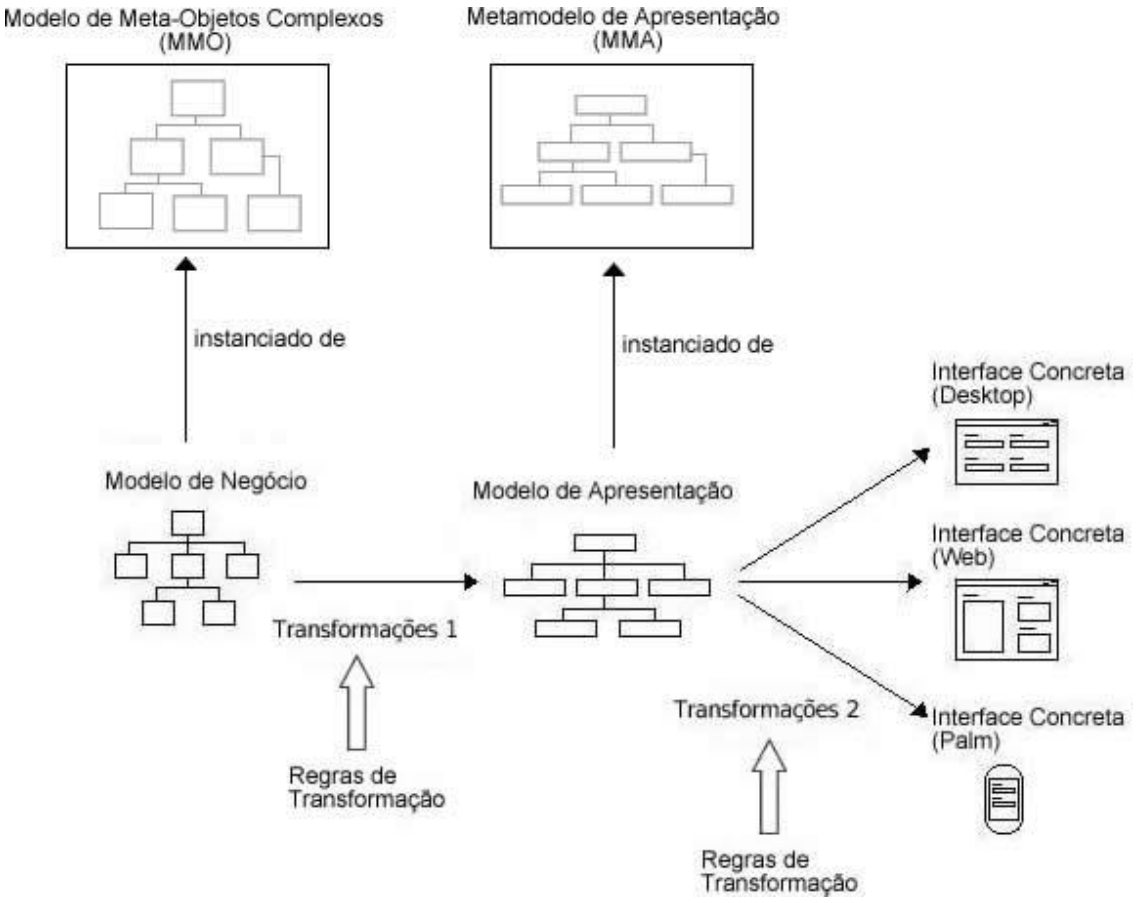

Figura 2. Visão geral da abordagem.

O MMO é um meta-modelo que representa e descreve objetos complexos de sistemas de informação. Seu objetivo é prover a representação do MN e consiste de entidades, relacionamentos entre entidades e propriedades das entidades. A Figura 3 apresenta a visão conceitual do MMO.

