Definição de Tipos de Dados na Modelagem de Negócios: uma proposta de ferramenta de captura de Metadados para geração automática de código

Marcia Nunes de Carvalho Leite

UFRJ – Universidade Federal do Rio de Janeiro

Mestrado em Informática

Prof. Dr. Amauri Marques da Cunha

Prof. Dr. Carlo Emmanoel Tolla de Oliveira

Rio de Janeiro

2006

Marcia Nunes de Carvalho Leite

DEFINIÇÃO DE TIPOS DE DADOS NA MODELAGEM DE NEGÓCIOS: uma proposta de ferramenta de captura de Metadados para geração automática de código

Número de volumes 1

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática, Universidade Federal do Rio de Janeiro – UFRJ, como parte dos requisitos necessários à obtenção do título de Mestre em Informática.

Orientadores: Amauri Marques da Cunha, Dr. Ing.
Carlo Emmanoel Tolla de Oliveira, Ph.D.

Rio de Janeiro
2006

Carvalho Leite, Marcia Nunes de.
Definição de Tipos de Dados na Modelagem de Negócios:
uma proposta de ferramenta de captura de Metadados
para geração automática de código / Marcia Nunes de Carvalho
Leite. Rio de Janeiro, 2006.
xi, 110 f. : il.

Dissertação (Mestrado em Informática) –
Universidade Federal do Rio de Janeiro, NCE, 2006.

Orientadores: Amauri Marques da Cunha,
Carlo Emmanoel Tolla de Oliveira

1. Definição de Tipos de Dados na Modelagem de Negócios.
2. Definição de Tipos de Dados com Metadados.
3. Informática – Teses.

I. Cunha, Amauri Marques (Orient.). II. Universidade Federal
do Rio de Janeiro. NCE. III. Título

FOLHA DE APROVAÇÃO

Marcia Nunes de Carvalho Leite

DEFINIÇÃO DE TIPOS DE DADOS NA MODELAGEM DE NEGÓCIOS: uma proposta de ferramenta de captura de Metadados para geração automática de código

Rio de Janeiro, 25 de Agosto de 2006.

_____

Amauri Marques da Cunha, Dr. Ing., UFRJ (co-orientador)

_____

Carlo Emmanoel Tolla de Oliveira, Ph.D., UFRJ (co-orientador)

_____

Maria Cláudia Reis Cavalcanti, D.Sc., IME (examinadora)

_____

Priscila Machado Vieira Lima, Ph.D., UFRJ (examinadora)

RESUMO

CARVALHO LEITE, Marcia Nunes de**. Definição de Tipos de Dados na Modelagem de Negócios**: uma proposta de ferramenta de captura de Metadados para geração automática de código. Orientadores: Prof. Dr. Amauri Marques da Cunha, Prof. Dr. Carlo Emmanoel Tolla de Oliveira. Rio de Janeiro: UFRJ/IM/DCC/NCE; 2006. Dissertação (Mestrado em Informática).

O objetivo desta dissertação é ensaiar mais um passo em direção à automação da geração de código para sistemas de informação em computador, visando ajudar a resolver uma parte desse problema que já vem persistindo desde os primórdios da Ciência da Computação. Com base em conceitos da Model Driven Architecture (MDA®) e utilizando diagramas da Unified Modelling Language (UML™) associados a um repositório de Metadados de um sistema de informação é usada uma linguagem formal – linguagem do Modelo de Objetos Relacionados (MOR™) – para descrição do Negócio. Assim, partindo de um nível alto de abstração, o Platform Independent Model (PIM) da MDA, podem ser desenvolvidos sistemas com qualidade e rapidez. Isso leva a uma redução de custo, principalmente em codificação e manutenção. A idéia principal é obter um Modelo de Negócio enriquecido com metadados, que se tornará uma fonte significativa para geração automática de código. O foco deste trabalho está em propor uma maneira simples e fácil para captura e descrição dos metadados dos Tipos Elementares de Dados identificados no modelo nível PIM. Um segundo passo é a geração de arquivos XML® a partir deste modelo. O arquivo resultante servirá de entrada para uma ferramenta de geração de código que gere o seu respectivo código em uma linguagem Orientada a Objeto escolhida para o desenvolvimento.

ABSTRACT

CARVALHO LEITE, Marcia Nunes de**. Data Types Definition within Business Modelling**: defining a tool to capture Metadata to provide automatic code generation. Orientadores: Prof. Dr. Amauri Marques da Cunha, Prof. Dr. Carlo Emmanoel Tolla de Oliveira. Rio de Janeiro: UFRJ/IM/DCC/NCE; 2006. Dissertação (Mestrado em Informática).

The aim of this dissertation is to introduce another step towards the automatic code generation for computer based information systems, trying to solve part of this issue that has persisted for 50 years. Based on Model Driven Architecture (MDA®) concepts and using some of the Unified Modelling Language (UML™) diagrams associated to a Metadata repository of the information system, we will use a formal language – Modelo de Objetos Relacionados language (MOR™) – to build the Business description. Therefore, from a high level of abstraction, Platform Independent Model (PIM), we will be able to develop quality systems faster. This will lead to cost reductions, especially in coding and maintenance. The main idea is to have a detailed Business Model with metadata that will be a significant source for automatic code generation. The focus of this work will be to propose a simple and easy way to capture and describe the metadata of the Elementary Data Types identified in that model. A second step will be to generate XML® files from this model. The resulting file will serve as the entry file to a code generation tool, which will import it and then generate the code in the chosen Object Oriented language.

Sumário

1 Introdução

Nesta dissertação pretendemos mostrar que o sistema de informação em computador precisa nascer num nível de abstração o mais alto possível, e ser detalhado ao máximo nos seus níveis subseqüentes, antes de iniciarmos o projeto físico. Para tal, precisamos, entre outras coisas, de mais tempo para entender o negócio e levantar os requisitos. E, para termos qualidade no que é produzido, é importante entender bem o que o cliente ou usuário necessita. Então, para garantir os três itens fundamentais no desenvolvimento de software – Prazo, Preço e Qualidade, a saída é automatizar mais a geração de código, que hoje é um fator crítico no ciclo deste desenvolvimento.

Segundo Pressman [PRESSMAN, 1998], o software de computador tornou-se uma força motora que dirige a tomada de decisão nos negócios: “o software é um transformador de informação, produzindo, gerando, adquirindo, modificando, exibindo, ou transmitindo informação, que é o produto mais importante da nossa época”. E o tempo para essa entrega é, na maioria das vezes, crucial.

A concorrência entre empresas desenvolvedoras de software é definida por custo e prazo e, em alguns casos, senão todos, pela qualidade. Por isso, precisamos desenvolver sistemas rapidamente, a baixo custo e com qualidade, uma vez que a importância do software continua a crescer. O programador solitário de antigamente foi substituído por uma equipe de especialistas, cada um concentrado numa parte da tecnologia a fim de produzir um sistema complexo. Porém, os questionamentos feitos àquele programador solitário ainda não obtiveram respostas convincentes ou soluções definitivas [PRESSMAN, 1998]:

- Por que leva tanto tempo para construir software?
- Por que os custos são tão altos?

1.1 A Evolução do Software

Pressman [PRESSMAN, 1998] conta um pouco sobre a história do produto esperado: o Software. No início da década de 80, percebemos uma compreensão da importância do software de computador. Atualmente, o software, já tendo ultrapassado o hardware como peça mais importante para o sucesso de sistemas, se tornou um fator diferenciador num mercado em que a concorrência está extremamente agressiva. Um dos principais objetivos é desenvolver sistemas amigáveis, isto é, fáceis de usar, e com alta qualidade, baixo custo e num prazo mais curto possível.

Ainda de acordo com Pressman, vemos na Figura 1 [PRESSMAN, 1998] a evolução do software de computador nas últimas décadas.

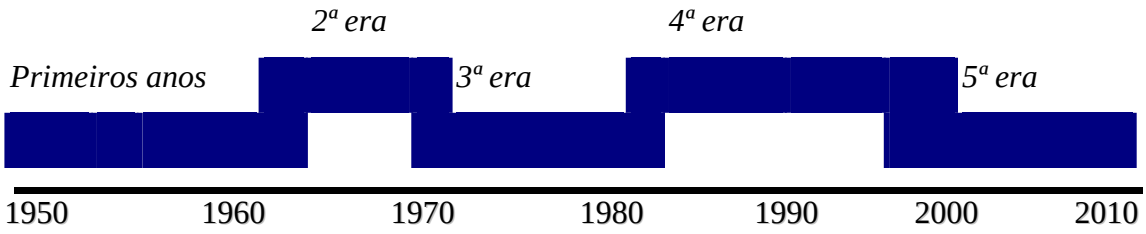


Figura 1 – Evolução do Software

Para cada uma das fases da Figura 1 temos:

- Primeiros anos - Orientação batch, software desenvolvido sob medida, distribuição limitada onde poucos tinham acesso ao software;
- 2ª Era - Multiusuário, Tempo Real, Bancos de Dados, produto de software;
- 3ª Era - Sistemas Distribuídos, “Inteligência Embutida”, hardware de baixo custo, impacto de consumo;

- 4ª Era - Sistemas *desktop* com maior capacidade de processamento, Tecnologias Orientadas a Objeto (OO), Sistemas Especialistas, Redes Neurais Artificiais, Computação Paralela;

- 5ª Era - Sistemas Web em larga escala, *Grid Computing*, *Peer to Peer*, Sistemas Autônomos, e aqui podemos incluir a "Geração Automática de Código"...

Nos primeiros anos, o software era projetado sob medida para uma aplicação, e tinha a sua distribuição limitada. Era usado pela própria pessoa ou organização, e por isso muitas vezes não havia documentação. As linguagens eram "de máquina", ou seja, de mais baixo nível.

A 2ª era foi caracterizada pelas *software-houses* e sistemas multiusuários. O problema da "manutenção de software" surgiu e começou a absorver recursos exageradamente, uma vez que as falhas detectadas eram alteradas e o software precisava ser redistribuído a todos os usuários. Começamos a observar também uma evolução nas linguagens de programação. O uso de linguagens tipo "Assembler" foi substituído pelo uso de linguagens procedurais.

A 3ª era da evolução do software começou em meados da década de 70, e ainda continua até hoje (em menor escala). Foi a era do uso de microprocessadores, computadores pessoais e estações de trabalho com grande capacidade de processamento, redes locais e globais. O software se diferenciava, o hardware já se tornava um produto primário. Com as linguagens de programação mais evoluídas, já não há mais uma preocupação em gerar código de máquina, pois essa atividade foi automatizada.

A 4ª era trouxe as tecnologias Orientadas a Objeto que estão ocupando o lugar das abordagens mais convencionais no desenvolvimento de software. Sistemas especialistas e inteligência artificial saem do laboratório e alcançam o mercado.

A 5ª era, ou a era em que vivemos hoje, chega com a responsabilidade de resolver alguns problemas, como por exemplo:

- A sofisticação do software ultrapassou a capacidade de construí-lo de maneira que possamos utilizar todo o potencial de processamento do hardware;
- A capacidade de manter os programas existentes é ameaçada por projetos ruins, rotatividade da equipe e recursos inadequados (qualidade baixa x custo baixo pode ser um motivo; pouco tempo para o entendimento dos requisitos, pode ser um outro motivo);
- A capacidade de construir programas não acompanha a demanda por novos programas (tempo elevado para codificação manual pode ser um dos fatores).

Ainda nesta última era, foi inserida a geração automática de código. Entretanto, já se observava uma tendência ou evolução neste sentido desde a 4ª era, possivelmente representada pela evolução das linguagens de programação, cada vez mais próximas da linguagem natural, e pelas ferramentas *Case* com geração automática parcial de código para criação das tabelas em bancos de dados relacionais.

Então, observando esse histórico, concluímos que houve uma mudança no foco em desenvolvimento de sistemas. Nos primeiros anos, o foco era em codificar programas usando instruções em baixo nível, linguagem “de máquina”, “Assembler”. Depois, o foco foi deslocado para a codificação usando linguagens de mais alto nível, automatizando a geração de código de máquina através de montadores e de compiladores. Atualmente, as linguagens estão muito mais evoluídas, se aproximando até mesmo da linguagem natural. E ainda há muito por vir.

1.2 O Problema

Nesta dissertação será abordada uma parte do problema da geração manual de código: os tipos de dados encontrados durante a modelagem do negócio, os quais freqüentemente possuem pouca ou nenhuma semântica para auxiliar na geração automática de código. A codificação manual dos sistemas de informação é um ponto crítico no desenvolvimento de sistemas, muitas vezes responsável por atrasos significativos. Portanto, é difícil gerar código automaticamente a partir de modelos pouco detalhados no que diz respeito aos tipos de dados.

Para se desenvolver sistemas pelos processos tradicionais (por exemplo: cascata, espiral ou incremental), vemos abaixo alguns dos problemas e desafios normalmente encontrados [PRESSMAN, 1998]:

- Pouco tempo para entender o negócio do cliente e efetuar o levantamento de requisitos – muitas vezes os cronogramas são apertados e/ou os clientes/usuários não estão disponíveis;
- Cliente/usuário também não conhece bem o negócio ou não sabe exatamente o que precisa;
- Usuário se sente ameaçado pela possibilidade do sistema substituir seu trabalho e não compartilha seu conhecimento - dificuldade de “gerenciar o conhecimento”;
- Distância grande entre “o quê” o cliente/usuário quer e “como” os analistas irão implementar;
- Geração manual de código em larga escala, mencionado anteriormente como nosso ponto de atenção neste trabalho.

Estes e outros problemas que ocorrem durante o desenvolvimento de sistemas levam a outros ainda maiores, como queda de produtividade e qualidade, softwares com baixa portabilidade e interoperabilidade, excessiva manutenção e documentação sempre defasada

[PRESSMAN, 2002]. Nesta dissertação será abordada uma parte do problema da geração manual de código: os tipos de dados que são normalmente encontrados durante a modelagem do negócio possuem pouca ou nenhuma semântica para auxiliar na geração automática de código. Vejamos a seguir, cada um destes maiores problemas em detalhes [KLEPPE, 2003].

1.2.1 Produtividade

O processo de desenvolvimento de sistemas, tal como conhecemos hoje, é freqüentemente voltado para utilizar uma modelagem de mais baixo nível, física, com muita codificação. Um processo típico inclui várias fases, como por exemplo:

1. Concepção e levantamento de requisitos
2. Descrição e Análise Funcional
3. Desenho
4. Codificação
5. Teste
6. Implantação

Usando o processo iterativo e incremental ou o tradicional cascata, documentos e diagramas são produzidos durante as fases 1 a 3. Isso inclui a descrição dos requisitos e muitas vezes diagramas usando a UML (vide item 2.6). A quantidade de papel gerada é geralmente enorme. Contudo, muitos artefatos gerados nestas fases são apenas papel e nada mais, e rapidamente perdem seu valor quando começa a codificação. As posteriores mudanças nos sistemas são quase sempre realizadas diretamente no código, fazendo com que os diagramas e documentos fiquem ainda mais defasados em relação ao sistema que está sendo desenvolvido. Esta prática dificulta cada vez mais a produtividade, causando um efeito “bola de neve”.

A qualidade é prejudicada quando as mudanças são feitas sem o conhecimento do negócio, ou com prazo irreal, com pouco tempo para testes, acarretando defeitos em outras partes do sistema que estavam funcionando antes da manutenção. Acreditamos que uma

possível solução para este problema seria haver mais tempo para entender o negócio, a fim de podermos desenhar e detalhar modelos mais próximos da realidade. Como isso consome mais tempo, então, uma boa alternativa seria automatizar uma fase que hoje é crítica e demorada, a codificação.

1.2.2 Portabilidade

A todo momento, novas tecnologias são inventadas e se tornam populares, por exemplo, Java, Linux, XML, .NET. As empresas por sua vez precisam seguir estas novas tecnologias, pois as ferramentas usadas anteriormente podem parar de receber suporte por parte dos fornecedores. Adicionalmente, as novas tecnologias também oferecem benefícios, facilitando a portabilidade e a troca de plataforma, por exemplo.

Nesta dissertação, é tratada a importante questão de produzir modelos legíveis pela máquina num nível de abstração mais alto, onde a mudança de tecnologia e/ou ferramentas não afetem o bom andamento do negócio. Assim, uma vez trocada a tecnologia, o sistema seria gerado novamente a partir do mesmo modelo, já nesta nova tecnologia.

1.2.3 Interoperabilidade

Sistemas de computador raramente vivem isolados. Eles precisam se comunicar com outros sistemas. Normalmente, estes diversos sistemas usam diversas tecnologias. Até mesmo um único sistema pode vir a usar várias tecnologias diferentes entre si. Por isso, é importante desenvolver componentes e interfaces para essa crescente combinação de tecnologias, permitindo assim a interoperabilidade dos sistemas.

Acreditamos que a Interoperabilidade torna-se ainda mais viável com a introdução de Metadados (vide item 4.4). As informações emitidas por um sistema de origem são documentadas com metadados para que possam ser entendidas no sistema de destino. É uma

espécie de referência cruzada, “De-Para”, de significado das informações, que possibilita a conversação entre sistemas.

1.2.4 Manutenção e Documentação

Muitos desenvolvedores acham que a sua principal tarefa é produzir código. Assim, surgem inúmeras desculpas para não documentar o sistema:

- Documentar durante o desenvolvimento “gasta” tempo e atrasa ou torna mais lento o processo, além de “não auxiliar” naquela tarefa principal;
- A documentação será usada apenas pelos desenvolvedores que virão depois;
- Não existe incentivo para documentar um sistema; isto é feito somente porque o gerente do projeto diz que é necessário e, a cada nova manutenção no sistema, a documentação precisa ser revista e alterada manualmente.

1.3 Hipótese

É claro que a forma de pensar apresentada no item anterior não está correta. O ideal é que possamos desenvolver softwares resistentes ao tempo e manuteníveis; e sem uma boa documentação isso se torna inviável. Uma solução para esse problema seria gerar novamente a documentação a partir do próprio código fonte, ou, considerando a idéia principal desta dissertação, a partir de um modelo processável pela máquina poderiam ser gerados ambos, código e documentação, a partir de uma alteração no modelo.

Se houver um detalhamento dos tipos de dados, teremos melhores condições para promover a geração automática de código. Acreditamos que capturar Metadados sobre os tipos de dados nos levará a uma facilitação dessa geração. Estes Metadados proporcionarão muito mais semântica ao tipo de dado.

1.3.1 Metodologia

De acordo com o que foi abordado anteriormente, é interessante haver um modelo que possa ser entendido pela máquina. Para desenhar um modelo deste tipo, é necessário pensar no nível mais alto de abstração e descer deste nível até um detalhamento satisfatório. Para isso, são usados conceitos de Orientação a Objetos, MDA (vide Capítulo 2), uma linguagem padronizada para representação, MOF e UML, descritos no contexto da MDA, além de um bom conhecimento sobre Modelo de Negócio (Capítulo 3). Para um entendimento mais completo e preciso, são abordados ainda outros assuntos relacionados no Capítulo 4, como Tipos Elementares de Dados, um conceito bastante interessante e útil, mas ainda não difundido, aprendido durante o curso de Modelagem de Negócio Orientada a Objetos (MNOO) [CUNHA, 2004], e Metadados para estes Tipos Elementares de Dados, por serem considerados assuntos fundamentais para o desenvolvimento da idéia exposta no Capítulo 5.

A seguir, apresentamos de maneira simplificada a organização do trabalho para alcançar um modelo detalhado no que diz respeito aos tipos de dados:

- Com base nos conceitos da MNOO, definir o Modelo do Negócio (nível de abstração mais alto) usando a linguagem UML;
- Utilizando uma ferramenta gráfica, desenhar o Diagrama de Classes com seus atributos, que serão detalhados utilizando uma linguagem formal. Neste estudo será utilizada a linguagem do Modelo de Objetos Relacionados (MOR) (item 3.3.2) em BNF equivalente, introduzida durante o Curso de MNOO [CUNHA, 2004];
- Identificar os Tipos Elementares de Dados do Sistema e desenhar o Diagrama de Tipos Elementares de Dados encontrados.

Após percorrer este caminho, chegamos ao nosso alvo: descrever uma ferramenta que possibilite a criação de uma Biblioteca de Tipos Elementares de Dados a partir de tipos de dados primitivos existentes num alto nível de abstração, de acordo com o MOF (item 2.5).

Esta ferramenta tem como objetivo principal facilitar o detalhamento de cada Tipo Elementar encontrado, analisando e capturando suas características de preenchimento e validações necessárias, com o auxílio de Metadados, os quais são traduzidos para a linguagem do MOR, em BNF equivalente. A partir deste ponto, ficaria a cargo de trabalhos futuros o desenvolvimento e implementação desta ferramenta, que teria como um de seus encargos a transformação da linguagem BNF equivalente para XML. Com o código XML gerado, e de acordo com a MDA (Capítulo 2), a responsabilidade será passada para uma ferramenta de geração de código a partir deste formato. Atualmente já existem estudos neste sentido. O AndroMDA [ANDROMDA, 2005] e o UMT-QVT [UMTQVT, 2006] são alguns exemplos desses estudos em desenvolvimento que são mencionados brevemente no Capítulo 6.

Acreditamos que, ao percorrer os passos aqui apresentados, é possível construir uma ponte de “mão-dupla” entre o Projeto Lógico e o Projeto Físico no que diz respeito aos Tipos Elementares de Dados identificados no levantamento de requisitos do sistema. Essa idéia parece automatizar um pouco mais a construção de sistemas de informação, bem como minimizar a repetição de partes do código, pois os Tipos Elementares [CUNHA, 2004] e suas características são colocados em evidência como classes a serem usadas por todos os sistemas. Além disso, pode ser obtida uma significativa melhora na confiabilidade dos dados e no entendimento do negócio, que são pontos fundamentais em um sistema informatizado.

2 MDA® (Model Driven Architecture)

"Um bom planejamento importa" [KLEPPE, 2003]. A preparação de projetos de software não é tempo perdido. Se a captura de requisitos for subestimada, certamente serão gastos esforços desenvolvendo coisas que o usuário não quer ou não precisa. No que se refere à MDA, este Capítulo baseia-se principalmente em Kleppe [KLEPPE, 2003], MDA Guide [OMG, 2003] e OMG [OMG, 2006].

Os sistemas de informação têm sido desenvolvidos, gerenciados e integrados ao longo do tempo, usando uma série de metodologias, ferramentas e *middleware*, e parece não ter fim essa contínua inovação. O que temos visto nos últimos anos é um movimento gradual para modelos mais completos semanticamente, bem como para padrões de representação e troca de dados. O tamanho e a complexidade do software tem aumentado significativamente nos últimos 20 a 25 anos. A variedade de tecnologias disponíveis para implementação pode provocar muita indecisão. Usaremos C++, Java, Visual Basic, C# ? .NET? Corba? JavaScript, ASP? JDBC? EJB? Atualmente, um simples projeto de software usa vários destes recursos, empregando-os onde melhor se aplicam.

A abordagem MDA procura criar um bom design, que cobre uma múltipla gama de tecnologias de implementação e auxilia na extensão do ciclo de vida do software. MDA é baseada numa padronização amplamente usada para visualização, armazenamento e troca de modelos e design de software. O mais conhecido destes padrões é a UML, que é focada mais adiante neste Capítulo. De fato, a MDA é apenas mais um evolucionário passo no desenvolvimento de software. A aparente "magia" na automação de software a partir de modelos é, verdadeiramente, apenas um outro nível de compilação [KLEPPE, 2003].

Vejamos dois conceitos fundamentais para o entendimento e desenvolvimento de sistemas: Modelo e Metamodelo.

- **Modelo**

Uma definição de “modelo” seria [ASKOXFORD, 2005]: “uma descrição simplificada (freqüentemente matemática) de um sistema ou processo usada para dar assistência em cálculos e predições”. Analisemos as palavras-chave desta definição: descrição, simplificada e assistência.

De acordo com essa definição, um modelo é uma *descrição* de um sistema, e não o próprio sistema, é apenas uma representação formal deste. O segundo aspecto da definição é que o modelo é uma representação *simplificada*. Não é tão complexo ou tão rico como o sistema que está sendo modelado. O ser humano é, de alguma forma, limitado na quantidade de informação que pode processar de cada vez. Quando pessoas trabalham em um problema complexo, elas querem (ou precisam) quebrá-lo em problemas menores, mais gerenciáveis. É necessário “Dividir para Conquistar”. Modelos são construídos para auxiliar esta tarefa.

A última palavra-chave: *assistência*. Modelos servem a um propósito específico, eles descrevem um domínio. São construídos para alcançar um objetivo específico. O objetivo não é construir o modelo, mas desenhar o sistema com auxílio do modelo.

Na tentativa de uma geração mais automática de código, o aspecto “representação simplificada” pode ser repensado, pois neste caso é importante que o modelo seja enriquecido com informações sobre seus elementos. A fim de realizar esse detalhamento, utilizaremos a linguagem do MOR e Metadados. Estes assuntos são tratados mais adiante, nos Capítulos 3 e 4, respectivamente.

Normalmente, um primeiro modelo é muito bruto, ignorando ainda muitos aspectos do negócio. O objetivo deste modelo é dar a todos os envolvidos (usuários, desenvolvedores, projetistas) a chance de expressar livremente os vários pontos de vista. Este modelo será refinado em um ou mais modelos, aumentando sua sofisticação e complexidade. Cada iteração incorpora mais elementos, até que todos os aspectos relevantes tenham sido

incorporados. Modelos são abstrações de coisas que existem na realidade, são, portanto, aproximações sucessivas dessa realidade.

Um modelo é sempre escrito numa linguagem. Uma das mais efetivas representações usadas em modelagem são os desenhos e gráficos. A mente tem mais facilidade em trabalhar com um gráfico do que com uma longa lista de instruções complexas. Dentro deste contexto, a linguagem UML é apresentada mais adiante.

Muitos sistemas existentes não têm nada a ver com o seu modelo inicial, ou porque o modelo foi desenhado, projetado e ignorado, ou talvez ele tenha sido seguido num primeiro momento e depois deixado de lado, ficando assim completamente desatualizado, obsoleto. A idéia da geração automática de código pode resolver este problema, uma vez que possibilita haver uma via em “mão-dupla”. Ao atualizar um modelo, geramos novamente o código e vice-versa. Esta idéia é explorada ao longo desta dissertação.

Existem poucas desculpas (ou nenhuma!) para construir software sem antes fazer um trabalho de análise e modelagem. A modelagem não nos leva apenas a sistemas que são mais fáceis de desenvolver, integrar e manter, mas também à habilidade de automatizar pelo menos parte da sua construção.

- **Metamodelo**

“Um metamodelo é um modelo explícito [...] uma definição precisa das regras e conceitos necessários para construir modelos semânticos, específicos dentro de um domínio de interesse” [METAMODEL, 2006]. Simplificando, é um modelo que descreve modelos. Por sua vez é também um modelo - um modelo dos tipos de informação que se pretende capturar.

Ultimamente a meta modelagem tem sido foco de atenção. Um dos principais motivos é a necessidade de subir o nível de abstração dos modelos [METAMODEL, 2006]. Isso ajuda, por exemplo, a abstrair os níveis de detalhe de integração e interoperabilidade, a visualizar

problemas por outro ângulo, a particionar sistemas complexos, a entender e descrever o domínio do problema.

Um modelo pode ter diferentes papéis (metamodelo, metametamodelo,...) dependendo do contexto [KLEPPE, 2003]. O termo "*meta*" descreve o papel que o modelo representa em relação a outro modelo. O metametamodelo se relaciona com o metamodelo da mesma maneira que o metamodelo se relaciona com o modelo. A Figura 2 ilustra alguns níveis de modelos [OMG, 2002].

