

UNIVERSIDADE ESTADUAL PAULISTA  
“JÚLIO DE MESQUITA FILHO”  
Instituto de Geociências e Ciências Exatas – IGCE  
Curso de Bacharelado em Ciências da Computação

EVALDO DIAS DE JESUS

**A CONTRIBUIÇÃO DE MODELOS UTILIZANDO UMA ABORDAGEM  
MODELO - CÓDIGO**

Trabalho realizado sob orientação do Prof. Dr. Frank Jose Affonso, DEMAC/IGCE  
Período: 17.08.2011 a 03.12.2011

Rio Claro - SP  
2011

## A CONTRIBUIÇÃO DE MODELOS UTILIZANDO UMA ABORDAGEM MODELO - CÓDIGO

Trabalho de Conclusão do Curso, Trabalho de Graduação, apresentado à disciplina ES/TG, do Curso de Bacharelado em Ciências da Computação, período Noturno, do Instituto de Geociências e Ciências Exatas da Universidade Estadual Paulista “Júlio de Mesquita Filho”, campus de Rio Claro, para apreciação segundo as normas estabelecidas pelo Conselho do Curso, em 27.11.2007.

---

Aluno: Evaldo Dias de Jesus

---

Orientador: Prof. Dr. Frank Jose Affonso  
Departamento de Estatística, Matemática Aplicada e Computação (DEMAC) / IGCE / UNESP

Rio Claro - SP  
2011

005 Jesus, Evaldo Dias de  
J58 A contribuição de modelos utilizando uma abordagem  
modelo - código / Evaldo Dias de Jesus. - Rio Claro : [s.n.],  
2011  
54 f. : il., figs.

Trabalho de conclusão de curso (bacharelado - Ciências  
da Computação) - Universidade Estadual Paulista, Instituto de  
Geociências e Ciências Exatas  
Orientador: Frank Jose Affonso

1. Engenharia de software. 2. Arquitetura. 3.  
Desenvolvimento. 4. Metodologia. 5. Linguagem. I. Título.

EVALDO DIAS DE JESUS

A CONTRIBUIÇÃO DE MODELOS UTILIZANDO UMA  
ABORDAGEM MODELO - CÓDIGO

Trabalho de Graduação apresentado ao Instituto de Geociências e Ciências Exatas - Câmpus de Rio Claro, da Universidade Estadual Paulista Júlio de Mesquita Filho, para obtenção do grau de Bacharel em Ciências da Computação.

Comissão Examinadora

Prof. Dr. Frank Jose Affonso (orientador)

Prof(a). Dr(a). Maria Cecilia Vecchiato Saenz Carneiro

Prof. Dr. Leandro Alves Neves

Rio Claro, 12 de dezembro de 2011.

## **AGRADECIMENTOS**

A Deus acima de tudo, pois somente ele poderia me ajudar e inspirar nos momentos difíceis.

Ao Prof. Dr. Frank J. Affonso, meu orientador, pela ajuda e contribuição fundamental ao longo desse semestre, mostrando o caminho das pedras.

À minha mãe, irmão e cunhada que sempre estiveram ao meu lado me ajudando, perseverando, incentivando e contribuindo com a minha formação acadêmica.

Ao meu pai, que contribuiu com minhas conquistas e ajudou a tornar o que sou.

## RESUMO

Em um mundo em que cada vez mais se trabalha com software, surge a necessidade constante de abordagens efetivas que favoreçam a reutilização de software. Essa reutilização deve estar alinhada a um conjunto de práticas, procedimentos e metodologias que originem um produto estável e de boa qualidade. Questões assim dão margem a novos estilos e abordagens na engenharia de software. Com isso, essa dissertação visa abordar conceitos relativos ao desenvolvimento e arquitetura orientada a modelos. A orientação a modelos traz aspectos significativos à automatização do desenvolvimento, onde modelos construídos durante a fase de especificação auxiliam o desenvolvimento. A definição de termos como modelo, arquitetura e plataforma faz com que o enfoque se torne mais claro, pois para MDA e MDD é importante a separação entre os quesitos técnicos e os de negócio. Processos importantes são abordados, para que se possa destacar os artefatos que são construídos em cada etapa do desenvolvimento orientado a modelo. As etapas do desenvolvimento: CSM, PIM, PSM e ISM, detalham o objetivo de cada fase na orientação a modelos, fazendo com que ao fim de cada estágio sejam produzidos artefatos que gradativamente venham a ser especializados. Os modelos são tratados por diferentes perspectivas de modelagem, abstraindo os conceitos e construindo um conjunto de detalhes que retrata um cenário específico. Essa retratação pode ser uma representação gráfica ou textual, no entanto, na maioria das vezes opta-se por uma linguagem de modelagem, por exemplo, a UML. Visando oferecer uma visão prática, essa dissertação expõe algumas ferramentas que favorecem a construção de modelos e a geração de código que auxilia o desenvolvimento, mantendo a documentação sistêmica. Finalmente, o trabalho apresenta um estudo de caso que menciona os aspectos teóricos discutidos ao longo da dissertação, com isso espera-se que a arquitetura e desenvolvimento orientado a modelos consigam expor características importantes a serem consideradas na engenharia de software.

