República Federativa do Brasil
Ministério do Desenvolvimento, Indústria e do Comércio Exterior
Instituto Nacional da Propriedade Industrial.

(21) **PI0909274-9 A2**

(22) Data de Depósito: 18/12/2009
(43) Data da Publicação: 16/08/2011
(RPI 2119)

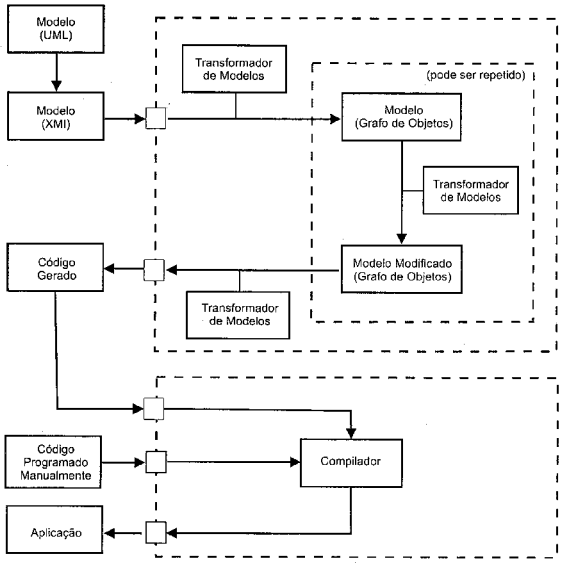
(51) Int.Cl.:
G06F 9/44 2006.01
G06Q 20/00 2006.01
G06Q 50/00 2006.01

(54) Título: **PROCESSO PARA GERAÇÃO AUTOMATIZADA DE SOFTWARE**

(73) Titular(es): COMPUGRAF TELECOM LTDA

(72) Inventor(es): JOSE CARLOS CORDEIRO MARTINS

(57) Resumo: PROCESSO PARA GERAÇÃO AUTOMATIZADA DE SOFTWARE. É proposto um processo de geração automática de programas fonte com base no MDA. Para orientar o processo de geração de código, são usadas regras de marcação padronizadas, inseridas em modelos UML por meio de lagged values, extensões e anotações. Essas regras indicam o interesse arquitetural e o padrão dc projeto que deverá ser usado ao gerar os programas fonte. O PROGERAS é proposta de extensão ao MDA, aplicando os conceitos do AOD e do POA para gerar programas fonte de forma automatizada a partir de um diagrama de classes (PIM). Com essa proposta, acredita-se ser possível não só obter um elevado nível de automação no processo de produção do software, forçar o uso de padrões, aumentar a qualidade e a consist6ncia, como também ganhar produtividade.



PI0909274-9

"PROCESSO PARA GERAÇÃO AUTOMATIZADA DE SOFTWARE"

INTRODUÇÃO/ESTADO DA TÉCNICA

No presente pedido é proposto um processo para geração de programas a partir de modelos UML, como sugerido no MDA (*Model-Driven Architecture*). O MDA é um processo de desenvolvimento de software criado e publicado pela OMG (*Object Management Group*), em 2001, e tem como base um conjunto de práticas e padrões para desenvolvimento de software por meio da transformação de modelos. O MDA está fundamentado em outros padrões também definidos pela OMG: UML (*Unified Modeling Language*) para modelagem, MOF (*Meta-Object Facility*) para definição de linguagens de modelagem, XMI (XML *Metadata Interchange*) para armazenamento de modelos e QVT (*Query View Transformation*) para transformação de modelos (Mellor *et alli.*, 2005).

O MDA é uma especialização do MDD (*Model-Driven Development*), uma abordagem na qual considera-se que os modelos de análise e projeto estão no mesmo nível dos programas fontes, buscnado transferir o trabalho da programação para a construção de modelos (Stahl *et alli.*, 2006). O objetivo do MDA e do MDD é desenvolver sistemas completos de forma automatizada por meio da ligação e transformação de modelos e no final gerar programas fonte. Na abordagem MDA estão definidas quatro categorias de modelos, que são o CIM (*Computer Indepent Model*), o PIM (*Platform Indepent Model*), o PSM (*Platform Specific Model*) e os programas-fonte. O CIM é usado para modelar os processos de negócio e a estrutura das informações. O PIM é um modelo utilizado para descrever a estrutura do sistema sem levar em consideração a plataforma na qual será implementado. Esse modelo pode consistir de diagramas de classes, estados, sequência, dentre outros. O PSM consiste de um refinamento do PIM, no qual é considerada uma plataforma específica e ele lida com o sistema operacional, linguagem de programação, tecnologia da plataforma (.NET, J2EE, CORBA, etc.), distribuição, dentre outros detalhes técnicos. No MDA o programa fonte também é considerado como sendo um modelo, que é o refinamento do PSM. As seguintes contribuições do MDA são destacadas em Biffl *et alli.* (2007):

- Melhoria da produtividade aliada à redução de prazo e custo, decorrente da automação da geração de modelos e programas fonte.

- Maior foco no negócio, decorrente da mudança do enfoque do desenvolvedor que passa a trabalhar principalmente no CIM e no PIM para construir modelos conceituais, ao invés de ter que se aprofundar em detalhes lógicos e técnicos.

- Aumento da portabilidade, visto que o mesmo PIM pode ser transformado em modelos PSM para diferentes plataformas e linguagens.

- Simplificação do processo de mudança, já que essas precisam ser feitas apenas no CIM e no PIM, e depois propagadas para os modelos PSM e programas fonte automaticamente.

- Aumento do reuso, por meio da reutilização das rotinas e regras de transformação e geração de programas fonte.

Sabe-se que o uso do MDA pode resultar em diversas melhorias, tais como: (i) melhorias no requisito de portabilidade, devido à separação do conhecimento de negócio de sua implementação numa tecnologia específica; (ii) aumento de qualidade, devido ao reuso de padrões e práticas bem testadas no processo de transformação; (iii) melhoria na facilidade de manutenção, devido à separação de interesses e obtenção de maior consistência e geração de trilhas entre modelos e programas fonte. Contudo, o MDA apresenta uma lacuna importante por não apresentar suporte adequado para elaboração do projeto técnico da arquitetura, uma vez que a concepção da arquitetura do software não é considerada na visão do MDA. A ausência dessa característica tem sido um campo fértil para pesquisas, que estão no campo do ACMDA (*Architecture-Centric* MDA) e *Aspect-Oriented* MDA, e que propõem o uso do AOD (*Aspect-Oriented Design*), do POA (*Pattern-Oriented Architecture*) e do AOP (*Aspect-Oriented Programming*) como meios para estender o MDA.

Na filosofia do *Pattern-Oriented Architecture* (POA), a arquitetura de um software pode ser construída por meio do reuso de soluções estruturais prontas, que são os padrões de projeto. O POA pode trazer várias contribuições para aumento da qualidade e produtividade, ou seja:

- Prover soluções bem conhecidas e testadas para o projeto da arquitetura, reduzindo o esforço e os riscos de especificação.
- Prover meios para documentar as decisões pertinentes à arquitetura.
- Facilitar a comunicação entre os participantes do projeto por meio de um vocabulário padronizado.
- Descrever os atributos de qualidade associados a cada solução.

Segundo o *Aspect-Oriented Design* (AOD), a arquitetura de um software pode ser projetada e implementada a partir da especificação de interesses arquiteturais (*architectural concerns*) como persistência, apresentação, auditoria, dentre outros. Esses interesses podem ser especificados e implementados por meio da aplicação de padrões de projeto. O AOD é derivado do *Aspect-Oriented Programming* (AOP) que é uma técnica de programação que permite endereçar umas das principais limitações da programação orientada a objetos, que é a modularização dos interesses arquiteturais que atravessam vários elementos da estrutura do software (*crosscutting concerns*), como persistência, tratamento de erros, segurança, dentre outros. O AOD provê mecanismos para identificar e projetar esses interesses e depois combiná-los com o resto dos programas. Dentre os principais benefícios do AOD, destacam-se:

- Centralizar a localização do código associado à implementação de uma funcionalidade específica, por meio de um aspecto. Por exemplo, no AOP o programa pertinente à persistência fica concentrado num aspecto, que depois é combinado com o resto dos programas no momento da compilação. Isso aumenta a coesão e reduz o acoplamento entre os elementos da arquitetura.
- Automatizar o processo de geração de programas, combinando múltiplos aspectos para gerar o software. Isso pode resultar em ganho de produtividade e qualidade, assim como redução de custo.

Na abordagem do MDA falta abordar a questão do projeto técnico da arquitetura, o que inclui o tratamento dos requisitos não-funcionais. Algumas pesquisas propõem a combinação do MDA com o AOD como meio para lidar com os interesses

transversais e outras propõem a combinação do MDA com o POA para lidar com as decisões pertinentes à arquitetura. O PROGERAS poderá ser usado como uma extensão ao MDA, adicionando os benefícios associados ao AOD e ao POA.

Apesar de o MDA apresentar uma lacuna quanto ao projeto da arquitetura, as ferramentas que o utilizam propiciam vários benefícios para os desenvolvedores de software. Há uma discussão sobre a adoção do MDA como meio de aumento de produtividade e também é apresentada uma avaliação baseada no uso do OptimalJ. Há uma lista extensa de ferramentas para geração de programas-fonte, compatíveis com o MDA e MDD. Algumas delas, como o CodeSmith, .netTiers, AndroMDA, MyGeneration, dentre outras, foram testadas durante o a concepção do processo para geração automatizada de programas fontes, objeto do presente pedido, doravante denominado PROGERAS, como meio para automatizar etapas do processo. Porém todas elas esbarram na limitação de não deixar explícitas as escolhas arquiteturais feitas pelos geradores ao criar os programas fonte e fornecem suporte limitado para que o arquiteto possa influenciar essas escolhas. Este tipo de abordagem carece de um mecanismo formal, padronizado e flexível que permita ao arquiteto indicar o que deve ser gerado, já que cada software tem necessidades especificas e não deveriam ser geradas nem mais nem menos linhas de programas e rotinas que o necessário.

BREVE DESCRIÇÃO DAS FIGURAS.

A figura 1 mostra esquematicamente o fluxograma do processo de geração de programas fonte, objeto do presente pedido, com o detalhamento e a marcação do modelo e geração do código;

A figura 2 mostra o processo de geração de programas-fonte de um software usando um gerador;

A figura 3 exibe diagrama de contexto da ferramenta de geração de código com base no PROGERAS;

A figura 4 mostra projeções do modelo de domínio por interesse arquitetural;

A figura 5 mostra *Templates* (Modelos padrões) implementados no protótipo da ferramenta de geração de programas-fontes (batizado de Genesys) criado para demonstrar o PROGERAS;

A figura 6 mostra um exemplo do Modelo de Domínio do Sistema de Reembolso de Despesas;

A figura 7 exibe um exemplo do Diagrama de Componentes do Sistema de Reembolsos;

A figura 8 mostra um Diagrama de Pacotes descrevendo as camadas do Sistema de Reembolsos;

DESCRIÇÃO DETALHADA DA INVENÇÃO

Processo para Geração Automatizada de Softwares

A arquitetura de um software consiste em uma estrutura ou estruturas do sistema que, por sua vez, se formam pelos elementos do software, das propriedades visíveis externamente demonstradas por esses elementos e nas relações entre eles. As necessidades do negócio vão determinar os atributos de qualidade que a arquitetura do sistema precisará apresentar. Somadas a outros requisitos técnicos (sistema operacional, linguagem, dentre outros) vão orientar o arquiteto na identificação dos interesses arquiteturais relevantes e na seleção das abordagens técnicas que serão utilizadas (padrões de projeto, algoritmos, estruturas de dados, dentre outras). Essas características técnicas são transversais aos requisitos funcionais, que descrevem as capacidades, serviços e o comportamento do software. O mapeamento das funcionalidades do sistema na estrutura de software vai determinar o suporte da arquitetura aos requisitos de qualidade.