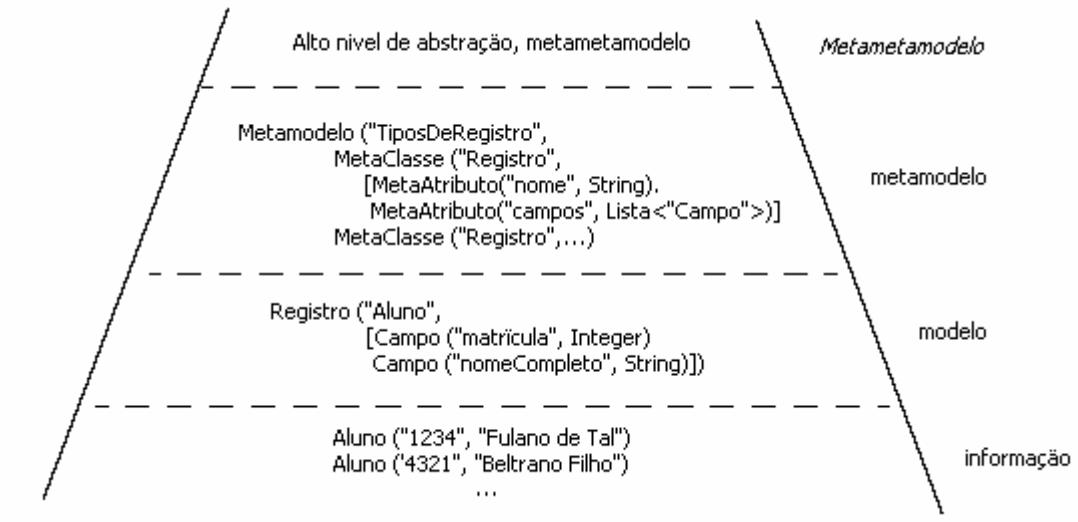


Figura 2 - Níveis de Modelos

Muitos metamodelos tendem a estar num nível conceitual, e não no nível lógico ou físico, porque bons metamodelos procuram apresentar muita distância de considerações de implementação tecnológica. Um metamodelo pode ser analisado sob três diferentes perspectivas [METAMODEL, 2006]:

1. Como um conjunto de blocos de construção e regras usados para construir modelos;
2. Como um modelo de domínio de interesse;
3. Como uma instância de um outro metametamodelo.

Um metamodelo permite definir uma linguagem. Exemplos: Ferramentas CASE, MOF, UML. Podemos dizer ainda que um metamodelo é uma coleção de conceitos que formam o vocabulário com o qual estamos raciocinando sobre um certo domínio [METAMODEL, 2006]. Ele determina o que se pode tratar com o software a ser construído.

A noção de semântica é muito importante para modelos em geral. O termo semântica [METAMODEL, 2006] reflete a necessidade de, não apenas modelar alguma coisa do mundo real, mas também o significado que esta coisa tem para o propósito do metamodelo. Uma ferramenta gráfica é um bom exemplo para ilustrar esse conceito. Se alguém desenhar uma caixa na tela, utilizando uma ferramenta dessas, ela entende que é uma caixa. Na perspectiva da ferramenta, a semântica é “caixa na tela”. Mas se alguém utiliza esta ferramenta gráfica para documentar, por exemplo, um fluxo de trabalho de uma empresa, então para a “caixa na tela” esse alguém tem em mente outro significado: ou um “processo” ou uma “atividade” [METAMODEL, 2006]. Vemos então que existem diferentes opiniões sobre o significado de uma determinada coisa. Cada significado depende da perspectiva usada na sua construção.

2.1 Definição e Objetivo da MDA

A MDA [KLEPPE, 2003] é um arcabouço para desenvolvimento de software definido pelo OMG (Object Management Group) e que se baseia nas tecnologias deste grupo, como o MOF e a UML, e para aplicações de Data Warehouse, o CWM. Alguns destes assuntos são tratados neste Capítulo. MDA surge como uma nova maneira de escrever especificações e desenvolver aplicações. O desenvolvimento baseado em MDA está focado em funcionalidade e comportamento de um sistema de informação ou aplicação distribuída. Podemos dizer que MDA é uma arquitetura para criação de bons projetos nos ambientes de tecnologia de informação multiplataforma existentes hoje.

A análise da sigla, "*Model Driven*" mostra que a MDA provê um significado para o uso de modelos. Estes modelos são usados para direcionar o entendimento, projeto, construção, entrega, operação e manutenções corretivas e evolutivas em sistemas informatizados. Vista também como uma abordagem para especificação de sistemas de tecnologia da informação, ela se propõe a separar uma especificação funcional da sua especificação de implementação em uma plataforma específica. Entendemos por plataforma neste contexto, um conjunto de subsistemas/tecnologias que provêem um conjunto coerente de funcionalidades através de interfaces e de padrões especificados, que qualquer desenvolvedor pode usar para construir sistemas sem se preocupar com os detalhes de implementação da funcionalidade [OMG, 2003].

Segundo Kleppe [KLEPPE, 2003], a MDA começa com a idéia já conhecida de separar a especificação do sistema de seus detalhes de implementação, e também, de como o sistema usa a capacidade da plataforma em que se encontra. Assim, ela provê uma abordagem para especificar sistemas independentes de plataforma, especificar plataformas, e escolher uma plataforma específica para um sistema.

Ainda de acordo com Kleppe [KLEPPE, 2003], um aspecto importante da MDA é que ela abrange o ciclo de vida completo do sistema, cobrindo análise e projeto, programação (teste, construção de componentes), entrega e gerenciamento. A MDA define uma arquitetura para estruturação de modelos que efetivamente separa as preocupações relevantes à integração, interoperabilidade e portabilidade. Ela provê uma visão de maior detalhamento - "*zoom in*" e uma mais abstrata, conceitual - "*zoom out*" de qualquer modelo do sistema, expondo detalhes de objetos e interações, com uma arquitetura que separa uniformemente as especificações de suas realizações.

A MDA enfatiza fortemente a criação de desenhos, não apenas papel, mas modelos legíveis pela máquina armazenados em repositórios padronizados [KLEPPE, 2003]. Assim,

esses modelos podem ser entendidos por ferramentas automáticas, gerando esquemas, esqueletos de código, codificação de testes, e até roteiros de código para integração de sistemas em múltiplas plataformas usadas em um projeto. A principal missão da MDA é resolver problemas de integração. Portanto, seus três objetivos principais são [OMG, 2003]: **Portabilidade, Interoperabilidade** e **Re-usabilidade**.

A chave para a MDA está na importância dos modelos no desenvolvimento de software. Sua promessa é permitir uma definição de aplicação que seja legível pela máquina e modelos de dados que permitam flexibilidade de longo prazo para [OMG, 2003]:

- *Implementação:* uma nova infra-estrutura de implementação (efeito de uma tecnologia mais recente) pode ser integrada ou alcançada por um design já existente;
- *Integração:* podem ser automatizadas a produção de pontes de integração de dados e a conexão para novas infra-estruturas de integração;
- *Manutenção:* a disponibilidade do desenho num formato legível pela máquina dá aos desenvolvedores acesso direto à especificação do sistema, fazendo da manutenção uma tarefa mais fácil;
- *Teste e Simulação:* desde que os modelos desenvolvidos possam ser usados para gerar código, eles podem igualmente ser validados em relação aos requisitos, testados sob várias infra-estruturas, e ainda podem ser usados para simular diretamente o comportamento do sistema que está sendo desenhado.

2.2 O Ciclo de Vida do Desenvolvimento em MDA

O ciclo de vida da MDA não é muito diferente do tradicional. As mesmas fases são identificadas, como podemos ver na Figura 3 [KLEPPE, 2003]. Uma das principais diferenças deste ciclo de vida está na natureza dos artefatos que são criados durante o processo de

desenvolvimento. Para a MDA, os artefatos são modelos formais, isto é, modelos que podem ser entendidos pelo computador.

Os três modelos: PIM, PSM e Código são definidos pela MDA e essenciais para ela. Existe ainda um modelo anterior a estes, o CIM – *Computation Independent Model* que não é descrito em detalhes dentro da MDA, e que está diretamente relacionado ao Modelo de Negócio. Veremos um pouco sobre ele no Capítulo 3.

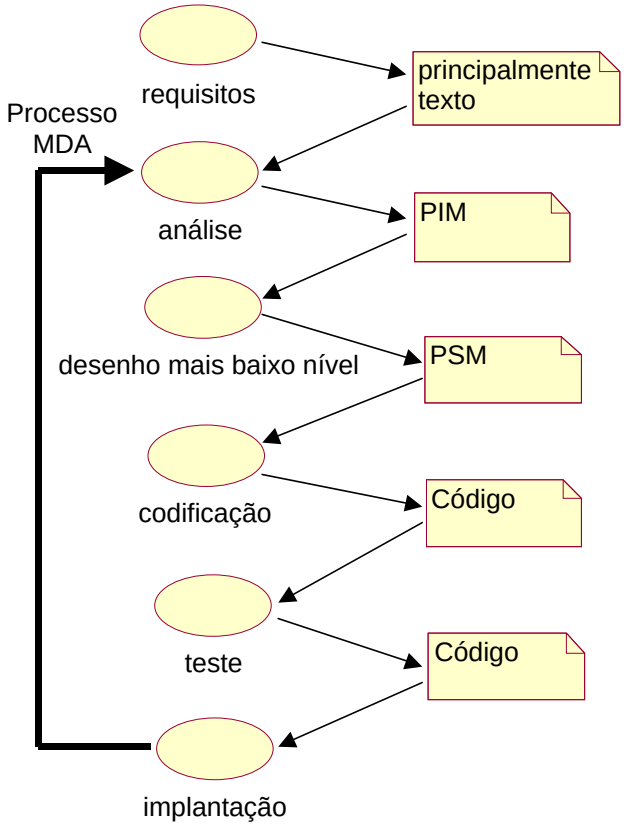


Figura 3 – Ciclo de Vida do Desenvolvimento de Software em MDA

A MDA explica como os três modelos: PIM, PSM e Código se relacionam [KLEPPE, 2003]. De acordo com a Figura 3, um PIM deve ser criado, então transformado em um ou mais PSMs , que por sua vez são transformados em código. O passo mais complexo no processo de desenvolvimento MDA é o momento no qual um PIM é transformado em um ou

mais PSMs. Muitos trabalhos neste sentido estão acontecendo com o objetivo de disponibilizar ferramentas capazes de transformar um modelo PIM em PSM.

O trabalho desta dissertação encontra-se na primeira fase do ciclo de vida do desenvolvimento da MDA. Detalharemos uma parte do Modelo PIM, no que se refere aos tipos de dados nele encontrados. Desta forma, estaremos contribuindo para que haja mais captura de informações para a próxima fase, que é a transformação em PSM. Vejamos um pouco mais sobre cada um destes modelos [KLEPPE, 2003] :

- **Modelo Independente de Plataforma (PIM – *Platform Independent Model*)**

Este primeiro modelo definido pela MDA é de um alto nível de abstração e é independente de qualquer tecnologia de implementação, e, portanto, independente de plataforma. Um PIM descreve um sistema de software que suporta alguns negócios. No PIM, o sistema é modelado do ponto de vista de como ele suporta melhor o negócio. Normalmente um PIM é escrito em UML. Neste alto nível de abstração, o modelo não é “apenas papel”, ele deve ser detalhado e consistente, diferentemente do modo de desenvolvimento tradicional.

Fazendo uma analogia ao Jornalismo, precisamos escrever modelos de alto nível que não representem apenas “manchetes”, mas que contenham a “notícia” na íntegra, em nível adequado de detalhes. Sendo assim, pode ser preciso descer vários níveis de abstração e detalhamento, mas tomando o cuidado para não explicitar comprometimento com nenhuma tecnologia.

- **Modelo de Plataforma Específica (PSM – *Platform Specific Model*)**

Num próximo passo, o PIM é transformado em um ou mais Modelos de Plataforma Específica - PSMs. Um PSM é feito sob medida para especificar o sistema em termos da construção da implementação. Ele contém termos específicos da tecnologia utilizada. Assim,

um PSM para banco de dados relacional, por exemplo, contém termos como "*table*" (tabela), "*column*" (coluna), "*primary key*" (chave primária).

É claro que o PSM faz sentido para desenvolvedores que detêm o conhecimento específico de uma plataforma. Geralmente, um PIM pode ser transformado em vários PSMs, um para cada plataforma específica escolhida para as partes correspondentes do sistema em desenvolvimento.

- **Código**

Este é o último passo no desenvolvimento – a transformação de cada PSM em código. Já que o PSM, por construção, se encaixa perfeitamente na tecnologia, esta transformação é relativamente direta.

2.3 Automação dos Passos de Transformação e Benefícios da MDA

O processo MDA pode ser bem parecido com o desenvolvimento tradicional. Contudo, existe uma diferença crucial. Tradicionalmente, as transformações de modelo para modelo e de modelo para código são feitas pelo desenvolvedor. Muitas ferramentas podem gerar algum código a partir de um modelo, mas isso não vai além da geração de alguns esqueletos de código vazios, onde a maior parte do trabalho deve ser feita à mão. Já as transformações MDA são sempre executadas por ferramentas. O que existe de novo são as transformações PIM → PSMs. Este é o ponto onde são gastos muitos esforços atualmente.

As ferramentas existentes hoje são capazes de gerar uma aplicação executando funcionalidades básicas a partir de um PIM, por exemplo, criando e modificando objetos em um sistema. Isso permite ao desenvolvedor ter um retorno imediato sobre o PIM em desenvolvimento, porque um "rascunho" do sistema resultante pode ser gerado automaticamente para cada versão de PIM.

Em termos de benefícios e melhorias no processo de desenvolvimento de software, vejamos como a MDA lida com os problemas levantados no Capítulo 1, segundo a visão de Kleppe [KLEPPE, 2003]:

- **Produtividade**

Em MDA, o foco do desenvolvedor é direcionado para o desenvolvimento do PIM. As transformações de PIM para PSMs precisam ser definidas, sendo esta uma tarefa difícil e especializada. Mas, cada transformação precisa ser definida apenas uma vez, podendo ser aplicada no desenvolvimento de diversos sistemas. A maioria dos desenvolvedores estará então focada no desenvolvimento dos PIMs. Os detalhes técnicos encontrados serão adicionados na especificação da transformação PIM → PSMs. Isso auxilia na produtividade de duas formas:

- Os desenvolvedores PIM têm menos trabalho a fazer, pois os detalhes tecnológicos não são considerados neste passo. No nível PSM e Código, há muito menos código a ser escrito porque boa parte será gerada a partir da descrição do PIM;
- Os desenvolvedores podem passar a dar mais atenção a resolver os problemas do negócio, traduzindo-os na modelagem do PIM e, como resultado disso, teremos um sistema que atende muito melhor às necessidades dos usuários e clientes finais.

Conforme vimos, para podermos ter um ganho na produtividade é necessário que o PIM não seja “apenas papel”, mas sim diretamente relacionado ao código gerado. A completeza e a consistência deste modelo são fundamentais para que ele possa sobreviver às mudanças de tecnologia.

- **Portabilidade**

Assim como a produtividade, a portabilidade é alcançada se focarmos no desenvolvimento dos PIMs. Os PSMs podem ser gerados automaticamente para diversas plataformas. Tudo aquilo que é especificado no nível PIM é completamente portável.

Para plataformas populares já existe um número de ferramentas de transformação disponíveis ou sendo desenvolvidas. Para as demais, talvez precisem ser desenvolvidas de acordo com a necessidade. Para novas tecnologias e plataformas que ainda estão por vir, a indústria de software precisará desenvolver as correspondentes transformações em tempo. Isso possibilita a portabilidade de sistemas prontos para uma nova tecnologia a partir dos PIMs existentes.

- **Interoperabilidade**

A Figura 4 mostra o método completo da MDA, com múltiplos PSMs gerados a partir de um mesmo PIM, e os possíveis relacionamentos entre tais PSMs, que na MDA são chamados de pontes [KLEPPE, 2003]. Quando PSMs estão em plataformas diferentes, eles não podem conversar entre si diretamente. É preciso transformar os conceitos de uma plataforma para os da outra. Para possibilitar essa interoperabilidade, gerando não só os PSMs mas também as pontes entre eles, é indispensável o uso de Metadados, que são discutidos no Capítulo 4.

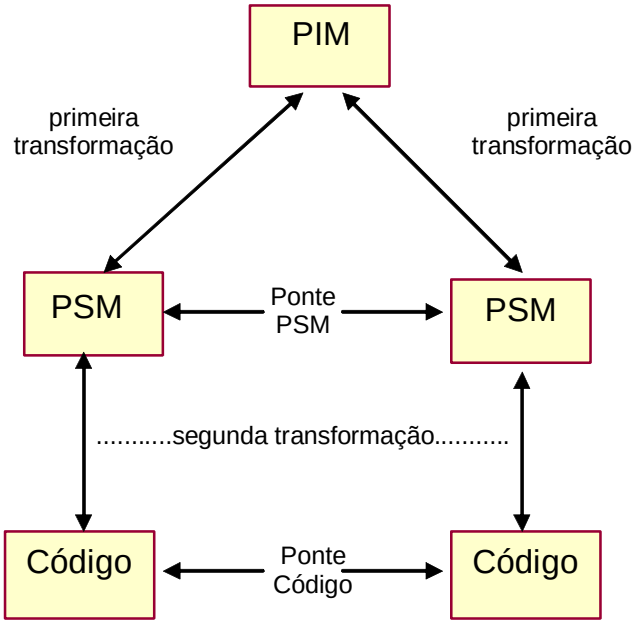


Figura 4 – Interoperabilidade na MDA usando pontes [KLEPPE, 2003]

- **Manutenção e Documentação**

A MDA preconiza que as mudanças no sistema sejam feitas no PIM, gerando novamente os PSMs e o Código. A documentação está no próprio PIM e nos PSMs, e as boas ferramentas devem conseguir manter este mapeamento PIM x PSMs. É bem verdade que, na prática, hoje muitas mudanças ainda são feitas no PSM, e o código é gerado novamente a partir deste. Quando essas mudanças precisarem ser refletidas no PIM, as boas ferramentas também podem ajudar a resolver o problema, mantendo a consistência entre os modelos. A Figura 5 [ABMANN, 2006] ilustra esse processo.

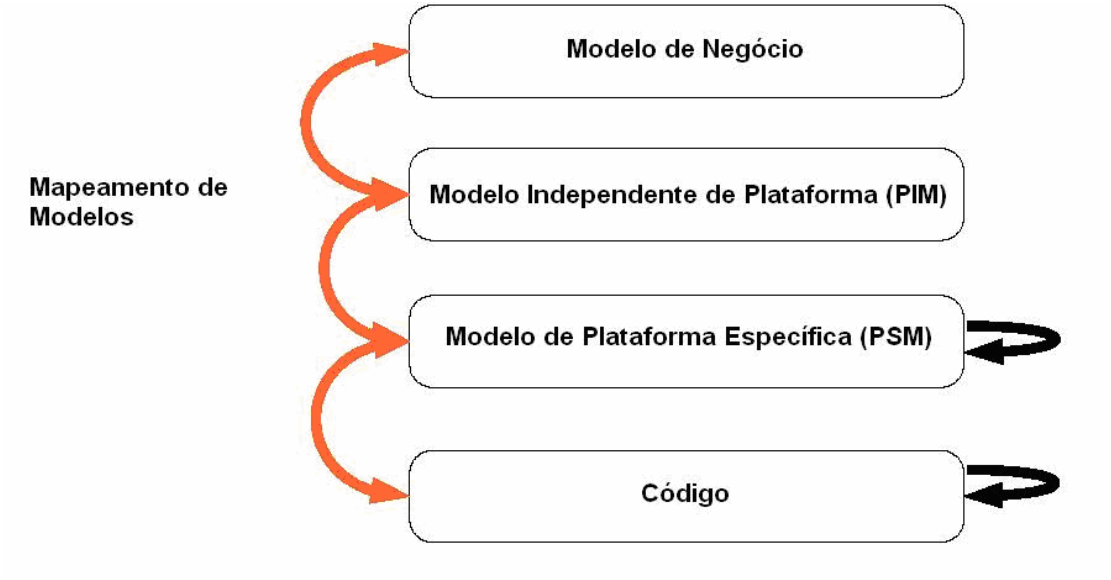


Figura 5 - Transformações em “mão-dupla”

2.4 Implementando um processo MDA

O que é necessário para implementar um processo em MDA [KLEPPE, 2003] :

1. Modelos de alto nível, escritos segundo uma linguagem padrão, que sejam consistentes, precisos e contendo informação suficiente sobre o sistema;
2. Um ou mais padrões, linguagens bem definidas para escrever modelos em alto nível;
3. Definições de como um PIM é transformado em um PSM que possa ser traduzido automaticamente (ferramentas de transformação PIM → PSMs);
4. Uma linguagem formal na qual sejam escritas estas definições, que serão interpretadas pelas ferramentas de transformação;
5. Ferramentas que implementam a execução das definições de transformação;
6. Ferramentas que implementem a execução da transformação de um PSM para código.

Para o modelo de alto nível citado no item 1, podemos considerar o modelo de negócio descrito no Capítulo 3, como sendo um *Computer Independent Model* - CIM [OMG, 2003]. No momento de construção de um CIM, deve ocorrer uma decisão sobre o que irá

fazer parte do sistema computacional e o que será deixado de fora. O modelo que descreve as partes componentes do sistema computacional, ao ser detalhado usando a UML, passaria a se caracterizar como um PIM.

Em resposta ao item 2, para o desenvolvimento da idéia desta dissertação, usamos o MOR, que é uma linguagem apresentada no Capítulo 3. A partir do item 3, já estaríamos fora do escopo desta dissertação, uma vez que estamos abordando a completeza do modelo no nível mais alto de abstração (CIM e PIM).

Muitos dos pré-requisitos relacionados à implementação de um processo em MDA já estão em desenvolvimento no OMG. Já se pode observar que as linguagens usadas na MDA deverão ter definições formais para que as ferramentas possam processá-las automaticamente. Vale lembrar que todas as linguagens padronizadas pelo OMG têm uma definição formal, como a UML e o MOF.

As definições de transformação PIM → PSM usadas em MDA ainda estão expressas de uma maneira não padronizada. Para permitir a padronização dessas definições, o OMG está trabalhando numa linguagem padrão para escrever definições de transformações chamada QVT (Query, Views and Transformations) [KLEPPE, 2003].

Segundo Kleppe [KLEPPE, pág.35], a UML tem hoje alguns pontos fracos que impedem a geração completa de um PSM a partir de um PIM. O problema está nos diagramas de comportamento ou dinâmicos. Existem vários diagramas para modelar a parte dinâmica do sistema, mas estes não são completos o suficiente e não utilizam uma linguagem formal para a geração do PSM. Um exemplo disso é o Diagrama de Caso de Uso da UML, discutido mais adiante, cuja descrição é feita através de uma linguagem natural.

As camadas existentes hoje entre a MDA e os sistemas de informação estão representadas na Figura 6 [OMG, 2006] a seguir:

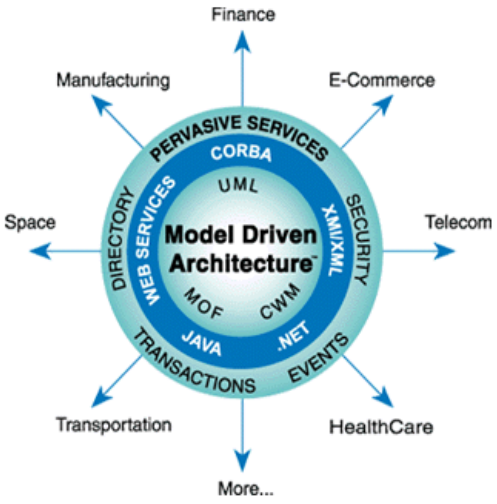


Figura 6 – Visão da MDA

A linguagem OCL (Object Constraint Language) [KLEPPE, 2003], antes usada para especificar apenas restrições nos modelos UML, apresenta-se atualmente bastante melhorada e útil. A combinação UML-OCL permite uma geração mais completa de PSMs e de código. Segundo Kleppe, atualmente esta é a melhor maneira de desenvolver PIM de alta qualidade e alto nível, porque apresenta um resultado de precisão, não-ambigüidade e consistência nos modelos, contendo mais informação sobre o sistema a ser implementado.

Complementando o histórico apresentado no Capítulo 1 e seguindo a abordagem MDA, o foco do desenvolvimento de sistemas deverá ser deslocado para outro nível. Hoje o foco ainda está na codificação. A tendência é que no futuro (bem próximo) este foco esteja voltado para o desenvolvimento do PSM, e, a partir daí, com o desenvolvimento de ferramentas de transformação padronizadas, o foco poderá então ser finalmente deslocado para o nível PIM.

MDA é uma tecnologia emergente que se encontra ainda na sua infância [KLEPPE, 2003]. Nem as linguagens nem as ferramentas estão suficientemente desenvolvidas para alcançar os 100% de geração automática de código que são prometidos pela MDA. Ainda são necessárias intervenções humanas no código gerado. Quanto mais completos e consistentes

forem os modelos desenvolvidos, melhor será o resultado das transformações nas fases da MDA. Para entender melhor os relacionamentos entre os vários padrões usados pela MDA, a Figura 7 mostra quatro níveis de abstração chamados na terminologia [OMG, 2003] do OMG de M0, M1, M2 e M3.

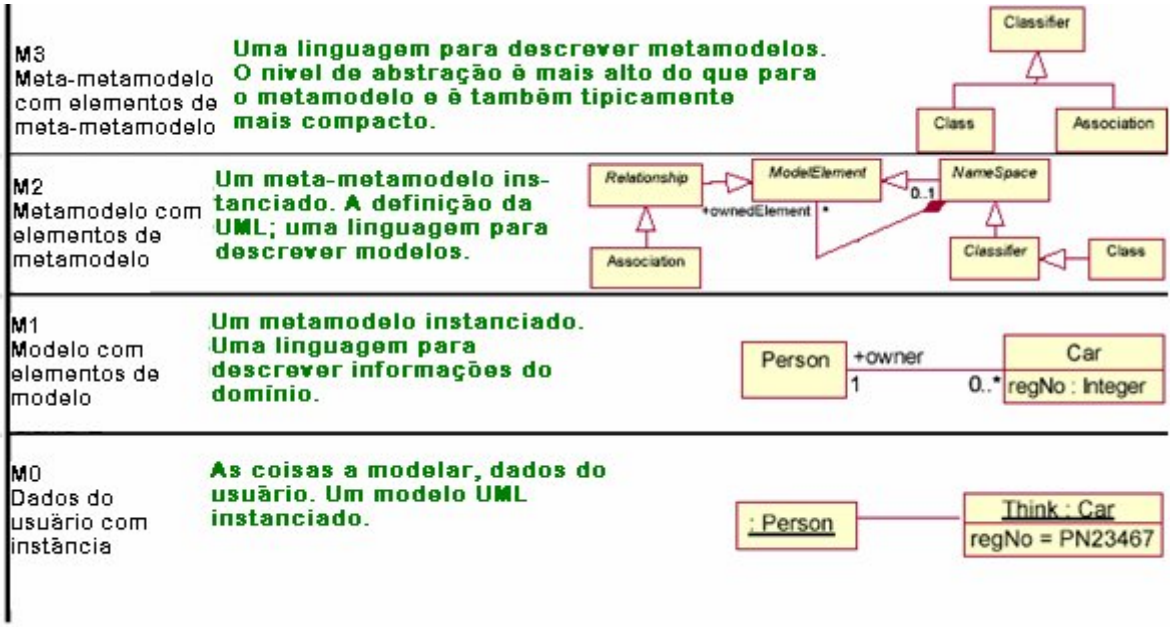


Figura 7 – The OMG 4-Layer Metamodel Architecture [NYTUM, 2005]

Assim, temos:

- **M3** - nível de metametamodelo – contém apenas o MOF, que é uma linguagem para descrever metamodelos;
- **M2** - nível de metamodelo – contém qualquer tipo de metamodelo, incluindo os metamodelos da UML e do CWM, que são linguagens para descrever modelos. É uma instância de M3;
- **M1** – nível de modelo – qualquer modelo com um correspondente metamodelo no nível M2. Descreve um domínio específico. É uma instância de M2;
- **M0** – nível concreto – qualquer situação real, única no tempo e no espaço, representada por um modelo do nível M1. Contém a própria instância do objeto.