**Palavras-chave:** Modelo. Arquitetura. Desenvolvimento. Software. Metodologia. Linguagem.

## **ABSTRACT**

In a world that is increasingly working with software, the need arises for effective approaches that encourage software reuse. The reuse practice must be aligned to a set of practices, procedures and methodologies that create a stable and high quality product. These questions produce new styles and approaches in the software engineering. In this way, this thesis aims to address concepts related to development and model-driven architecture. The model-driven approach provides significant aspects of the automated development, which helps it with produced models built in the specification phase. The definition of terms such as model, architecture and platform makes the focus becomes clearer, because for MDA and MDD is important to split between technical and business issues. Important processes are covered, so you can highlight the artifacts that are built into each stage of model-driven development. The stages of development: CSM, PIM, PSM and ISM, detailing the purpose of each phase in oriented models, making the end of each stage are gradually produced artifacts that may be specialized. The models are handled by different prospects for modeling, abstracting the concepts and building a set of details that portrays a specific scenario. This retraction can be a graphical or textual representation, however, in most cases is chosen a language modeling, for example, UML. In order to provide a practical view, this dissertation shows some tools that improve the construction of models and the code generate that assists in the development, keeping the documentation systemic. Finally, the paper presents a case study that refers to the theoretical aspects discussed throughout the dissertation, therefore it is expected that the architecture and the model-driven development may be able to explain important features to consider in software engineering.

**Keywords:** Model. Architecture. Development. Software. Methodology. Language.

## LISTA DE ILUSTRAÇÕES

|                                                            |    |
|------------------------------------------------------------|----|
| Figura 1 – Fluxo do MDD.....                               | 20 |
| Figura 2 – Diagrama de modelagem.....                      | 23 |
| Figura 3 – Expressões em OCL.....                          | 24 |
| Figura 4 – Restrição para construção de interface.....     | 25 |
| Figura 5 – Fluxo de geração.....                           | 26 |
| Figura 6 – Representação de um diagrama em XML.....        | 28 |
| Figura 7 – Especificação em formato XSD.....               | 31 |
| Figura 8 – Diagrama de classes UML com base em um XML..... | 32 |
| Figura 9 – Relacionamento dos itens do <i>Ecore</i> .....  | 33 |
| Figura 10 – Hierarquia de classes no <i>Ecore</i> .....    | 35 |
| Figura 11 – Fluxo da metodologia.....                      | 37 |
| Figura 12 – Modelo PIM de especificação.....               | 40 |
| Figura 13 – Modelo PSM de especificação.....               | 41 |