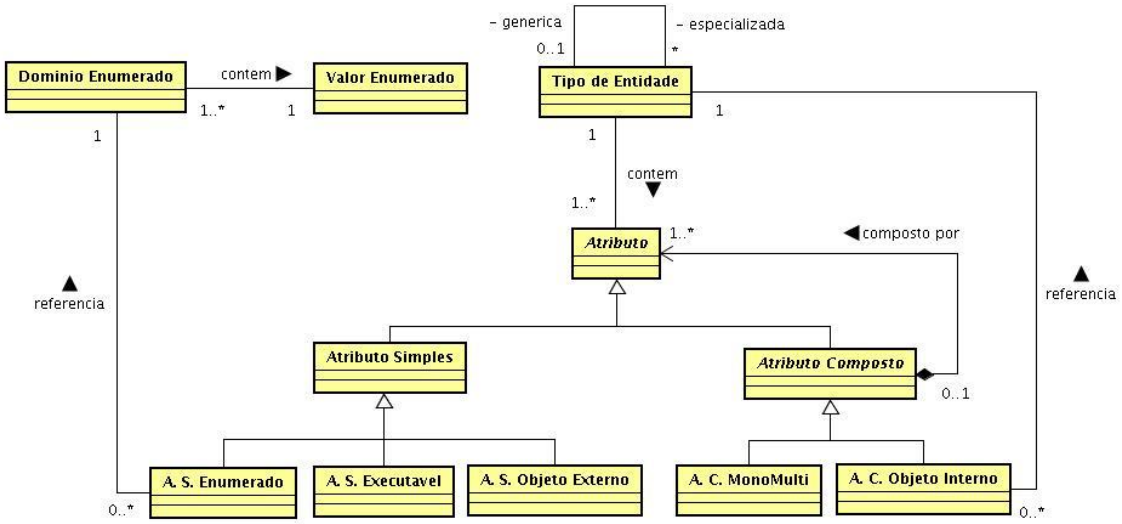

Figura 3. Visão conceitual do MMO.

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

Os Atributos Simples podem ser dos tipos Enumerado, Executável ou Objeto Externo. Os Enumerados são atributos cujos valores podem ser somente aqueles contidos no domínio enumerado pré-definido. Os do tipo Executável descrevem atributos cujo objetivo é executar programas dentro do sistema. Os atributos do tipo Objeto Externo são aqueles que contêm informações sobre arquivos ditos como externos ao sistema, como arquivos de texto e mídia (imagem, áudio e vídeo). Os Atributos Compostos são aqueles formados de um ou mais atributos. O Composto do tipo Objeto Interno contém a propriedade de referenciar 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 MN que representa uma entidade com suas diversas características e associações. Posteriormente, essa entidade será transformada em uma tela CRUD pelo mecanismo interno do GDIG.

O MMA é um meta-modelo utilizado para construir MAs. É voltado para interface gráfica, descrevendo informações sobre aparências e comportamentos. A Figura 4 ilustra o MMA.