Em princípio, poderíamos continuar adicionando mais níveis, mas na prática isso não é muito útil, pois não haveria fim. Ao invés de definir um nível M4, o OMG decidiu que todos os elementos do nível M3 devem ser definidos como instâncias de conceitos do próprio M3. Essa separação entre os níveis é artificial e não muito rígida. O essencial é o relacionamento entre esses modelos, ou seja, um modelo é “instância de” um outro modelo. Se todo elemento tem um meta-elemento, classificador através do qual os metadados podem ser acessados, qualquer modelo pode ser construído e qualquer sistema pode ser descrito.

Na realidade, todos os elementos de todos os níveis existem no mundo real e, portanto, pertencem ao nível M0. Alguns elementos de M0 são classificações de outros elementos e, desta forma, são classificados como nível M1, assim como existem classificações sobre essas classificações que pertenceriam ao nível M2 e assim por diante. Tudo depende do ponto de vista em que nos situamos. A Figura 8 mostra o Diagrama de Venn [NYTUM, 2005], que ilustra os subconjuntos de elementos nos níveis abordados.

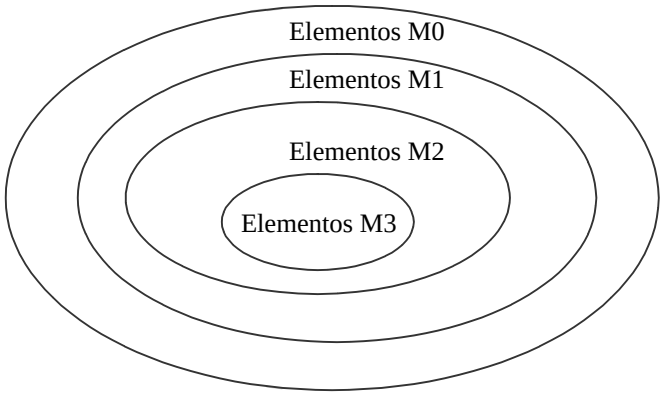


Figura 8 – Relacionamentos de Subconjuntos entre M0, M1, M2 e M3.

Vejamos um pouco mais sobre os níveis M3 – MOF e M2 – definição da UML.

2.5 MOF™ (Meta-Object Facility)

O MOF é um padrão do OMG que define uma linguagem abstrata para especificação de metamodelos [OMG, 2002]. Desta forma, o MOF é um *metametamodelo* que pretende suportar um amplo conjunto de modelos e aplicações.

De acordo com a Figura 7, o MOF situa-se no nível M3. O MOF é definido usando o próprio MOF. Isso mostra que o seu modelo é suficientemente expressivo para meta modelagem. Ele é orientado a objeto por natureza. É a linguagem usada para descrever a UML e o CWM, e serve como um metametamodelo comum para estes dois metamodelos. O MOF define os elementos essenciais, a sintaxe e a estrutura dos metamodelos que são usados para construir modelos orientados a objeto de sistemas distintos.

A especificação do MOF provê [OMG, 2002]:

- Uma definição formal do metametamodelo do MOF, isto é, a linguagem abstrata para especificar metamodelos MOF;
- Um modelo abstrato dos objetos MOF genéricos e suas associações;
- Um conjunto de regras para mapeamento de qualquer metamodelo baseado em MOF para uma interface independente de linguagem (definida em CORBA IDL);
- Um conjunto de interfaces padrão que pode ser usado para manipular metamodelos interoperáveis e seus correspondentes modelos;
- Definição de um repositório padrão para metamodelos e, portanto, para modelos;
- Regras que definem o ciclo de vida, composição e semântica dos elementos do metamodelo;
- Um formato XMI para troca de informações entre metamodelos MOF (o XMI provê um mapeamento do MOF para o XML);
- Uma hierarquia de interfaces.

O MOF especifica um pequeno, mas completo, conjunto de conceitos de modelagem que podem ser usados para expressar modelos de informação. O ponto forte do MOF é que ele permite, além de outras coisas, que metamodelos diferentes (representando domínios diferentes) sejam usados de forma a possibilitar a interoperabilidade.

Os principais conceitos do MOF são Classes, Associações, Tipos de Dados e Pacotes. A seguir, são apresentados os principais conceitos da OMG sobre as Classes, Associações e Tipos de Dados [OMG, 2002]:

- **Classes**

A Classe, como no sentido mais corriqueiro da palavra, é utilizada para descrever um conjunto de objetos que possuem propriedades em comum, tais como identidade, estado e comportamento. Ela é usada para modelar os metaobjetos do MOF. As classes podem ter três tipos de informações: Atributos, Operações e Referências. Elas podem conter também exceções, constantes, tipos de dados, restrições e outros elementos.

Um Atributo define um item conceitual que representa uma propriedade de estado em cada instância da classe, a qual possui as seguintes características:

- Nome – deve ser único no escopo dos atributos de uma determinada classe;
- Tipo – pode ser um tipo de dado ou um tipo complexo (de outra classe);
- Indicador “modificável” (“isChangeable”) – determina se o usuário possui uma operação para modificar o valor do atributo;
- Indicador “derivado” (“isDerived”) – determina se o conteúdo é parte de um estado explícito da instância da classe ou derivado de outro estado;
- Multiplicidade – determina valores mínimos e máximos de ocorrências do atributo.

Cada operação especifica o nome e a assinatura através dos quais um comportamento é invocado. Uma operação possui as seguintes propriedades:

- Nome – único no escopo da classe;
- Lista de parâmetros posicional contendo o nome do parâmetro, o tipo (dados ou classe), direção (entrada, saída ou ambos), multiplicidade;
- Um tipo de retorno opcional;
- Uma lista de exceções que podem ser apresentadas.

A referência é a informação da relação de conhecimento da instância da classe em relação às instâncias de outras classes do modelo. A classe pode ou não ter referências para outras classes.

O MOF permite que cada classe possa herdar propriedades de uma ou mais classes. Seguindo a mesma linha da UML, o modelo MOF usa o verbo “generalizar” para descrever o relacionamento de herança (por exemplo, uma superClasse generaliza uma subClasse). Pelo caminho inverso é usado o verbo “especializar” (por exemplo, uma subClasse “especializa” uma superClasse).

Uma subClasse herda todo o conteúdo de suas superClasses. Mas devemos observar algumas restrições:

- Uma Classe não pode se autogeneralizar, direta ou indiretamente;
- Uma Classe não pode generalizar uma outra classe se a subClasse contém um elemento do modelo com o mesmo nome do elemento do modelo contido ou herdado da superClasse, isto é, não é permitida a sobreposição de propriedades;
- Quando uma classe tem múltiplas superClasses, os nomes dos elementos dos modelos herdados não podem ter o mesmo nome, a menos que as superClasses estejam herdando nomes de uma classe ancestral comum.

As classes podem ser definidas como Classe Abstrata, Raiz ou Folha. A Classe Abstrata é usada somente para herança. Nenhum metaobjeto pode existir correspondendo a uma classe abstrata, isto é, a classe abstrata não é instanciável. Para as Classes Raiz e Folha

temos: quando uma classe é declarada como folha, fica proibida a criação de subClasses para ela; por outro lado, ao declarar uma classe como raiz proíbe-se a criação de superClasses para ela.

- **Associações**

A associação representa um relacionamento entre instâncias de duas classes. Ela pode representar ainda se um relacionamento é identificador, isto é, se uma das classes participantes não existe fora do escopo da outra.

Conceitualmente, as associações não possuem identidade de objeto e, portanto, não podem ter atributos e operações. Cada associação MOF possui duas extremidades, cada uma com as seguintes propriedades:

- Nome para a extremidade – único dentro da associação;
- Tipo para a extremidade – precisa ser uma classe;
- Especificação da multiplicidade – conceitualmente semelhante à multiplicidade de atributos, porém se refere à quantidade de ocorrências de instâncias de classes associadas;
- Especificação da agregação – o MOF suporta 2 tipos de semântica: composição (todo-parte, onde existe uma dependência de existência entre as classes associadas – relacionamento identificador) e não-agregação ou composição (não existe amarração ou dependência de existência no relacionamento entre as instâncias da classe);
- Navegabilidade – determina “quem conhece quem” no relacionamento, podendo ser para uma das extremidades da associação ou para ambas, caracterizada pela existência de uma referência.

- **Tipos de Dados**

Definições de metamodelo freqüentemente precisam usar valores, tanto para atributo como para parâmetro de operação. O MOF provê o conceito de meta modelagem de um tipo de dado para dar suporte a essa necessidade.

Os tipos de dados podem representar tipos primitivos e complexos. Os tipos primitivos, como string, boolean e inteiro, são a base para expressar seu conteúdo. O MOF define seis tipos de dados padrão, que são considerados adequados para meta modelagem neutra, sem tecnologia definida. São eles [OMG, 2002]:

1. **Boolean** – usado para representar valores verdadeiros ou falsos. Domínio: ‘true’ e ‘false’
2. **Integer** – subconjunto de números inteiros de 32 bits (complemento a dois), dentro do intervalo -2^{31} a $+2^{31} - 1$
3. **Long** – subconjunto de números inteiros de 64 bits (complemento a dois), dentro do intervalo -2^{63} a $+2^{63} - 1$
4. **Float** – subconjunto de números racionais que correspondem aos valores representáveis como números de ponto flutuante IEEE de precisão simples (para maiores detalhes, consultar ANSI/IEEE Standard 754-1985).
5. **Double** – subconjunto de números racionais que correspondem aos valores representáveis como números de ponto flutuante IEEE de precisão dupla (para maiores detalhes, consultar ANSI/IEEE Standard 754-1985).
6. **String** – conjunto infinito de seqüências de caracteres de 16 bits.

Os tipos complexos são, por exemplo, classes, estrutura de tipos, coleções e tipo alias.

Na Figura 9, vemos a classe DataType, suas subclasses e classes relacionadas [OMG, 2002]. DataType é a superClasse das classes que representam tipos de dados MOF e tipos de

dados construtores. Dentre estes tipos, vamos considerar especialmente os 6 (seis) tipos primitivos apresentados anteriormente.

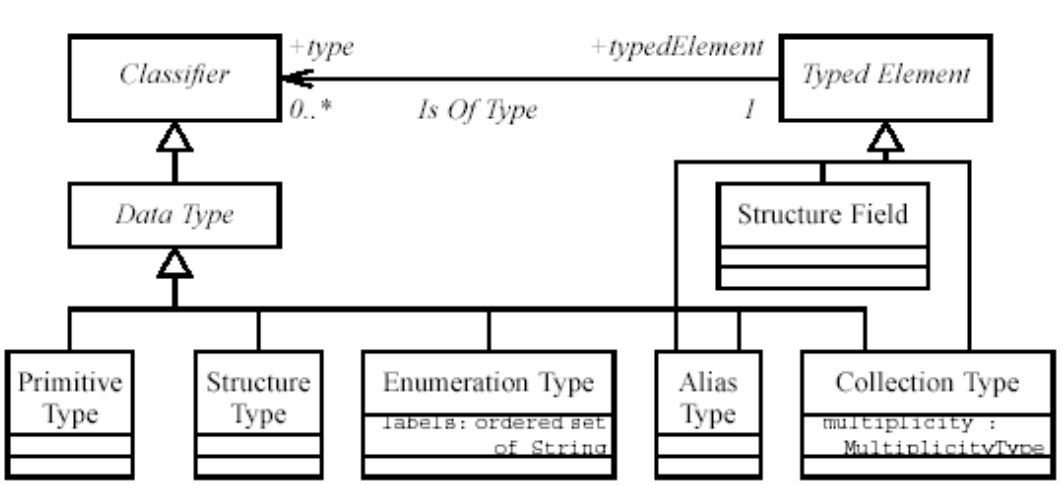


Figura 9 – Elementos de Tipos de Dados MOF

O papel do MOF dentro da MDA é fornecer os conceitos e ferramentas para formular as linguagens de modelagem [KLEPPE, pág.134]. Usando a definição de linguagem de modelagem do MOF (metamodelo no nível M2), podemos definir transformações entre as linguagens de modelagem. Sem um padrão para descrever metamodelos, as transformações não podem ser definidas e a abordagem MDA seria difícil de realizar.

Sendo assim, o MOF é vital para disponibilizar a tecnologia para a MDA. A Figura 6, apresentada anteriormente, mostra o MOF como um dos “ingredientes” centrais para suporte à MDA.

2.6 UML™ (Unified Modelling Language)

A UML é uma linguagem [OMG, 2005] também definida pelo Object Management Group (OMG), que vem promovendo um bom design, com uma linguagem comum visual, amplamente aceita e entendida. Ela vem resolver o problema da torre de Babel das notações pré-UML [FOWLER, 2003], apesar de muitas diferenças ainda persistirem entre o que as

ferramentas gráficas mostram e permitem quando são desenhados diagramas UML. Mesmo com essas diferenças, houve um aumento na modelagem visual de sistemas.

Na maioria dos projetos de desenvolvimento de sistemas, infelizmente, a modelagem visual tem sido referenciada meramente como “uma maneira de desenhar figuras” que depois precisam ser trabalhadas e detalhadas para serem traduzidas em código. A proposta desta dissertação é uma contribuição para que estes modelos sejam mais bem definidos a fim de que possam ser interpretados automaticamente pelo computador. Assim, é necessária uma linguagem bem definida. Uma tal linguagem é aquela que tem sintaxe (formato) e semântica (significado) bem definidas.

Uma das desculpas mais freqüentes para não se dar muita atenção ao design é que modelos compreensíveis são “apenas papel”, e o esforço despendido na sua criação poderia ser mais bem usado escrevendo código real. Isso ocorre em parte porque não são aproveitadas todas as potencialidades da linguagem. É bom lembrar que bons modelos de software fazem a codificação ficar mais fácil, e, em muitos casos, mais precisa.

A UML é a linguagem de modelagem padrão no nível M2 (ver Figura 7), definida usando-se o MOF, isto é, a UML é uma instância do MOF. Atualmente, é a linguagem de modelagem mais usada pela comunidade de tecnologia da informação.

A especificação da UML define uma linguagem gráfica para visualização, especificação, construção e documentação de artefatos para sistemas de objetos distribuídos. Ela é baseada no paradigma da orientação a objetos. Essa especificação inclui:

- Uma definição formal do metamodelo da UML;
- Uma notação gráfica para expressar modelos UML;
- Um conjunto de interfaces para representar e gerenciar modelos UML;
- Um formato XMI para troca entre modelos UML.

A UML foi definida de modo a ser uma linguagem de modelagem visual fácil de entender e semanticamente rica, incorporando as boas práticas do desenvolvimento de software. Ela permite a modelagem de arquitetura, objetos, interação entre objetos, aspectos de modelagem de dados do ciclo de vida da aplicação, bem como aspectos de projeto de desenvolvimento baseado em componentes.

O objetivo deste Capítulo é resumir apenas alguns dos pontos mais importantes da UML, que são necessários para o entendimento da idéia desta dissertação. Para uma abordagem mais completa sobre o assunto, sugerimos a leitura de [FOWLER, 2003] e [RUMBAUGH, 2000] .

2.6.1 Diagramas

Um modelo UML pode ser tanto PIM como PSM [OMG, 2005]. E um modelo bem documentado pode ser uma boa fonte para geração de código. A UML ajuda não só a fazer desenhos “ocos”, mas também a especificar, visualizar e documentar modelos de sistemas. Pode ser usada também para modelagem de negócios e para modelagem de sistemas computacionais e não-computacionais.

A UML possui treze (13) tipos de diagramas-padrão divididos em três (3) categorias [OMG, 2005]:

- **Diagramas Estruturais** – inclui Diagrama de Classe, Diagrama de Objeto, Diagrama de Componentes e Diagrama de Pacotes;
- **Diagramas Comportamentais** – inclui o Diagrama de Caso de Uso, Diagrama de Atividades, Diagrama de Colaboração e Diagrama de Estado;
- **Diagramas de Interação** – inclui o Diagrama de Seqüência, Diagrama de Comunicação.

Dentre eles, os principais são o Diagrama de Classe, que mostra características estáticas, o Diagrama de Caso de Uso e o Diagrama de Atividades. Esses diagramas podem mostrar muita ou pouca informação, de acordo com a necessidade. Assim, é possível preparar vários modelos incrementais com a mesma ferramenta.

O enfoque deste Capítulo é no Diagrama de Classe, para representação do modelo de dados orientado a objeto que serviu de fonte para a idéia desta dissertação, e no Diagrama de Caso de Uso, para representação de processos de negócio e entendimento do negócio. São apresentadas também algumas ferramentas utilizadas para o desenho desses diagramas.

- **Diagrama de Classe**

O ponto mais forte da UML é a modelagem dos aspectos estruturais de um sistema. Isso é feito principalmente através do uso de modelos de classe. A Figura 10 mostra um exemplo do Diagrama de Classe em UML.

Explorando os elementos do Diagrama de Classe apresentado na Figura 10 e ilustrando a especificação do MOF (item 2.5), vemos as Classes representadas por retângulos. Na parte superior do retângulo aparece o nome da classe, logo abaixo seus atributos e em seguida suas operações. As linhas que ligam as classes representam as Associações. A seta que aparece na extremidade de algumas linhas representa a navegabilidade. As cardinalidades 0..*, 0..1, 1..1, 1..* determinam a multiplicidade na associação.

Figura 10 - Exemplo de Diagrama de Classe

- **Diagrama de Caso de Uso**

O Diagrama de Caso de Uso permite capturar eventos de negócio, analisando como objetos externos interagem com o sistema. Ele representa uma seqüência particular de

interações entre o sistema e um ator, que pode ser um usuário final ou um sistema externo ao sistema que está sendo analisado.

Podemos usar o Diagrama de Caso de Uso para analisar requisitos do sistema e para ajudar a definir o escopo do mesmo. É um diagrama que pode ser validado com o usuário devido à sua simplicidade de apresentação. Um Diagrama de Caso de Uso principal pode ser detalhado em diversos diagramas de caso de uso mais específicos. A Figura 11 mostra um exemplo de Diagrama de Caso de Uso simplificado para uma Livraria Virtual segundo a UML. Este exemplo contempla apenas as principais funcionalidades do sistema, tais como Efetuar Pedido, Processar Pedido e Encomendar Publicações, sem considerar neste momento o relacionamento comercial com os fornecedores.

O Diagrama de Caso de Uso deve conter os seguintes elementos [AGILE, 2005]:

- **Casos de Uso** – um caso de uso descreve uma seqüência de ações que provê algo de valor mensurável para um ator. Ele é representado graficamente como uma elipse horizontal;
- **Atores** – Um ator representa uma pessoa, uma organização ou um sistema externo (ao sistema em questão) que desempenha um papel em uma ou mais interações com o sistema. É representado graficamente como um “bonequinho”;
- **Associações** – Associação entre ator e caso de uso é representada como uma linha sólida no diagrama, conectando um ao outro, com uma seta opcional em uma das extremidades. A seta é freqüentemente usada para indicar a direção de invocação do relacionamento ou para indicar o ator principal dentro do caso de uso. Alguns autores preferem não utilizá-la para evitar confusões do tipo: “Para que lado fica a seta?”. Uma associação existe sempre que um ator está envolvido com uma transação descrita pelo caso de uso.

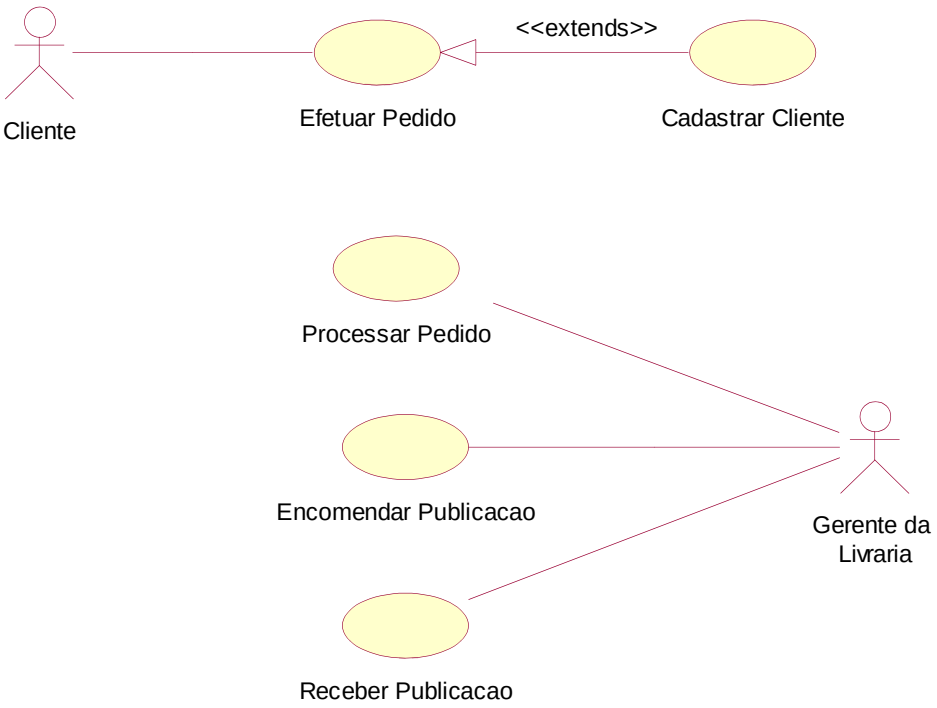


Figura 11 - Exemplo de Diagrama de Casos de Uso para uma Livraria Virtual

Opcionalmente pode conter também:

- **Caixas limite de sistema** – São pouco usadas. Podemos desenhar um retângulo em torno dos casos de uso que fazem parte do escopo do sistema. Tudo que estiver dentro da caixa representa funcionalidade que está no escopo do sistema. Pode ser utilizada também para delimitar as funcionalidades a serem entregues em cada versão.
- **Pacotes** – Pacotes ou *Packages* são construções UML que permitem a organização de elementos do modelo (como os casos de uso) em grupos. Pacotes são desenhados como pastas de arquivos e podem ser usados em qualquer diagrama UML. No caso de diagramas muito grandes, o agrupamento facilita a visualização e análise.

- **O Caso de Uso**

Depois de pronto o Diagrama de Caso de Uso, é hora de detalhar cada um dos casos de uso encontrados. Não há um padrão abrangente ou linguagem formal para esse detalhamento. Sendo assim, diversos autores sugerem como o caso de uso pode ser apresentado. A Figura 12 ilustra uma descrição de caso de uso.

Efetuar Pedido
ID: UC 1
Pré-condições:
- Não aplicável.

Pós-condições:
- Pedido pronto para ser processado e estoque atualizado.

Ator(s)	**Resposta**
O cliente se identifica Ator seleciona “Efetuar Pedido” Escolhe Publicação (ões) Finaliza Pedido Seleciona forma de pagamento e informa detalhes para pagamento Confirma Pedido	Caso o cliente não seja cadastrado, efetua seu cadastro através do Caso de Uso *Cadastrar Cliente*. Verifica que o cliente existe. Mostra publicações disponíveis Inclui Publicação (ões) como Item (ns) do Pedido para o cliente. Solicita confirmação para finalizar o pedido. Informa cálculo de frete Informa cálculo de desconto Informa cálculo de sumário de custos Solicita informação de pagamento Valida detalhes de pagamento Pede confirmação para efetuar o pedido. Inclui Pedido, Item(s) do Pedido Atualiza Estoque Provê confirmação com Número do Pedido.

Figura 12 – Exemplo de Descrição de Caso de Uso

Neste momento, a partir da descrição do caso de uso, podem ser identificadas as entidades do mundo real que aparecem nessa descrição e que precisam ser representadas na

persistência do sistema de informação. Elas darão origem ao Diagrama de Classes persistentes do sistema.

- **Algumas ferramentas**

Existem algumas ferramentas disponíveis hoje que auxiliam no desenho dos diagramas da UML. Visando a geração automática de código, um diagrama UML pode ser "traduzido" para XML no intuito de gerar um arquivo que possa ser lido por uma ferramenta de geração de código a partir deste formato. Esquemas XML podem ser derivados automaticamente de um modelo UML. Várias ferramentas efetuam esta derivação, dentre elas [IBM, 2006]:

- **The Eclipse Modeling Framework (EMF)**, conhecido formalmente como "The IBM XMI® Toolkit", inclui um gerador de código que deriva um esquema XML a partir de um modelo;
- **IBM Rational Rose**, que inclui RoseScript, uma linguagem de script que pode manipular um modelo UML e portanto salvá-lo como esquema XML;
- **Poseidon for UML** possui uma opção de geração de código que é facilmente customizada para gerar esquemas XML;
- **Codagen** oferece geração de código sofisticada em uma ferramenta UML;
- **Enterprise Architect**, ferramenta gráfica para desenhar diagramas UML com opção de geração de código XML e XMI.

- **O papel da UML na MDA**

A UML é a tecnologia facilitadora chave para a MDA, participando da sua fundação [KLEPPE, 2003]. Toda aplicação usando MDA é baseada num modelo UML independente de plataforma. A UML é uma linguagem bastante rica para representar tanto modelos Platform Independent Model (PIM), quanto modelos Platform Specific Model (PSM), contendo a

mesma informação que uma implementação, mas na forma de um modelo UML ao invés de código. Como vimos anteriormente na Figura 6, a UML está, junto com o MOF e o CWM, no núcleo da MDA.

Alguns pontos fortes são ressaltados como justificativa para a UML ser a linguagem escolhida para representar modelos desenvolvidos segundo a MDA [FRANKEL, 2003]:

- Separação da sintaxe abstrata da sintaxe concreta – a UML apresenta um modelo formal da sua semântica. Este modelo define os conceitos da UML e os associa com a sintaxe abstrata ao invés da sintaxe da notação gráfica. A notação gráfica é uma sintaxe concreta para expressar a sintaxe abstrata.
- Extensibilidade – permite a criação de linguagens especializadas para os vários aspectos dos sistemas. Esse é um dos requisitos-chave para um ambiente MDA, com cada linguagem apoiando a especificação de um aspecto particular do sistema;
- Suporte para Modelagem Independente de Plataforma – a UML, por sua natureza, suporta a criação de especificações de sistema que são independentes de plataforma, ao mesmo tempo em que se mantêm precisas e formais.

3 Modelo de Negócio

O conceito de Modelagem de Negócio, ou de Domínio como chamam alguns autores, apresenta-se de diversas formas. Isto gera interpretações diferentes sobre Modelagem de Negócio: o que é, qual o seu uso, como fazê-la, porque fazê-la e em quais circunstâncias [OLDFIELD, 2005]. A seção 3.1 apresenta um breve conjunto de respostas a estas questões.

3.1 Definição

O Modelo de Negócio [CUNHA, 2004] representa aspectos da empresa que interessam à engenharia de negócio. A empresa é modelada por seus processos de negócios, em duas visões:

- Externa - que permite entender as relações de negócios com o mundo exterior;
- Interna - que mostra o que tem de ser feito dentro da empresa para satisfazer às relações de negócios.