## LISTA DE ABREVIATURAS E SIGLAS

**BNF** – *Backus-Naur Form*

**CIM** – *Computation Independent Model*

**CWM** – *Common Warehouse Metamodel*

**DSL** – *Domain Specific Language*

**EMF** – *Eclipse Modeling Framework*

**GMF** – *Graphical Modeling Framework*

**GUI** – *Graphical User Interface*

**ISM** – *Implementation Specific Model*

**MDA** – *Model Driven Architecture*

**MDD** – *Model Driven Development*

**MDE** – *Model Driven Engineering*

**MDSD** – *Model Driven Software Development*

**MOF** – *Meta-Object Facility*

**OCL** – *Object Constraint Language*

**OMG** - *Object Management Organization*

**PIM** – *Platform Independent Model*

**PSM** – *Platform Specific Model*

**UML** – *Unified Modeling Language*

**XSD** – *XML Schema Definition*

**XML** – *Extensible Markup Language*

**XMI** – *XML Metadata Interchange*

## SUMÁRIO

|     |                                              |    |
|-----|----------------------------------------------|----|
| 1   | INTRODUÇÃO.....                              | 11 |
| 1.1 | Objetivo.....                                | 12 |
| 1.2 | Metodologia .....                            | 12 |
| 1.3 | Organização do trabalho .....                | 13 |
| 2   | ABORDAGENS PARA A ORIENTAÇÃO A MODELOS ..... | 15 |
| 2.1 | Abordagens.....                              | 15 |
| 2.2 | Modelo .....                                 | 16 |
| 2.3 | Arquitetura orientada a modelo .....         | 16 |
| 2.4 | Desenvolvimento orientado a modelo .....     | 18 |
| 3   | LINGUAGEM ESPECÍFICA DE DOMÍNIO .....        | 21 |
| 4   | LINGUAGEM DE RESTRIÇÃO DE OBJETOS .....      | 22 |
| 4.1 | Restrições na modelagem.....                 | 23 |
| 4.2 | Abordagem OCL e o padrão MDA.....            | 25 |
| 5   | TRANSFORMAÇÃO COM ACCELEO .....              | 26 |
| 5.1 | Objetivo.....                                | 26 |
| 5.2 | Recursos.....                                | 27 |
| 6   | MODELAGEM COM EMF.....                       | 29 |
| 6.1 | Objetivo.....                                | 29 |
| 6.2 | Recursos.....                                | 30 |
| 6.3 | O meta-modelo <i>ECORE</i> .....             | 33 |
| 7   | METODOLOGIA.....                             | 36 |
| 8   | ESTUDO DE CASO.....                          | 38 |
| 8.1 | Construção do CIM .....                      | 38 |
| 8.2 | Construção do PIM .....                      | 39 |
| 8.3 | Construção do PSM .....                      | 40 |
| 8.4 | Construção do ISM .....                      | 41 |

|    |                                                                                |    |
|----|--------------------------------------------------------------------------------|----|
| 9  | CONCLUSÃO .....                                                                | 43 |
| 10 | REFERÊNCIAS.....                                                               | 44 |
| 11 | APÊNDICES .....                                                                | 45 |
|    | Apêndice A – Representação XMI do esquema biofábrica .....                     | 45 |
|    | Apêndice B – <i>Template</i> para geração das classes Java <i>bean's</i> ..... | 47 |
|    | Apêndice C – <i>Template</i> para geração da camada de acesso a dados.....     | 48 |
|    | Apêndice D – Classes geradas.....                                              | 50 |

## 1 INTRODUÇÃO

A maneira de fazer software vem agregando importantes inovações, que fazem com que o software deixe de ser um simples produto. A reescrita de um componente não pode exigir um retrabalho a cada concepção de um novo sistema.

A automatização no processo de desenvolvimento favorece novos métodos durante a fase de implementação, entre eles está a reutilização de software. Os benefícios da reutilização estão vinculados a um maior dinamismo e qualidade do produto, propondo um desenvolvimento aos moldes da linha de produção. Essa abordagem reutiliza conceitos e incorpora artefatos que foram devidamente utilizados e testados em outras aplicações. Com isso, essa maneira de fazer software se tornou um ponto relevante na linha da engenharia de software, e o desenvolvimento orientado a modelo faz parte dessa perspectiva.

Mesmo com a crescente necessidade de demanda por software, ainda não é possível definir um padrão para abordagens de reúso, e a qualidade dos processos na engenharia de software ainda é tratada numa visão inicial. O reúso na maioria das vezes está vinculado às abordagens técnicas, em que o vê ligado principalmente ao código fonte. A engenharia de software proclama abordagens diversificadas, que se fazem necessárias no levantamento de artefatos, visando favorecer o fator construção/concepção de um domínio.

Em 2001 notou-se uma crescente necessidade em padronizar o desenvolvimento de sistemas, com isso o OMG (*Object Management Organization*) definiu as diretrizes do MDA (*Model Driven Architecture*), que veio a separar o conceito de negócio junto às questões tecnológicas, havendo assim um maior enfoque no domínio ou negócio. Através de modelos é possível abordar cenários e comportamentos que o sistema passa a exercer, e assim conceber um modelo de domínio mais eficaz que vem a apoiar não somente o entendimento do negócio, mas também propiciar um desenvolvimento conciso. O principal fator motivador: modelo e código devem ser facilitadores durante a construção do domínio.

Conceitos do MDA buscam separar a abordagem de domínio da tecnológica, com isso surgem estratégias de definição como PIM (*Platform Independent Model*), PSM (*Platform Specific Model*) e CIM (*Computation Independent Model*), que tratam respectivamente do modelo independente de plataforma, do modelo específico de plataforma e da visão que o usuário possui em relação ao negócio. Critérios assim são amplamente comentados e lidados no padrão MDA, fazendo assim com que haja separação nas perspectivas de abordagem de cada elemento. Os ideais expostos por esse padrão trazem para a metodologia adotada uma categorização por hierarquias, pois cada critério retrata uma abordagem diferente ao nível do sistema. Esses conceitos serão detalhados nesse trabalho, para que se possa entender corretamente cada componente do MDA.

## 1.1 Objetivo

Este trabalho tem por objetivo apresentar linhas da engenharia de software, expondo principalmente o desenvolvimento e arquitetura orientada a modelos. A partir de conceitos e estratégias de modelagem, a construção de artefatos orientados a modelos permitem a concepção de perspectivas especialistas para a elaboração do modelo de domínio.