A arquitetura de um software decorre de múltiplos interesses, alguns associados às funções de infra-estrutura (persistência, geração de *logs*, tratamento de exceções, dentre outros) e outros associados às funções de negócio, sendo que os primeiros são construídos

para suportar o segundo. A forma como os interesses arquiteturais são especificados varia em função dos atributos de qualidade que o software precisa atender. Cada interesse pode ser implementado por meio do uso de táticas arquiteturais bem conhecidas (padrões de projeto), que permitem maximizar certos atributos de qualidade. Um padrão de projeto pode estar associado a várias táticas, e vice-versa. Um padrão de projeto também pode estar associado a mais de um interesse, e vice-versa. Por exemplo, numa implementação o desenvolvedor pode criar um mecanismo para gerar trilha de *log* (para suportar a segurança) ou trilha de testes (para suportar a facilidade de ser testado).

No processo de análise o arquiteto terá que compreender os requisitos envolvidos numa implementação e fazer as escolhas de quais combinações de táticas deverão ser aplicadas para que o sistema possa atingir os seus objetivos de qualidade e de funcionalidade. No PROGERAS os padrões de projeto selecionados para implementação de cada interesse arquitetural são indicados no próprio modelo, por meio de um mecanismo de marcação. O método de escolha das táticas está fora do escopo deste processo, no qual é considerada apenas a marcação e a transformação de modelos. Posteriormente uma ferramenta (um gerador de programas fontes) vai ler os modelos PIM marcados e transformá-los para programa fonte usando os mecanismos de combinação aspectual.

Requisitos do Processo

Quanto ao mecanismo de marcação de modelos e geração de programas, o processo foi concebido para atender aos seguintes requisitos:

- Usar uma estrutura padronizada para marcação dos modelos com base no recurso de *tagged values*, estereótipos e anotações do UML.
- Usar ferramentas para automatizar o processo de geração de programas fonte.
- Projetar a estrutura do software a ser gerado com base em interesses arquiteturais e padrões de projeto. Os padrões são escolhidos para atender a certos atributos de qualidade que o software precisa respeitar.
- Implementar cada interesse arquitetural por meio da aplicação de um ou mais padrões de projetos.

- Os interesses arquiteturais podem atravessar uns aos outros, sendo assim, o processo deve ter mecanismos para combinar dois ou mais interesses e, por conseguinte, dois ou mais padrões de projeto, no processo de geração de programas fonte.

- Para cada padrão de projeto há um ou mais *templates*, sendo eles ativados em função das marcas inseridas no modelo de entrada. Cada marca indica um interesse junto e um padrão de projeto.

- Os *templates* podem receber parâmetros de entrada para direcionar a ferramenta de geração de programas fonte, por exemplo, uma *string* de conexão, um *flag* indicando se um determinado atributo de uma classe é persistente ou não, dentre outros. A regra de marcação deve permitir que seja indicado o interesse arquitetural, o padrão de projeto e também os parâmetros. Juntos, esses elementos compõem um aspecto que será aplicado ao modelo para geração dos programas pertinentes.

- O processo não se limita a um estilo arquitetural específico, como LAYER ou FILTERS AND PIPES ou BLACKBOARD, por exemplo. O estilo arquitetural é imposto por meio dos padrões de projeto que o arquiteto aplica aos modelos, por meio das marcações.

- O processo não deve limitar-se a uma linguagem, plataforma ou *framework* específico.

- O processo deve poder ser usado em conjunto com diferentes metodologias, como RUP, XP, Scrum, dentre outras.

Um processo de desenvolvimento de software define um arcabouço para as tarefas necessárias para construir um sistema. O processo fornece um roteiro, composto por uma série de passos previsíveis, que ajuda o desenvolvedor a criar, dentro do tempo especificado, um produto de software com qualidade esperada. Os desenvolvedores adaptam o processo às necessidades específicas de cada projeto. Tipicamente um processo é composto pelos seguintes tipos de atividades:

- **Comunicação**: atividades associadas à comunicação com o cliente e levantamento de requisitos. Por exemplo, identificação e reuniões com os interessados no projeto, documentação das características e funcionalidades desejadas, priorização dos requisitos, investigação das restrições e limitações que serão colocadas no sistema, dentre outras.

- **Planejamento**: atividades para estabelecer um plano de trabalho, que descreve as tarefas, os riscos, os recursos, produtos de trabalho, dentre outros elementos.

- **Modelagem**: atividades pertinentes à criação de modelos. Basicamente existem duas atividades de modelagem, a análise e o projeto técnico. As tarefas de análise envolvem, dentre outras, levantamento, elaboração, negociação, especificação e validação de requisitos para criação de modelos de análise. As tarefas técnicas envolvem, dentre outras, criação do modelo de dados, projeto arquitetural e de componentes.

- **Construção**: atividades de geração de código (manual ou automático) e os testes necessários.

- **Implantação**: atividades associadas à entrega do software ao cliente, que faz avaliação e fornece *feedback*.

No PROGERAS as atividades têm foco no trabalho técnico de engenharia, pertinentes à modelagem e construção. Sendo assim, para obter-se um melhor resultado este processo pode ser combinado com outro que o complemente, como o SCRUM ou PMI/PMBook, que tem enfoque nas atividades de gerenciamento e comunicação – ambos se completariam.

Passos do Processo

O processo PROGERAS consiste de uma sequência de passos para obterem-se programas fonte de forma automática, por meio do uso de uma ferramenta, a partir de um diagrama de classes UML. Os passos básicos do processo estão ilustrados na figura 1.

Passo 1 – Levantar Requisitos. A primeira tarefa consiste em identificar os requisitos do software que será desenvolvido: requisitos funcionais, não-funcionais e técnicos. No

PROGERAS não é definido como estes requisitos devem ser levantados. O trabalho pode ser feito por meio de casos de uso, cenários, histórias ou por meio de qualquer outro mecanismo. Este trabalho normalmente é executado por um Analista de Negócios.

Passo 2 – Criar o Modelo de Domínio. O próximo passo é elaborar o Modelo de Domínio, por meio de um diagrama de classes UML, no qual são indicadas as entidades de negócio e as relações entre elas. Este trabalho é executado manualmente por um Analista de Sistemas.

Passo 3 – Projetar Arquitetura. O projeto da arquitetura é elaborado por meio da identificação e especificação dos interesses arquiteturais. Para cada interesse arquitetural deve ser associado um padrão de projeto que permita que o software possa atingir os objetivos de qualidade especificados. No PROGERAS, para cada padrão de projeto/interesse haverá um *template* para geração de programas fonte. O Projeto da Arquitetura é executado por um Arquiteto de Software.

Passo 4 – Marcar o Modelo de Domínio. O próximo passo consiste em fazer a marcação do Modelo de Domínio. Os elementos do diagrama de classes (domínios, classes, atributos e operações) devem ser marcados com base no projeto da arquitetura e no padrão de Regras de Marcação do PROGERAS. As marcações indicam o interesse arquitetural e o padrão de projeto selecionado para sua implementação. Juntas, essas duas informações vão permitir que a ferramenta selecione e execute o *template* correspondente. Este trabalho também é executado pelo Arquiteto de Software.

Passo 5 – Gerar programas fontes. Depois que os diagramas forem marcados, eles serão processados por uma ferramenta, que receberá como entrada, um ou mais diagramas de classes (com marcações) correspondentes ao modelo PIM e, como saída, vai gerar tanto programas fonte numa linguagem específica quanto outros modelos mais detalhados que demonstram a arquitetura do sistema após a aplicação dos padrões de projeto. A ferramenta vai processar os modelos e criará outros modelos novos, específicos para cada interesse, que detalham a arquitetura do software. Será criado um modelo PIM para cada interesse arquitetural (persistência, apresentação, etc.). Esses modelos mostram a estrutura da arquitetura, não considerando a tecnologia que será utilizada na implementação. Este

trabalho também é executado pelo Arquiteto de Software. Estas entradas e saídas estão ilustradas na figura 2, na qual é mostrado o diagrama de contexto de uma ferramenta de geração de código compatível com o PROGERAS.

Ao processar as marcas associadas ao PROGERAS, o interesse arquitetural e o padrão de projeto indicados vão orientar a ferramenta na seleção e na execução do *template* pertinente. Um elemento, como uma classe, pode conter múltiplas marcações, cada uma associada a um *template* diferente. Por exemplo, uma classe pode ter uma marcação para gerar o programa fonte associado à classe da camada de negócio (interesse de negócio), outra para persistência (interesse de persistência) e outra para gerar telas de pesquisa e edição (interesse de apresentação).

Dessa forma a ferramenta, por meio dos *templates*, vai gerar os programas fonte de infra-estrutura da aplicação e depois as regras de negócio deverão ser inseridas manualmente por um programador. Para isso, o programador deverá inserir novas linhas de programa, em áreas protegidas, nos arquivos gerados pela ferramenta ou criar novas classes para herdar as geradas pela ferramenta. Quando o programa fonte for regerado, a ferramenta preservará o conteúdo inserido manualmente, demarcado por essas áreas.

Passo 6 – Adicionar linhas de programa manualmente. A ferramenta pode não gerar todo o programa fonte e um programador terá concluir o trabalho manualmente. Nesta etapa é necessário assegurar que o código adicionado manualmente não seja substituído pelo código regerado. Para isso as novas linhas de programa devem ser adicionadas em áreas protegidas, demarcadas nos arquivos gerados pela ferramenta.

Passo 7 – Gerar o software executável. Finalmente os programas gerados pela ferramenta, mais aqueles construídos manualmente por um programador devem ser combinados num projeto e compilados para dar origem ao aplicativo executável.

Por meio do processo PROGERAS é proposto um complemento à visão MDA, cujo foco é definir padrões para modelagem e transformação de modelos. Neste processo é sugerida uma estrutura padronizada de regras de marcação, para indicar os padrões que serão usados na implementação de cada interesse arquitetural e uma seqüência de passos a serem seguidos para marcar o modelo e transformá-lo em programas fonte.

Ferramenta de Geração de Programas Fonte (robôs)

No PROGERAS as ferramentas de geração de programas fonte também são chamadas de robôs. Estas ferramentas recebem como entrada diagramas UML marcados e como saída geram programas fonte.

Alguns interesses arquiteturais, como persistência e auditoria, podem se cruzar. Por exemplo, pode ser necessário gerar *log* de auditoria sempre que objetos de determinada classe forem persistidos. Neste caso estão sendo cruzados os interesses de persistência e geração de *log*. A ferramenta precisará ter mecanismos para combinar dois ou mais padrões de projeto, por meio da combinação dos *templates* pertinentes. O AOD fornece um mecanismo para essa finalidade, que é o uso de pontos de junção e adendos. No PROGERAS os *templates* são tratados como adendos, e dentro de cada *template*, devem ser definidos pontos de junção. A ferramenta fará a combinação aspectual por meio da combinação dos *templates*.