O modelo representa também a estrutura de comando da empresa, através de uma visão gerencial que mostra como o recurso humano é estruturado para ações de resultado e autogestão. Nessa visão administrativa, o Modelo de Negócio seria um modelo de mais alto nível da empresa, contendo todos os processos necessários para sua sobrevivência. Isso inclui os Processos Chave de Negócio e os Processos Empresariais de Negócio [CUNHA, 2004]:

- **Processo Chave de Negócio** – É voltado para o cliente e rende dinheiro para a empresa. Para cada (tipo de) produto que a empresa negocia existe um processo chave de negócio. Para cada (tipo de) cliente existe um processo chave de negócio.
- **Processo Empresarial de Negócio** - É voltado para o dono de empresa, consome dinheiro, e é necessário para constituir e manter viva a própria empresa. O processo empresarial de negócio tem como cliente o dono de empresa, o produto é

um serviço que a empresa presta ao dono de empresa, e o valor trocado é oriundo do capital ou do lucro da empresa.

Assim, a Modelagem do Negócio envolve o entendimento da estrutura e da dinâmica da organização, garantindo que clientes, usuários e desenvolvedores tenham a mesma visão da organização para a qual será construído o sistema de informação [CUNHA, 2004].

Segundo Oldfield [OLDFIELD, 2005], o Modelo de Domínio é um modelo dentro do qual uma empresa conduz seus negócios. O Modelo de Domínio para uma empresa pode ser muito semelhante ao de uma outra empresa que conduz seus negócios dentro do mesmo ramo de atividade. Os dois modelos podem possuir diversas partes em comum. Porém, quando o nível de detalhamento aumenta, podem existir diferentes idéias sobre o que deve constituir esse modelo. Neste momento, ele pode deixar de ser semelhante ao Modelo de Negócio. Freqüentemente, o termo “Modelo de Domínio” é usado para se referir ao Diagrama conceitual de Classes encontrado durante a análise.

Acreditamos que o termo Modelo de Domínio diz respeito a uma parte genérica do negócio que estamos modelando. Poderíamos dizer que se refere aos requerimentos conceituais de um sistema geral para uma determinada área de negócio. Sendo assim, o Modelo de Negócio de uma empresa deve ser considerado mais abrangente e detalhado que o Modelo de Domínio.

3.2 Objetivo

O Modelo de Negócio serve como ponto de partida para se entender a empresa, ainda num nível alto de abstração. É preciso ter uma visão geral do negócio para escolher o que será transformado em sistema computacional. O Modelo de Negócio [CUNHA, 2004] representa os aspectos da empresa que interessam à engenharia de negócio. A empresa é modelada por seus processos de negócios, numa visão externa, que permite entender as relações de negócios

com o mundo exterior, e numa visão interna, que mostra o que tem de ser feito dentro da empresa para satisfazer às relações de negócios.

Alguns autores acham que o Modelo de Domínio [OLDFIELD, 2005] tem como objetivo separar o que não varia muito do que costuma variar. As entidades no domínio, por se situarem em um nível mais abstrato, normalmente não mudam, e nem seus relacionamentos. Da mesma forma, ainda num nível abstrato, as responsabilidades executadas por estas entidades no domínio não mudam muito. Por sua vez, os aspectos arquiteturais, como persistência, apresentação, gerenciamento, e o uso do sistema, estes sim podem mudar (e quase sempre isso acontece!).

A Orientação a Objetos apresenta como um de seus objetivos a flexibilidade dos sistemas, estruturando-os em torno destes aspectos invariáveis com o tempo. A Modelagem de Domínio facilita a identificação de tais aspectos.

3.3 Modelando o Negócio

Dentro da visão da MDA [OMG, 2003], o Modelo de Negócio pode ser considerado como um CIM – Computation Independent Model. Sob este ponto de vista, o foco é o próprio negócio, que se constitui no ambiente do sistema e que determina os seus requisitos. Os detalhes de estrutura e processamento estão escondidos ou ainda não determinados. Às vezes, um modelo CIM é confundido com um Modelo de Domínio. Isto pode ser considerado incorreto, quando se supõe que o modelo CIM deve abranger, além das ações para tratamento de informações, todas as ações físicas e materiais necessárias à realização do negócio.

O CIM tem o papel muito importante de construir uma ponte entre aqueles que são especialistas sobre o domínio (o negócio) e seus requisitos, e aqueles que são especialistas em projeto e construção de artefatos que satisfazem os requisitos do domínio [OMG, 2003]. Os requisitos do sistema podem ser modelados dentro de um CIM, descrevendo a situação na

qual o sistema será usado. O modelo gerado desta maneira é independente, por construção, de como o sistema de informação é implementado.

Para a definição e o detalhamento do modelo deve ser usado um vocabulário e uma linguagem que sejam familiares às pessoas especialistas no negócio, vale dizer, no domínio. Este vocabulário e a linguagem correspondente devem fazer parte do que chamamos de Universo de Discurso. Para o detalhamento, neste trabalho adotamos a linguagem do MOR (Modelo de Objetos Relacionados) [CUNHA, 2004].

3.3.1 Universo de Discurso

Universo de Discurso é o vocabulário juntamente com a forma de expressão lingüística usados por um grupo de pessoas para se referir a uma realidade [CUNHA, 2004]. A adoção de um universo de discurso, que é realizada com o intuito de entender melhor o cliente, deve ser cuidadosa, pois o emprego errado da linguagem pode levar à ambigüidade. Na modelagem de negócio emprega-se o universo de discurso para [CUNHA, 2004]:

- Nomear as peças de modelagem;
- Descrever o que essas peças representam;
- Buscar aumentar a expressividade do modelo e facilitar a verificação de sua fidelidade à realidade.

3.3.2 MOR (Modelo de Objetos Relacionados)

Segundo Cunha [CUNHA, 2004], “O MOR é uma referência conceitual e um instrumento usado para descrever e compreender um universo de objetos associados por relações de conhecimento e classes associadas por relações de herança. É um sistema fechado constituído de objetos, classes, relações de conhecimento, relações de herança e uma máquina teórica de objetos.” De acordo com esta descrição, o MOR é puramente orientado a objeto.

- **A linguagem do MOR**

Devido à necessidade de padronização, será usada uma linguagem formal, padronizada para este estudo - a linguagem do MOR (Modelo de Objetos Relacionados) [CUNHA, 2004] para descrição e detalhamento dos modelos, na descrição dos elementos dos diagramas - definição de Classes, atributos, operações e relações de conhecimento, casos de uso do processo e atores envolvidos. Esta linguagem utiliza a notação BNF (Backus-Naur Form) equivalente e a notação de conjuntos da Matemática para a descrição dos Tipos Elementares de Dados, que são o tema do próximo Capítulo, que detalha como o MOR os descreve.

A linguagem BNF permite a definição de aspectos léxicos e sintáticos de forma concisa e é utilizada em compiladores. A notação BNF equivalente no MOR é usada para descrever regras de formação (sintaxe) de seqüências de símbolos através da combinação de construtores, terminais, conectivos booleanos “E” e “OU”, lógica de repetição e recursividade. Uma seqüência é válida quando obedece completamente à regra de formação. Descrever um conjunto de seqüências de símbolos é descrever a regra de formação de seus elementos.

O próximo Capítulo apresenta a notação BNF utilizada pelo MOR em mais detalhes, bem como um exemplo prático da sua utilização no detalhamento dos tipos de dados durante a modelagem de um sistema de informação.

4 TED (Tipos Elementares de Dados)

Este Capítulo completa o referencial teórico para o melhor entendimento da proposta de ferramenta apresentada no Capítulo 5. Como referências, foram consultadas principalmente as transparências de aulas de Fundamentos de MNOO [CUNHA, 2004] e de MNOO [CUNHA, 2004]. Os Tipos Elementares de Dados (TED) são nosso principal objeto de estudo na busca por facilitar uma geração automática de código.

A idéia básica de “tipificar” os dados, é a de acrescentar mais semântica aos dados que serão disponibilizados ou armazenados ao longo do ciclo de vida do sistema computacional. Durante este estudo, constatamos que os tipos de dados sofreram poucas modificações ao longo dos anos no que diz respeito à sua semântica.

4.1 Definição

Tipo Elementar de Dado ou simplesmente Tipo Elementar é um tipo de dado derivado dos tipos de dados primitivos conhecidos (Boolean, Integer, Long, Float, Double, String) ou derivado de um outro Tipo Elementar conhecido, isto é, já criado anteriormente. O Tipo Elementar é um tipo complexo. Ele representa um conjunto de valores e as operações reconhecidas como válidas sobre elementos desse conjunto [CUNHA, 2004]. Em outras palavras, os Tipos Elementares são tipos de dados definidos através de metadados que vão dar mais semântica ao dado, ajudando a garantir a sua confiabilidade. |Pode-se dizer que o Tipo Elementar é o tipo de dado “lapidado” para atender perfeitamente às características do sistema em questão. Uma vez que ele seja definido, não será necessário especificar suas regras de preenchimento e validação em qualquer outra parte do código, somente na sua própria classe.

De acordo com o MOR, os Tipos Elementares são descritos por classes usadas somente para esta finalidade [CUNHA, 2004], denominadas "*elementary type class*".

- O nome da classe é o nome do Tipo Elementar;

- A descrição semântica da classe é a descrição semântica do Tipo Elementar;
- A operação do objeto classe é usada para descrever a operação suportada pelo Tipo Elementar;
- A operação "*value*" do objeto classe:
    - *value* (valor : Symbols) : NomeTipoElementar - onde Symbols representa o conjunto de todos os caracteres representáveis. Vale a pena ressaltar que o MOR possui uma biblioteca de tipos elementares considerados básicos, constituída do tipo “Symbols” e dos seguintes tipos derivados dele por herança, cada um com sua respectiva semântica:
        - Numeric - representa o conjunto de todos os números;
        - Natural – representa o conjunto de todos os números naturais;
        - Positive – representa o conjunto de todos os números inteiros positivos, isto é, a união do conjunto dos números naturais com o valor “zero”;
        - Boolean – representa os valores enumerados: ‘true’ (verdadeiro) e ‘false’ (falso);
    - Retorna o próprio conteúdo do parâmetro valor, se ele pertencer ao conjunto de valores do Tipo Elementar.
    - Retorna "" (nulo) se o conteúdo do parâmetro valor não pertencer ao conjunto de valores do Tipo Elementar.

Tomemos como exemplo um Banco. As informações que circulam nos seus sistemas informatizados são, em grande parte, numéricas, como por exemplo, Saldo de Conta, Saldo de Aplicação X, Saldo de Aplicação Y, Valor de CPMF, Valor de Depósito, Valor de

Transferência, Valor de Retirada, Saldo de Poupança. Pode-se observar que em todas essas informações há um tipo de dado numérico. Mas, seria essa a única característica entre essas informações que poderiam ser colocadas em evidência? Certamente, não. Caso essas informações apresentem sempre o mesmo tamanho máximo de dígitos inteiros e decimais, esta regra também pode ser colocada em evidência.

Assim, descobrindo as características dos atributos, podem ser encontradas várias semelhanças entre eles, a fim de colocá-las em evidência na definição de um Tipo Elementar, para que essas características possam ser herdadas e reutilizadas toda vez que surgir um outro atributo com aquelas mesmas características. Desta forma, suas regras de preenchimento ficarão centralizadas e escritas apenas uma vez na classe de Tipo Elementar gerada, proporcionando um melhor entendimento do dado e facilitando futuras manutenções.

No exemplo do Banco, é possível a criação de um Tipo Elementar “ValorEmReal”, que possua as características comuns aos atributos exemplificados. São elas: tipo de dado numérico, e, por suposição, um número máximo de dígitos inteiros igual a 10 (dez) e um número de dígitos decimais igual a 2 (dois). Os atributos que representam valores, Saldo de Conta, Saldo de Aplicação X, Saldo de Aplicação Y, Valor de CPMF, Valor de Depósito, Valor de Transferência, Valor de Retirada, Saldo de Poupança, seriam associados a esse Tipo Elementar. Por exemplo, no caso de ocorrer um aumento do número máximo de dígitos inteiros para aqueles atributos, apenas o Tipo Elementar seria modificado.

Pode-se representar, de forma simplificada, através do metamodelo da Figura 13, o relacionamento entre as classes de Tipo Elementar (TED) e os Atributos de uma Classe. Um TED pode herdar diretamente de um Tipo Primitivo. Em alguns casos, pode haver um TED Especializado que herda de TED. Em outros, pode ocorrer um Tipo Elementar derivado de uma Agregação ou de uma Composição de Tipos Elementares. As reticências, logo abaixo da classe TED Especializado, indicam uma simplificação do desenho. A intenção é mostrar que

pode existir uma hierarquia de TEDs Especializados. Ainda por questões de simplificação e didática, também não aparecem possíveis relacionamentos de agregação e composição respectivamente das classes TED Agregado e TED Composto com a classe TED Especializado. Vale ressaltar que a classe Atributo pode estar relacionada às diferentes classes de TED, mas que esses relacionamentos são mutuamente exclusivos.

Fazendo-se uma comparação com o MOF, o diagrama da Figura 13 seria análogo ao nível M2.

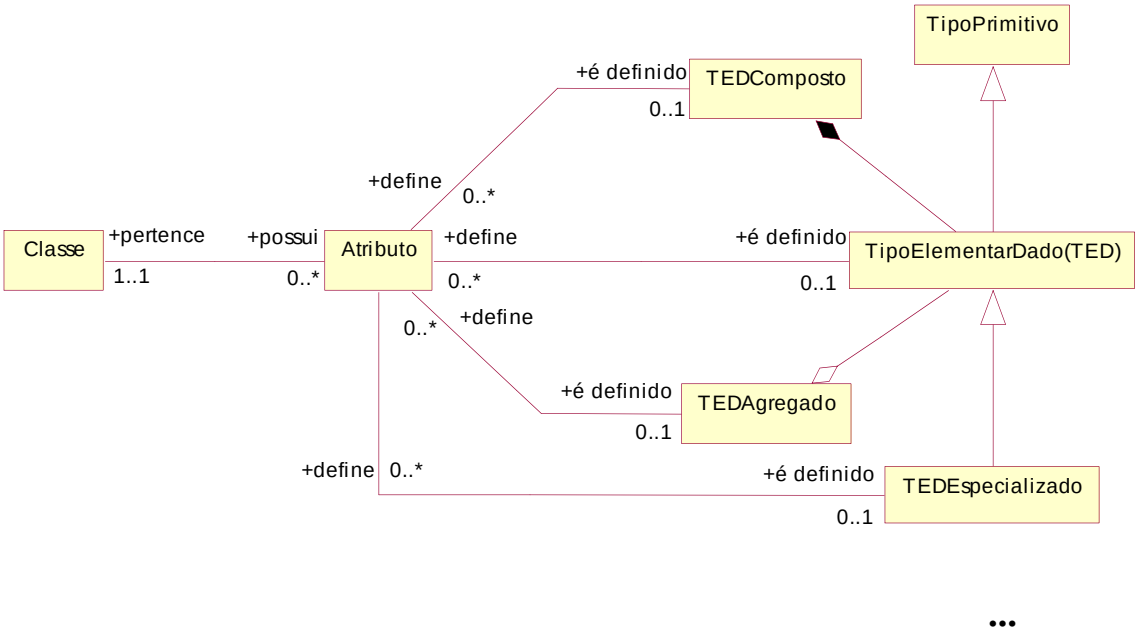


Figura 13 – Metamodelo num nível de abstração análogo ao nível M2 do MOF

Revendo o Diagrama de Classes apresentado anteriormente na Figura 10, como exemplo de nível M1, pode-se construir o diagrama da Figura 14, onde cada um dos atributos está agora associado ao seu respectivo Tipo Elementar de Dado:

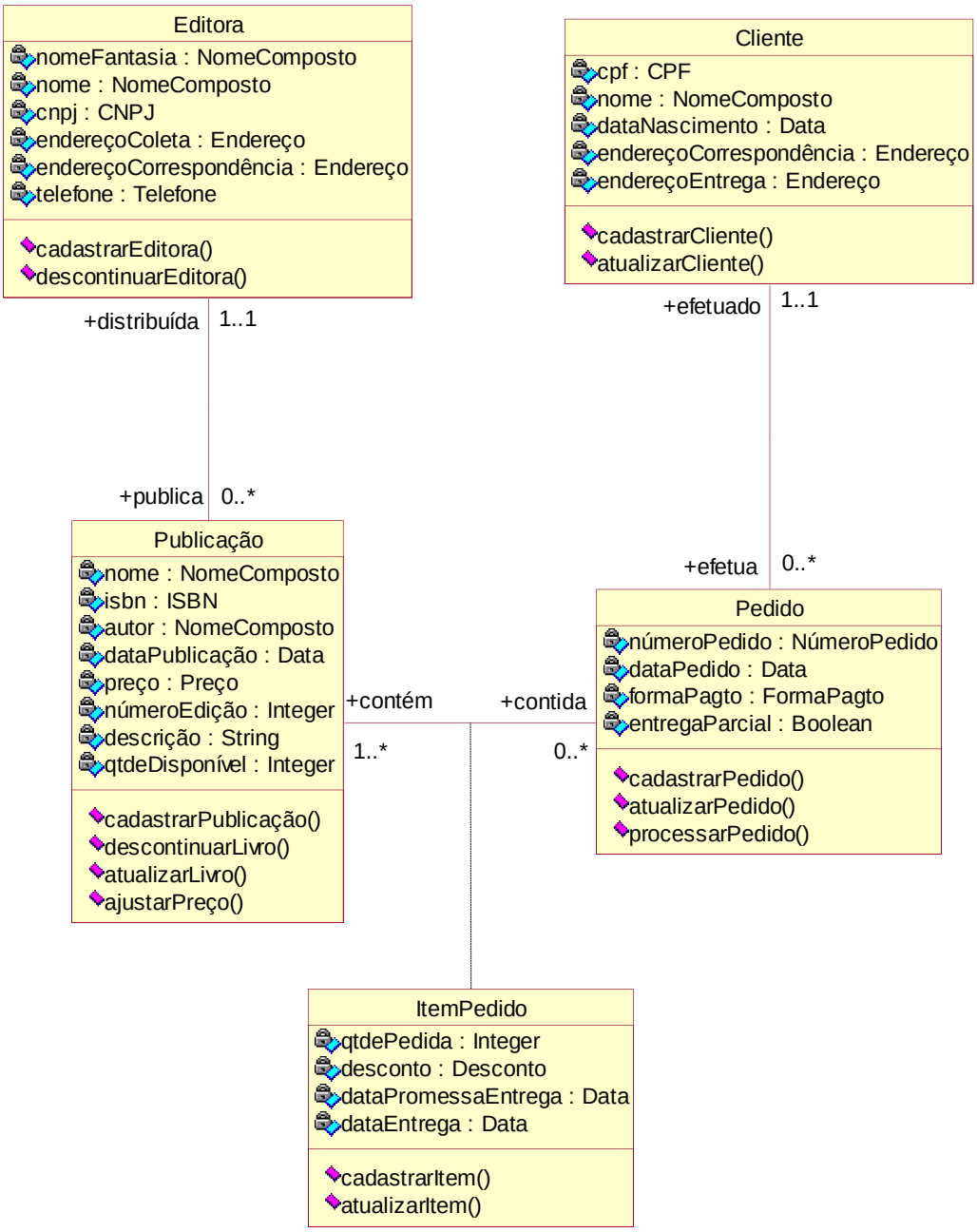


Figura 14 – Diagrama de Classes do sistema - nível M1

Os Tipos Elementares podem ser representados num outro Diagrama de Classes, ilustrado na Figura 15. A correlação entre esses dois diagramas é esclarecida no Capítulo 5, onde cada atributo do Diagrama de Classes do sistema é analisado para a geração do Tipo Elementar através da ferramenta proposta. No momento, o objetivo é mostrar que, dado um

Diagrama de Classes do sistema, é possível existir um diagrama paralelo que o complementa com a definição de TEDs.

É importante ressaltar que o diagrama de Tipos Elementares contém apenas classes abstratas, que foram criadas para reunir as definições das características de todos os atributos do Diagrama de Classes do sistema.

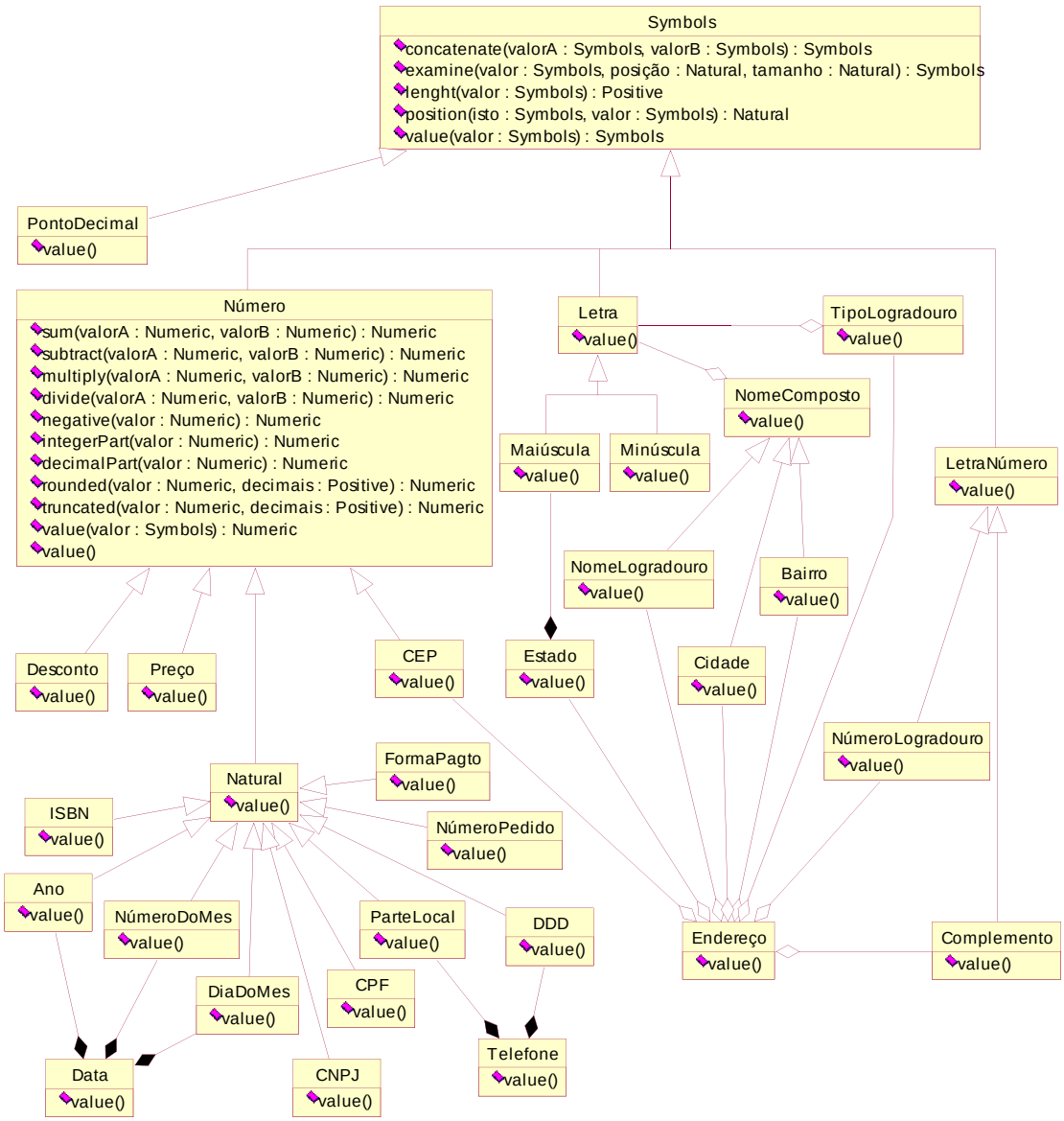


Figura 15 – Diagrama de Tipos Elementares

Estes dois diagramas do nível M1 têm um relacionamento entre si: cada atributo do Diagrama de Classes do sistema passa a ter o seu domínio definido por uma classe correspondente do Diagrama de Tipos Elementares. Este relacionamento pode ser representado no nível de abstração (M2 do MOF) da Figura 13. É possível também representar num só diagrama os relacionamentos entre os dois diagramas, onde cada Classe é resultante da agregação dos Tipos Elementares dos seus respectivos atributos. Assim, cada classe pode ser também um Tipo Elementar, neste caso, um Tipo Elementar composto.

A Figura 16 ilustra um possível mapeamento das classes entre os dois diagramas, considerando apenas a classe Cliente como exemplo.

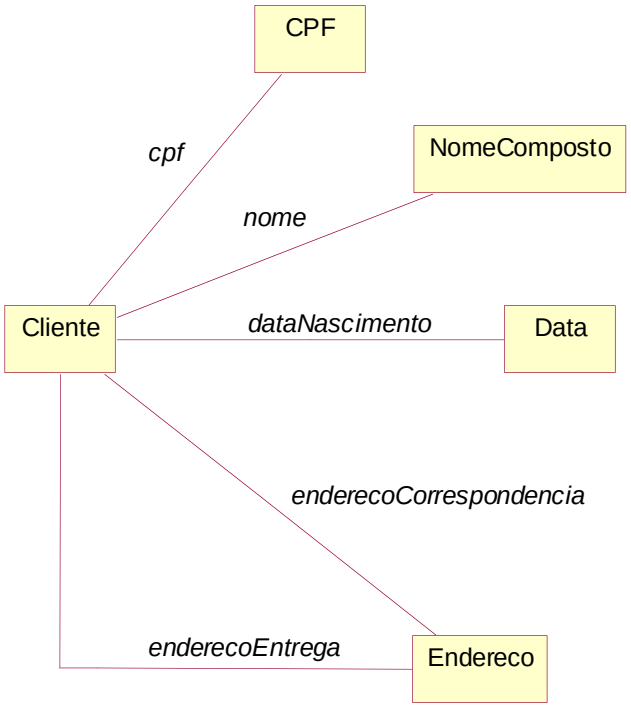


Figura 16 – Rede de Conhecimento entre os Tipos Elementares

Neste caso observa-se a formação de uma rede de classes num ambiente isotrópico. Esse ambiente homogêneo apresenta os Tipos Elementares e as classes, que agora podem ser consideradas Tipos Elementares complexos resultantes da agregação de vários Tipos Elementares.

O modelo da Figura 16 mostra uma visão que difere das anteriores, pois não apresenta atributos nas classes. Estes atributos estão representados através dos relacionamentos. Assim, temos, de maneira simplificada, uma Oração completa: Sujeito + Predicado, onde o Sujeito é a Classe de origem e o Predicado é o Relacionamento + Classe destino.

O Tipo Elementar de Dado deve possuir [CUNHA, 2004]:

a) **Nome** - Identifica o Tipo Elementar. É um resumo da descrição semântica. Uma palavra ou palavras emendadas por letra maiúscula. Exemplo: DiaDoMes .
b) **Descrição Semântica** - É uma descrição informal do conjunto de valores usando termos do universo de discurso dos valores. Deve conter exemplos de valores válidos. Exemplo para DiaDoMes: “1”, “30”.
c) **Descrição de Domínio de Valores** - Para conjuntos finitos e pequenos os valores são enumerados. Para conjuntos grandes ou infinitos duas técnicas são usadas:
    i. Notação BNF equivalente (Backus-Naur Form) – apresentada no item 4.3.
    ii. Notação de conjunto (matemática).
d) **Descrição de Operações Reconhecidas** - Cada operação aceita pelo Tipo Elementar é uma operação do objeto classe, descrita como uma operação de classe.
e) **Diagrama de Classes Abstratas** representando a Hierarquia de Herança de Tipos Elementares de Dados.