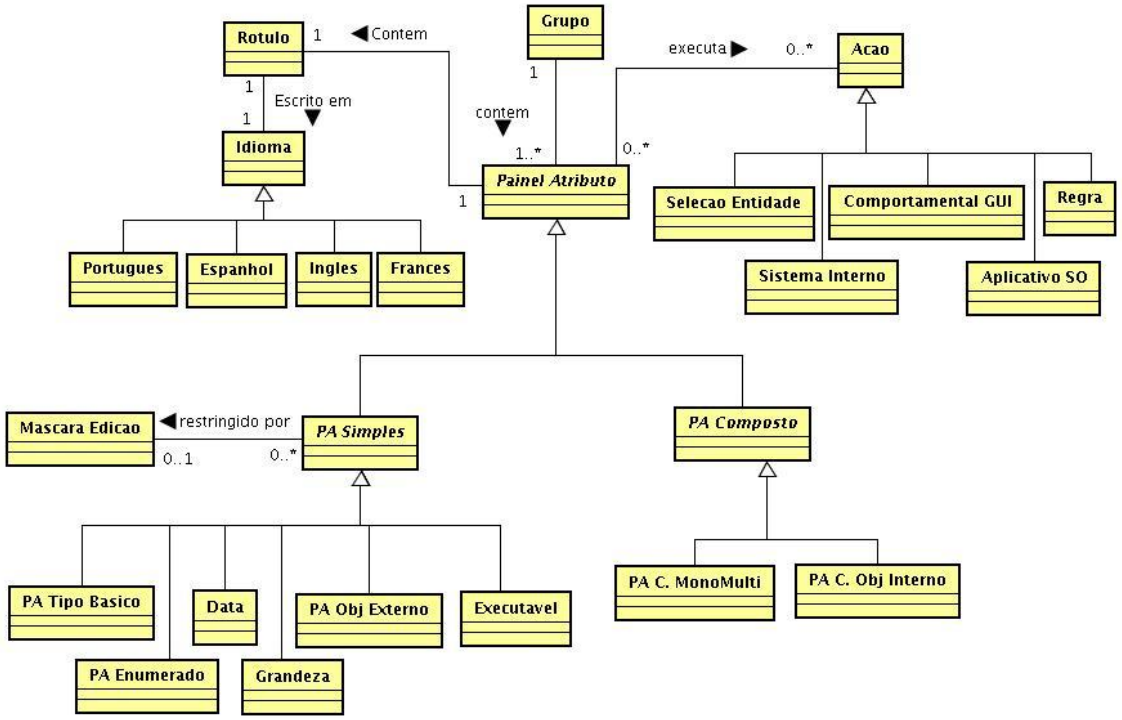

Figura 4. Visão conceitual do MMA.

Cada Painel Atributo descreve um campo de formulário dentro da interface CRUD a ser gerada. Para que o GDIG possa ter uma referência de agrupamento desses campos, cada Painel Atributo deve fazer parte de um Grupo.

Cada Painel Atributo é composto por um rótulo, que pode ser definido em quatro idiomas (português, espanhol, inglês ou francês), e, de acordo com a especialidade do Painel Atributo, o GDIG cria um conjunto de componentes gráficos que permite ao usuário inserir e gerenciar dados dentro das janelas. Existem vários tipos de Painéis Atributos específicos, cada um responsável por referenciar diferentes tipos de campos no formulário.

O Painel Atributo Simples é aquele indivisível, atômico. O Painel Atributo Tipo Básico descreve campos que podem receber quaisquer tipos de valores, o Enumerado aqueles que podem conter somente valores de um conjunto pré-definido, o Data aqueles cujo valor é uma data formatada de acordo com o idioma, o Grandeza aqueles cujo valor representa uma medida, o Objeto Externo aqueles cujo valor referencia arquivos externos como arquivos de mídia e o Executável aqueles que executam uma ação específica como, por exemplo, abrir um sub-sistema.

O Painel Atributo Composto MonoMulti é aquele que contém outros Atributos dentro dele, chamados, nesse caso, de sub-atributos. O Objeto Interno contém atributos (referencia dados) de uma entidade já cadastrada no sistema.

Cada Painel Atributo pode executar ações. Existem ações para seleção de entidade, executada somente pelo Painel Atributo Composto Objeto Interno. Outra ação é a de comportamento da GUI. Essa consiste na implementação do padrão *Observer* [Gamma 1998], onde um componente gráfico, ao modificar seu valor, aciona outros componentes que executam ações específicas. E outra ação é a regra, explicada

anteriormente. Essas ações provêm propriedades de relacionamentos para a interface gráfica.

Os Modelos de Negócio e de Apresentação são armazenados em um banco de dados. Existem outras formas de armazenamento como *Annotations* [Naked 2008] e arquivos XML [Sbfb 2008]. Porém, decidiu-se pelo armazenamento num banco de dados devido à vantagem de se obter independência referente a linguagens de programação ou outras tecnologias. O armazenamento em *Annotations*, por exemplo, torna os metadados presos à linguagem Java.

Ao contrário dos modelos, a interface não é armazenada em disco. Não são criados código-fontes para uma janela gerada pelo GDIG. Todo o processo de leitura do modelo, criação e organização dos componentes gráficos são feitos em tempo de execução e, após o fechamento da GUI, não ocorre nenhum tipo de registro de sua estrutura visual. As únicas informações utilizadas para a construção das GUIs são aquelas dos modelos. A Figura 5 mostra uma visão do armazenamento dos modelos e geração da GUI em tempo de execução.