Na figura 2.2 está ilustrada a estrutura de um gerador. A ferramenta vai ler o diagrama de classes com as marcações, gravado em formato XMI (conforme definido no padrão MDA), e armazená-lo internamente, por meio de um grafo de objetos. Esse modelo será processado e convertido, também por meio de um módulo de transformação, em um novo modelo. Para realizar essa tarefa, a ferramenta percorrerá todos os *tagged values* de cada elemento (domínios, classes, atributos, relações e operações) para identificar aqueles pertinentes ao PROGERAS. A ferramenta vai aplicar o padrão de projeto indicado para cada interesse. Ao aplicar o padrão de projeto, novas classes e domínios serão criados, dando origem a um novo modelo. Posteriormente esse novo modelo será processado pelo gerador de código que produzirá os programas fonte para uma plataforma e linguagem específicas. Depois os programas gerados automaticamente e os criados manualmente pelo desenvolvedor serão combinados e compilados para gerar o aplicativo completo.

Uma ferramenta de geração de programas fonte compatível com o PROGERAS trabalhará com múltiplos *templates*, um para cada padrão de projeto e interesse arquitetural. Por exemplo, para o interesse de persistência, haverá um *template* que implementa o padrão de projeto DAO com acesso ao banco de dados via ODBC e outro para o DAO que usa a API do *driver* nativo. A mesma abordagem pode ser aplicada a

outros interesses arquiteturais, como auditoria e apresentação. O *template* pode gerar programas fonte de auditoria para gravar *log* em arquivo TXT ou XML. O interesse de apresentação pode ser implementado com MVC, mas pode haver um *template* para web e outro para aplicações *desktop*. Cada *template* está associado a um padrão de projeto diferente. Estes elementos estão representados na figura 2.3.

Regras de Marcação

No PROGERAS, as marcas indicam um interesse arquitetural, como proposto no AOD, e o padrão de projeto que será usado na sua implementação, como proposto no POA. As marcas são inseridas no modelo por meio de *tagged values* ou anotações e têm a forma **Variável := Expressão.** A variável indica o interesse (aspecto arquitetural) e a expressão indica o mecanismo de implementação que pode ser um valor atômico ou pode conter um conjunto de um ou mais parâmetros, separados por ponto e vírgula. Cada parâmetro tem a forma **Parâmetro = Valor**, representando o nome do parâmetro e o valor correspondente. Uma marca tem a seguinte estrutura:

```
aspAspecto:={ Pattern=NOME;[outros parâmetros do pattern]}
```

Seguem alguns exemplos de marcas do PROGERAS:

```
aspPresentation:={Pattern=MVCWEB; Operations=CRUD;
Screen=Form}

aspPersistency:={Pattern=DAODB}

aspAuditity:={Pattern=LOGTXT;Layer=DAL}
```

No primeiro exemplo, a marca está associada ao interesse de apresentação e indica que para esse interesse deve ser aplicado o padrão de projeto MVCWEB (MVC para telas html). Essa marca, que deve ser aplicada às classes, tem dois parâmetros: o *Operations*, que indica as operações de CRUD (*Create, Read, Update* e *Delete*) que devem ser ativadas na tela, e o *Screen* que indica o formato da tela (**Screen=Form** para formulário e **Screen=Grid** para tabela).

O próximo exemplo é uma marcação do interesse de persistência, que será implementado por meio do padrão de projeto DAO, usando banco de dados relacionais.

Neste padrão, para cada classe marcada como persistente no Modelo de Domínio, será criada uma classe responsável por ler e gravar os objetos correspondentes no banco de dados na camada de persistência. O exemplo indica que será aplicado o padrão de projeto DAO para bancos de dados relacionais (*Pattern*=DAODB), mas poderiam ser usados outros padrões aplicando DAO para XML, para CSV, dentre outros. O *template* DAO pode ter múltiplos parâmetros, como os exemplos indicados na tabela 2.1.

No Modelo de Domínio são inseridas marcas para indicar os interesses arquiteturais relevantes no projeto da aplicação e os padrões de projeto aplicados na implementação de cada um deles. Quando uma ferramenta processa o Modelo de Domínio e cria novos modelos, um para cada interesse arquitetural, uma entidade do modelo aparece em um ou mais modelos. Uma entidade pode ser marcada com mais de um interesse, como mostrado na figura 4, por exemplo:

- Interesse negócio, para indicar que é uma entidade de negócio.
- Interesse de persistência, para indicar que será persistida no banco de dados.
- Interesse de apresentação, para indicar que será visualizada e atualizada por meio de telas (interface com usuário).
- Interesse de monitoração, para indicar que as operações de construção e destruição de objetos serão monitoradas.
- Dentre outros.

Quando um *template*, é construído, nas suas definições deve ser indicado o nome do padrão de projeto associado, o nome do interesse arquitetural, a lista de parâmetros permitidos e a relação dos arquivos com os *templates* de programas fontes.

Na figura 5 estão indicados alguns exemplos de interesses arquiteturais e padrões de projeto. Para cada interesse arquitetural pode haver uma ou mais implementações de padrões de projeto. Cada implementação está associada a um *template*, que será usado pela ferramenta para geração de programas fonte. Por exemplo, existem três implementações persistência: DAODB, DAOXML e DAOCSV. O *template* DAODB gera código para persistência de bancos de dados relacionais usando o padrão de projeto DAO

(*Data Access Objects*), o DAOXML gera código para persistir objetos em arquivos XML e o CSV em arquivos CSV (*Coma-Separated Values*).

O nome do interesse arquitetural será atribuído arbitrariamente pelo desenvolvedor que constrói o *template*. Contudo, a estrutura dos nomes deve ser padronizada, para facilitar a sua memorização e o trabalho de marcação de modelos. Devem-se evitar dois nomes diferentes para o mesmo interesse arquitetural, pois isso geraria ambigüidade e dificultaria o uso do processo e da ferramenta de geração de programas fontes.

Na figura 5 há algumas sugestões de nomes, que são compostos pelo prefixo "asp" (de "aspecto") seguido do nome do interesse. Abaixo, foi usada a língua inglesa para que o processo possa ser entendido mais facilmente por desenvolvedores de outros países.

Classe	aspPersistency	{ Pattern = DAODB; [Operations = **[C][R][U][D]**; ReadAll=Yes|No; AccessMethod=DML|USP] } **Operations** → operações que serão criadas no programa fonte, pode ser ***Create***, ***Read***, ***Update*** e ***Delete***. **ReadAll** → indica que a entidade terá a opção de ler todos os objetos do repositório numa única consulta. **AccessMethod** → indica o método de acesso aos registros das tabelas do banco de dados. Pode ser DML (*Data Manipulation Language*) para comandos SELECT ou USP (*User Stored Procedures*) para acesso por meio de *Procedures* SQL.
Atributo	aspPersistency	{ *Pattern* = NONE } Para que um atributo seja tratado como não persistente, é necessário desassociar o padrão de projeto associado à persistência.
Domínio	aspPersistency	{Pattern = DAODB} **Pattern** → nome do padrão de projeto. Quando um domínio recebe essa marcação, então todas as classes contidas nele serão tratadas como persistentes. Se uma delas não for pertinente, então ela deverá ser marcada com *Pattern* = NONE.

Tabela 1 – Exemplo de parâmetros de um *template* para gerar código de persistência.

Conclusão

O PROGERAS aborda a transformação de modelos PIM marcados para programas fonte, aplicando uma transformação direta (não são geradas versões intermediárias de modelos). O processo consiste em uma seqüência de passos para se obter programas de forma automática, por meio do uso de uma ferramenta a partir de um diagrama de classes UML marcado.

No PROGERAS, o desenvolvimento começa com a elaboração do Modelo de Domínio e a próxima tarefa consiste da criação da estrutura do software. Essa estrutura será projetada tomando como referência os interesses arquiteturais relevantes, que podem ser identificados e projetados com base nos requisitos não funcionais que a aplicação precisa atender, o que pode ser feito por meio de cenários.

Os elementos do diagrama de classes (domínios, classes, atributos e operações) são marcados para indicar os padrões de projeto que se deseja utilizar no projeto da arquitetura. As marcações indicam o interesse arquitetural e o padrão de projeto que será utilizado. Juntas, essas duas informações vão permitir que uma ferramenta selecione e execute o *template* correspondente.

Uma ferramenta de geração de programas fonte compatível com o PROGERAS poderá trabalhar com múltiplos *templates*, um para cada padrão de projeto. Por exemplo, para o interesse arquitetural de persistência, haverá um *template* para implementar o padrão de projeto DAO com acesso ao banco de dados via ODBC e outro para o DAO que usa a API do *driver* nativo. No capítulo seguinte é mostrado um exemplo de uso do processo e das regras de marcação.

EXEMPLOS

Exemplo de Uso do Processo de Geração Automatizada de Programas Fonte

A descrição a seguir mostra um exemplo de aplicação do processo PROGERAS. Este exemplo consiste de um software real, que foi construído para a Compugraf® Comunicação Empresarial para automatizar o processo de reembolso de despesas de funcionários. Nesta empresa os funcionários podem fazer reembolsos de despesas e de estudos. As despesas reembolsadas são aquelas que os funcionários incorrem

ao atender um cliente ou fornecedor, como deslocamento, hospedagem, refeição, dentre outras. A empresa também reembolsa cursos universitários, línguas e cursos técnicos.

O exemplo aqui apresentado foi simplificado (devido á restrição de tamanho deste texto), o projeto original aborda também reembolso para terceiros (prestadores de serviços), reembolso de deslocamentos com veículo próprio, notas de débito entre empresas, múltiplas alçadas de aprovação, dentre outras funções. O sistema original tem aproximadamente 40 entidades de negócio.

1.1. Contexto

A empresa tem aproximadamente 200 funcionários, que estão espalhados em quatro filiais, dispersas geograficamente. Três delas estão no estado de São Paulo e uma no Rio de Janeiro. Os funcionários de todas as filiais podem fazer reembolsos.

1.1.1. Despesas

Depois de pagar com recursos próprios uma despesa para o atendimento de um cliente ou fornecedor, o funcionário deve prestar contas à empresa para receber de volta o valor gasto. Para isso é necessário apresentar o "Relatório de Prestação de Contas de Despesas" com os seguintes dados:

- Identificação do funcionário
- Identificação do cliente ou fornecedor
- Identificação do projeto (opcional)
- Descrição da finalidade da despesa
- Relação dos comprovantes associados (cupons ficais, notas fiscais, recibos, etc.)

O funcionário deve apresentar este relatório, mais os comprovantes em anexo, ao seu gerente para aprovação, por meio de uma assinatura. Na ausência de um gerente um diretor da empresa também pode aprovar. Depois de aprovado o relatório deve ser encaminhado à área financeira, que vai contabilizar as despesas (por tipo: refeição, combustível, estacionamento, etc.) e provisionar o pagamento. Caso algum comprovante não seja válido, o relatório poderá ser recusado pela área financeira e o funcionário terá

que refazê-lo. Os pagamentos ocorrem duas vezes por semana, às terças-feiras e às quintas-feiras.