Comparando Tipo de Dado com Tipo Elementar de Dado:

- Dada uma classe ContaCorrente e um atributo “NúmeroContaCorrente” pertencente a esta classe, pode-se associar o Tipo de Dado “*Integer*” ao atributo.
- Dados o mesmo atributo e a mesma classe, pode-se ter uma classe representando um Tipo Elementar de Dado: NúmeroContaCorrente, definido como um valor “*Integer*” com tamanho de 5 (cinco) posições acompanhado de um dígito verificador, que pode ser um “*Integer*” com tamanho de 1 (uma) posição. Ainda

se pode definir o dígito verificador também como uma classe e disponibilizar uma operação para o seu cálculo.

Como se pode constatar nesse exemplo, o Tipo Elementar de Dado possui mais semântica e, conseqüentemente, pode proporcionar mais confiabilidade de conteúdo do que o Tipo de Dado puro. Está claro que o uso do tipo *Integer* é inadequado, deixando a desejar por não possuir características específicas. Por isso, sugiro o uso do tipo *Natural* do MOR apresentado anteriormente, pois ele pode resolver os problemas relacionados ao tamanho do campo e ao sinal (positivo ou negativo), atribuindo mais semântica ao dado modelado.

4.2 Principais funções dos Tipos de Dados

As funções principais que os tipos provêem (ou deveriam prover) são [WIKIPEDIA,2006]:

- **Segurança** - permite codificar operações que podem ser válidas ou não para um determinado contexto, aumentando a confiabilidade do dado;
- **Otimização** - o tipo do dado pode ser informação útil para o compilador;
- **Documentação** - melhora a documentação do código, melhora a semântica;
- **Abstração (ou Modularidade)** - permite pensar num nível mais alto ou para expressar interface entre sistemas.

Analisando estas funções com uma visão mais crítica, podemos concluir que os tipos de dados – primitivos e complexos – propostos pelo MOF no item 2.5 não atendem completamente às funções apresentadas acima e não são suficientes para uma boa geração automática de código. Por quê? Dado o exemplo anterior do tipo de dado númeroContaCorrente, observamos:

- Quanto à Segurança – um campo Integer aceita operações de soma, subtração, divisão e multiplicação, o que não é coerente para um número de conta corrente.

- Quanto à Documentação – apenas dizemos que o tipo é Integer. Onde está a semântica do dado?
- Quanto à Abstração – pensando num nível de abstração mais alto, na visão de negócio, o atributo númeroContaCorrente é um “número de conta corrente” com tamanho definido, com um dígito verificador, e não um “Integer”.

Os tipos de dados são usados hoje de forma muito genérica, dificultando a geração e o armazenamento de dados confiáveis. A programação de regras para garantir a consistência e integridade no preenchimento destes dados muitas vezes transforma funções simples em programas ilegíveis, complexos, e, conseqüentemente, de difícil manutenção.

A idéia dos Tipos Elementares ajuda a limitar corretamente o preenchimento dos dados, ainda num nível de abstração alto, que é o do domínio da aplicação. É neste nível que pode ser bem identificada a natureza do dado, suas características e restrições. Desta forma, são evitadas redundâncias no código, facilitando a manutenção do sistema.

4.3 Notação BNF (Backus-Naur Form) equivalente

A notação BNF [CUNHA, 2004] é usada para descrever a sintaxe, isto é, a regra de formação, de seqüências de caracteres (símbolos) através da combinação de:

a) Terminais - seqüência de símbolos escrita entre aspas indicando que a seqüência válida contém exatamente a seqüência de símbolos entre as aspas na mesma posição. Exemplo: “janeiro”

b) Construtores - denominação genérica para um conjunto de seqüências de símbolos que é descrito por uma regra de formação. Notação: nomeConstrutor = regraDeFormação. Exemplo: dígitoZero = “0”

c) Lógica OU - Uma lista de partes, terminal ou construtor ou repetição, separadas por barra (/) indica que somente uma delas pode participar da seqüência válida naquela

posição. Exemplos: dígitoNatural = "1" / "2" / "3" / "4" / "5" / "6" / "7" / "8" / "9"
dígito = dígitoZero / dígitoNatural

d) Lógica E - sucessão de partes, construtor, terminal, repetição ou grupo alternativo indicando que a seqüência válida é formada pela concatenação das possíveis partes, na mesma ordem em que aparecem. Exemplo: númeroDoisDígitos = dígito dígito

e) Lógica de repetição - repetição de partes, construtor ou terminal, significa que a seqüência válida é formada pela concatenação sucessiva das possíveis partes, limitadas por um número mínimo e um número máximo de vezes que a parte pode aparecer. Um número máximo arbitrariamente grande será denotado pela letra N. Exemplo: númeroInteiro = digito*(1-N)

f) Recursividade - o nome do construtor aparecendo indiretamente na regra de formação do próprio construtor. Esta regra tem que conter, obrigatoriamente, uma alternativa terminal para interromper a recursividade.

Uma seqüência de caracteres é válida quando obedece completamente à sua regra de formação. Para ilustrar a utilização dessas regras, a Figura 17 mostra a notação BNF equivalente descrita em BNF equivalente [CUNHA, 2004].

```
notaçãoBNFequivalente = nomeTipoElementar/ nomeConstrutor "=" construtor*(1-N)
construtor = construtorOpcional / construtorObrigatório
construtorObrigatório = construtorRepetitivo / grupoAlternativo / nomeConstrutor /
nomeTipoElementar /
constante / constanteSymbol / construtorSymbol / construtorExcept
construtorOpcional = " [" construtorObrigatório*(1-N) "]"
construtorRepetitivo = nomeConstrutor / constante "*" "(" limiteInferior "-" limiteSuperior ")"
grupoAlternativo = construtorObrigatório construtorAlternativo*(1-N)
construtorAlternativo = "/" construtorObrigatório
constante = aspa valor aspa
aspa = symbol(")
valor = símboloGráfico*(1-N)
símboloGráfico = except(")
constanteSymbol = "symbol" "(" symbol() ")"
construtorSymbol =  "symbol()"
construtorExcept = "except" "(" symbol() ")"
nomeConstrutor = Letra*(1-N)
nomeTipoElementar = Maiúscula  [ Letra*(1-N) ]
limiteInferior = Natural
limiteSuperior = Natural / "N"
Letra = Maiúscula / Minúscula
Maiúscula = "A" / "B" / "C" / "D" / "E" / "F" / "G" / "H" / "I" / "J" / "K" / "L" / "M" / "N" / "O"
/ "P" / "Q" / "R" / "S" / "T" / "U" / "V" / "W" / "X" / "Y" / "Z" / "Á" / "À" / "Â" / "Ã" / "É" /
"Ê" / "Í" / "Ó" / "Ô" / "Õ" / "Ú" / "Ü" / "Ç"
Minúscula = "a" / "b" / "c" / "d" / "e" / "f" / "g" / "h" / "i" / "j" / "k" / "l" / "m" / "n" / "o" / "p"
/ "q" / "r" / "s" / "t" / "u" / "v" / "w" / "x" / "y" / "z" / "á" / "à" / "â" / "ã" / "é" / "ê" / "í" / "ó"
/ "ô" / "õ" / "ú" / "ü" / "ç"
Natural = DigitoNatural  [ Digito*(1-N) ]
DigitoNatural = "1" / "2" / "3" / "4" / "5" / "6" / "7" / "8" / "9"
Digito = DigitoNatural / "0"
```

Figura 17 - Notação BNF equivalente descrita em BNF equivalente

Para auxiliar na descrição dos Tipos Elementares, é possível adotar a já conhecida notação de conjunto da Matemática com suas operações: União, Interseção, Diferença, além de limites como =, >, <, <= , >=. A Figura 17a apresenta a notação de conjunto descrita em BNF equivalente [CUNHA, 2004].

```
notaçãoDeConjunto = nomeTipoElementar "=" descriçãoConjunto / expressãoConjunto
descriçãoConjunto = nomeConjunto "|" restriçãoConjunto
restriçãoConjunto = restriçãoLimite / restriçãoTamanho
restriçãoLimite = restriçãoLimiteSuperior / restriçãoLimiteInferior /
restriçãoLimiteAmbos
restriçãoLimiteSuperior = nomeConjunto operadorLimite valor
restriçãoLimiteInferior = valor operadorLimite nomeConjunto
restriçãoLimiteAmbos = valor operadorLimite nomeConjunto operadorLimite valor
valor = aspa símboloGráfico*(1-N) aspa
aspa = symbol(")
símboloGráfico = except(") / aspaDupla
aspaDupla = symbol(") symbol(")
operadorLimite = "<" / "<="
restriçãoTamanho = "length" "(" nomeConjunto ")" "=" Natural
expressãoConjunto = nomeConjunto operaçãoConjunto*(1-N)
operaçãoConjunto = "UNION" / "INTERSECTION" / "DIFFERENCE" nomeConjunto
nomeConjunto = Letra*(1-N)
```

Figura 17a - Notação de conjunto descrita em BNF equivalente

A linguagem do MOR, mencionada no item 3.3.2 do Capítulo anterior, utiliza essa notação BNF equivalente para descrição dos Tipos Elementares de Dados.

4.4 Metadados

Desde os tempos antigos, a informação é usada para classificar, organizar e pesquisar. Na antiga Suméria, as placas de argila eram identificadas por fios coloridos conforme o tipo, e arrumadas em prateleiras com indicações escritas ao lado. Os escribas romanos amarravam molhos de documentos relacionados, etiquetavam-nos e penduravam-nos do teto. A diferença agora é que a informação é eletrônica, dispersa e cresce a uma velocidade exponencial.

Gilliland-Swetland em seu artigo *Defining Metadata* [SWETLAND, 2005] comenta:

> Até meados dos anos 1990, “metadados” era um termo usado principalmente por comunidades envolvidas com o gerenciamento e interoperabilidade de dados geoespaciais, [...] Para essas comunidades, “metadados” se referia a um conjunto de padrões bem como documentação adicional interna e externa e outros dados necessários para identificação, representação, interoperabilidade, gerenciamento técnico, performance e uso de dados contidos em sistemas de informação.

- **Definição**

Metadado é um termo atualmente entendido de diversas formas pelas diversas comunidades de profissionais que desenham, criam, descrevem, preservam e usam sistemas de informação e recursos informacionais. Sua importância cresceu exponencialmente nos últimos anos, principalmente com o desenvolvimento de sistemas para Web. Desde o surgimento dos bancos de dados tem sido ressaltada a importância da documentação dos sistemas e dos próprios bancos utilizados por eles. Atualmente, essa documentação se tornou ainda mais vital para a longevidade dos sistemas de informação.

O prefixo “Meta” vem do grego e significa “além de”, “uma natureza de uma ordem mais alta ou tipo mais fundamental”. Então, de forma simplificada e econômica, Metadados são “dados sobre os dados”, “informações sobre informações”, informações que caracterizam os dados, ou informações usadas para prover documentação para esses dados. Metadados respondem às perguntas: quem, o quê, quando, onde, por quê e como, sobre todas as faces do dado que está sendo documentado. Auxiliam na publicação e suporte dos dados que uma organização produz.

Metadados descrevem as propriedades de um objeto, que por sua vez também é um dado. Talvez, uma maneira de entender melhor o “todo”, seria pensar que metadados são o somatório total do que se pode dizer sobre um objeto de informação em qualquer nível de abstração. Considera-se neste contexto um objeto de informação como “um item ou conjunto de itens que possam ser endereçados e manipulados por um sistema como um objeto distinto” [SWETLAND, 2005].

Em geral, todo objeto de informação apresenta 3 aspectos [SWETLAND, 2005]:

- Conteúdo – o que o objeto contém ou a que se refere;
- Contexto – indica quem, o quê, por quê, onde e como, associados à criação do objeto;

- Estrutura – conjunto formal de associações entre os objetos de informação.

O papel da Estrutura tem crescido conforme a capacidade de processamento dos computadores se torna mais poderosa e sofisticada. Observa-se que quanto mais estruturado o objeto de informação, mais essa estrutura pode ser explorada para pesquisa, manipulação e inter-relacionamento com outros objetos de informação [SWETLAND, 2005]. Porém, metadados não servem apenas para descrição.

Vejamos um exemplo prático - imaginemos um cartaz onde somente apareça o número: ***2,1321***. O que isto significa? Provavelmente pode significar muitas coisas ou até mesmo nada. No entanto, o fato é que este dado existe. Se acrescentarmos ao mesmo cartaz outro dado relacionado ao primeiro, tal como a palavra **"Dólar" - *2,1321 Dólares***, essa nova informação sobre o dado lhe conferiu um significado. Assim, pode-se dizer que ao dado foi acrescentada uma informação a partir de um outro dado. A forma de representação do número também é outro metadado, ou seja, indica que o número é do tipo decimal.

Nessas interpretações, metadados não apenas identificam e descrevem um objeto de informação, mas também documentam como o objeto se comporta, suas funções e uso, seus relacionamentos com outros objetos de informação, e como este deve ser gerenciado. Fazendo uma breve analogia à Biologia e à Literatura, talvez seja possível dizer que metadados são o DNA do objeto de informação adicionado à sua biografia.

- **Fontes de Metadados**

Os metadados podem surgir de vários locais durante o decorrer do projeto. Por exemplo, eles podem vir de repositórios de ferramentas CASE, os quais geralmente já estão estruturados, facilitando a integração da origem dos metadados com o seu respectivo repositório. Essa fonte de metadados é bastante significativa.

Outros dados que devem ser guardados no repositório de metadados são os materiais que surgirão das entrevistas com usuários e outras pessoas envolvidas com o desenvolvimento

do sistema. Destas entrevistas, normalmente são obtidas informações preciosas que não estão documentadas em nenhum outro lugar, além de regras para validação dos dados. O volume de metadados que podem ser gerados é muito grande.

Todas as fases de um projeto, desde a modelagem até a visualização da informação, podem criar ou consultar os metadados. Neles estarão contidas informações como atributos das tabelas, cálculos necessários, descrições, periodicidade das cargas, histórico de mudanças, definições e regras de negócio, detalhes de segurança, informação de domínios, *tags* XML entre outras. A Figura 18 resume a coleta de metadados durante essas fases.

FASE 1 ***Levantamento de Dados***	***FASE 2*** ***Modelagem de Dados***	***FASE 3*** ***Projeto Físico***	***FASE 4*** ***Implementação***	***FASE 5*** ***Implantação***
Documentos gerados nesta fase: • Atas de reuniões (encontros com os usuários), • Documento de levantamento de dados, entre outros documentos.	Documentos gerados nesta fase: • Modelo de Domínio do Negócio e Modelo lógico do banco de dados, definição das classes e suas relações de conhecimento.	Documentos gerados nesta fase: • Modelo físico do banco de dados, geração do banco de dados, entre outros documentos.	Documentos gerados nesta fase: • Documento de onde virão as informações, as possíveis transformações que cada informação poderá sofrer, com suas regras de negócio, entre outros documentos.	Documentos gerados nesta fase: • Documentação de aprovação de todo o processo por parte do usuário.

Figura 18 - Fases do Desenvolvimento

Algumas informações que os metadados podem conter:

- A estrutura dos dados segundo a visão do programador;
- A estrutura dos dados segundo a visão do analista;
- A fonte de dados que alimenta o banco de dados;
- O Modelo conceitual de dados.

Adicionalmente, metadados podem ser oriundos de regras de negócio da empresa e de mudanças que elas eventualmente tenham sofrido, e também da freqüência de acesso aos dados. No contexto desta monografia são extraídos metadados para os tipos de dados encontrados durante a modelagem do negócio. A ferramenta proposta no próximo Capítulo é focada na captura dos metadados de cada um dos atributos adotados durante o projeto do sistema de informação, visando facilitar seu entendimento e possibilitar uma geração mais automática de código.

- **Importância dos Metadados**

Informações não registradas estão fadadas a perder o seu valor. Sem metadados, ou seja, sem semântica, não há como saber o que um objeto representa. Os metadados mantêm o valor do dado, garantindo o seu uso adequado e seguro ao longo do tempo. As organizações que possuem dados bem documentados não ficam vulneráveis e propensas a perder o conhecimento do negócio quando um empregado se aposenta ou aceita uma outra oferta de trabalho. Considerando que a informação é o patrimônio mais valioso de uma organização, metadados não são apenas mais uma opção para as organizações, mas uma necessidade. Essa é uma boa justificativa para que todos os envolvidos dediquem um pouco mais do seu tempo na obtenção das informações necessárias para a criação dos Tipos Elementares.

Toda informação sobre um determinado objeto faz parte do negócio. Desta forma, precisa ser documentada como tal. “Conhecimento é poder”. Por outro lado, a falta de conhecimento sobre os dados de uma organização pode levar à duplicação de esforços e outros tipos de desperdício. O valor da informação aumenta com o seu uso e quando combinada com outra informação.

Uma informação pode provocar o surgimento de outra informação. O seu uso tende a resultar em mais informação. A transformação de dados em informação tem sido fortemente impulsionada pela tecnologia que apóia as pessoas: Dados + Contexto = Informação

Estes avanços tecnológicos estão sendo absorvidos pela organização. A chamada tecnologia da informação ajuda a gerenciar não só a informação, mas também como as pessoas capturam, transformam e criam a informação e, além disso, como a compartilham com outros: Informação + Experiência = Conhecimento.

O conceito de metadados surgiu em função das necessidades das organizações em conhecer mais profundamente os dados que mantêm. Organizações que não documentam seus dados ficam sujeitas à superposição de esforços de coleta e manutenção de dados, vulneráveis a problemas de inconsistências e, principalmente, assumem um alto risco pelo não uso ou pelo uso impróprio da informação.

Portanto, metadados são essenciais para o adequado entendimento do dado. E isso se aplica aos tipos de dados. Boolean, integer, long, float, double e string isoladamente não possuem o significado suficiente para o que realmente se pode esperar do dado. É necessário capturar mais informações para que se possa tirar melhor proveito na automatização da produção do código. Vale lembrar que uma análise do custo-benefício sempre deve ser feita, pois criar bons metadados requer tempo e dinheiro.

Dentro do contexto da MDA e MOF, para que os metadados compartilhados sejam entendidos por todos os componentes participantes, um sistema baseado em MDA requer que seus componentes utilizem uma linguagem formal (sintaxe e semântica) para representação de metadados [POOLE, 2005]. Tal linguagem padronizada pode ser a XML.

4.5 XML® (eXtensible Markup Language)

E**X**tensible **M**arkup **L**anguage (XML) [XML, 2003] é derivada do SGML (ISO 8879), expressa em um formato texto simples, bastante flexível. Originalmente projetada para atender aos desafios de publicação eletrônica em larga escala, XML está sendo responsável por um importante papel: o compartilhamento e a troca de uma ampla variedade de dados,

hoje especialmente no ambiente Web, e também em outros ambientes de sistemas computacionais.

XML é uma linguagem ‘markup’, isto é, uma linguagem de marcação, para documentos contendo informações estruturadas. Praticamente todos os documentos possuem alguma estrutura. Uma informação estruturada apresenta um conteúdo (palavras e figuras principalmente) e alguma indicação do papel que cada um desses itens de conteúdo exerce. A especificação da XML estabelece um padrão para a utilização de marcas identificadoras dos itens de conteúdo do documento.

XML 1.0 define a linguagem XML usando a gramática BNF [XML, 2003]. Então, considerando que XML originou-se do BNF, e que o BNF é a linguagem utilizada pelos compiladores, a transformação dos aspectos sintáticos e léxicos escritos em BNF é praticamente direta para XML. Esse é o motivo pelo qual XML é usada no desenvolvimento da idéia dessa dissertação, como linguagem para expressar os dados de entrada e saída da ferramenta proposta, e para a troca de informações entre ela e as ferramentas de desenho de diagramas em UML e posterior geração de código.

5 Proposta de uma Ferramenta de Captura de Metadados - GeraTED

Imagine-se uma forma de escrever uma descrição de uma aplicação computacional, que seja legível pela máquina, de modo que o computador possa gerar a aplicação automaticamente. Desta forma, durante um teste do sistema, seria possível efetuar, iterativamente, melhorias sucessivas na aplicação. Ao ser identificado um problema, a especificação poderia ser editada e alterada de acordo com a solução dada, e então a aplicação seria re-gerada... e em pouco tempo, ela estaria pronta para um novo teste ! Conforme o Capítulo 2, esta é a proposta da MDA.

Quando desenvolvedores de software se deparam com esta idéia, freqüentemente observam: “Isso nunca vai funcionar. Não se pode gerar um sistema completo a partir de um modelo. Sempre será preciso ajustar o código”. Este pensamento ainda se justifica porque, geralmente, os modelos produzidos não contêm o mínimo de informação necessária para uma geração automática de código. Apesar da resistência que se pode encontrar, a proposta da MDA já apresenta alguns resultados concretos. Mas ainda há espaço para automatizar mais.

Esta proposta é viável, contudo, depende de um comprometimento em fazer não apenas mais um modelo, mas sim “O modelo”, detalhado no nível PIM. É necessário produzir um modelo que, ao ser detalhado, possa ser entendido pela máquina. E para auxiliar nesse trabalho, mais uma vez é preciso “Dividir para conquistar”.

A MDA ainda está em sua infância, mas já mostrou a sua capacidade de mudar radicalmente a forma como os sistemas são desenvolvidos. Está surgindo um novo paradigma, em que o desenvolvimento de sistemas está deslocando seu foco da codificação para a elaboração de modelos “executáveis”. Já podem ser alcançados grandes ganhos, em termos de produtividade, portabilidade, interoperabilidade e esforços menores em manutenção.

A proposta de construção de uma ferramenta gráfica é de grande valia para facilitar o detalhamento do diagrama de Tipos Elementares de Dados (TED) apresentados no Capítulo anterior. O Tipo Elementar será resultante do somatório: ‘tipo de dado + metadados” dentro do contexto do sistema em desenvolvimento. Apesar de ser evidente, convém ressaltar que a idéia de implementar TED está inteiramente alinhada com a abordagem da MDA.

5.1 Objetivo e Visão Geral

A ferramenta aqui proposta, de captura de metadados, visa facilitar a Geração Automática de Código dos Tipos Elementares de Dados. Ela está sendo chamada de GeraTED – Gerador de Tipos Elementares de Dados, e consiste basicamente em apresentar, ao interlocutor envolvido com o desenvolvimento do sistema, conjuntos de perguntas genéricas relevantes para cada atributo das classes do sistema, levando em consideração o tipo de dado primitivo associado àquele atributo. Assim sendo, para cada atributo são feitas perguntas de acordo com o seu tipo primitivo, como, por exemplo:

- Tamanho mínimo e máximo de caracteres esperado durante o preenchimento;
- Domínio de Valores, ou intervalo de valores (mínimo e máximo);
- Exceções de Valores ou Caracteres (valores ou caracteres não permitidos);
- Se for um campo decimal (tipo *float* ou *double*), quantas posições decimais?
- Se for um campo numérico (tipo *integer, float, double*), aceita valor negativo?
- Aceita valor nulo ou é obrigatório o preenchimento?
- Regras de preenchimento em geral.

A partir das respostas a estes questionamentos, a GeraTED irá gerar as regras de preenchimento dos Tipos Elementares. Será utilizada a linguagem do MOR, apresentada no Capítulo 3, item 3.3.2, para a geração dessas regras, pois ela abrange de forma bastante clara e satisfatória a construção das mesmas em BNF equivalente. Vale lembrar que o usuário da

GeraTED não precisará conhecer a notação ou sintaxe da linguagem do MOR (lembrando da MDA, “a automação consiste em mais um nível de compilação”). Ao final do levantamento dessas características, através do preenchimento do questionário, o próximo passo seria gerar o XML correspondente, sobre o qual seria feita a futura geração automática de código.

De acordo com o Capítulo 2, um Modelo PIM desenvolvido através de uma ferramenta poderá servir de entrada para o PSM, sendo necessário escolher a plataforma e também o programa de transformação. Assim, a partir de um modelo de tipos primitivos e elementares pode-se gerar mais código automaticamente. Toda a parte de verificação de domínio de atributos pode ser expressa em um modelo de alto nível, como o PIM.

Partindo do princípio de “Dividir para conquistar”, o foco do trabalho aqui apresentado é dar uma atenção maior aos tipos de dados identificados durante o levantamento do sistema. Afinal, eles farão parte do sistema e sua importância é bastante significativa, conforme ressaltado no Capítulo 4, em termos de garantia de integridade, confiabilidade e outras características fundamentais a considerar em persistência de dados.

De maneira sucinta, o objetivo principal almejado é a definição pormenorizada dos domínios dos atributos a serem manipulados, utilizando o conceito de Tipos Elementares de Dados. Isto requer a captura e a utilização de metadados, através de uma linguagem para descrição dos Tipos Elementares resultantes, que neste trabalho é a linguagem do MOR. Para a captura destas informações foi definida e desenhada a ferramenta GeraTED, visando apoiar essa tarefa.

5.2 Estudo de Caso

Para chegar ao nosso objetivo, percorremos um caminho que contou com o auxílio de todo o embasamento teórico explanado até aqui, principalmente MDA, MOF, UML, Modelo de Negócio (incluindo o MOR), Metadados e Tipos Elementares de Dados. Faremos uso do

Diagrama de Classes da Livraria juntamente com a Biblioteca de Tipos Elementares, apresentados nas Figuras 14 e 15 respectivamente, como um estudo de caso para mostrar a utilização da idéia proposta. Também foi construída uma ferramenta para auxiliar no desenvolvimento da idéia, com exemplos práticos como protótipo de telas e criação de classes de tipos elementares de dados.