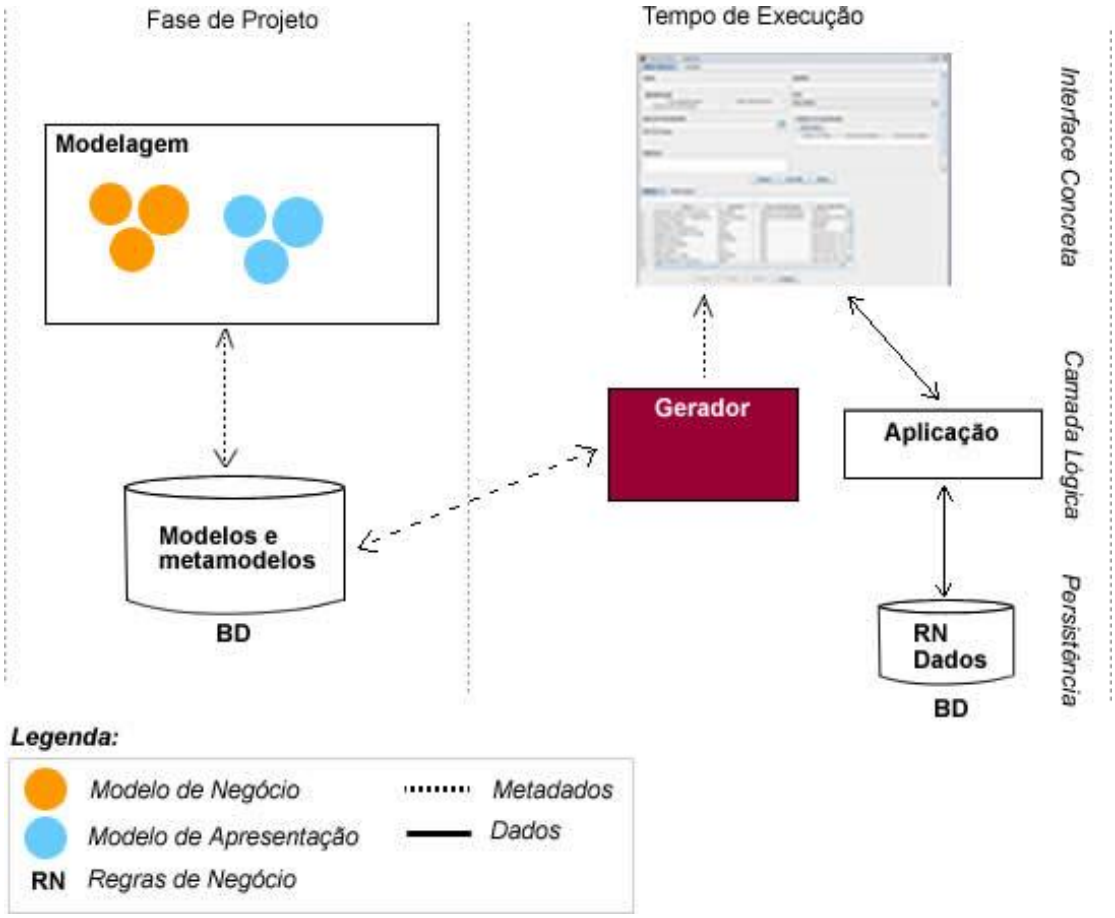

Figura 5. Armazenamento dos modelos e geração da GUI.

Na fase de projeto, o projetista da interface gráfica cria os MN e MA e armazena-os no banco de dados. Tempos depois, quando o usuário final requisitar a abertura da janela como, por exemplo, clicando em um menu, o GDIG receberá essa requisição, obterá do banco de dados os modelos referentes à janela (entidade), a

montará e a mostrará ao usuário. Todo esse processo, desde o clique no menu até a visualização pelo usuário, será feito em tempo de execução.

No meio dessa seqüência de passos poderá ser requerida da camada de aplicação algumas regras de negócio pré-persistidas, de acordo com a necessidade do usuário

A Figura 6 traz um exemplo de interface gerada. A parte com borda hachurada delimita campos do formulário. Cada Painel Atributo é mapeado para um campo que contém um rótulo e um conjunto de *widgets*. Os campos são organizados da esquerda para direita, de cima para baixo, formando um formulário com duas colunas e N linhas.