Com o intuito de esclarecer e exemplificar os conceitos apresentados, algumas análises são tratadas juntamente com a exposição do estudo de caso, que visa detalhar as abordagens orientadas a modelos e a metodologia adotada durante esse trabalho.

No estudo de caso, cenários reais podem fazer uso de uma perspectiva de orientação a modelos, onde o maior objetivo é apresentar a eficiência e técnica que tal abordagem traz no sentido de automatizar processos e análises, que estariam naturalmente numa linha manual, sofrendo por ventura com uma defasagem sistêmica.

A referência a conceitos da MDA e MDD (*Model Driven Development*) desperta uma visão conceitual sobre questões e estratégias adotadas na orientação a modelos, tratando de abordagens importantes como: DSL (*Domain Specific Language*) e OCL (*Object Constraint Language*). Cada um desses conceitos é responsável por complementar o entendimento diante das questões de modelagem, ou seja, oferecem subsídios que são de fundamental importância para que se inicie qualquer perspectiva orientada a modelos.

Por meio da UML (*Unified Modeling Language*) serão criados modelos de domínio que visam expor, através de um enfoque alto nível, os benefícios oferecidos para apresentar a visão do sistema e como cada módulo contribui durante a fase de especificação. O modelo construído em UML tem como objetivo oferecer insumos necessários para análises que expõem os conceitos do MDA e/ou MDD. A modelagem fará com que se possa extrair recursos das ferramentas que seguem os padrões de enfoque desse trabalho, podendo mostrar como a automatização está inserida num desenvolvimento orientado a modelos.

A principal estratégia do presente trabalho está em abordar uma visão da orientação a modelo no segmento gráfico – código através de ferramentas e *frameworks* específicos, que de posse de uma especificação em alto nível faz a geração em código fonte, expondo assim o real intuito de uma abordagem baseada em métodos por modelos.

## 1.2 Metodologia

A metodologia usada está vinculada a padrões estabelecidos pela MDA e MDD, que visam oferecer qualidade no ciclo de vida em desenvolvimento de aplicações, dados e padrões de integração de aplicações, mostrando boas práticas na arquitetura de software, modelagem, gerenciamento de meta-dados e transformação de modelos, favorecendo o

desenvolvimento de modelos que passam a ser construídos em fase inicial e posteriormente facilitam os níveis de desenvolvimento de software.

Para esclarecer tal metodologia, o estudo de caso será exposto inicialmente em uma abordagem alto nível, em que especifica o cenário principal do sistema, onde exemplificações em UML serão utilizadas para que se possa entender a interação e relacionamento de cada objeto, fornecendo as propriedades e funcionalidades de cada. De posse dos últimos detalhes, será possível construir a perspectiva de domínio através de ferramentas adequadas.

As práticas serão expostas com a ajuda da ferramenta *Eclipse Modeling* e tomando como modelo de domínio a UML, sendo que a manipulação desse modelo será tratada pelo EMF. Esse é um *framework* de modelagem e geração de código que facilita a confecção de modelos baseados nos aspectos sistêmicos, podendo gerar e customizar os diagramas que irão compor cada modelo.

O projeto EMF (*Eclipse Modeling Framework*) reúne os padrões adotados pelo MDA, suporta os conceitos da metodologia e expõe os objetivos acerca da boa prática de desenvolvimento, documentação e arquitetura, fazendo uso de modelos UML para modelagem de dados nos projetos de desenvolvimento e nas ferramentas de integração. Um modelo de domínio ou meta-modelo é usado para servir como base na geração de código, compondo assim um detalhamento alto nível ou abstração sobre o cenário a ser tratado.

### **1.3 Organização do trabalho**

Esta seção descreve como este trabalho está organizado e o que se espera de cada capítulo.

O capítulo 2 apresenta as abordagens de modelagem, que são praticadas por algumas multinacionais de software, conceitos iniciais sobre a modelagem, a estrutura e o que é arquitetura e desenvolvimento orientado a modelos.

No capítulo 3 são apresentados conceitos sobre a linguagem específica de domínio, como ela contribui com a modelagem e com a especificação do domínio.

O capítulo 4 apresenta os conceitos da linguagem de restrição a objetos, que acaba incorporando certas regras aos objetos, tornando-os mais restritivos e próximos do domínio. Algumas exemplificações são apresentadas, demonstrando como uma restrição valoriza a modelagem e como se pode construir restrições em OCL.

O capítulo 5 inicia uma abordagem mais prática, onde se detalha o objetivo da ferramenta Acceleo. Trata dos recursos que contribuem com a modelagem e, como a automatização na geração de código favorece o desenvolvimento.

No capítulo 6 uma segunda ferramenta de modelagem é apresentada, o EMF. São detalhados recursos de modelagem e meta-modelo *Ecore*.