Aproveitando o conteúdo aprendido durante o Curso de MNOO [CUNHA,2004] e seguindo os conceitos da MDA, temos os seguintes passos:

- Elaborar um Modelo de Negócio – PIM para Estudo de Caso de uma Livraria virtual, representando-o através de um Diagrama de Caso de Uso, idêntico ao da Figura 11 do Capítulo 2;

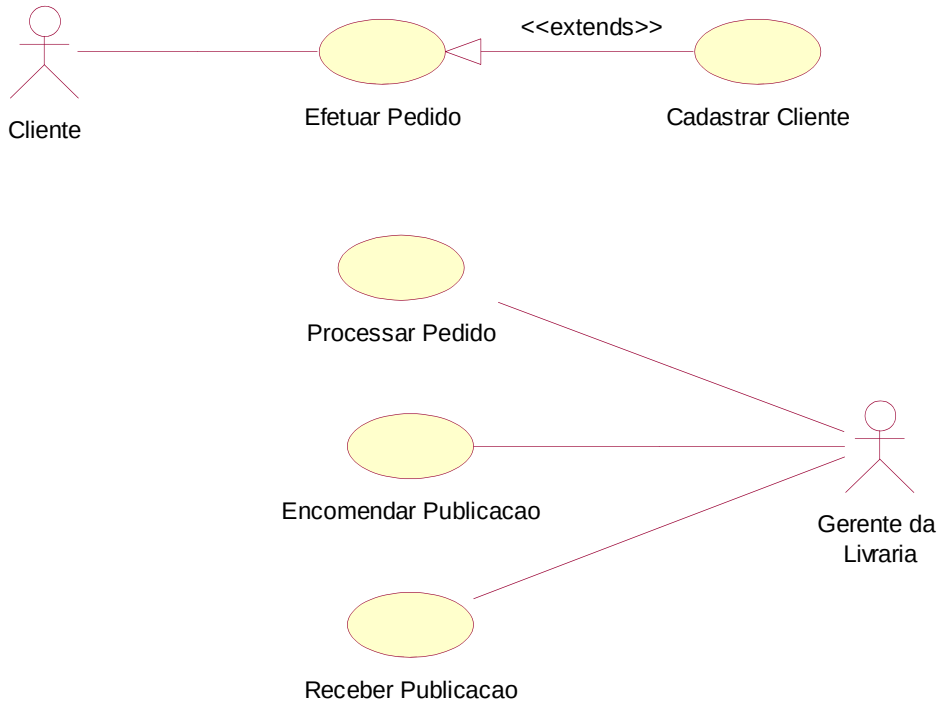


Figura 19 – Diagrama de Caso de Uso de uma Livraria Virtual

- Desenhar seu Diagrama de Classe – vide a Figura 10, sem a definição de tipo de dado para cada atributo, a qual representa uma possível solução para o negócio da livraria virtual;
- Identificar, para cada atributo de cada classe, o tipo de dado primitivo correspondente dentre os 6 apresentados pelo MOF, que são: *Integer, String, Float, Long, Double, Boolean.* Partindo do Diagrama de Classe da Figura 10, foi obtido o da Figura 20;

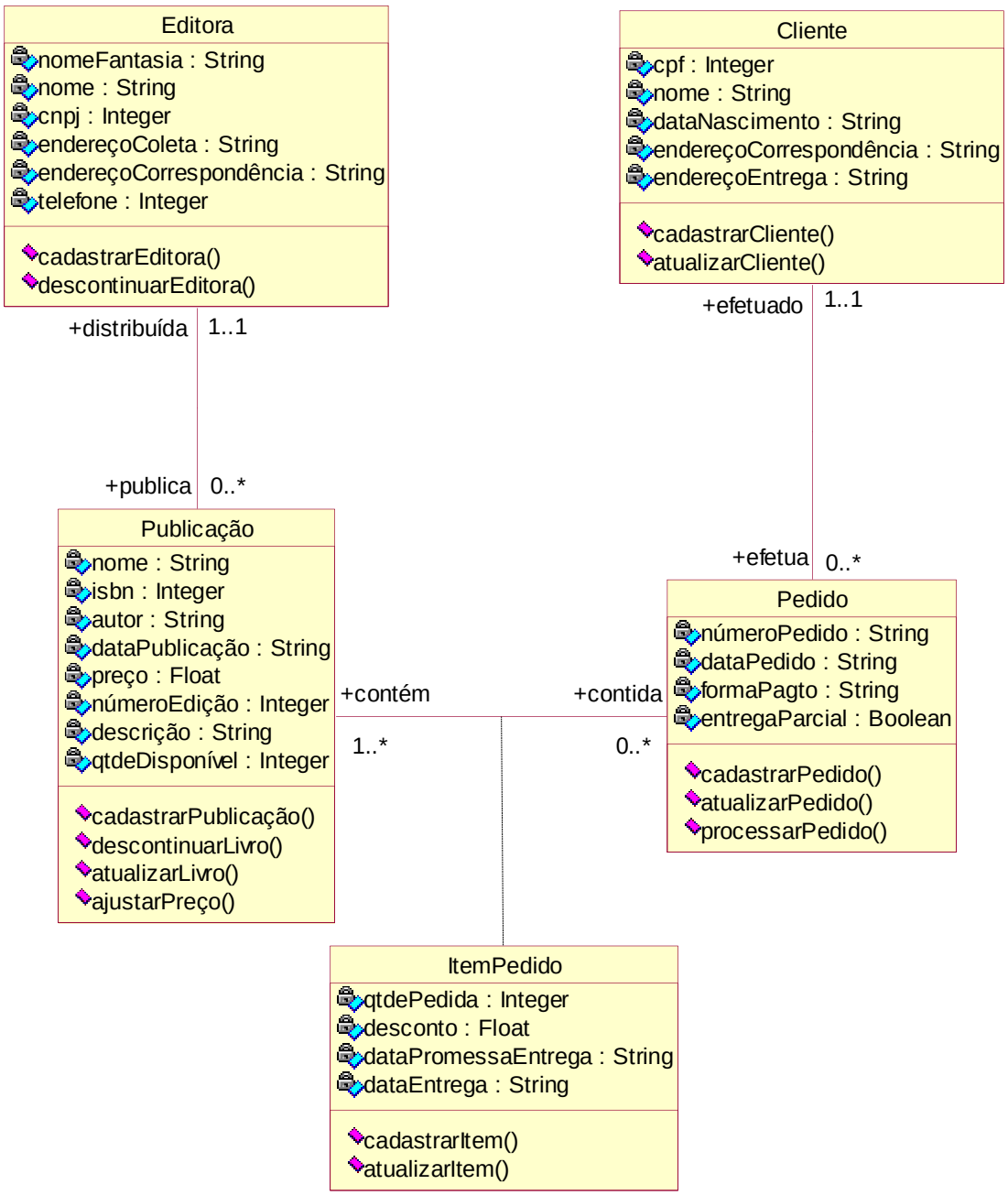


Figura 20 – Diagrama de Classe com tipos de dados

- Utilizar a ferramenta GeraTED (que é detalhada mais adiante no item 5.3) para auxiliar na descrição dos atributos das classes do sistema a partir da identificação e captura de seus Metadados, que serão traduzidos para a linguagem MOR, em BNF equivalente.

Nota: O usuário da GeraTED não precisa conhecer essa linguagem, pois são apresentados questionários para a captura dos metadados de acordo com o tipo de dado em questão. Estes metadados podem ser, por exemplo: domínio de valores, tamanho máximo e mínimo do tipo de dado, regras e restrições de preenchimento, entre outras informações;

- Alimentar a GeraTED com um arquivo XML, que contém as classes do sistema em desenvolvimento;
- Responder o questionário apresentado para cada tipo primitivo associado a um atributo. Assim, cada atributo terá um Tipo Elementar de Dado associado a ele, armazenando suas características relevantes, ao invés de puramente um tipo primitivo. Os Tipos Elementares de Dados encontrados farão parte de uma hierarquia de classes de Tipos Elementares de Dados, isto é, uma Biblioteca de Tipos Elementares. Esta Biblioteca poderá ser reutilizada por outros sistemas de informação;
- Gerar, através da GeraTED, o código XML com a definição dos Tipos Elementares criados e das classes com os seus respectivos atributos associados àqueles tipos;

Ao final desse procedimento, o código XML resultante serve de entrada para a obtenção do(s) PSM(s), através da transformação PIM → PSMs. Esta etapa está fora do escopo deste trabalho, ficando como sugestão para trabalhos futuros.

5.3 Especificação da GeraTED

Aqui é apresentada a proposta de especificação da ferramenta GeraTED, para auxiliar na captura de metadados para os Tipos Elementares. Ela deve receber como dados de entrada:

- O Diagrama de Classes do sistema em desenvolvimento, em formato da linguagem XML, isto é, suas classes + atributos com tipo primitivo associado;

E deverá fornecer como resultado:

- XML das classes + atributos com Tipo Elementar associado + classes de Tipos Elementares gerados para os atributos com seus metadados e regras de preenchimento, que foram originalmente escritos pela GeraTED utilizando a linguagem do MOR, em BNF equivalente.

O esquema geral do funcionamento da GeraTED, em uma visão externa, está ilustrado na Figura 21.

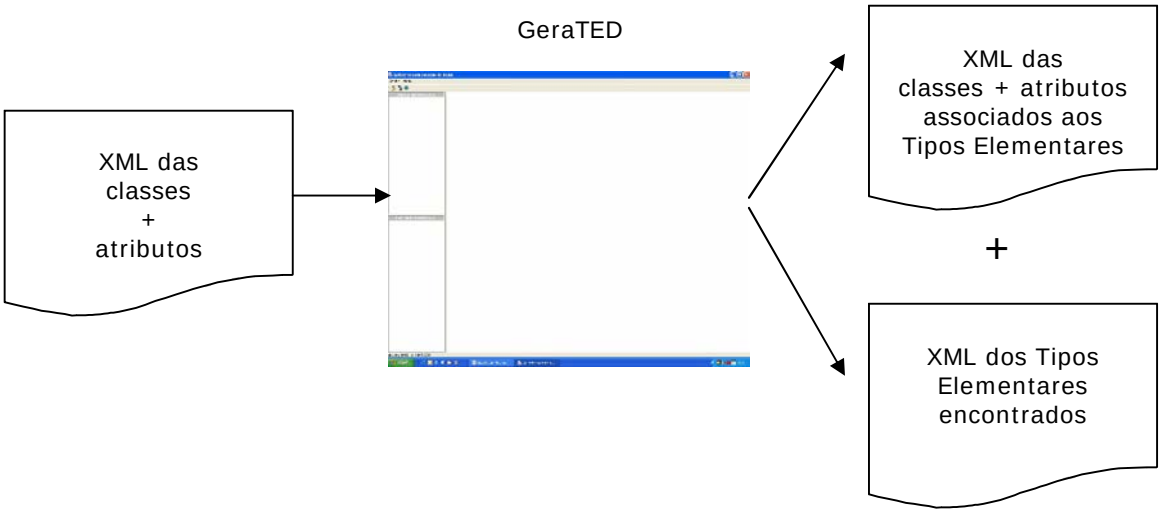


Figura 21 – Esquema de funcionamento geral da ferramenta GeraTED

Complementando a Figura 21, a Figura 22 mostra o fluxo completo normal do funcionamento da GeraTED:

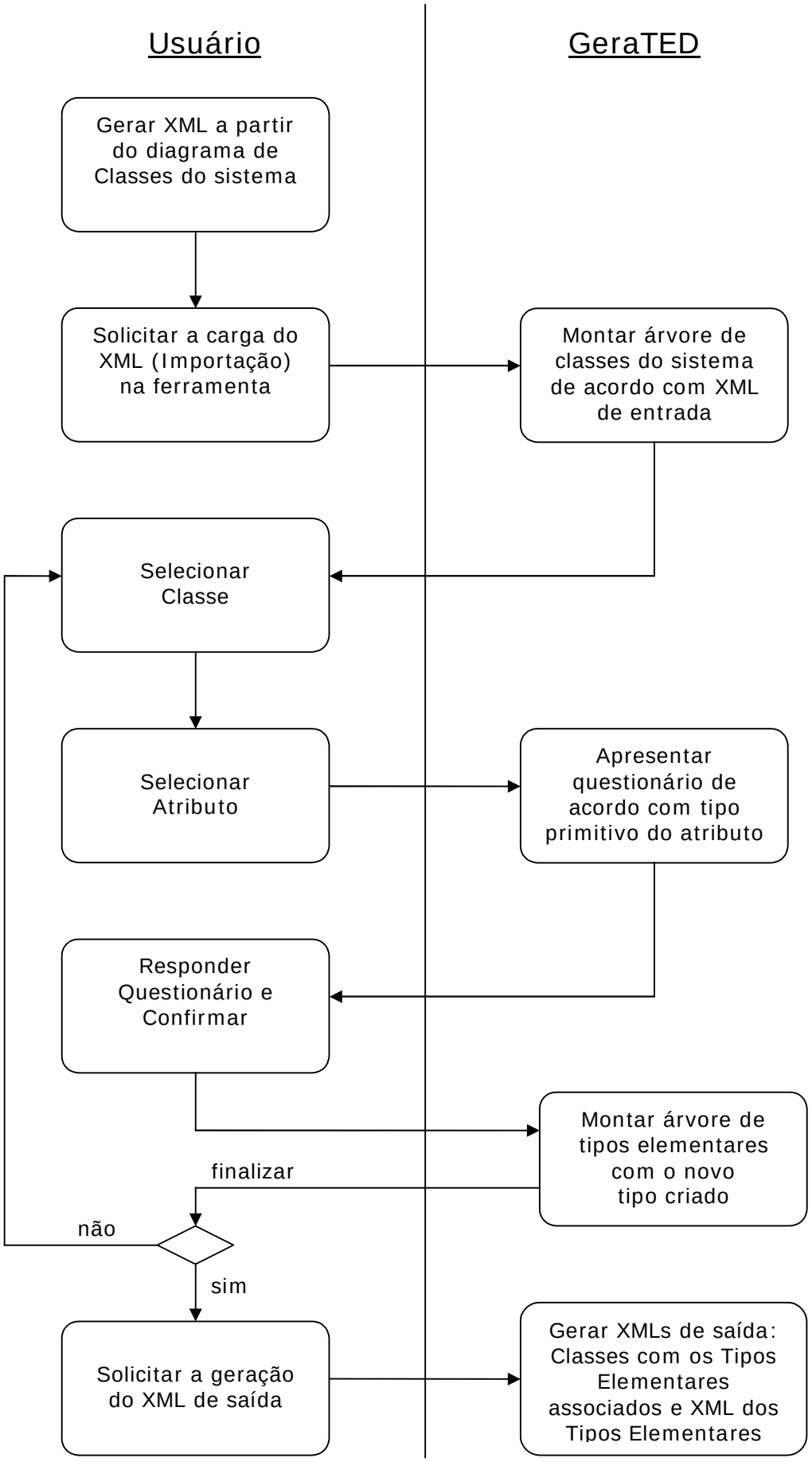


Figura 22 – Fluxo de funcionamento da GeraTED

Assim, o usuário da GeraTED poderá criar quaisquer Tipos Elementares identificados no levantamento do sistema, percorrendo os passos apresentados no item 5.2. A GeraTED ajuda na captura dos metadados dos atributos das classes do sistema, e, como resultado, gera novas classes, uma para cada Tipo Elementar encontrado. A Figura 15, apresentada na seção 4.1, mostra o Diagrama de Hierarquia de Tipos Elementares encontrados para o exemplo do sistema da livraria da Figura 14. Tomemos como exemplos esses diagramas para ilustrar as entradas e as saídas da GeraTED.

A Figura 23 ilustra um protótipo de tela principal onde o usuário pode escolher as opções de importar e exportar arquivos XML.

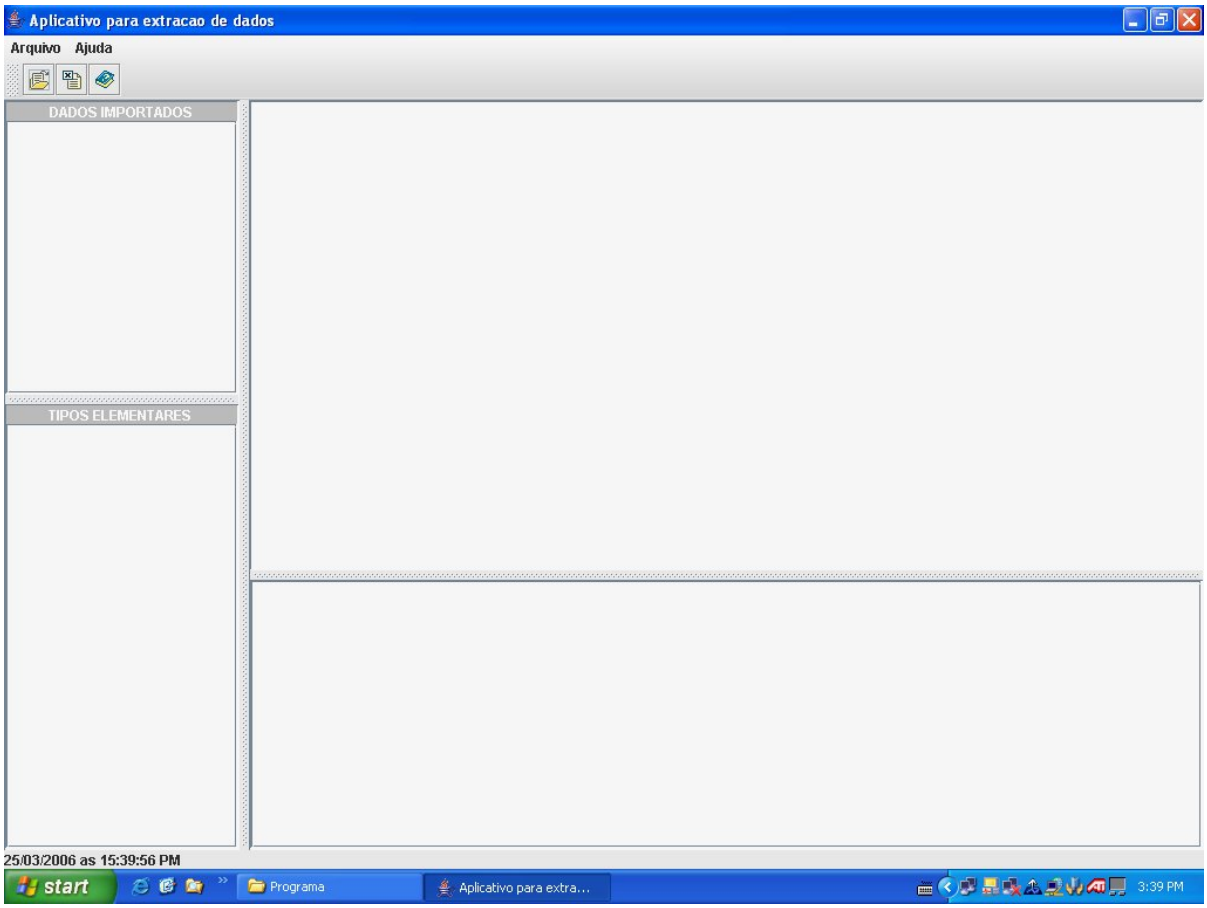


Figura 23 – Protótipo de Tela inicial da GeraTED

Uma explicação da Figura 23:

- Na parte superior esquerda aparecem as funcionalidades básicas de importar e exportar XML, pelo menu Arquivo ou através dos botões.

- Na metade superior esquerda da tela estão as classes importadas do Diagrama de Classes do sistema em desenvolvimento, na grade “Dados Importados”.
- Na parte inferior esquerda situam-se os Tipos Elementares que estão sendo gerados, na grade “Tipos Elementares”.
- O lado superior direito da tela é reservado para a apresentação do questionário correspondente ao atributo selecionado de uma determinada classe.
- O lado inferior direito da tela mostra a descrição em XML do Tipo Elementar, obtida após a sua geração.

Ao clicar no botão superior esquerdo – Importar Arquivo XML, a janela abaixo é apresentada (Figura 24) para a seleção do arquivo contendo a estrutura das classes em XML. As Figuras 24a e 24b mostram exemplos de arquivos XML de entrada.

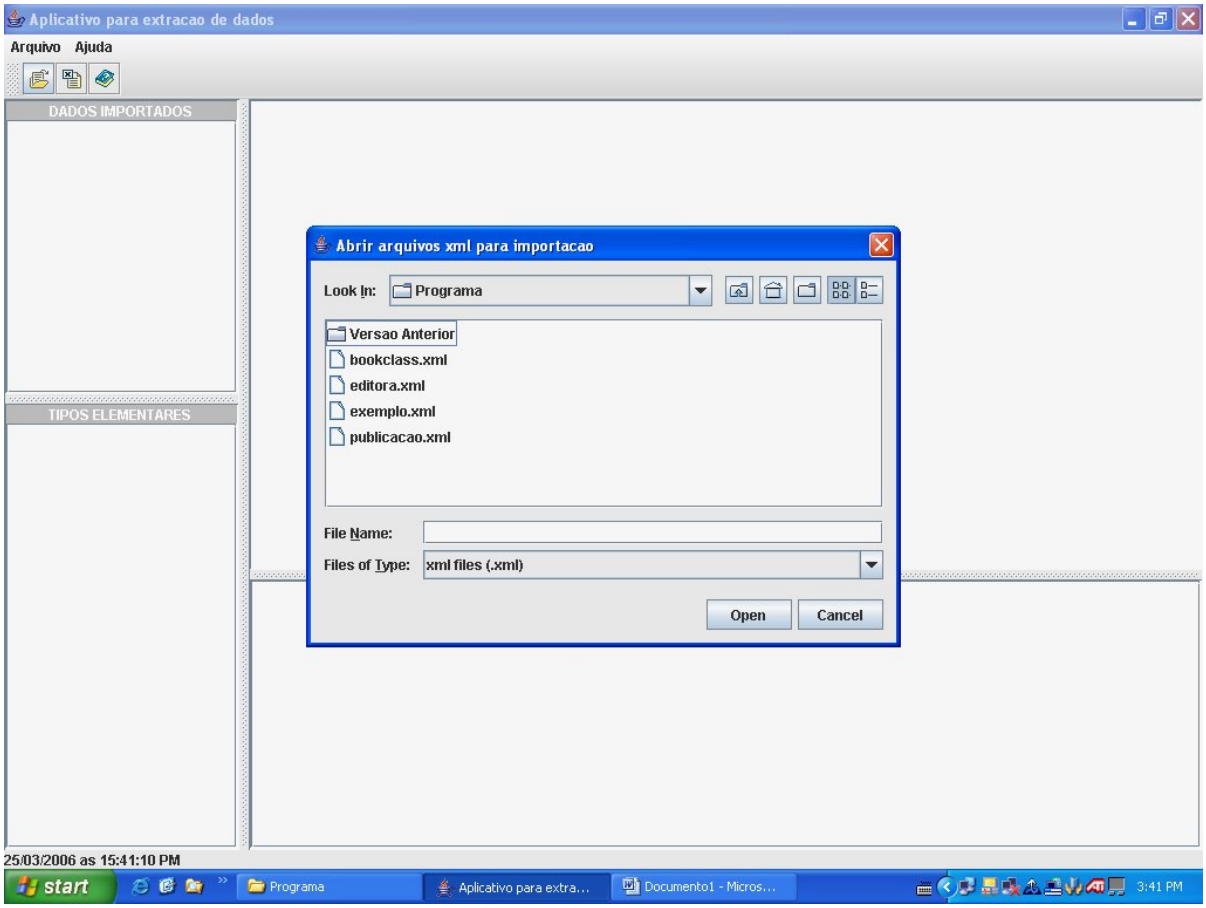


Figura 24 – Tela de importação de estrutura de classes XML

```
<?xml version="1.0" ?>
- <Classe nome="Publicacao">
  - <Atributo>
      <Nome>nome</Nome>
      <Tipo>String</Tipo>
    </Atributo>
  - <Atributo>
      <Nome>isbn</Nome>
      <Tipo>String</Tipo>
    </Atributo>
  - <Atributo>
      <Nome>autor</Nome>
      <Tipo>String</Tipo>
    </Atributo>
  - <Atributo>
      <Nome>dataPublicacao</Nome>
      <Tipo>String</Tipo>
    </Atributo>
  - <Atributo>
      <Nome>preco</Nome>
      <Tipo>Float</Tipo>
    </Atributo>
  - <Atributo>
      <Nome>numeroEdicao</Nome>
      <Tipo>Integer</Tipo>
    </Atributo>
  - <Atributo>
      <Nome>descricao</Nome>
      <Tipo>Double</Tipo>
    </Atributo>
  - <Atributo>
      <Nome>qtdeDisponivel</Nome>
      <Tipo>String</Tipo>
    </Atributo>
</Classe>
```

Figura 24a - Arquivo XML de entrada – Classe Publicação

```
<?xml version="1.0" ?>
- <Classe nome="Editora">
  - <Atributo>
      <Nome>nomeFantasia</Nome>
      <Tipo>String</Tipo>
    </Atributo>
  - <Atributo>
      <Nome>nome</Nome>
      <Tipo>String</Tipo>
    </Atributo>
  - <Atributo>
      <Nome>cnpj</Nome>
      <Tipo>String</Tipo>
    </Atributo>
  - <Atributo>
      <Nome>enderecoColeta</Nome>
      <Tipo>String</Tipo>
    </Atributo>
  - <Atributo>
      <Nome>enderecoCorrespondencia</Nome>
      <Tipo>Integer</Tipo>
    </Atributo>
  - <Atributo>
      <Nome>telefone</Nome>
      <Tipo>Integer</Tipo>
    </Atributo>
</Classe>
```

Figura 24b - Arquivo XML de entrada - Classe Editora

Após a importação do arquivo XML, a árvore de classes do sistema é carregada na GeraTED – vide Figura 25. A partir deste ponto, o usuário pode selecionar cada atributo de cada classe para detalhamento.

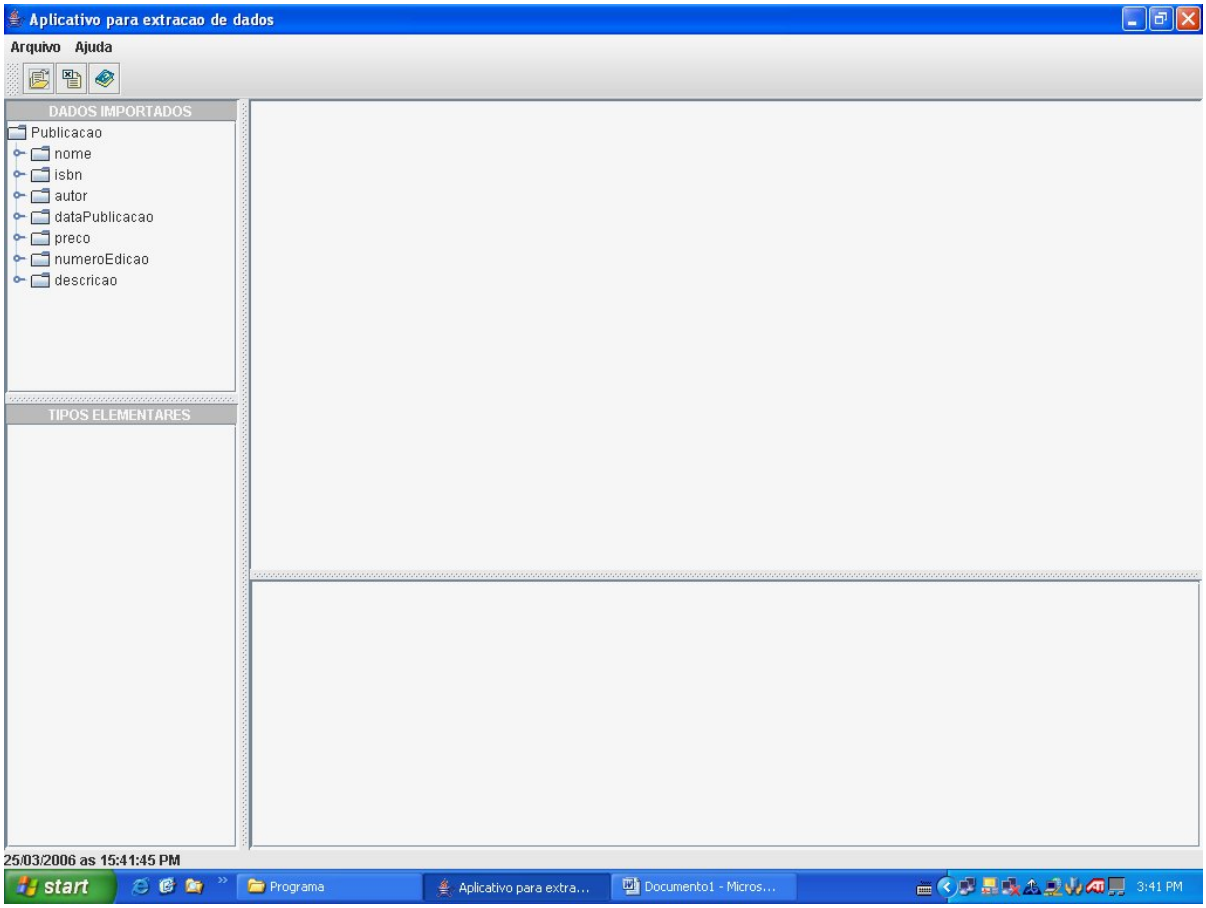


Figura 25 – Protótipo de Tela contendo árvore de dados importados pela GeraTED

De acordo com o tipo do atributo selecionado, um questionário correspondente é apresentado para preenchimento. Mais adiante no item 5.3.1 o questionário é mostrado por inteiro. A Figura 26 ilustra a apresentação do questionário para um atributo do tipo *Integer*.