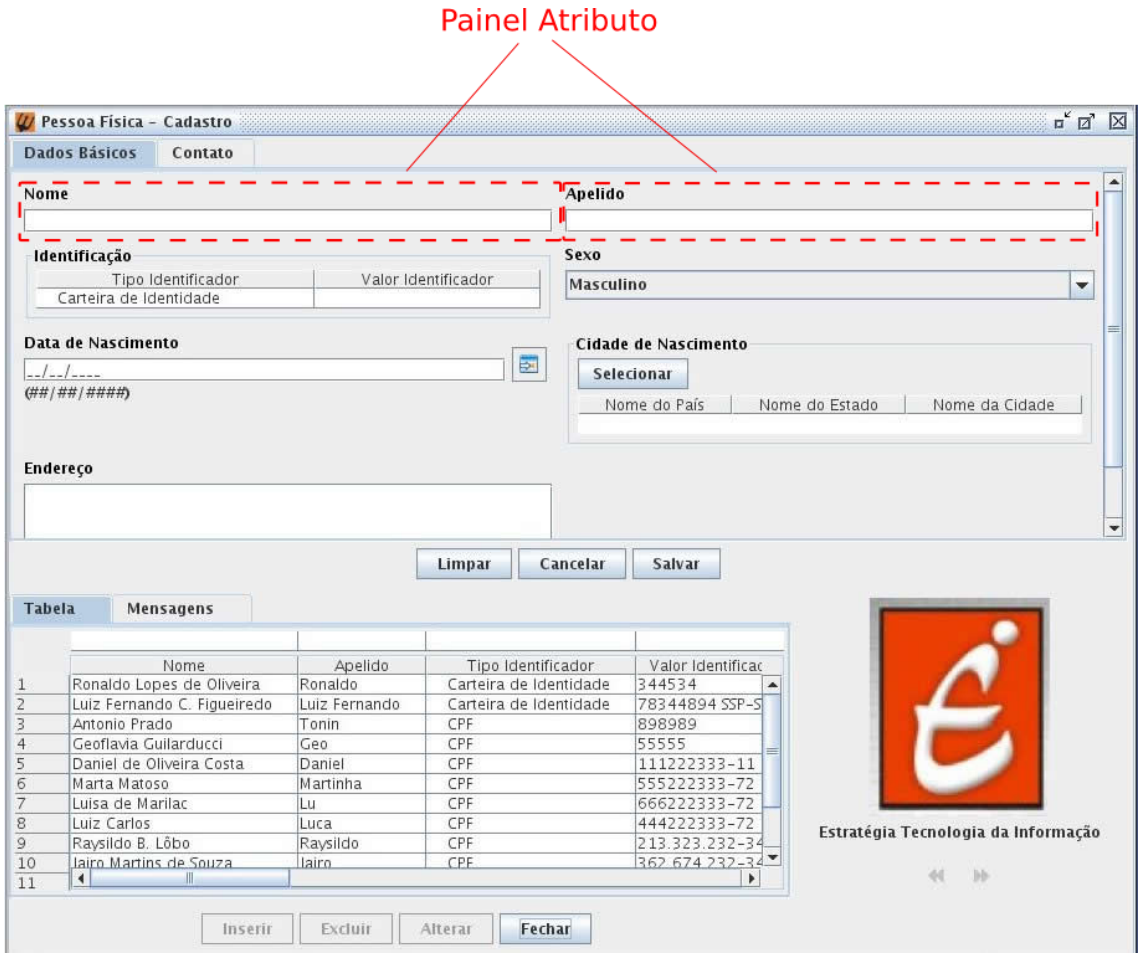

Figura 6. Tela gerada – Interface Concreta.

3.1. Exemplo prático da geração de um componente da GUI

Com o intuito de concretizar o entendimento do mecanismo de geração de GUIs, simularemos a geração de um componente gráfico de uma janela. Como todos os componentes são criados através do mesmo mecanismo, basta demonstrar a geração de um componente para entender a geração de toda a janela. Tomando como referência a tela da Figura 6, simularemos a geração do campo “Data de Nascimento”.

A Figura 7 traz o elemento dentro do MN que representará a “Data de Nascimento”.

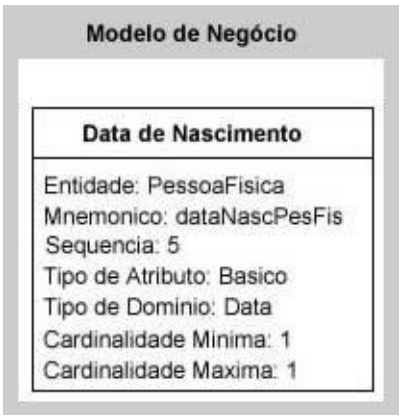

Figura 7. Modelo de Negócio referente ao campo “Data de Nascimento”.