Caso o funcionário conheça antecipadamente o valor da despesa, ele poderá pedir um adiantamento em dinheiro. Este pedido é feito por gerente ou diretor usando um formulário específico, no qual são indicadas as seguintes informações:

- Identificação do funcionário
- Descrição do motivo da despesa
- Valor

O formulário deve ser encaminhado ao departamento financeiro, que vai provisionar o pagamento. Depois o funcionário terá que prestar contas por meio do "Relatório de Prestação de Contas de Despesas". Caso o funcionário ainda seja credor, ele receberá o valor da diferença, caso ele seja devedor ele terá que devolver o saldo.

1.1.2. Estudos

A empresa tem a política de reembolsar despesas com estudos para todos os funcionários que tenham no mínimo seis meses de registro. O padrão é reembolsar o dois terços do valor do curso, sendo que a somatória dos reembolsos de um mês não pode ultrapassar 20% do valor do salário bruto do funcionário. A critério do gerente, o curso pode ser reembolsado integralmente e o valor não será abatido da bolsa de estudos (20% do salário).

Somente determinados tipos de cursos podem ser reembolsados, que são aqueles associados à função que o funcionário desempenha na empresa. Podem ser cursos universitários, cursos de línguas e cursos técnicos. Cabe ao gerente do funcionário avaliar o curso e autorizar o reembolso, na ausência do gerente um diretor pode fazer a aprovação. Para ter direito ao reembolso, quando o funcionário planejar fazer um curso ele deverá antes preencher um "Formulário de Solicitação de Reembolso de Estudo", no qual deverá informar os seguintes dados:

- Identificação do funcionário

- Descrição do curso
- Instituição
- Duração do curso
- Datas e valores das parcelas
- Percentual de reembolso
- Valor a ser abatido da bolsa de estudos

O funcionário deve apresentar este formulário ao seu gerente para aprovação, por meio de uma assinatura. Na ausência de um gerente um diretor da empresa também poderá aprovar. Ao aprovar o gerente deverá indicar no formulário se o reembolso deverá ser abatido da bolsa de estudos e se o reembolso é integral (e não de dois terços). Depois o formulário deverá ser encaminhado ao departamento pessoal. Somente cursos previamente aprovados poderão ser reembolsados.

Posteriormente, o funcionário deverá efetuar o pagamento de cada parcela com recursos próprios e depois solicitar o reembolso. Para receber o reembolso, o funcionário terá que preencher o "Relatório de Solicitação de Reembolso de Estudos", indicando as seguintes informações:

- Identificação do funcionário
- Descrição do curso
- Instituição
- Data e valor da parcela

O formulário deverá ser encaminhado ao departamento pessoal, que calculará o valor do reembolso, considerando a bolsa mensal e os valores previamente aprovados pelo gerente do funcionário (ou um diretor). O departamento pessoal encaminhará para o departamento financeiro uma solicitação para que o pagamento seja contabilizado e provisionado.

1.2. Requisitos

O processo PROGERAS começa com o levantamento dos requisitos funcionais, não-funcionais e tecnológicos, que vão orientar o arquiteto na especificação da arquitetura e, por conseguinte, na marcação do modelo.

1.2.1. Requisitos Funcionais

Considerando a sistema de reembolsos, o software deverá apresentar as seguintes funcionalidades:

- Consultar o banco de dados de funcionários (identificação, departamento, gerente, salário, data de contratação, conta corrente).
- Prover mecanismo para um gerente ou diretor solicitar um adiantamento para o funcionário.
- Prover mecanismo para o departamento financeiro contabilizar e provisionar o pagamento do adiantamento ao funcionário.
- Prover mecanismo para o funcionário fazer a prestação de contas de despesas.
- Prover mecanismo para o gerente do funcionário, ou um diretor da empresa, aprovar um relatório de prestação de contas.
- Prover mecanismo para o departamento financeiro contabilizar e provisionar o pagamento de despesas ao funcionário.
- Prover mecanismo para o funcionário solicitar a aprovação de um curso.
- Prover mecanismo para o gerente do funcionário, ou um diretor, aprovar o curso, o valor do reembolso (dois terços ou integral) e indicar se o curso será abatido da bolsa ou não.
- Prover mecanismo para o departamento pessoal controlar o uso da bolsa de estudos pelos funcionários.
- Prover mecanismo para o funcionário solicitar o reembolso de um curso.

- Prover mecanismo para o departamento financeiro contabilizar e provisionar o pagamento de um curso ao funcionário.
- Prover mecanismo para parametrização dos reembolsos: percentual da parcela, percentual do salário, datas de pagamento, dentre outras.

1.2.2. Requisitos Não-funcionais

O sistema deve respeitar aos seguintes requisitos:

- Funcionalidade:
  - Precisão – os valores dos reembolsos devem ser calculados com precisão de duas casas decimais.
  - Interoperabilidade – o sistema deverá ser integrado ao sistema de contabilidade e folha de pagamento, sem denegrir o desempenho ou a segurança destes sistemas, para consulta e atualização de dados.
  - Segurança – deve haver um controle de segurança parametrizável de modo que somente as pessoas autorizadas possam aprovar reembolsos e pagamentos. Somente os usuários autorizados poderão visualizar os salários dos funcionários da empresa e alterar os parâmetros do sistema.
- Confiabilidade:
  - Recuperabilidade – em caso de falha o sistema não poderá ficar fora do ar por mais que 24 horas.
- Facilidade de Uso (Entendimento, Aprendizado, Operação):
  - O sistema será utilizado por todos os funcionários da empresa, espalhados geograficamente. Os funcionários não poderão ser treinados individualmente no uso do sistema, sendo assim este deve ser de fácil uso e aprendizado.
- Facilidade de Manutenção:

- Facilidade de ser Analisado – o sistema será instalado nos terminais de todos usuários espalhados geograficamente. No caso de algum erro normalmente não será possível enviar um desenvolvedor ao local para analisar e resolver o problema. Desta forma o sistema precisará gerar logs que possam ser enviados para área de TI para que possam ser analisados, que indiquem as operações efetuadas pelo sistema mais os dados associados.

- Facilidade de mudança – na fábrica de software da Compugraf® trabalham aproximadamente 30 desenvolvedores (na data de elaboração desde texto) e qualquer um deles deve poder fazer eventuais manutenções neste sistema.

1.2.3. Requisitos Tecnológicos

A empresa já possui outros sistemas em produção, e o sistema de reembolsos deverá ser compatível com a plataforma existente:

- Banco de Dados MS-SQL Server® 2005.
- Ser programado em C# com .NET 3.5.
- Usar a tecnologia WPF para construção das telas e respeitar o padrão de cores, fontes, dentre outros da empresa (no .NET é chamado de tema).
- Ser distribuído através intranet usando a tecnologia *ClickOnce*.
- Poder ser executado em estações com Windows XP®, 2MB de memória e usar no máximo 2M do disco rígido.
- Fazer controle de acesso por meio dos dados cadastrados no *Active Directory*.

1.3. Modelo de Domínio

O modelo de domínio, representado na figura 3.1, é compostos pelos domínios "Funcionário", "Reembolso" e "Financeiro". No do domínio "Funcionário" estão as entidades associados aos dados dos funcionários que fazem reembolso e aos chefes que aprovam:

- Funcionario: um funcionário da empresa, pode ser chefe ou não. Nesta tabela existem os atributos nome, departamento, cargo, salário, data de contratação, gerente do departamento.

- Empresa: dados da empresa onde o funcionário está registrado. A Compugraf® é composta por cinco diferentes razões sociais.

- DadosFuncionario: dados do funcionário específicos ao processo de reembolso de despesas e estudos, como valor mensal da bolsa de estudos, percentual de reembolso por curso, e se o funcionário está autorizado a fazer reembolso de estudos.

- CursoFuncionario: registro de um curso do funcionário, para que ele possa fazer o reembolso de estudos.

- ParcelaCurso: parcelas do curso, para o funcionário solicitar reembolso

No domínio "Reembolso" estão as entidades associadas às solicitações de reembolso e as aprovações:

- TipoDespesa: tipos de despesas para solicitação de reembolso, como combustível, hospedagem, taxi, refeição, dentre outras.

- Projeto: projetos executados pela empresa, aos quais podem estar associados uma ou mais despesas de funcionário.

- ComprovanteDespesa: registro de um comprovante de despesa, como um cupom fiscal ou uma nota fiscal. Cada comprovante deve estar associado a um tipo específico de despesa.

- Reembolso: solicitação de reembolso de despesa, que deve estar associado a um ou mais comprovantes quando se tratar de reembolso de despesas ou a uma parcela de curso, quando for reembolso de estudos. Quando for uma despesa pode estar associado a um projeto.

- Adiantamento: registro de um adiantamento dado a um funcionário.

Finalmente há o domínio "Parametros", onde estão as entidades associadas à parametrização do sistema:

- ParametrosCalculo: parâmetros para o cálculo dos valores de reembolso de estudos e datas de pagamento, e para contabilização dos reembolsos.

1.4. Projeto da Arquitetura

O Sistema de Reembolsos deverá ser integrado com outros três sistemas, descritos na figura 3.2:

- **Contabilidade**: integração por meio de um WebService, no qual há um serviço para enviar para a contabilidade os dados de um pedido de reembolso ou adiantamento.

- **Folha de Pagamento**: neste sistema estão contidas, dentre outras, as entidades funcionário e departamentos. A integração deverá ser feita por meio de uma DLL, na qual estão contidas os objetos de negócio deste sistema. O Sistema de Reembolsos vai instanciar objetos e chamar os métodos pertinentes.

- **Projetos**: neste sistema estão as entidades de projetos, itens do projeto, condições de pagamentos dentre outras. A integração será feita da mesma forma que com a Folha de Pagamento.

O Sistema de Reembolsos será estruturado em camadas, por meio do padrão de projeto LAYERS. Este padrão arquitetural é usado para dividir a arquitetura em camadas, de modo a facilitar a implementação e manutenção do sistema. Criando blocos com baixo acoplamento e alta coesão. As camadas são implementadas para se comunicarem umas com as outras respeitando uma seqüência específica de mensagens. Num projeto bem construído uma camada A usa os serviços da camada B, mas a camada B não usa os serviços de A – comunicação flui numa única direção. O Sistema de Reembolsos será composto pelas seguintes camadas:

- *User Interface Layer* (UIL) ou Camada de Interface com Usuário – nesta camada estão encapsuladas as telas do sistema, usadas pelos usuários.

- *Application Layer* (APL) ou Camada de Aplicação – nesta camada estão os objetos e serviços específicos da aplicação, como mecanismos para geração de relatórios, função de login dos usuários, dentre outros.

- *Business Rule Layer* (BRL) ou Camada de Regras de Negócio (também conhecida como Camada de Serviços de Negócio) – nesta camada são providas as funcionalidades administrativas e operacionais que o sistema precisa suportar. Por exemplo, criação de reembolso, aprovação, concessão de adiantamentos, dentre outros.

- *Business Entitties Layer* (BEL) ou Camada de Entidades de Negócio – nesta camada residem as entidades de negócio com as quais os serviços de negócio operam. Por exemplo Reembolso, Comprovante, Funcionário, dentre outros.