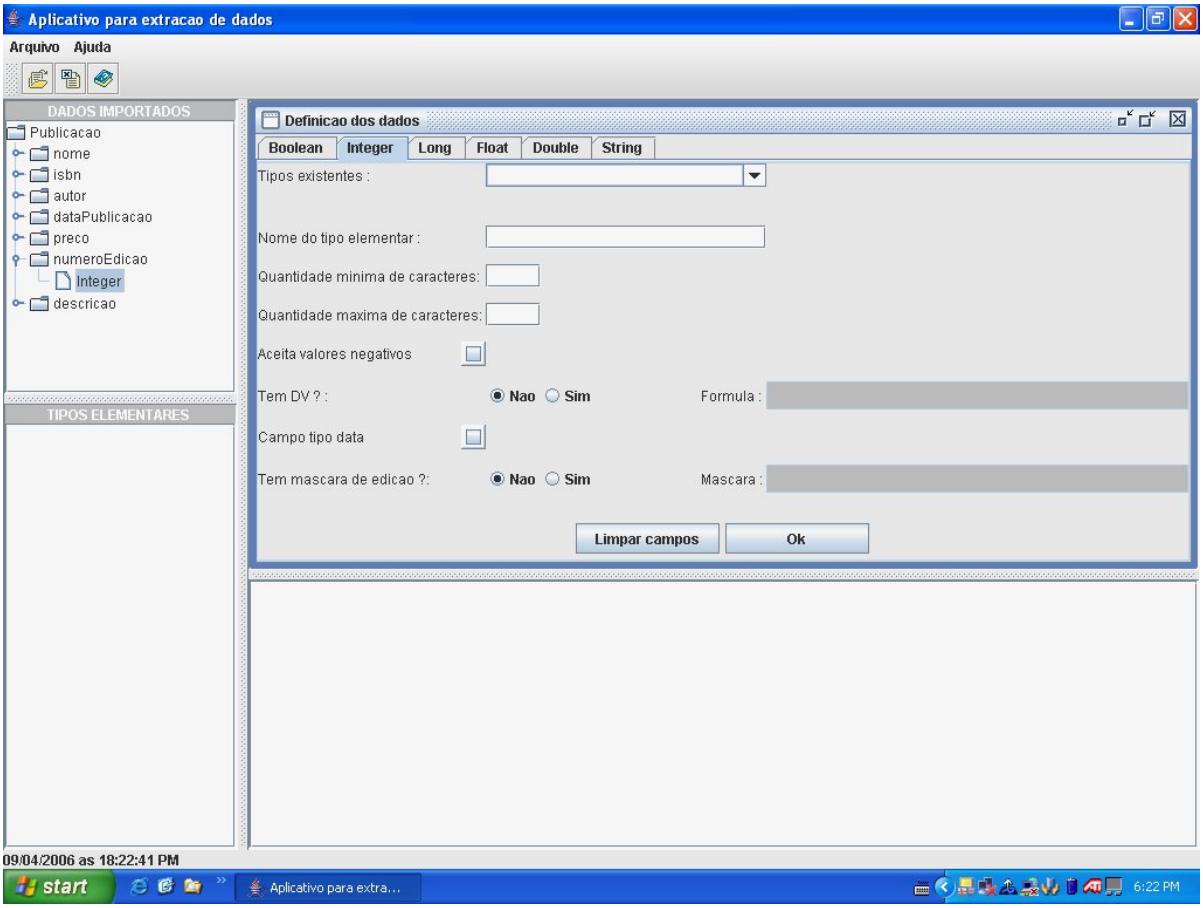


Figura 26 – Tela de Captura de Metadados para o tipo de dado *Integer*

Neste momento, o usuário da GeraTED deve responder às perguntas apresentadas de acordo com a necessidade do negócio em estudo. O objetivo aqui é capturar tantas informações quantas o usuário puder fornecer para o enriquecimento do Tipo Elementar a ser gerado. As perguntas não respondidas não farão parte das características do Tipo Elementar referente ao atributo.

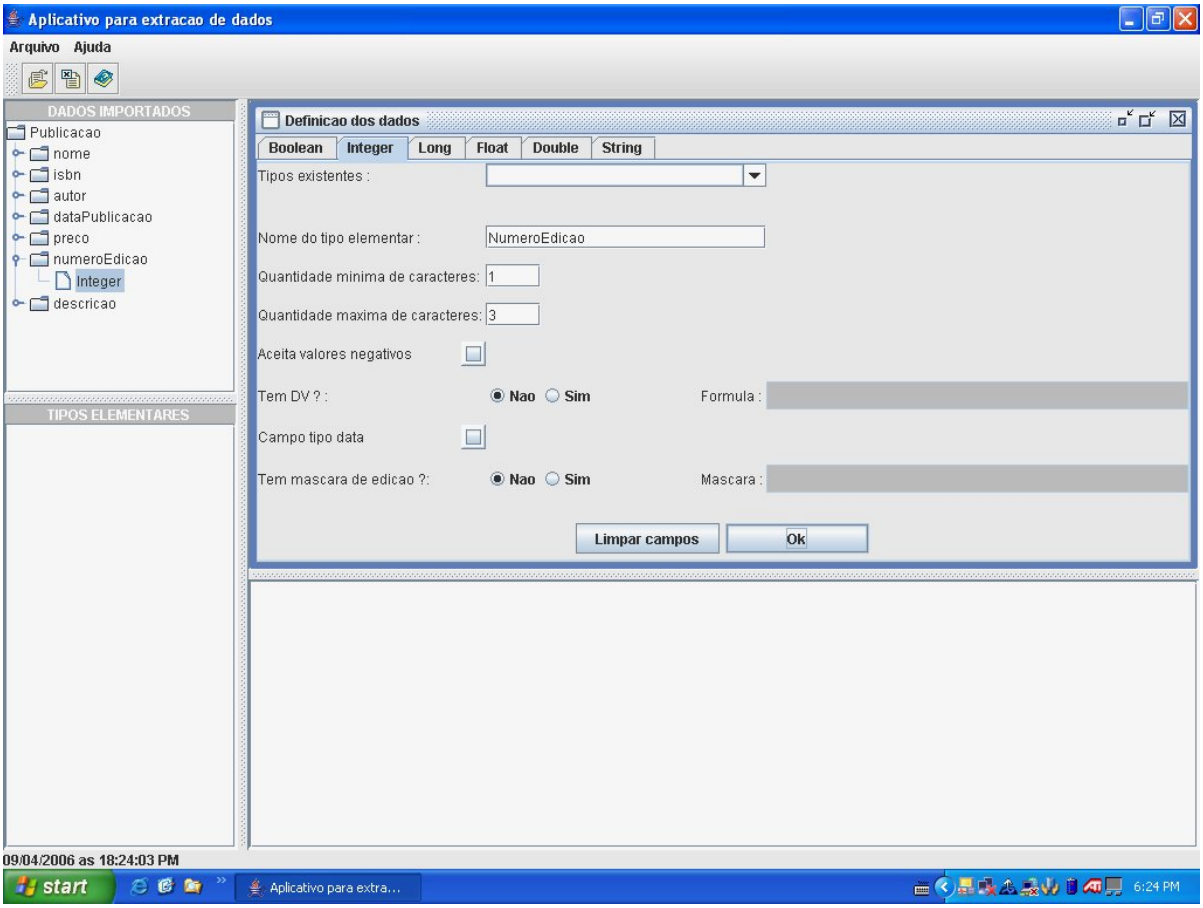


Figura 26a – Tela de Captura de Metadados para o tipo de dado *Integer*, atributo numeroEdicao

Após responder às perguntas apresentadas, vide exemplo na Figura 26a, o usuário da GeraTED confirma e um novo Tipo Elementar é criado, conforme mostra a Figura 27. É necessário repetir este procedimento para todos os atributos das classes importadas. Assim, cada atributo será analisado e seus metadados capturados para a criação dos Tipos Elementares correspondentes. Vale lembrar da possibilidade de reutilização: todos os Tipos Elementares encontrados podem ser reaproveitados por outros sistemas ou até mesmo por outro(s) atributo(s) do mesmo sistema. Isso faz com que o mesmo Tipo Elementar possa ser criado uma única vez para todo(s) o(s) sistema(s).

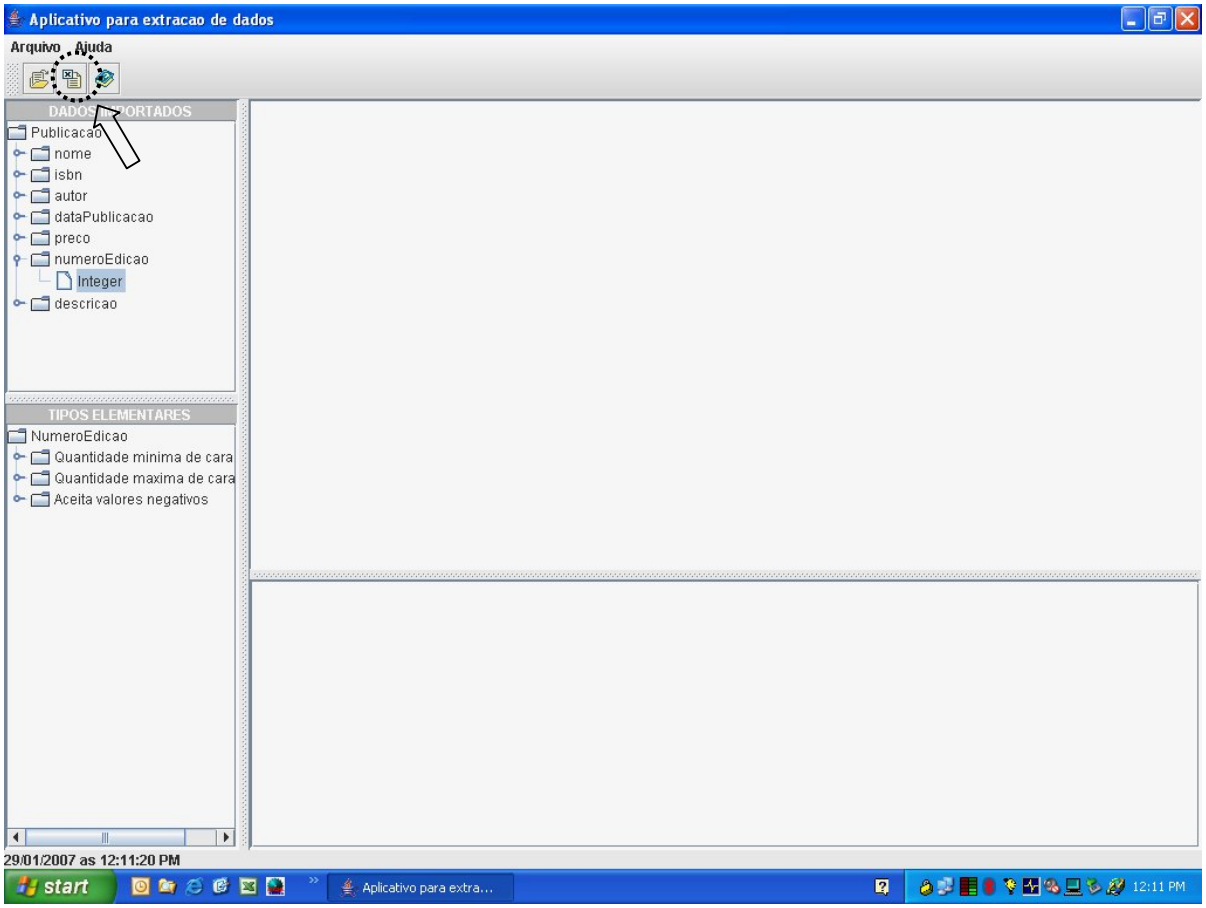


Figura 27 – Após a Captura de Metadados para o atributo numeroEdicao, a criação do Tipo Elementar

Após a captura dos metadados para todos os atributos das classes importadas, é hora de gerar o arquivo de saída. Para isso, é necessário clicar no segundo botão que está no canto superior esquerdo da tela, assinalado na Figura 27 – Exportar XML, ou acionar a opção adequada através do menu Arquivo.

A Figura 28 mostra a confirmação da geração do arquivo XML. Para visualizá-lo, basta assinalar “Yes” na caixa de mensagem.

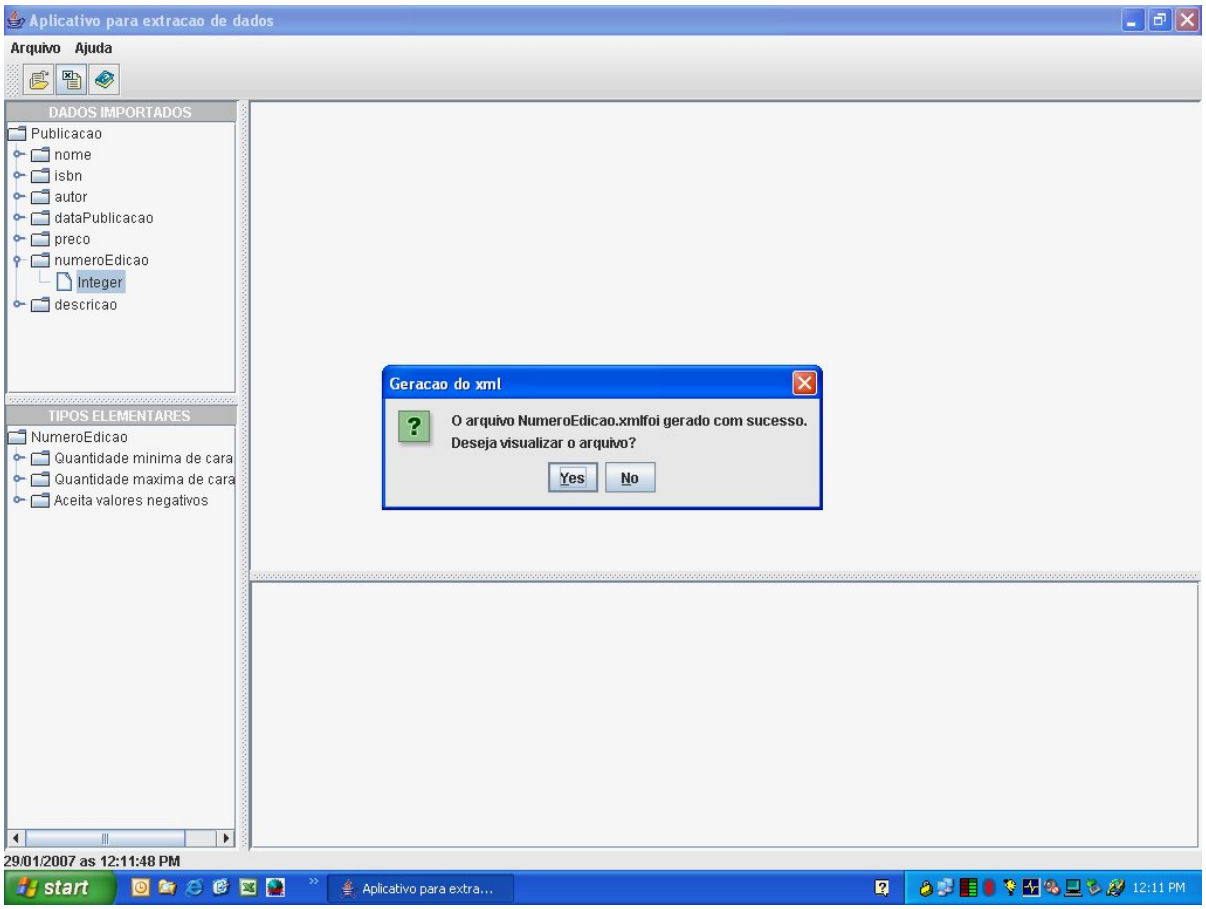
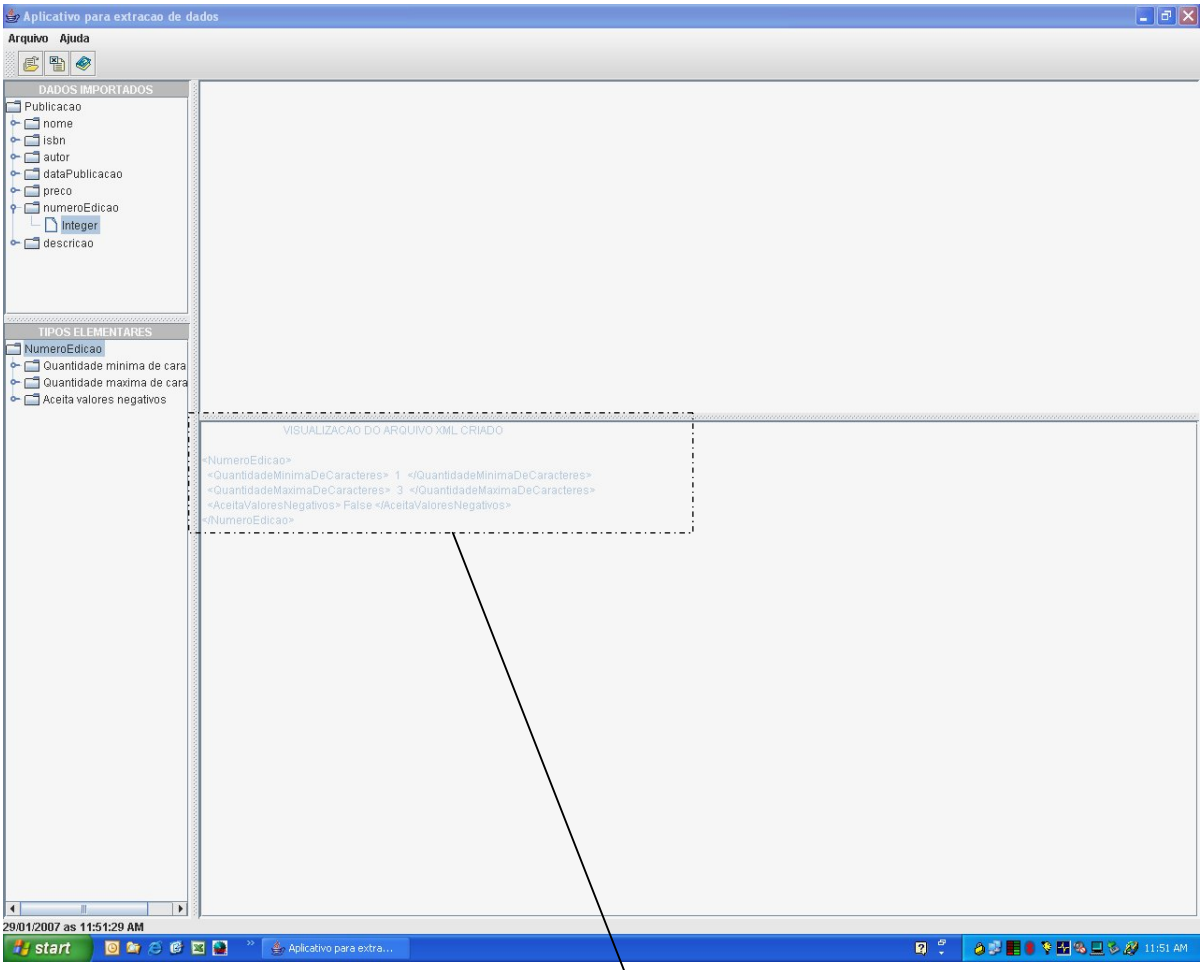


Figura 28 – Geração do arquivo de saída para o Tipo Elementar NumeroEdicao

Assim, a Figura 28a ilustra a geração do arquivo de saída para o Tipo Elementar NumeroEdicao. Após a criação de todos os Tipos Elementares, pode-se obter o arquivo XML da Figura 28b, que contém a classe Publicação com os seus atributos já associados aos Tipos Elementares criados. Na Figura 28b aparece claramente o relacionamento existente entre os atributos e os Tipos Elementares, mostrando uma representação do diagrama apresentado na Figura 13, da seção 4.1 do Capítulo 4.




```
<NumeroEdicao>
<QuantidadeMinimaDeCaracteres>1</QuantidadeMinimaDeCaracteres>
<QuantidadeMaximaDeCaracteres>3</QuantidadeMaximaDeCaracteres>
<AceitaValorNegativo> False </AceitaValorNegativo>
</NumeroEdicao>
```

Figura 28a – Visualização do arquivo XML gerado para o Tipo Elementar NumeroEdicao

```
<?xml version="1.0" ?>
- <Classe nome="Publicacao">
  - <Atributo>
      <Nome>nome</Nome>
      <Tipo>NomeComposto</Tipo>
    </Atributo>
  - <Atributo>
      <Nome>isbn</Nome>
      <Tipo>ISBN</Tipo>
    </Atributo>
  - <Atributo>
      <Nome>autor</Nome>
      <Tipo>NomeComposto</Tipo>
    </Atributo>
  - <Atributo>
      <Nome>dataPublicacao</Nome>
      <Tipo>Data</Tipo>
    </Atributo>
  - <Atributo>
      <Nome>preco</Nome>
      <Tipo>Preco</Tipo>
    </Atributo>
  - <Atributo>
      <Nome>numeroEdicao</Nome>
      <Tipo>NumeroEdicao</Tipo>
    </Atributo>
  - <Atributo>
      <Nome>descricao</Nome>
      <Tipo>Descricao</Tipo>
    </Atributo>
  - <Atributo>
      <Nome>qtdeDisponivel</Nome>
      <Tipo>Qtde</Tipo>
    </Atributo>
</Classe>
```

Figura 28b – XML de saída para a classe Publicação com seus Tipos Elementares associados

A Figura 28c mostra alguns dos Tipos Elementares criados que estão relacionados com as classes do sistema. Neste exemplo, constam os Tipos Elementares que foram criados após a captura de metadados dos atributos da classe Publicação. Cada Tipo Elementar é

representado por uma classe em XML. Estas classes, por sua vez, possuem informações importantes para auxiliar numa futura geração automática de código. Pode-se imaginar que a classe do Tipo Elementar NomeComposto, por exemplo, possuiria um método VerificarPreenchimento para o caso de preenchimento obrigatório, contendo funções tais como VerificarQuantidadeMínimaDeCaracteres e VerificarQuantidadeMáximaDeCaracteres. Os metadados mostrados no exemplo da Figura 28c são meramente ilustrativos.

```
<NomeComposto>
  <PreenchimentoObrigatorio>S</PreenchimentoObrigatorio>
  <QuantidadeMinimaDeCaracteres>2</QuantidadeMinimaDeCaracteres>
  <QuantidadeMaximaDeCaracteres>60</QuantidadeMaximaDeCaracteres>
</NomeComposto>

<ISBN>
  <PreenchimentoObrigatorio>S</PreenchimentoObrigatorio>
  <QuantidadeMinimaDeCaracteres>1</QuantidadeMinimaDeCaracteres>
  <QuantidadeMaximaDeCaracteres>10</QuantidadeMaximaDeCaracteres>
</ISBN>

<Data>
  <TipoData>S</TipoData>
</Data>

<Qtde>
  <QuantidadeMinimaDeCaracteres>1</QuantidadeMinimaDeCaracteres>
  <QuantidadeMaximaDeCaracteres>5</QuantidadeMaximaDeCaracteres>
</Qtde>

<Preco>
  <QuantidadeMinimaDeCaracteres>1</QuantidadeMinimaDeCaracteres>
  <QuantidadeMaximaDeCaracteres>12</QuantidadeMaximaDeCaracteres>
  <NumeroCasasDecimais>2</NumeroCasasDecimais>
</Preco>

<NumeroEdicao>
  <ValorDefault>1</ValorDefault>
  <QuantidadeMinimaDeCaracteres>1</QuantidadeMinimaDeCaracteres>
  <QuantidadeMaximaDeCaracteres>3</QuantidadeMaximaDeCaracteres>
  <AceitaValorNegativo> False </AceitaValorNegativo >
```

Figura 28c – XML de saída dos Tipos Elementares gerados

Com este exemplo pode-se concluir que, durante o levantamento de informações para a construção de sistemas em uma organização, é construída uma biblioteca de Tipos Elementares de Dados. Esta biblioteca será bastante rica e útil, pois servirá de base para a reutilização dos Tipos Elementares e, adicionalmente, poderá ser utilizada para estabelecer uma padronização destes Tipos Elementares de Dados nos diversos sistemas da organização.

O Diagrama de Tipos Elementares de Dados da Figura 15 pode ser considerado como um metamodelo de classes de Tipos Elementares. Ele ilustra graficamente a biblioteca resultante do trabalho de captura de metadados. Após a definição dos Tipos Elementares de Dados através da GeraTED, passam a existir insumos suficientes para desenhar um Diagrama de Hierarquia de Tipos Elementares de Dados semelhante ao apresentado na Figura 15 do Capítulo 4.

5.3.1 Questionário

O questionário aqui apresentado foi formulado no intuito de auxiliar na captura de metadados. Através de perguntas simples e genéricas, podem ser obtidas informações relevantes que serão de grande valia para a geração automática de código. Como se pode observar no exemplo da Figura 26a, o questionário é apresentado de acordo com o tipo de dado do atributo, considerando os 6 tipos primitivos apresentados pelo MOF.

As perguntas estão divididas em 2 grupos:

I. <u>Perguntas comuns a todos os tipos de dados:</u>

a. Preenchimento Obrigatório – Sim/Não
b. Valor Default – disponibilizar campo com o tamanho máximo do atributo
c. Tipos Enumerados – captar os valores possíveis (domínio de valores). Nota: neste caso, as demais perguntas se tornam desnecessárias e não serão respondidas.

II. <u>Perguntas especificas para cada tipo de dado:</u>

<u>Tipo Primitivo:</u> *Integer*

1. Quantidade de caracteres mínima para preenchimento;
2. Quantidade de caracteres máxima para preenchimento;
3. Aceita valores negativos?
4. Tem Dígito Verificador? Em caso afirmativo, qual é a fórmula?
5. Usar separadores para mostrar o campo? Em caso afirmativo, qual é a máscara de edição?
6. Este campo é uma data? Caso afirmativo, fazer restrições de acordo... dia 1 a 31, mês 1 a 12 e ano 0 a 3000, por exemplo. Criticas de dias / mês. Qual é a máscara de edição? Esta pergunta é mutuamente exclusiva com a nº. 5.
7. Este campo representa hora ? Caso afirmativo, fazer restrições de acordo... hora: 00 a 23, minutos: 00 a 59 e segundos: 00 a 59. Qual separador deve ser utilizado ?
   Ex: ":" (dois pontos).
8. Este campo representa data e hora? Caso afirmativo, efetuar críticas apresentadas nos itens 6 e 7.

<u>Tipo Primitivo:</u> *String*

1. Quantidade de caracteres mínima para preenchimento
2. Quantidade de caracteres máxima para preenchimento
3. Caracteres não permitidos
4. Caracteres que deverão aparecer no preenchimento (obrigatórios)
5. Inicia com maiúsculas?

6. Quantidade mínima de conjuntos de caracteres separados por um caracter “espaço em branco” (ex: nome e 1 sobrenome)
7. Quantidade máxima de conjuntos de caracteres

Tipo Primitivo: *Long*

As mesmas perguntas apresentadas para o tipo *Integer* deverão ser apresentadas para o tipo *Long.*

Tipo Primitivo: *Float*

1. Quantidade de caracteres mínima para preenchimento
2. Quantidade de caracteres máxima para preenchimento
3. Número de casas decimais
4. Sinal (‘+’ ou ‘ ‘ para positivo, ‘-‘ para negativo)
5. Separador (ponto ou vírgula)
6. Representa valor em moeda? Em caso afirmativo, qual é o símbolo (ex: R$, U$, £, €)

Tipo Primitivo: *Double*

As mesmas perguntas apresentadas para o tipo *Float* deverão ser apresentadas para o tipo *Double.*

Tipo Primitivo: *Boolean*

- (nenhuma pergunta específica foi identificada até o momento, além das perguntas comuns aos tipos de dados).