O capítulo 7 detalha a metodologia usada no estudo de caso, apresentando as etapas de modelagem. Ao fim do capítulo uma representação esquemática é apresentada, para que se possa entender a interação e papel de cada elemento da metodologia.

Ao longo do capítulo 8 são detalhadas todas etapas de modelagem que são defendidas pelo MDA e MDD. Esse capítulo apresenta os artefatos que são esperados por cada fase e como cada um colabora com a automatização do processo de desenvolvimento.

Finalmente, no capítulo 9 são apresentadas as conclusões obtidas nessa monografia.

## 2 ABORDAGENS PARA A ORIENTAÇÃO A MODELOS

Este capítulo apresenta os princípios básicos que circundam o MDD, tratando diferentes visões adotadas por corporações que lidam com esse segmento de abordagem, mas tendo, sobretudo como foco principal o detalhamento de conceitos ligados à arquitetura orientada a modelos do OMG.

### 2.1 Abordagens

Não apenas o OMG investiu no desenvolvimento orientado a modelos, mas outras organizações decidiram construir os seus próprios padrões de visão orientada a modelos. Belix (2006, p. 40) apresenta três diferentes abordagens que são responsáveis por construir critérios diferentes de uma mesma perspectiva de análise.

A Microsoft definiu as diretrizes da orientação a modelos através do *Software Factories*, que diferentemente de outras abordagens, não faz uso da UML, mas incorpora as práticas da DSL, a maior justificativa para não fazer uso da linguagem de modelagem mais conhecida, é o fato de alegar que essa é totalmente inconsistente. Para a Microsoft não é possível utilizar a UML em modelos focados em domínio, visto que se trata de uma linguagem para propósitos gerais, preferindo adotar a DSL que na prática é uma linguagem de processamento para máquinas, em que a partir de um modelo pode-se construir estruturas que serão reconhecidas como modelos de domínio.

Para Belix (2006, p. 40), uma segunda abordagem que é interessante ao aspecto da modelagem é o MDSD (*Model Driven Software Development*), que é amplamente conhecido por se tratar de uma especificação mais abrangente que o MDA. Essa abordagem incorpora níveis de abstração elevados, não estando preso especificamente a uma linguagem de modelagem ou modelo de domínio. Na verdade, é proposto o uso de princípios da modelagem e meta-modelagem, possuindo alguns padrões ditados pelo OMG, mas não permanece restrito a esses.

Uma característica comum aos métodos é o fato de que ambos fazem uso da linguagem de domínios específicos, uma vez que essa é uma importante ferramenta para lidar com as linhas de produção de software. O principal objetivo dessas soluções é fornecer mecanismos independentes, que sirvam como base para produção de sistemas por meio do segmento de modelagem. Diante de algumas noções dos métodos anteriores, pode-se compará-los com a abordagem amplamente divulgada e discernida pelo OMG, o padrão MDA.

## 2.2 Modelo

O foco principal do MDA está nos modelos, que segundo Mignon (2007, p. 09), deixam de ocupar uma posição secundária na engenharia de software para integrarem principalmente o desenvolvimento baseado em modelos. A modelagem é responsável por reunir a especificação e descrição do sistema, incorporando aspectos do ambiente em que serão inseridos, pois a contextualização influencia no cenário e modelos a serem elaborados. Um modelo pode ser uma definição gráfica (*designs*), através de simples desenhos de elaboração ou diagramas, ou até mesmo uma descrição textual para especificação alto nível.

Caliari (2007, p. 07) aborda que através de linguagens de modelagem, como a UML, o modelo fornece uma abstração acerca do sistema e os desenvolvedores conseguem extrair informações sobre o comportamento e interação, podendo ressaltar aspectos relevantes ou críticos de um ponto de vista.

O OMG emprega a modelagem do MDA com o uso da linguagem de modelagem unificada, uma vez que essa é uma linguagem de modelagem de propósito geral, podendo ser usada em qualquer nível do desenvolvimento e servindo de propósito para qualquer abordagem de domínio.

Caliari (2007, p. 10) comenta que a meta-modelagem oferece subsídios para a construção de linguagens de modelagem, servindo como modelo para o processamento de padrões especificados por linguagens de modelagem. Meta-modelos são responsáveis por estabelecer definições, regras e semânticas que devem ser seguidas pelo seu respectivo modelo. O OMG define que o meta-modelo da UML é derivado a partir do MOF (*Meta-Object Facility*), que é um gerenciador de meta-dados que oferece serviços de modelagem para a construção de meta-modelos, sendo de grande importância para oferecer interoperabilidade na elaboração de modelos de domínio.

## 2.3 Arquitetura orientada a modelo

Para Belix (2006, p. 29), o OMG através do MDA propõe uma padronização para possibilitar a longo prazo, maior flexibilidade em processos de implementação, integração, manutenção, teste e simulação, indicando que é possível haver a integração entre sistemas através de pontes, reunindo assim diferentes infraestruturas.