- *Data Access Layer* (DAL) ou Camada de Persistência de Objetos – nesta camada estão implementadas as funções de leitura e gravação de objetos de negócio, que estão implementados na camada BEL.

- *Infra-structure Layer* (IFL) ou Camada de Infra-estrutura – nesta camada estão implementados serviços como tratamento de erros, geração de logs, controle de acessos (segurança) dentre outros que são utilizados pelas demais camadas.

- *Data Base Layer* (DBL) ou Camada de Banco de Dados – camada associada ao banco de dados, que no Sistema de Reembolsos será implementada por meio do MS-SQL Server®.

De modo a reduzir o acoplamento entre as camadas, cada uma delas será acessada pelas demais por meio de uma fachada, aplicando-se o padrão de projeto FACADE. Este padrão faz com que exista um único ponto de acesso, reduzindo o acoplamento entre as partes do sistema.

Com base nos requisitos apresentados (funcionais e não-funcionais) os seguintes interesses arquiteturais podem ser identificados: apresentação, persistência, controle de acesso, geração de logs e tratamento de erros. No PROGERAS não há imposição de um conjunto padrão de passos para identificação dos interesses arquiteturais,

neste projeto foi usada uma abordagem para seleção de aspectos na programação com AspectJ. Cada interesse deve ser especificado e construído como um aspecto separado da arquitetura, que deverá ser implementado por meio de um padrão de projeto. Cada padrão de projeto é definido para maximizar ou minimizar algumas características da aplicação, como maximizar o desempenho, minimizar o uso de recursos, facilitar a manutenção, dentre outros, que são os requisitos não-funcionais. No PROGERAS também não é imposto um roteiro para seleção dos padrões, no projeto do Sistema de Reembolsos foi usada uma abordagem que propõe a criação de cenários de atributos de qualidade, para escolha dos mecanismos que serão usados na construção do software (implementação dos interesses arquiteturais).

O interesse de apresentação aplica-se a todas as entidades que poderão ser cadastradas pelos usuários, como o reembolso e os comprovantes, por exemplo. Para cada entidade que apresentar este interesse deverá ser criada uma ou mais telas, conforme a necessidade. As telas, que serão encapsuladas na Camada de Interface com Usuário, devem ser construídas por meio de um padrão de projeto específico, o MODEL-VIEW-CONTROLER ou MVC. Neste padrão estão claramente separadas a máscara da tela (VIEW), as funções de controle (CONTROLER) e as entidades de negócio (MODEL). O MVC simplifica as mudanças de aparência nas telas (*look-and-feel skins*), a personalização de preferências de usuários, e as mudanças estruturais. Por meio deste padrão estes objetivos podem ser atingidos sem que seja necessário fazer mudanças nas funções centrais do sistema e nas regras de negócio.

O interesse de persistência aplica-se às entidades que serão persistidas em banco de dados. Neste projeto o mecanismo de persistência que será implementado por meio do padrão de projeto DATA ACCESS LAYER (DAL) e DATA ACCESS OBJECTS (DAO). As classes que serão geradas para esta finalidade serão implementadas na Camada de Persistência. A camada de persistência (padrão de projeto DAL), tem a função de abstrair a tecnologia de banco de dados que será usada para a função de persistência dos objetos. O padrão de projeto DAO provê uma interface abstrata para algum tipo de banco de dados (neste exemplo o MS-SQL Server®) ou mecanismo de persistência, sem expor os detalhes técnicos associados. Para cada objeto de negócio persistente é criado uma ou mais classes nas quais estão implementadas as funções de persistência. A maior vantagem deste

padrão é proporcionar uma separação clara e simples das camadas de negócio e persistência, que podem evoluir independentemente. Por meio do DAO mudanças numa camada normalmente resultam em pequeno impacto na outra.

Por meio do interesse arquitetural de controle de acesso, que está associado ao requisito de segurança, deverá ser possível indicar quais usuários podem acessar cada entidade. Este interesse aplica-se somente às entidades críticas, como a entidade funcionário, por exemplo, onde estão registrados informações sigilosas como o salário de cada colaborador. Para este interesse deverá ser utilizado o padrão de projeto FIREWALL PROXY. Por meio deste padrão a permissão de acesso pode ser verificada cada vez que um serviço é requisitado, como os serviços providos pelas camadas de persistência e apresentação, por exemplo. As classes geradas ficarão contidas na Camada de Infra-estrutura.

O interesse de monitoração pode ser aplicado às classes mais acessadas, a fim de monitorar-se o volume de acessos de consultas, atualizações e inclusões. Desta forma será possível avaliar-se, em tempo de execução, a quantidade de recurso e processamento consumido. A monitoração deverá ser feita por meio do padrão HEART BEAT. Por meio deste padrão é possível monitorar uma tarefa específica, para saber se e como ela está funcionando. As classes geradas ficarão contidas na Camada de Infra-estrutura.

O interesse de tratamento de erros será utilizado em todo o sistema a fim de padronizar o mecanismo utilizado para lidar com as situações de erro. A função de tratamento de erros será implementada por meio dos padrões de projeto RECOVERY BLOCKS, MINIMIZE HUMAN INTERVENTION, MAINTENACE INTERFACE, SOMEONE IN CHARGE, ESCALATION e FAULT OBSERVER. Estes padrões provêem meios para detectar os erros, gerar *log* e notificar alguém sobre sua ocorrência. As classes geradas ficarão contidas na Camada de Infra-estrutura.

Finalmente, o interesse de geração de *logs* será utilizado pelos mecanismos associados aos demais interesses para gerar *logs* para futura análise, pelo administrador do sistema. Poderão gerar *logs*, por exemplo, das funções de tratamento de erros e de monitoração. A função de geração de *logs* deverá ser implementada por meio do padrão de

projeto LEADER/FOLLOWERS, que permite gerar *logs* com pouco impacto no desempenho do sistema. Estas funções serão implementadas na Camada de Infra-estrutura.

1.5. Marcação do Modelo

Os artefatos do Modelo de Domínio (domínios, classes, atributos e relações) serão marcados para indicar os padrões que serão usados na implementação de cada interesse arquitetural. Alguns exemplos de marcas associadas à classe "Reembolso", estão indicadas na tabela 2.

Todas as classes de negócio serão marcadas com o aspecto "aspBusinessEntity", mais o padrão "BEL", para gerar os programas fonte da camada de entidades de negócio (BEL). Este padrão tem um parâmetro adicional que é o rótulo (*label*) da classe, este rótulo será usado como título das telas e em mensagens de erro e *log*. Para os atributos marcados também existem outros parâmetros para indicar se o atributo aceita valor nulo (Null=true), rótulo, regras de consistência (*validation*) e tamanho do campo (*size*). Associado ao padrão "BEL" e interesse "aspBusinessEntity" deve haver um conjunto de *templates* que serão processados por uma ferramenta para gerar os programas fonte.

Por meio da parametrização do interesse de persistência o arquiteto vai indicar o mecanismo que deve ser usado para persistência dos objetos. Para usar o padrão DAO com banco de dados relacionais, é indicado o padrão DAODB com o parâmetro "Write=true" para indicar que este classe permite atualização. Para indicar que devem ser criados mecanismos para carregar objetos com base no atributo "TipoReembolso", este atributo é marcado com o parâmetro "ReadBy=true". Para indicar os mecanismos da camada de armazenamento de dados, que será encapsulada no banco de dados, indica-se o interesse arquitetural de armazenamento de dados. O padrão "DDL+DML" é usado para gerar os *scripts* para criação das tabelas e índices (DDL – *Data Definition Language*) mais os *scripts* para criar as *stored procedures* para fazer INSERT, DELETE e UPDATE nestas tabelas.

O mecanismo de controle de acesso é gerado na camada de infra-estrutura (IFL), por meio da marca associada ao interesse arquitetural "aspUserAccessControl", que será construído por meio do padrão de projeto FIREWALL PROXY. Na camada de

persistência de todas as classes marcadas com este padrão, será inserido uma verificação de permissão de acesso dos usuários, em todas as funções de leitura, atualização, exclusão e inclusão.

Classe	Negócio	aspBusinessEntity := {Pattern=BEL; Label=Pedido de Reembolso}
	Persistência	aspPersistency := {Pattern=DAODB; Write=true}
	Armazenamento de dados	aspDatabase := {Pattern=DDL+DML}
	Controle de acesso	aspUserAccessControl := {Pattern=UAC}
	Trilha para auditoria	aspTracking := {Pattern=LOGCG}
Atributo DataAprovacaoChefe	Negócio	aspBusinessEntity := {Null=true; Label=Data de aprovacao do chefe}
Atributo ValorReembolso	Negócio	aspBusinessEntity := {Validation=GreaterThanOrEqualTo(0)}
Atributo TipoReembolso	Negócio	aspBusinessEntity := {Size=2; Validation=List(DE,ES); Label=Tipo do reembolso}
	Persistência	aspPersistency := {Pattern=DAODB; ReadBy=true}

Tabela 2 – Exemplos de marcas associadas à classe "Reembolso"

O mecanismo de geração de *log* é especificado por meio do interesse "aspTracking". O padrão "LOGCG" foi definido na Compugraf® para indicar, em cada registro de cada tabela do banco de dados, os dados da última atualização feita no registro: nome do usuário, data e hora, identificação do sistema usado e motivo da atualização.

1.6. Geração dos Programas Fonte

Associado a cada padrão há um ou mais *templates*, que serão processados por uma ferramenta para gerar os programas fonte. Por exemplo, para o interesse de persistência, padrão DAODB, foram criados os seguintes *templates*:

- **DAO.cs** – este *template* é processado uma vez para cada entidade persistente, para criar a classe responsável pelas funções de persistência. O nome da classe gerada terá o prefixo "dao", como "daoReembolso.cs".

- **DaoFacade.cs** – este *template* é processado apenas uma vez para gerar uma fachada, por meio do padrão de projeto FACADE, para a camada de persistência (DAL). Na fachada serão gerados os métodos para carregar, atualizar, apagar e incluir objetos, como "Reembolso_ReadObject(long id)" e "Reembolso_WriteObject(beReembolso oReembolso)".

- **Repository.cs** – este *template* é processado apenas uma vez para gerar uma classe onde estão os métodos de acesso ao banco de dados. Tem métodos para conectar, desconectar, iniciar uma transação, dentre outros.

A ferramenta de geração de programas fonte vai receber como entrada as definições do projeto (linguagem de programação, diretórios onde os arquivos serão armazenados, dentre outras informações) e o diagrama de classes UML. Depois vai processar o diagrama e montar internamente um grafo com as classes e os interesses especificados. Para cada interesse a ferramenta vai elencar os padrões parametrizados e vai processar os *templates* associados. A estrutura de uma ferramenta está representada nas figuras 2.2 e 2.3.

Quando for necessário usar um padrão para o qual ainda não existem os *templates*, estes precisarão ser criados antes que os programas fonte sejam gerados. Pode ser necessário, inclusive, definir-se um novo interesse arquitetural, por exemplo, o interesse de distribuição, serialização de objetos, dentre outros.

1.7. Programação Manual

As regras de negócio precisarão de programadas, pois o PROGERAS permite especificar e gerar somente as rotinas de infra-estrutura. Por exemplo, para criar um objeto para reembolso de estudos, é necessário calcular o valor do reembolso (normalmente dois terços da parcela) e a somatória dos reembolsos já pagos no mês. Neste projeto partes do programa serão construídas manualmente: algumas telas, os relatórios e alguns *scripts* SQL (por exemplo os *scripts* de *backup*).