O atributo “Data de Nascimento” faz parte da entidade cujo mnemônico é “PessoaFisica” e é identificado através de um identificador único, que neste caso é “dataNascPesFis”.

O atributo tem o valor seqüencial igual a cinco. Essa informação é utilizada pelo GDIG para determinar a posição do campo no formulário. As informações contidas nos campos “Tipo de Atributo” e “Tipo de Domínio” servem de parâmetro para a definição futura do tipo de componente gráfico a ser criado. A cardinalidade mínima e máxima informam a quantidade máxima e mínima de valores que este atributo pode ter. Neste caso, uma Pessoa Física terá uma e somente uma data de nascimento.

O próximo passo consiste na criação do MA. O GDIG executa essa tarefa obtendo o tipo de domínio “Data” e mapeando o atributo para o Painel Atributo Data. Nesse momento, o projetista define algumas outras informações como, por exemplo: se o campo será editável ou não, se deve ser oculto e quais tipos de ações ele executará. No caso, está ligado a uma ação do tipo “Regra de Validação”. Essa regra define que uma data de nascimento não pode ter valor maior que o dia atual do sistema. A geração do modelo de apresentação referente ao atributo “dataNascPesFis” é mostrada na Figura 8.

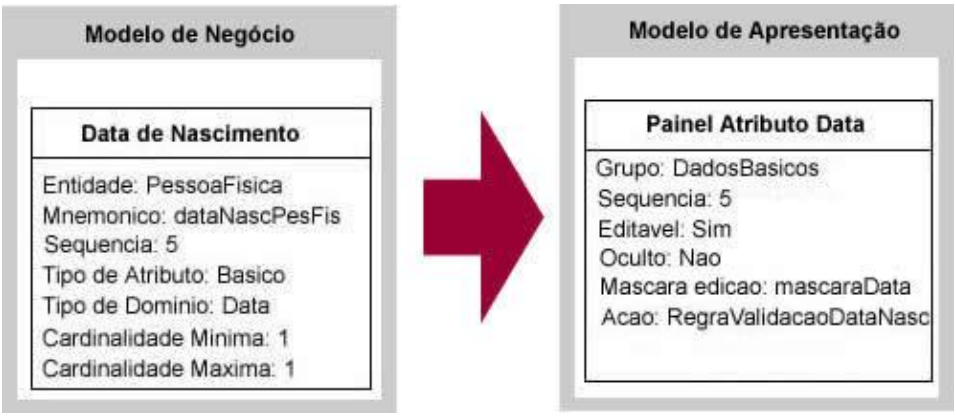

Figura 8. Transformação do Modelo de Negócio para o Modelo de Apresentação.

Até agora todas as ações foram executadas na fase de projeto e os modelos armazenados no banco de dados. Com isso, a interface gráfica está pronta para ser gerada.

Em tempo de execução, quando um usuário executa uma ação para abrir a janela "Pessoa Física" (como clicar num menu, por exemplo), o gerador dinâmico obtém o modelo de apresentação referente à entidade "PessoaFisica", interpreta as informações de cada atributo, cria seus componentes correspondentes e insere-os na janela. No caso do atributo "Data de Nascimento" são criados um rótulo no idioma desejado pelo usuário, um campo de texto com uma máscara de edição do tipo data e um botão para que o usuário possa escolher a data através de um calendário. A Figura 9 mostra os componentes gráficos criados.

Figura 9. Componentes gráficos criados para o campo de data.

O gerador executa esses passos para cada atributo do modelo de apresentação, montando a interface gráfica e apresentando-a ao usuário final, como mostra a Figura 6.

4. Resultados empíricos

Para demonstrar o aumento na produtividade do desenvolvimento das interfaces gráficas, a seguinte metodologia foi executada. Primeiramente, definiu-se uma tela CRUD de baixa complexidade a ser gerada. A tela consiste em cinco campos de texto para a entrada de dados e numa tabela contendo os dados já cadastrados, como mostrado na Figura 10.