Os princípios do MDA se apoiam na estrutura de modelos, sendo que propõe um desenvolvimento de software apoiado em padrões de modelagem, utilizando UML, MOF ou CWM (*Common Warehouse Metamodel*). Esse fato defende uma postura evolucionária para

que a automação por modelos faça-se gerar o software a partir de modelos pré-concebidos, pois se acredita que o modelo é a melhor forma de refletir o aspecto sistêmico.

A proposta de modelagem lança bases para um problema bem conhecido no desenvolvimento de software: a defasagem que os modelos sofrem em relação a constante mudança no código fonte, que por sua vez não é atualizada em seus modelos de especificação. Com isso, essa abordagem acaba gerando um fator motivador, já que a alteração do modelo será responsável por auxiliar no processo de implementação, diminuindo assim a taxa de retrabalho.

Importantes definições do OMG, em relação ao MDA, são apresentadas em Caliari (2007, p.07), são elas:

- Arquitetura: Trata dos componentes e estrutura de um sistema, como esses se relacionam ou interagem de acordo com a necessidade do sistema. Uma arquitetura dirigida a modelos deve gerenciar modelos que se aplicam corretamente às necessidades esperadas, garantindo que hajam regras fundamentais na interação das partes do sistema.
- Plataforma: É um conjunto que engloba subsistemas e tecnologias, que farão com que determinadas funções possam ser usadas através de interfaces, fazendo assim com que outras aplicações possam consumir serviços externos, sem se preocupar com o modo com que tais foram implementados, ou seja, os serviços são encapsulados.

A perspectiva de modelagem vem a agregar e contribuir com toda uma linha de especificação sistêmica, oferecendo benefícios que são evidentes a cada estágio. Na manutenção há uma maior facilidade por parte do desenvolvedor em entender cenários até então desconhecidos, visto que possuirá um *design* atualizado que será reflexo da especificação do sistema, e sua atualização será de crucial importância para geração de novos artefatos. Durante a fase de teste e simulação é notório que os modelos são responsáveis pela geração de código compatível com o negócio proposto, sendo muito mais simples de avaliar uma especificação, testar cenários contra várias infraestruturas e simular diretamente o comportamento do sistema.

Um ponto chave, apresentado por Belix (2006, p. 32) e Mignon (2007, p. 08), está no fato de empregar níveis de abordagem que separam totalmente o critério de negócio ou domínio, do aspecto técnico, criando assim conceitos importantes: PIM ou modelo independente de plataforma, PSM ou modelo específico de plataforma e CIM ou modelo independente de computação. Esses são conceitos fundamentais presentes no MDD que fazem com que cada interação esteja estruturada em uma etapa específica do processo de desenvolvimento, conforme é descrito na próxima seção.

## 2.4 Desenvolvimento orientado a modelo

O desenvolvimento por meio da modelagem estabelece diretrizes aos moldes do OMG, empregando diretivas e conceitos durante a análise e desenvolvimento de um sistema. Os conceitos mais influentes em um desenvolvimento orientado a modelo são:

- CIM: Etapa livre de qualquer aspecto computacional, onde é necessário que ocorra os levantamentos de requisitos do sistema e o ambiente de operação, sendo que haja a construção de modelos que estruturam o comportamento operacional, sem qualquer vinculação com tecnologia ou plataforma. Essa etapa produzirá artefatos que exprimem o real objetivo do domínio, sendo de primordial importância às etapas posteriores.
- PIM: Especifica um modelo não focado em plataforma, pois a proposta é fazer uso dos requisitos levantados pelo CIM, focando apenas nas operações do sistema. No entanto, existe um grau de detalhamento técnico que é especificado ao modelo. Com isso, tem-se a abordagem de operações que não mudarão em uma possível mudança de plataforma, retratando assim um aspecto de portabilidade. Nessa fase faz-se o uso de linguagens de modelagem genéricas ou uma linguagem específica de domínio para o requisito proposto.
- PSM: Especifica como os requisitos levantados junto ao PIM são utilizados em uma plataforma específica, aplicando critérios de uma ou outra plataforma. Nessa etapa os modelos são focados em cenários tecnológicos e como serão aplicados a determinadas ferramentas. Para que se alcance esse estágio, é aplicado o critério de transformação ou mapeamento entre modelos, onde a partir do modelo PIM é possível conceber um PSM ou diretamente o código. A aplicação da transformação ou mapeamento de modelos faz com que a partir de um modelo fonte seja alcançado um modelo alvo. Num mapeamento, é usado um modelo de plataforma que será responsável por agregar todos fatores técnicos, representando os módulos presentes na plataforma ou serviços que são utilizados pelo novo modelo.