1.8. Conclusão

Acima, foi apresentado um exemplo de aplicação do PROGERAS, e foi demonstrado como o processo deve ser usado. Entretanto, fica implícito para aqueles versados na técnica, que o exemplo acima não constitui uma limitação da invenção, sendo observado que o processo para a geração de programas fonte, aqui denominado PROGERAS, permite que seja possível especificar processos de negócio, de modo que as rotinas associadas também possam ser geradas automaticamente, sendo a sua limitação unicamente de acordo com as reivindicações anexas.

Por meio do processo PROGERAS é proposto um processo de geração automática de programas fonte com base no MDA. Para orientar o processo de geração de código, são usadas regras de marcação padronizadas, inseridas em modelos UML por meio de *tagged values*, extensões e anotações. Essas regras indicam o interesse arquitetural e o padrão de projeto que deverá ser usado ao gerar os programas fonte. O PROGERAS é proposta de extensão ao MDA, aplicando os conceitos do AOD e do POA para gerar programas fonte de forma automatizada a partir de um diagrama de classes (PIM). Com essa proposta, acredita-se ser possível não só obter um elevado nível de automação no processo de produção do software, forçar o uso de padrões, aumentar a qualidade e a consistência, como também ganhar produtividade.

Lista de Abreviaturas e Siglas

ACMDA	*Architecture-Centric Model-Driven Architecture*
ADL	*Architecture Description Language*
AOAD	*Aspect-Oriented Architecture Design*
AOD	*Aspect-Oriented Design*
AOM	*Aspect-Oriented Modeling*
AORE	*Aspect-Oriented Requirements*
AOP	*Aspect-Oriented Programing*
AOSD	*Aspect-Oriented Software Development*
CIM	*Computer-Independent Model*
CSV	*Comma Separated Value*
CTI	*Computer Telephony Integration*
IDE	*Integrated Development Environment*
GUI	*Graphic User Interface*
LOC	*Lines Of Code*
MDA	*Model-Driven Architecture*
MDD	*Model-Driven Development*
MDE	*Model-Driven Engineering*
MOF	*Meta-Object Facility*
PIM	*Platform-Independent Model*
OMG	*Object Management Group*
POA	*Patter-Oriented Architecture*
PSM	*Platform-Specific Model*
PROGERAS	Processo para geração automatizada de software
UML	*Unified Modeling Language*
XMI	XML *Metadata Interchange*

CAMPOS da Figura 6:

Campo A

- Sequencia : int = 1
- DataVencimento : Date
- Valor : float = 0

- ValorReembolso : float : = 0
- FlagAbaterAuxilioEstudo : boolean = true
- DataAprovacaoChefe : Date

Campo B

- Curso : String
- Instituicao : String
- DataInicio : Date
- DuracaoMeses : int

Campo C

- MaxAuxilioEstudo : float
- PercentualReembolsoCursos : float
- FazReembolsoEstudo : boolean
- UserId : String

Campo D

-DataPedido : Date
- DataAprovacaoChefe : Date
- DataAprovacaoDiretor : Date
- DataAprovacaoFinanc : Date
- DataPagamento : Date
- Finalidade : String
- ValorReembolso : Float
- DataContabilizacao : Date
- TipoReembolso : String

Campo E

- Numero : String
- DataEmissao : Date
- Valor : float
- Fornecedor : String

- Itinerário : String
- Quilometragem : float = 0
- Observacao : String

Campo F

- Data : Date
- Valor : float
- Finalidade : String
- DataVencimento : Date
- DataContabilizacao : Date

Campo G

- PercenbtualReembolsoEstudo : float = 0.66F
- PercenbtualMaxReembolsoSobreSalario : float = 0.2F
- DiaPagamentoReembolsoEstudo : int = 20
- DiaSemanaPagamentoReembolsoDespesa : int = 3
- PeriodoMesesParaInicioReembolsoEstudo : int = 6
- IntervaloContabilizacao : int
- EventoAdiantamentoCLT : int
- EventoAdiantamentoKFD : int
- EventoReembolsoDespesaCLT : int
- EventoReembolsoDespesaKFD : int
- EventoAbatimentoemFolhaCLT : int
- EventoAbatimentoemFolhaKFD : int
- EventoReembolsoEstudoCLT : int
- EventoReembolsoEstudoKFD : int
- EventoNotaDebito : int
- LimiteAprovacaoDiretor : float
- DiasParaSolicitacaoReembolsoEstudo : int

REIVINDICAÇÕES

1) "PROCESSO PARA GERAÇÃO AUTOMATIZADA DE SOFTWARE", caracterizado por compreender as etapas de:

a) levantamento de requisitos;

b) criação de modelos de domínio;

c) projeto de arquitetura;

d) marcação do modelo de domínio;

e) geração de programas fonte;

f)adição manual de linhas de programa; e

g) geração de software executável.

2) "PROCESSO", de acordo com a reivindicação 1, caracterizado pelo projeto de arquitetura compreender o uso de *templates* para geração do programa fonte.

3) "PROCESSO", de acordo com a reivindicação 2, caracterizado pelos *templates* consistirem como adendos nos quais são definidos pontos de junção de interesses arquiteturais.

4) "PROCESSO", de acordo com a reivindicação 1, caracterizado pela marcação de modelo de domínio utilizar elementos do diagrama de classes marcados com base no projeto de arquitetura e no padrão de regras de marcação do processo selecionado para a sua implementação permitindo que a ferramenta selecione e execute o *template* correspondente.

5) "PROCESSO", de acordo com a reivindicação 4, caracterizado pelas marcas serem inseridas no modelo por meio de *tagged values* ou anotações, tendo a forma Variável = Expressão.

6) "PROCESSO", de acordo com a reivindicação 1, caracterizado pela geração de programas compreender o uso de uma ferramenta que recebe os diagramas de classes com marcações, gerando programas fonte numa linguagem específica e novos modelos que demonstram a arquitetura do sistema.

7) "PROCESSO", de acordo com a reivindicação 1, caracterizado pela geração do software executável ser realizado pela ferramenta de geração de fonte (robôs) que recebem os diagramas UML marcados como entrada e geram os programas fonte.

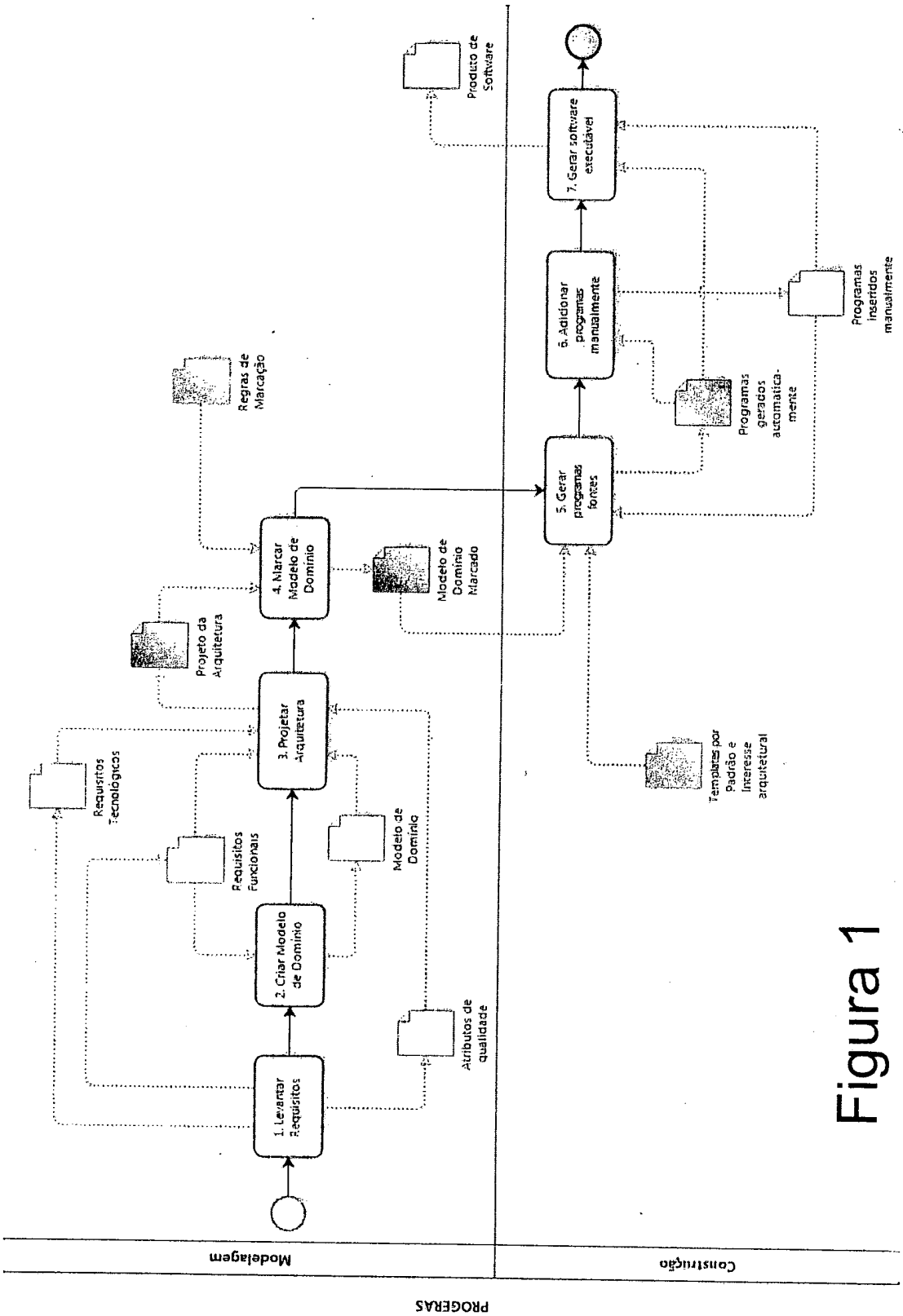

PROGERAS
Modelagem
Construção
1. Levantar Requisitos
2. Criar Modelo de Domínio
3. Projetar Arquitetura
4. Marcar Modelo de Domínio
5. Gerar programas fontes
6. Adicionar programas manualmente
7. Gerar software executável
Requisitos Tecnológicos
Requisitos Funcionais
Modelo de Domínio
Atributos de qualidade
Projeto da Arquitetura
Regras de Marcação
Modelo de Domínio Marcado
Templates por Padrão e Interesse arquitetural
Programas gerados automatica-mente
Programas inseridos manualmente
Produto de Software