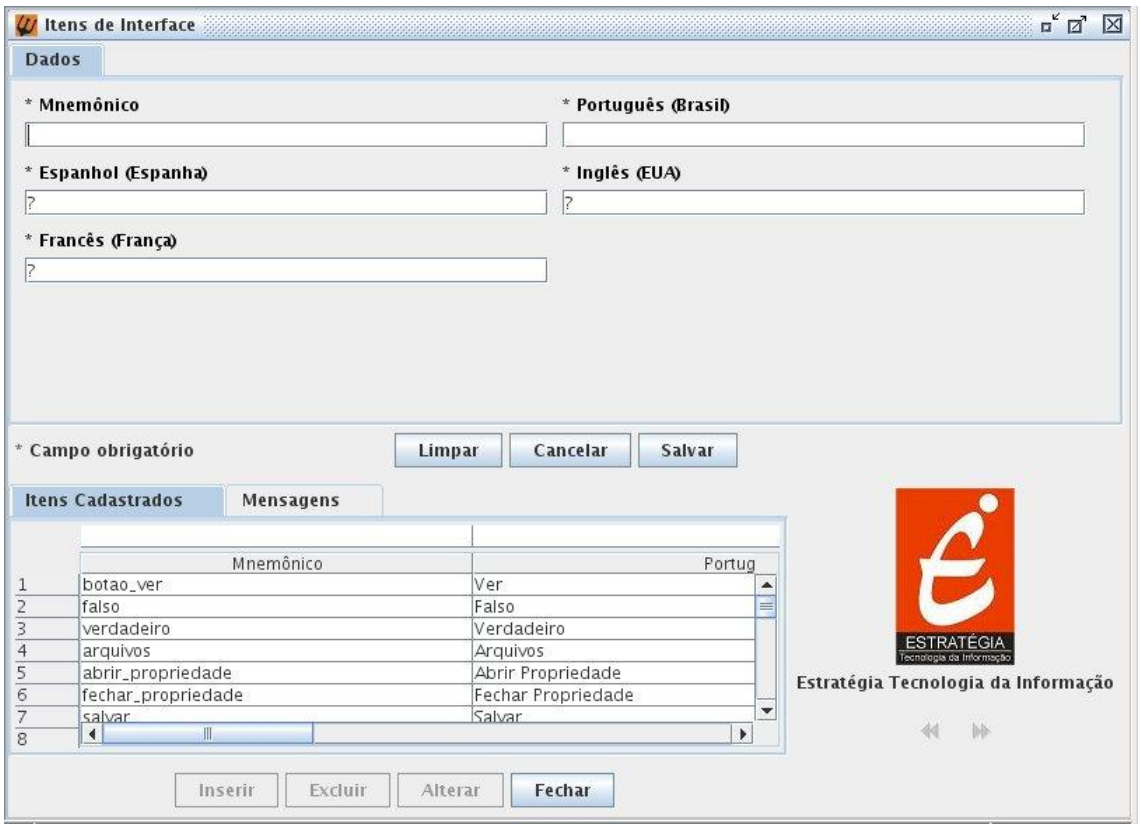

Figura 10. Tela de complexidade simples para a coleta dos dados.

A tela foi gerada de três formas: por meio de codificação manual utilizando o NetBeans IDE 6.7.1 como editor; através de editor visual, onde o programador tem o trabalho somente de escolher e arrastar os componentes gráficos para formar a tela utilizando também a ferramenta NetBeans IDE 6.7.1; e 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) e, em cada geração, foi medido o tempo em minutos (min). A Tabela 1 mostra os dados coletados para cada tipo de geração executada por cada um dos desenvolvedores.

Tabela 1. Dados de produtividade coletados durante o experimento.

Tipo de geração	**Tempo (minutos)**			
	D1	**D2**	**D3**	**D4**

Manual	382 min	516 min	315 min	411 min
Editor Visual	163 min	154 min	130 min	181 min
GDIG	24 min	31 min	27 min	23 min

Calculou-se então a média aritmética de cada tipo de geração:

- Manual: 1624 min / 4 = 406 min
- Editor Visual: 628 min / 4 = 157 min
- GDIG: 105 min / 4 = 26,25 min = 27 min

Pelos dados obtidos, conclui-se que a geração da interface gráfica pela ferramenta GDIG é, no mínimo, quinze vezes mais rápida que a geração manual e quase seis vezes mais rápida que a geração auxiliada por editores visuais. Essa diferença na produtividade tende a ser ainda maior, visto que o GDIG gera também os comportamentos da tela e as conexões entre as demais camadas do sistema, características que não foram medidas devido a fatores explicados no início dessa seção.