No desenvolvimento orientado a modelo é necessário primeiramente que se tenha uma especificação formal do que se espera do sistema, devendo fazer um levantamento dos requisitos. A proposta do OMG através do MDA é que no estágio CIM, ou também denominado modelo de domínio, haja uma contextualização do cenário proposto, incorporando questões do ambiente de execução e deixando de lado as soluções técnicas para possíveis problemas. O modelo independente de computação é responsável por

abordar aspectos no nível do usuário, focando em aspectos de domínio e fornecendo detalhes funcionais, que são usados nas próximas etapas.

A próxima etapa faz uso de uma perspectiva voltada a quesitos de serviços oferecidos, havendo assim a necessidade de transformação do modelo CIM para PIM. Conforme descrito em Mignon (2007, p. 13), “[...] *transformação de modelos é um processo de conversão de um modelo em outro modelo do sistema [...]*” é através dessa transformação que se pode aperfeiçoar o CIM e gerar um PIM mais refinado com maior riqueza de detalhes do sistema. Geralmente trata-se de um processo manual em que se define através de uma linguagem de modelagem genérica os comportamentos que fazem parte do novo modelo. Através de sucessivas transformações, cada etapa do MDA é alcançada e a produção de artefatos de meta-modelagem auxiliará na próxima etapa, pois cada atividade complementa todo processo, sendo que as transformações são vias de mão dupla, podendo alcançar um CIM a partir de um PIM. A aplicação de sucessivas transformações faz gerar ao final o código do sistema, que se baseia em modelos construídos ao longo do processo.

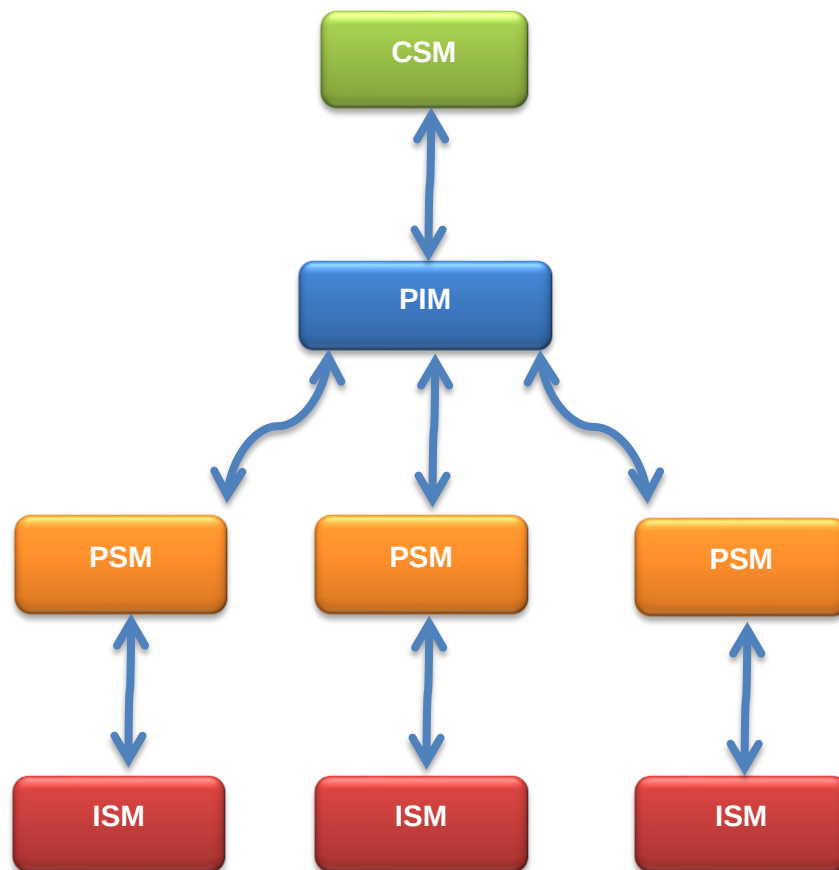
Para alcançar o estágio PIM, deve-se aplicar uma transformação que consiga extrair os detalhes do sistema em um nível sistêmico livre de perspectivas de plataforma, podendo haver definições genéricas de serviços que são suportados por qualquer plataforma. Nessa etapa é importante que haja uma avaliação do modelo, para que seja constatado que a modelagem possa ser aplicada a qualquer plataforma. Geralmente a representação do modelo é confeccionada em UML, já que essa ilustra bem os propósitos sistêmicos de uma forma genérica e livre de plataforma.