Figura 1

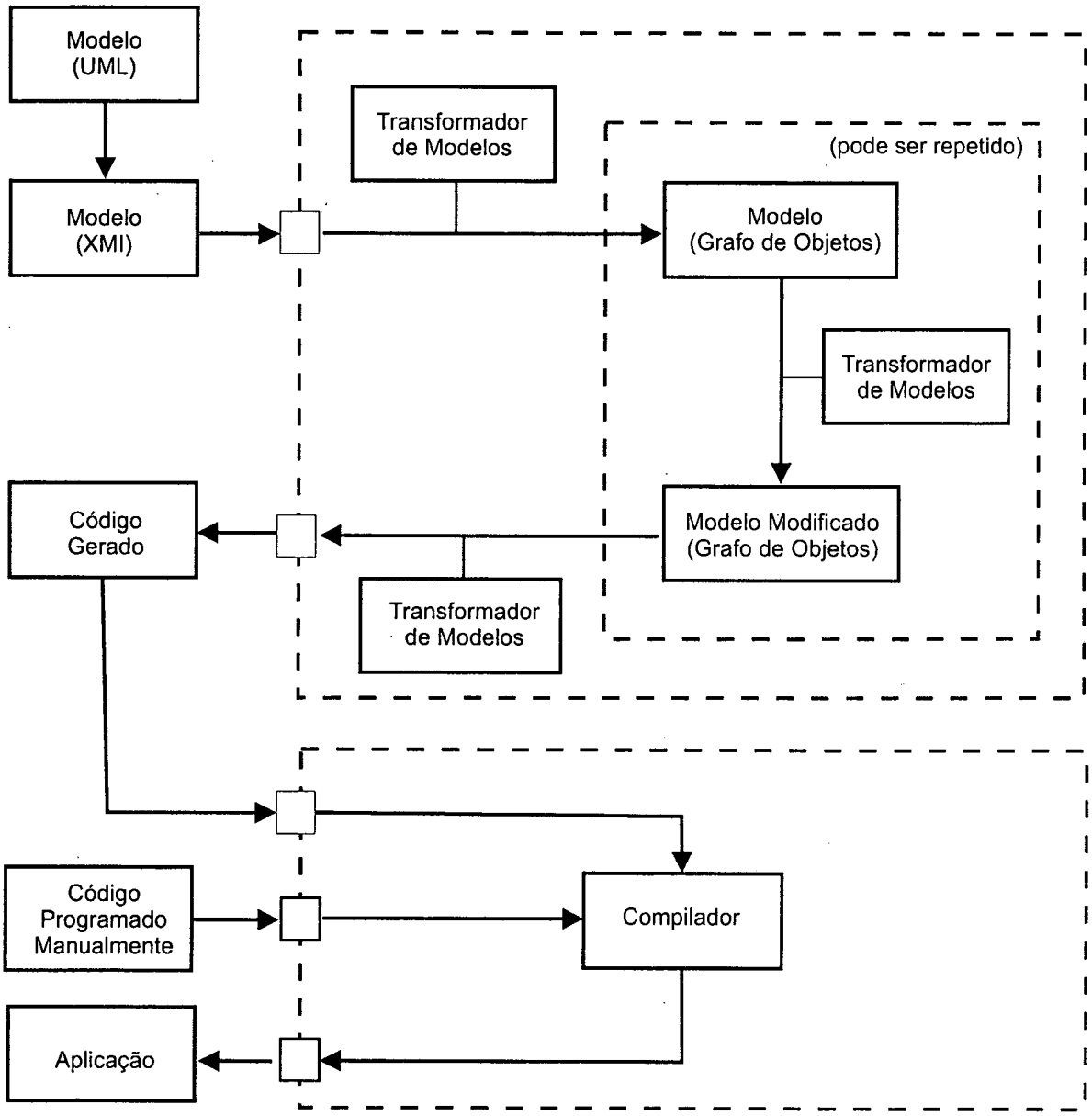

Modelo
(UML)
Modelo
(XMI)
Transformador
de Modelos
(pode ser repetido)
Modelo
(Grafo de Objetos)
Transformador
de Modelos
Modelo Modificado
(Grafo de Objetos)
Transformador
de Modelos
Código
Gerado
Código
Programado
Manualmente
Compilador
Aplicação

Figura 2

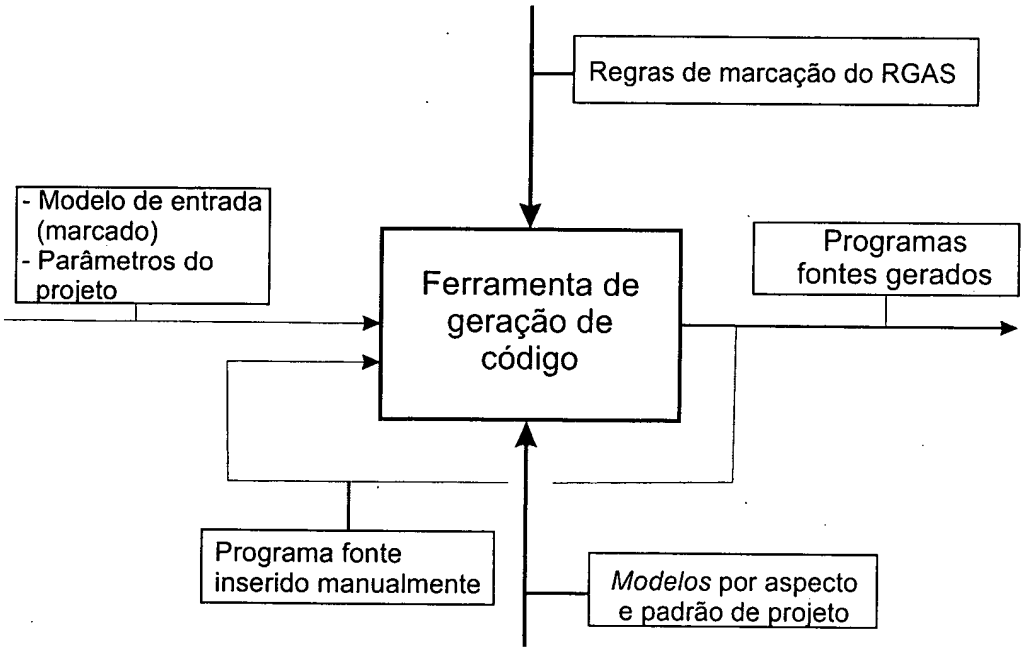


Figura 3

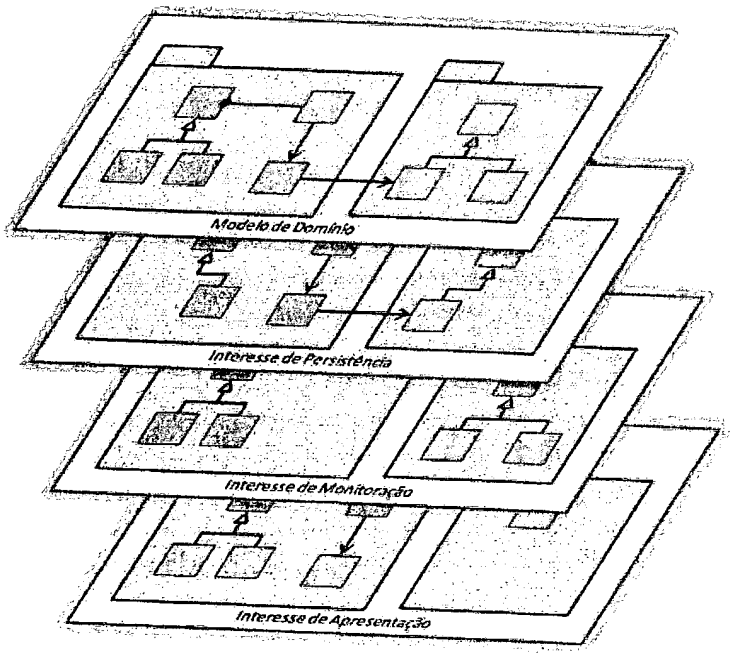


Figura 4

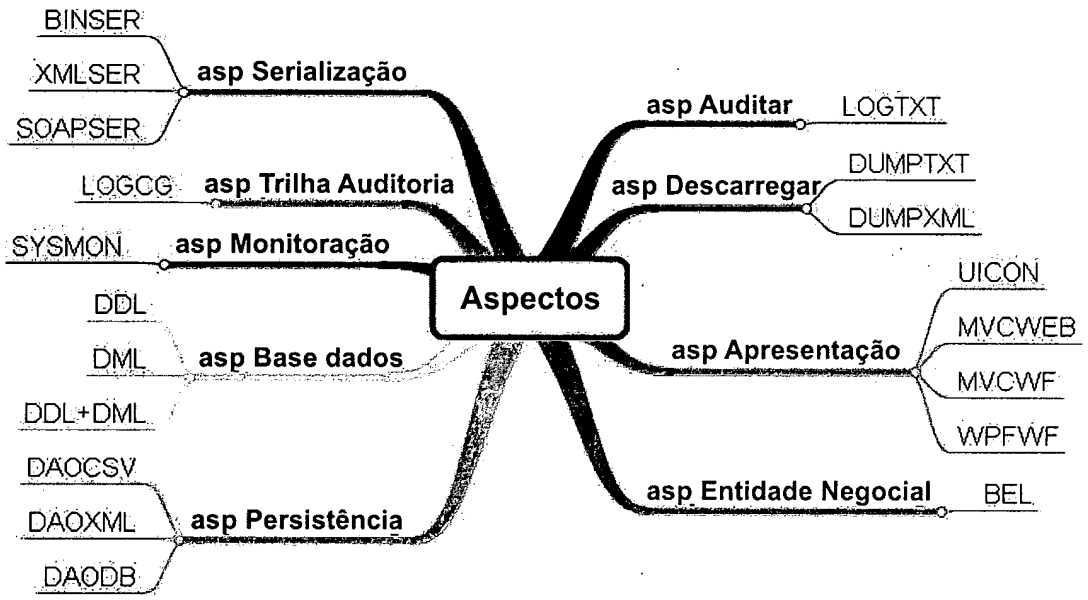

Aspectos
asp Serialização
BINSER
XMLSER
SOAPSER
asp Trilha Auditoria
LOGCG
asp Monitoração
SYSMON
asp Base dados
DDL
DML
DDL+DML
asp Persistência
DAOCSV
DAOXML
DAODB
asp Auditar
LOGTXT
asp Descarregar
DUMPTXT
DUMPXML
asp Apresentação
UICON
MVCWEB
MVCWF
WPFWF
asp Entidade Negocial
BEL