- **Convertendo para a linguagem do MOR em BNF equivalente**

O MOR está sendo usado como linguagem formal, de base, para dar uma interpretação rigorosa ao XML resultante. A conversão das perguntas do questionário para a linguagem do MOR em BNF equivalente foi feita de acordo com a seguinte tabela De → Para:

Questionário	MOR (em BNF equivalente)
Preenchimento obrigatório?	PreenchimentoObrigatório = Boolean
Tipos Enumerados?	TipoEnumerado = Symbols / null
Quantidade mínima de caracteres?	QuantidadeMinimaDeCaracteres = Natural / “0” / null
Quantidade máxima de caracteres?	QuantidadeMaximaDeCaracteres = Natural / null
Aceita valores negativos?	AceitaNegativo = Boolean
Tem dígito verificador? Fórmula?	TemDV = Boolean Fórmula = Symbols
Máscara de Edição?	Máscara = Symbols
Atributo somente data?	TipoData = Boolean
Atributo somente hora?	TipoHora = Boolean
Atributo data e hora?	TipoDataHora = Boolean
Lista de caracteres não permitidos?	CaracterProibido = Symbols
Caracteres obrigatórios no preenchimento?	CaracterObrigatório = Symbols
Inicia com maiúsculas?	InicialMaiúscula = Boolean
Quantidade mínima de conjuntos de caracteres?	QuantidadeMinimaConjCaracter = Natural / “0” / null
Quantidade máxima de conjuntos de caracteres?	QuantidadeMaximaConjCaracter = Natural / null
Número de casas decimais?	NumeroCasasDecimais = Natural / “0”
Sinal?	Sinal = null / “+” / “-”
Caracter separador?	Separador = “.” / “,”
Valor moeda? Qual o símbolo?	ValorMoeda = Boolean Símbolo = “R$” / “US$” / “£” / “€”

Tabela 1 – DE→PARA : Perguntas do questionário na linguagem do MOR

Vale ressaltar que, de acordo com as respostas às perguntas da tabela 1, mais informação poderá ser deduzida no futuro, através de referências cruzadas, como por exemplo a quantidade mínima de caracteres não poderá ser nula (ausência de valor) ou “0” (zero), se o campo for de preenchimento obrigatório.

A tabela 2, a seguir, mostra a biblioteca de Tipos Elementares criados com seus respectivos valores (*value*) na linguagem do MOR, já considerando suas características, tais como tamanho máximo e mínimo, preenchimento obrigatório, tipos enumerados (domínio).

Nome do Tipo Elementar	Descrição na linguagem do MOR em BNF equivalente (value)
Ano	Ano = Natural | 1900 <= Natural <= 2100
Bairro	Bairro = NomeComposto
CEP	CEP= zero / Natural | 1<= Natural <=99999999
Cidade	Cidade = NomeComposto
CNPJ	CNPJ = Natural | 1<= Natural <=99999999999999
Complemento	Complemento = “ ” / LetraNúmero
CPF	CPF = Natural | 1<= Natural <=99999999999
Data	Data = DiaDoMes simbol(“/”) NúmeroDoMes simbol(“/”) Ano
DDD	DDD = Natural | 10<= Natural <=99)
Desconto	Desconto = zero / zero PontoDecimal digito*(1-N)
DiaDoMes	DiaDoMes = Natural | 1<= Natural <= 31
Preço	Preço = zero PontoDecimal(“,”) dígito*(2) / dígitoNatural [dígito*(1-N)] PontoDecimal(“,”) dígito*(2)
Endereço	Endereço = TipoLogradouro NomeLogradouro (“,”) NúmeroLogradouro [“/” Complemento](“- ”) Bairro (“-”) Cidade (“-”) Estado (“-”) CEP
Estado	Estado = “AC” / “AL” / “AM” / “AP” / “BA” / “CE” / “DF” / “ES” / “GO” / “MA” / “MG” / “MS” / “MT” / “PA” / “PB” / “PE” / “PI” / “PR” / “RJ” / “RN” / “RO” / “RR” / “RS” / “SC” / “SE” / “SP” / “TO”
FormaPagto	FormaPagto = 1 / 2 / 3 / 4
ISBN	ISBN= Natural | 1<= Natural <=9999999999
LetraNúmero	LetraNúmero = Número UNION Letra
Natural	Natural = digitoNatural [digito*(1-N)]
NomeComposto	NomeComposto = Maiúscula Minúscula*(1-40) [[Symbols(“ “) [Maiúscula] Minúscula*(1-40)] *(1-10)]
NomeLogradouro	NomeLogradouro = NomeComposto
NúmeroDoMes	NúmeroDoMes = Natural | 1<= Natural <= 12
NúmeroLogradouro	NúmeroLogradouro = LetraNúmero
NúmeroPedido	NúmeroPedido = Natural | 1<= Natural <=999999999999
PontoDecimal	PontoDecimal = Symbols (“,”) / Symbols (“.”)
Telefone	Telefone = DDD (Natural | 20000000<= Natural <=99999999)
TipoLogradouro	TipoLogradouro = “R” / “Av” / “Pç” / “Lgo” / “Ld” / “Tv”

Tabela 2 – “Value” do Tipo elementar escrito na linguagem do MOR

Observações sobre a tabela 2:

- Os tipos enumerados (com valores pré-definidos) são meramente ilustrativos, podendo variar de acordo com o sistema;
- Os Tipos Elementares que se encontram sublinhados representam os Tipos Elementares encontrados na modelagem do sistema. Os demais tipos (não sublinhados) foram criados no sentido de facilitar a descrição dos anteriores e para aumentar a reutilização.

Podemos concluir que, a partir das respostas ao questionário de acordo com a tabela 1 De-Para, poderemos obter regras para o preenchimento dos Tipos Elementares como mostra a tabela 2, na linguagem do MOR. Vale reforçar que essas regras, uma vez capturadas, serão escritas apenas uma vez e utilizadas por todo o sistema na validação dos dados garantindo principalmente sua integridade e facilitando sua reutilização.

6 Conclusões

Apesar de todos os avanços tecnológicos ocorridos nos últimos tempos, a geração automática de código a partir dos tipos de dados ainda é limitada. Se for possível agregar a esses tipos de dados a semântica adequada, com seus respectivos metadados, poderá ser gerado mais código automaticamente. Por isso, é importante a solução proposta nesta monografia, que é o levantamento de informações a respeito dos atributos das classes do sistema para a geração de um diagrama de Tipos Elementares de Dados (TEDs), ainda durante a concepção do sistema. Isso proporciona dados mais confiáveis, minimizando também a codificação de regras de preenchimento, domínio de valores e outras restrições sobre os tipos de dados. Todas estas informações passam a ser concentradas em um único lugar, onde são definidos formalmente os TEDs,

A solução aqui apresentada, apesar de parecer simples, não é usual, no sentido de que até o próprio usuário final pode facilmente utilizar a GeraTED para definir seus dados. É necessário que haja um comprometimento por parte de todos os envolvidos, sejam usuários finais, gerentes, Analistas de Negócios e Analistas de Sistemas para o bom funcionamento desta idéia e, conseqüentemente, seja alcançado um melhor entendimento e mais qualidade da aplicação a ser desenvolvida.

A especificação dos domínios de valores nas fases mais iniciais do desenvolvimento de sistemas ainda precisa ser mais praticada. Esta especificação permite o compartilhamento e a validação de um conhecimento fundamental que todos os envolvidos possuem acerca do domínio da aplicação. Além disso, este entendimento acordado por todos os envolvidos precisa ser passado para o sistema computacional para que ele possa armazenar, pelo menos em parte, a capacidade de interpretar diferentes situações que ocorrem no negócio. Porém, até o presente momento, as ferramentas CASE não oferecem a possibilidade de detalhamento dos tipos de dados associados aos atributos encontrados.

6.1 Resultados Esperados

Os principais resultados esperados neste trabalho são os seguintes:

- Mais tempo útil empregado no entendimento do sistema – uma vez que os desenvolvedores passam a focar no modelo nível PIM, detalhando-o para a sua transformação em PSM e obter mais geração automática de código;
- Redução da geração de código manual;
- Prover insumos para a geração automática de código em larga escala e com rapidez, usando as ferramentas de transformação disponíveis;
- Integridade de documentação entre o projeto lógico e físico - uma via em “mão-dupla” – o código pode se transformar em uma cópia fiel do que foi especificado no nível mais alto de abstração;
- A documentação se integra como parte do sistema de informação e pode ser extraída diretamente da ferramenta CASE e dos bancos de dados;
- Mais encapsulamento dos dados;
- Melhoria da integridade e da confiabilidade nos dados armazenados;
- Manutenibilidade aumentada e facilitada – para atualizar um Tipo Elementar de dado, a alteração é feita em apenas um lugar, na própria classe.

6.2 Trabalhos Relacionados

Muitos esforços vêm acontecendo no sentido de automatizar a geração de código. Alguns dos trabalhos que vêm apresentando uma boa evolução são UMT QVT e AndroMDA. Embora os resultados destes projetos sejam bastante promissores, não se pode observar neles, pelo menos até o presente, um tratamento especial para os tipos de dados com o objetivo de lhes dar melhor semântica. Vejamos um pouco sobre cada um deles.

6.2.1 UMT QVT

A UMT-QVT é uma ferramenta de transformação de modelos e geração de código de modelos UML baseados em XMI. É uma iniciativa de código aberto para dar suporte ao desenvolvimento direcionado a modelos e a MDA [UMTQVT, 2006]. Ela provê um ambiente no qual novos geradores podem ser adicionados (ou "plugados"). O ambiente de desenvolvimento desta ferramenta é implementado em Java. Os geradores devem ser implementados em XSLT ou Java. Para facilitar a visualização do código gerado, ela utiliza o XMI Light, como é possível observar na Figura 29, que mostra a tela principal da ferramenta [UMTQVT, 2006]:

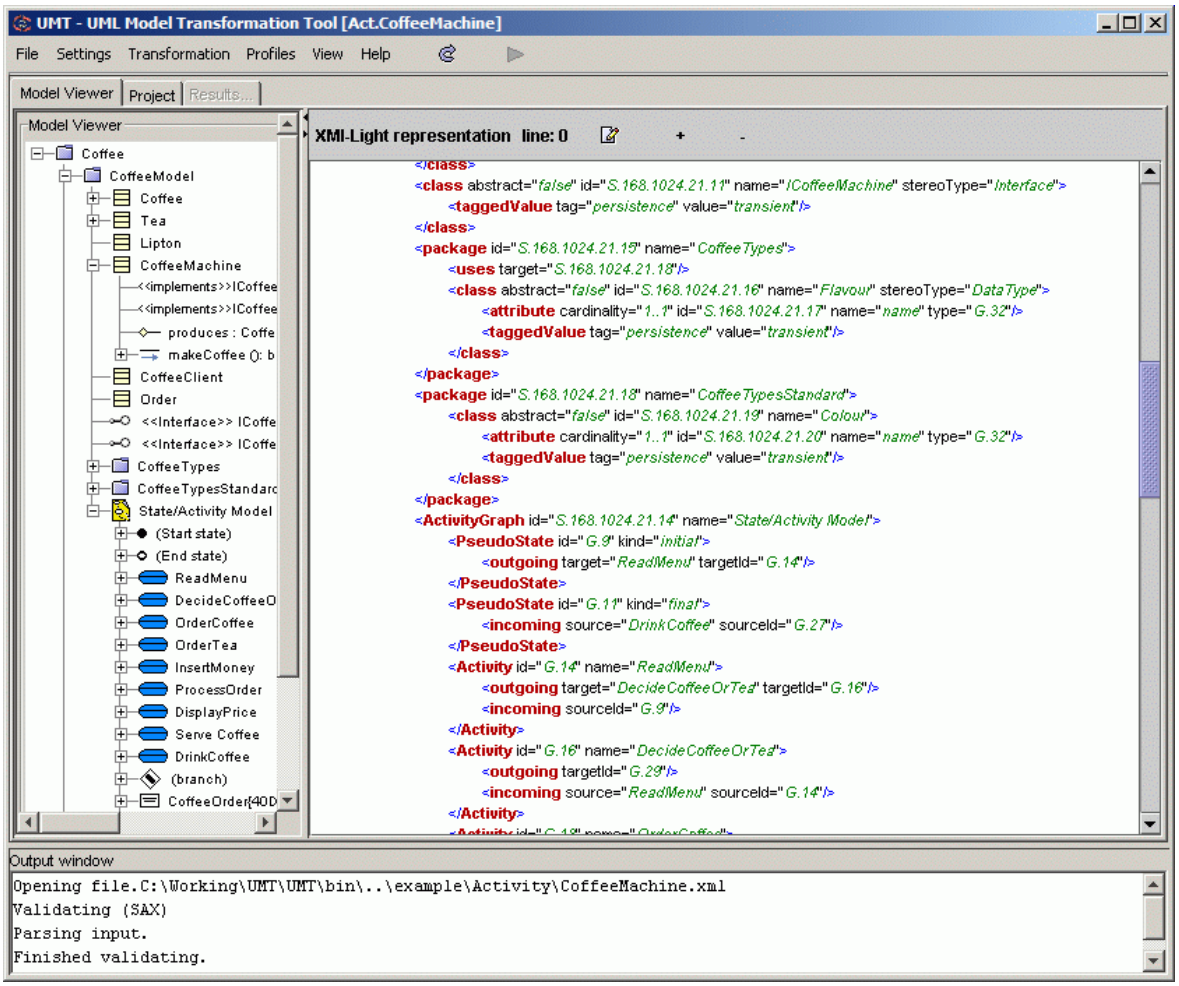


Figura 29 – Tela principal da UMT-QVT

6.2.2 AndroMDA

O projeto AndroMDA [ANDROMDA, 2005] deu origem a uma ferramenta que gera componentes J2EE rapidamente e em conformidade com modelos desenhados em UML. Ele procura seguir rigorosamente as recomendações e as transformações da MDA. A Figura 30 ilustra o esquema de funcionamento desse projeto.

AndroMDA também é conhecida como uma ferramenta de geração de código que recebe um Modelo UML como entrada e produz o código fonte como saída. Usando uma série de arquivos padrão (que podem ser customizados se necessário), AndroMDA pode criar código fonte, a partir de um modelo UML, em qualquer linguagem de programação orientada a objeto. Com esta finalidade, existem padrões para geração do código em Java e, em particular, para a produção de código J2EE. [ANDROMDA, 2005]

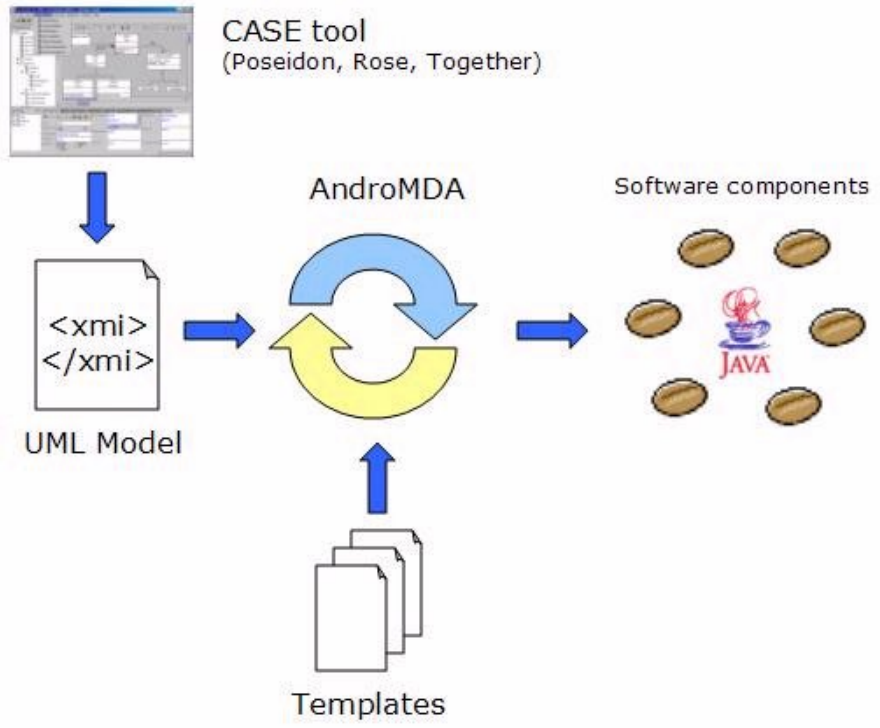


Figura 30 – Esquema de funcionamento do AndroMDA [ANDROMDA, 2005]

Existem dois componentes primários usados no sistema:

1. A máquina de geração de código AndroMDA, e
2. O Sistema de gerenciamento e construtor de projeto Maven da Apache. Este é um componente opcional, uma vez que o AndroMDA pode ser chamado diretamente a partir da linha de comandos. Contudo, ele é usado para simplificar a utilização do AndroMDA.

Em teoria, AndroMDA pode usar o arquivo de saída de qualquer ferramenta de modelagem UML como ponto de partida para a geração de código, desde que a ferramenta possa produzir um arquivo de saída no formato XMI. Na prática, como AndroMDA é um projeto em andamento, no momento da escrita desta dissertação, é recomendado o uso do *MagicDraw* como ferramenta UML. Como ela é a ferramenta usada pelos desenvolvedores desse projeto, a sua integração com AndroMDA é a que possui um suporte melhor, em comparação com outras alternativas.

6.3 Trabalhos Futuros

O leitor pode estar pensando: “capturados os metadados, e agora, como utilizá-los?”. A próxima etapa, de acordo com a MDA, é a transformação do PIM para PSM ou PSMs. Esta é a etapa mais complexa de todo o processo. Nesse momento, os metadados capturados são analisados para a elaboração de rotinas específicas de transformação e posterior geração automática de código. Assim, após a escolha da tecnologia a ser utilizada, como o gerenciador de banco de dados e o ambiente de programação, começa o aproveitamento de cada detalhe obtido durante o levantamento inicial e descrição do PIM.

Por exemplo, é desejável que, a partir do XML de saída da ferramenta contendo a definição dos Tipos Elementares, tal como ilustrado na Figura 28c do Capítulo 5, uma ferramenta de transformação possa produzir a codificação das classes para o sistema. Desta

forma, cada uma das características levantadas no questionário (vide item 5.3.1) poderá também dar origem a métodos que serão responsáveis pelas críticas de preenchimento dos dados nas classes.

Podemos esperar a geração de métodos como:

- ValidaNumeroMaximoCaracteres, para um Tipo Elementar que possua atributo QuantidadeMaximaDeCaracteres;

- ValidaNumeroMinimoCaracteres, para um Tipo Elementar que possua atributo QuantidadeMinimaDeCaracteres;

- VerificaPreenchimento, para um Tipo Elementar que possua valor “true” para o atributo PreenchimentoObrigatorio;

- ValidaData, para um Tipo Elementar que possua o atributo Data = “true”;

-ValidaNumeroDeCasasDecimais para um Tipo Elementar que possua definido o NumeroDeCasasDecimais; e assim por diante.

Uma opção de melhoria para a ferramenta GeraTED seria a utilização do XMI Light, como o da UMT-QVT. Na nossa proposta apresentada no Capítulo 5, a GeraTED recebe XML e retorna XML. Entretanto, ela poderia ser adaptada para utilizar também o XMI Light. Desta maneira, seria consideravelmente aumentada a compatibilidade com as ferramentas de desenho existentes no mercado.

Estendendo a idéia apresentada no Capítulo 5, pode-se desenvolver um mecanismo que, dado um sistema já em produção, seja capaz de identificar, através do conteúdo de seus atributos, um Tipo Elementar no qual ele se enquadre – uma espécie de Engenharia Reversa de Tipos Elementares. A partir daí, mecanismo poderia passar a fazer a validação do preenchimento desses atributos, emulando uma nova camada entre o banco de dados e a aplicação, onde os dados seriam criticados.

Desta forma, qualquer preenchimento que apresente não-conformidade com o universo já cadastrado, seria identificado e o usuário seria notificado para que pudesse acertar o conteúdo do atributo ou mudar as especificações do sistema. Por exemplo, um nome de pessoa que apresentasse vários sobrenomes. Se na base de dados o tamanho do maior nome pudesse conter, por exemplo, 8 sobrenomes e o novo nome contém 9 ou mais, o sistema autônomo identificaria esta anormalidade e enviaria uma mensagem de rejeição ao usuário. Ocorreria um procedimento análogo, caso um nome apresentasse uma seqüência de 3 caracteres iguais ou aparecesse um nome sem sobrenome, e assim por diante.

Uma outra idéia seria ampliar o conceito desse mecanismo proposto no parágrafo anterior, concedendo-lhe um certo grau de autonomia. Por exemplo, quando esse novo mecanismo autônomo fosse aplicado em um sistema em uso, ele estaria pronto para se modificar, adequando-se, moldando-se ao sistema durante o seu ciclo de vida, possibilitando redefinições “em tempo de execução” nos Tipos Elementares de Dados existentes. Neste caso, o cuidado a ser tomado é o de que as alterações no domínio de valores devem ser compatíveis com os dados pré-existentes na base de dados, de modo a não permitir o surgimento de inconsistências. Assim, poderíamos ter uma Biblioteca “Móvel” de Tipos Elementares de Dados que se auto-ajustaria de acordo com as mudanças ocorridas no sistema, seja para atender novos requisitos ou devido a mudanças nos requisitos já existentes.

Uma dificuldade encontrada durante o desenvolvimento do protótipo executável da ferramenta foi a não utilização de um banco de dados para armazenamento dos Tipos Elementares de Dados criados. A manipulação dos Tipos Elementares novos fica um pouco limitada ou dificultada utilizando-se apenas o XML. Uma possível melhoria na ferramenta seria a manipulação dos Tipos Elementares via uma camada de persistência. Isto facilitaria tanto no armazenamento e na recuperação quanto numa possível ordenação e busca por tipos existentes, principalmente para sistemas maiores. Além do mais, abriria o leque para uma

reutilização desses tipos, graças à possibilidade de compartilhamento do banco de dados dos Tipos Elementares criados através da ferramenta.

A linguagem do MOR poderia ser integrada com a abordagem de Ontologias. Desta forma, seria utilizado o formalismo da linguagem e favorecida a criação de bibliotecas de Tipos Elementares para determinados domínios. Poderiam haver bibliotecas de Tipos Elementares voltadas para saúde, transportes, educação, entre outros, onde uma mesma pessoa, por exemplo, poderia desempenhar diferentes papéis, como paciente, passageiro e aluno. Dependendo do domínio considerado. Como é possível a reutilização de alguns Tipos Elementares. no caso deste exemplo, o Tipo Elementar NomeComposto poderia ser comum a todos os domínios, ao pertencer a um domínio mais genérico, podendo, eventualmente, ser reutilizado através da sua descrição formal.

O questionário apresentado na seção 5.3.1 do Capítulo 5 é incremental. Podem haver bancos de perguntas elaboradas sob medida para o negócio em estudo. O próprio usuário final poderia criar suas perguntas e domínio de respostas possíveis, de acordo com o negócio em questão, utilizando até mesmo o seu universo de discurso. Para isso, a GeraTED precisaria ser redesenhada a fim de oferecer parametrização de perguntas e respostas, com a criação de um banco de perguntas.

A GeraTED apresenta a seguinte limitação neste momento: quando os Tipos Elementares resultantes podem ser agregados ou compostos, isto é, quando um Tipo Elementar derivado diretamente de um dos tipos primitivos pode servir de base para a criação de um outro Tipo Elementar ainda mais especializado, a ferramenta não está preparada para representá-los. Como a hierarquia pode continuar descendo o nível de especialização, a ferramenta pode ser melhorada no intuito de representá-la.

Um Modelo de Negócio bem detalhado pode servir como uma fonte rica de informação para geração automática de código. A chave para isso está na captura e

gerenciamento inteligentes dos Metadados, conforme foi mostrado ao longo dessa dissertação. Uma ferramenta que auxilie na captura desses Metadados durante o levantamento de requisitos do sistema de informação, além de melhorar enormemente o entendimento dos dados levantados, será certamente um diferencial em direção à automação do código. Portanto, a contribuição apresentada é um passo importante na melhoria da produtividade para construção de aplicações de software.

Referências Bibliográficas

Agile Modelling - **UML 2 Use Case Diagram Overview**. disponível em: http://www.agilemodeling.com/artifacts/useCaseDiagram.htm. Acesso em 07/07/2005

AndroMDA – **Components quickly with AndroMDA**. disponível em: http://www.andromda.org. Acesso em: 28/11/2005

AskOxford.com - **The Compact Oxford English Dictionary of Current English**. disponível em: http://www.askoxford.com/concise_oed/model?view=uk. Acesso em 05/07/2005

Aßmann, U. - **Model-Driven Architecture (MDA) and Ontologies** . disponível em: http://www.ida.liu.se/~uweas/Rewerse/I3/Kickoff/mda-introduction.pdf. Acesso em: 04/03/2006

Cunha, A. M. – **Notas de aula do curso de Modelagem de Negócio Orientada a Objeto**. NCE/UFRJ, 2004.

Fowler, M. - **UML Distilled: A Brief Guide to the Standard Object Modeling Language**. Boston: Addison-Wesley Professional - 3ª Edição, 2003. 192p.

Frankel, David S. – **Model Driven Architecture Applying MDA to Enterprise Computing.** Indianapolis: Wiley Publishing Inc., 2003. 328p.

Gilliland-Swetland, A. J. – **Setting the Stage – Defining Metadata** - disponível em: http://www.getty.edu/research/conducting_research/standards/intrometadata/2_articles/index.html. Acesso em: 16/06/2005

IBM - **Working XML: UML, XMI®, and code generation, Part 1.** disponível em: http://www-106.ibm.com/developerworks/library/x-wxxm23. Acesso em: 03/03/2006

Kleppe, A., Warmer, J., Bast, W. – **MDA Explained: The Model Driven Architecture - Practice and Promise**. Boston : Addison-Wesley Pub Co, 2003. 170p.

Metamodel.com – **Metamodel**. disponível em: http://www.metamodel.com. Acesso em 05/04/2006

Nytum, J.P. - **The UML Metamodel Architecture**. disponível em: http://fag.grm.hia.no/ikt2340/topic/modelling/Metamodel.pdf. Acesso em 28/09/2005

Oldfield, P. - **Domain Modelling**. disponível em: http://www.aptprocess.com/whitepapers/DomainModelling.pdf. Acesso em 07/07/2005

OMG - **Introduction to OMG's Unified Modeling Language™ (UML®)**. disponível em: http://www.omg.org/gettingstarted/what_is_uml.htm. Acesso em: 13/07/2005

OMG - **MDA Guide Version 1.0.1**. Needham, 2003. disponível em: http://www.omg.org/docs/omg/03-06-01.pdf. Acesso em: 14/06/2005

OMG - **MDA - "The Architecture of Choice for a Changing World®"**. disponível em: http://www.omg.org/mda. Acesso em: 16/06/2006

OMG - **Meta-Object Facility (MOF)**, version 1.4 (download). Needham, 2002. disponível em: http://www.omg.org/technology/documents/formal/mof.htm Acesso em: 15/06/2005

Poole, J.D. - **Model-Driven Architecture: Vision, Standards And Emerging Technologies**. disponível em: http://www.cwmforum.org/Model-Driven%20Architecture.pdf. Acesso em: 07/07/2005

Pressman, R. S. - **Engenharia de Software**. Rio de Janeiro: Makron Books - 3ª Edição, 1998. 1056p.

Pressman, R. S. - **Engenharia de Software**. Rio de Janeiro: McGraw-Hill - 5ª Edição, 2002. 843p.

Rumbaugh, J., Booch, G., Jacobson, I. - **UML: Guia do Usuário**. Rio de Janeiro: Campus, 2000. 474p.

UMT-QVT Homepage **– UMT QVT Tool –** disponível em: http://umt-qvt.sourceforge.net/. Acesso em: 23/06/2006

W3C - **Extensible Markup Language (XML)**. disponível em: http:// www.w3.org/XML. Acesso em: 06/03/2006

Wikipedia - **Type system**. disponível em: http://en.wikipedia.org/wiki/Type_system. Acesso em 15/05/2006