5. Conclusões

Este artigo tratou da geração de interfaces gráficas com o usuário dirigidas a modelos. Para tanto, foi criado um gerador dinâmico cujo mecanismo consiste no recebimento, como entrada, de modelos de alto nível de abstração. Estes, por sua vez, passam por transformações, gerando modelos específicos que são usados para a construção da interface do tipo CRUD para *Desktop* utilizando componentes gráficos do *Swing*.

Reduzimos o problema chave da geração de GUIs: 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 [France 2007].

O fator da eficiência é citado inclusive em [Jmatter 2008], 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 dez vezes comparado aos métodos tradicionais de desenvolvimento. Nossa ferramenta está acima desta média, mostrando-se quinze vezes mais eficiente em relação à codificação manual de interfaces e quase seis vezes em relação à geração auxiliada por editores visuais.

Além da eficiência no desenvolvimento de software, outras contribuições são verificadas como a (i) redução de código-fonte; (ii) diminuição da complexidade na geração das interfaces e do sistema como um todo; (iii) geração de visuais e comportamentos padrões da interface, fazendo com que a mesma tenha um alto nível de

usabilidade; (iv) maior influência das partes interessadas no decorrer do projeto, pois as mudanças têm um menor custo; e (v) reusabilidade dos modelos em outros sistemas ou módulos do mesmo sistema.

O trabalho pode ser estendido de várias maneiras: implementando a geração para outras plataformas alvo como Web e Palm, a comunicação com outras ferramentas através da importação e exportação de modelos e mecanismos inteligentes de automatização da manutenção e da organização dos componentes.

Referências

Alvarenga, G. G.: Uma Abordagem para Tratamento de Regras de Negócio em Sistemas de Informação. Goiânia, Instituto de Informática, Universidade Federal de Goiás, 2007.

Bendsen, P.: Model-driven business UI based on maps. In Proceedings of the 2004 ACM SIGMOD international Conference on Management of Data (Paris, France, June 13 - 18, 2004). SIGMOD '04. ACM, New York, NY, 887-891

Bordin, M. and Vardanega, T.: Real-time Java from an automated code generation perspective. In Proceedings of the 5th international Workshop on Java Technologies For Real-Time and Embedded Systems (Vienna, Austria, September 26 - 28, 2007). JTRES '07, vol. 319. ACM, New York, NY, 63-72

Collignon, B., Vanderdonckt, J., and Calvary, G.: An intelligent editor for multi-presentation user interfaces. In Proceedings of the 2008 ACM Symposium on Applied Computing (Fortaleza, Ceara, Brazil, March 16 - 20, 2008). SAC '08. ACM, New York, NY, 1634-1641

France, R. and Rumpe, B.: Model-driven Development of Complex Software: A Research Roadmap. In 2007 Future of Software Engineering (May 23 - 25, 2007). International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 37-54

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. M.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1998)

Jmatter, http://jmatter.org/. Acessado em: 15 de Junho de 2008.

Lehtonen, M., Petit, R., Heinonen, O., and Lindén, G.: A dynamic user interface for document assembly. In Proceedings of the 2002 ACM Symposium on Document Engineering (McLean, Virginia, USA, November 08 - 09, 2002). DocEng '02. ACM, New York, NY, 134-141

Myers, B. A.: User interface software tools. ACM Trans. Comput.-Hum. Interact. 2, 1 (Mar. 1995), 64-103

Naked objects, http://www.nakedobjects.org/home/index.shtml. Acessado em: 11 de Outubro de 2008.

Obrenovic, Z. and 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

Open xava, http://www.gestion400.com/web/guest/home. Acessado em: 15 de Outubro de 2008.

Roma framework, http://www.romaframework.org/. Acessado em: 18 de Setembro de 2008.

Sbfb - Swing Bean Form Builder, https://sbfb.dev.java.net/. Acessado em: 10 de Setembro de 2008.