Figura 5

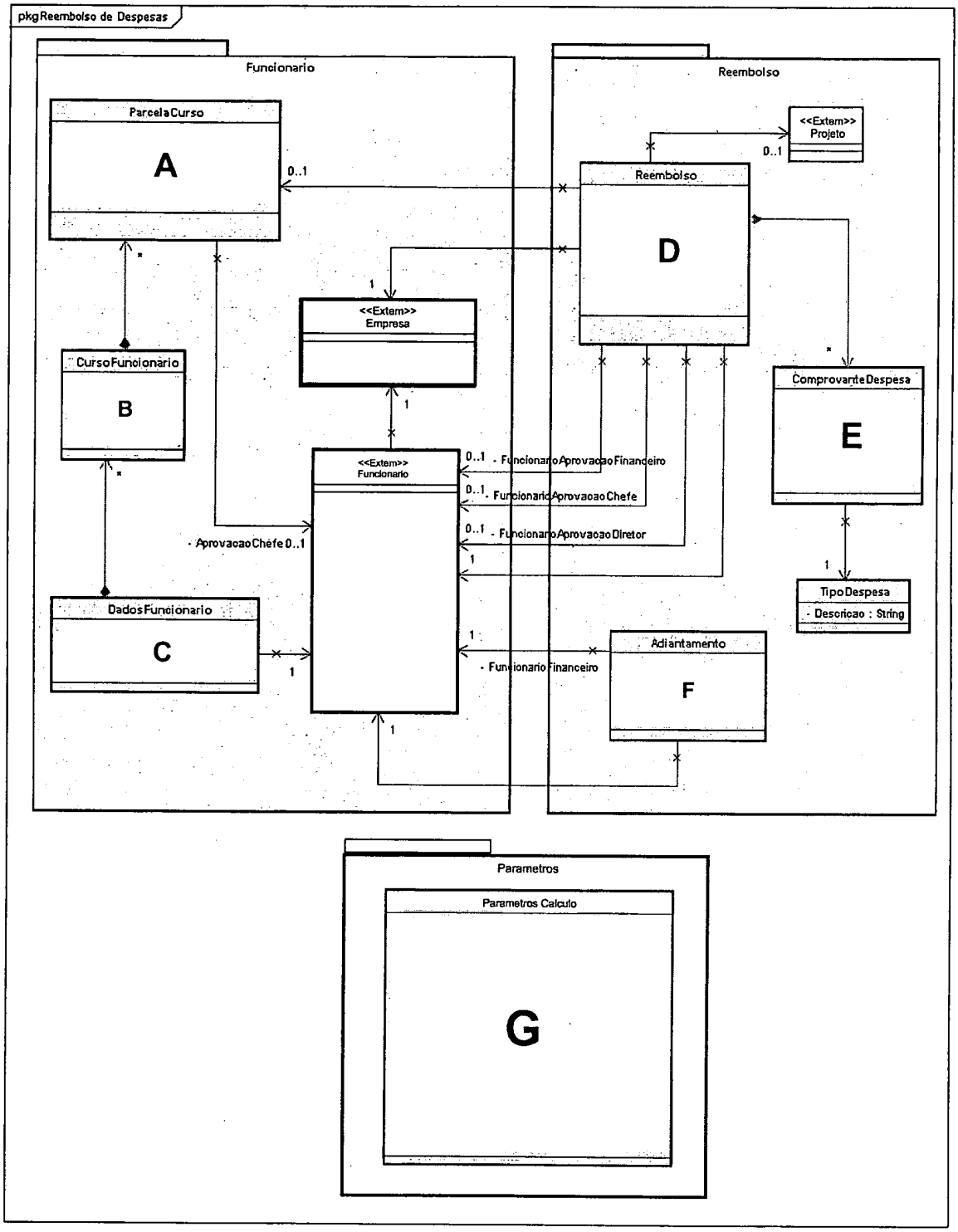

pkgReembolso de Despesas
Funcionario
Reembolso
ParcelaCurso
A
0..1
<<Extern>>
Projeto
0..1
Reembolso
D
<<Extern>>
Empresa
CursoFuncionario
B
ComprovanteDespesa
E
<<Extern>>
Funcionario
0..1 - FuncionarioAprovacaoFinanceiro
0..1 - FuncionarioAprovacaoChefe
0..1 - FuncionarioAprovacaoDiretor
- AprovacaoChefe 0..1
TipoDespesa
- Descricao : String
DadosFuncionario
C
Adiantamento
F
- FuncionarioFinanceiro
Parametros
Parametros Calculo
G

Figura 6

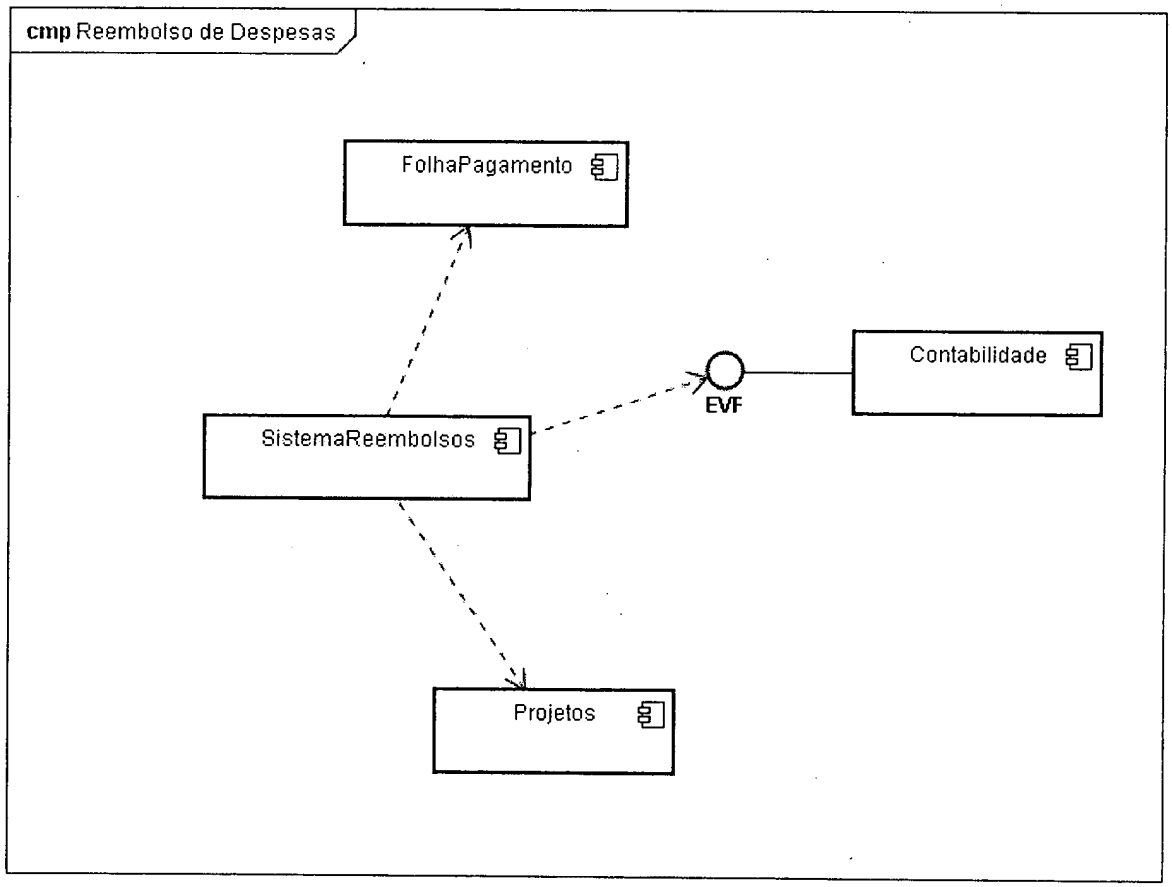

cmp Reembolso de Despesas
FolhaPagamento
Contabilidade
EVF
SistemaReembolsos
Projetos

Figura 7

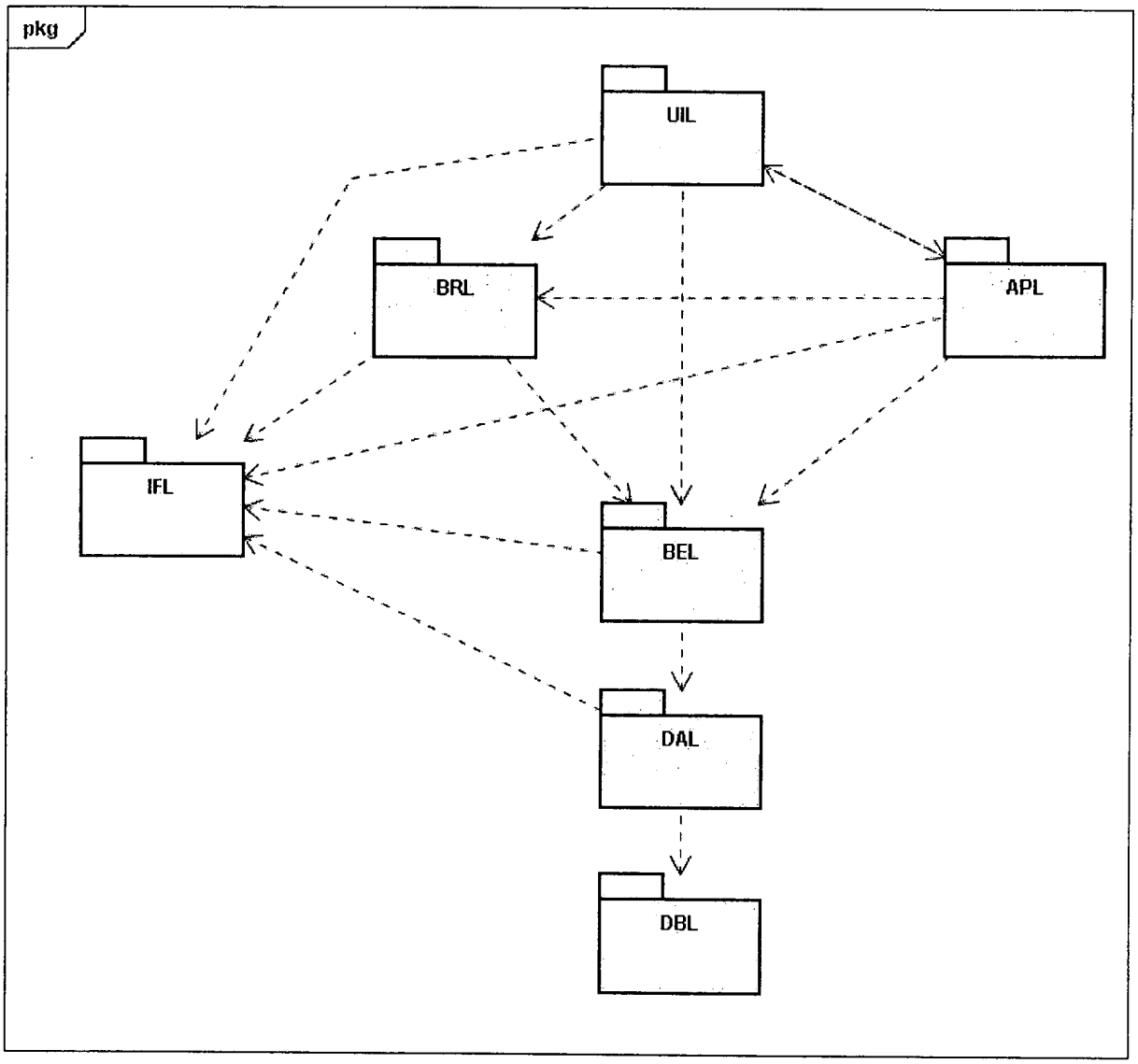


Figura 8

RESUMO

"PROCESSO PARA GERAÇÃO AUTOMATIZADA DE SOFTWARE"

É proposto um processo de geração automática de programas fonte com base no MDA. Para orientar o processo de geração de código, são usadas regras de marcação padronizadas, inseridas em modelos UML por meio de *tagged values*, extensões e anotações. Essas regras indicam o interesse arquitetural e o padrão de projeto que deverá ser usado ao gerar os programas fonte. O PROGERAS é proposta de extensão ao MDA, aplicando os conceitos do AOD e do POA para gerar programas fonte de forma automatizada a partir de um diagrama de classes (PIM). Com essa proposta, acredita-se ser possível não só obter um elevado nível de automação no processo de produção do software, forçar o uso de padrões, aumentar a qualidade e a consistência, como também ganhar produtividade.