Com o PIM prontamente detalhado, aplica-se uma nova transformação onde o intuito é gerar um novo modelo, o PSM. A transformação possui como objetivo o detalhamento dos conceitos até essa etapa construídos, que são livres de plataforma, para uma perspectiva de plataforma de execução. Mapeamentos são utilizados para prover especificações que possam transformar o PIM em PSM, ou seja, o modelo resultante será focado em quesitos técnicos específicos. No mapeamento o modelo de plataforma é utilizado para que consiga reunir informações suficientes quanto aos critérios técnicos, detalhando os diferentes tipos de elementos da plataforma. O modelo finalizado interage fornecendo meios tecnológicos para a abordagem de ferramentas, que fazem o tratamento do modelo.

Com o PSM construído é possível aplicar duas abordagens: gerar um novo PSM ou praticar a geração de código. A primeira abordagem trata-se de um refinamento ou revisão dos detalhamentos da plataforma e visa construir um modelo focado em algum aspecto em especial. A segunda simplesmente faz uso do modelo para geração de código, entrando em uma quarta etapa do processo, o ISM (*Implementation Specific Model*), que na verdade é a transformação da especificação sistêmica em código fonte. Na transição PSM – ISM há uma

transformação interlinguagem, onde o modelo passa a ser concretizado e concebido, transitando entre a linguagem de modelagem para a linguagem específica de plataforma.

A **Figura 1** resume as transições ocorridas em cada etapa do MDD, e ilustra que todas transformações são vias de mão dupla, já que a partir de uma especificação técnica é possível alcançar uma especificação em modelo alto nível livre de plataforma.



**Figura 1** - Fluxo do MDD.

### 3 LINGUAGEM ESPECÍFICA DE DOMÍNIO

O foco do MDA está no uso da UML que é uma linguagem de modelagem de propósito geral, uma vez que essa possui uma grande capacidade em retratar os diversos cenários, como: perspectivas na área da medicina, telecomunicação, educação, finanças entre outras diversas. Por ser uma linguagem de propósito geral, a UML acaba por não englobar aspectos que são específicos de uma determinada abordagem, porque estaria integrada com situações mais gerais no escopo do ambiente (MIGNON, 2007, p. 47).

Em vista de tal restrição da UML são propostas as linguagens específicas de domínio ou DSL's, que possuem a capacidade de descrever cenários com maior exatidão e focar em situações específicas, capturando assim a semântica do domínio. O termo domínio é um pouco vago, mas pode ser encarado como a abordagem de qualquer elemento, cenário ou sistema para um determinado assunto.

As linguagens específicas de domínio mais conhecidas são: APT para controle numérico e a BNF (*Backus-Naur Form*) que é a linguagem para especificação de gramáticas (LUCREDIO, 2009, p. 54). As DSL's podem se apresentar de dois modos: textual ou visual. O primeiro modo permite a especificação do domínio na forma de linguagens programáticas, que serão processadas pela máquina, podendo assim redigir programas. O segundo trata-se da construção de diagramas e modelos que possuem um tratamento especial na abordagem focal.

As DSL's são formadas por três elementos: sintaxe abstrata, concreta e semântica, onde a sintaxe abstrata dita os conceitos do domínio, as restrições e relacionamentos que serão propostos através da linguagem (LUCREDIO, 2009, p. 55). Um exemplo claro são os meta-modelos que definem as estruturas que irão compor os modelos, ou seja, o MOF pode ser encarado como uma sintaxe abstrata do modelo UML. Na sintaxe concreta há a preocupação em empregar estruturas que possam ser assimiladas facilmente na linguagem: símbolos ou ícones gráficos, que possuem atributos como cor, posição ou tamanho. A semântica se preocupa no correto significado dos itens presentes na sintaxe abstrata, sendo que no MDD ela possui quatro categorias: denotativa, operacional, tradutiva e pragmática.

Um dos itens mais importantes no MDD trata-se da abordagem tradutiva que é retratada pela semântica de uma linguagem específica de domínio, em que faz com que haja a tradução de um modelo para uma linguagem. Um exemplo são as ferramentas NetBeans e Visual Studio que fazem uso de um domínio GUI (*Graphical User Interface*), oferecendo interação com o usuário. Cada interação é responsável por gerar código XML (*Extensible Markup Language*) que define cada estado ou característica dos objetos, para que posteriormente possam ser utilizados por linguagens de propósito geral como: C#, C++, Java, UML entre outras.

## 4 LINGUAGEM DE RESTRIÇÃO DE OBJETOS

Como comentado em seções anteriores, a UML é muito útil para perspectivas de modelagem de propósito geral, no entanto, algumas abordagens adicionais necessitam ser inseridas na modelagem para que se possa de fato retratar o cenário o mais próximo possível do que será proposto.

O surgimento da linguagem de restrição de objetos ou OCL contribui para que a modelagem possa atribuir comportamentos que retratem o mais próximo da realidade algumas características dos objetos que circundam o sistema, ou seja, torná-los mais restritos e complementares às suas características.

A OCL é uma linguagem formal puramente declarativa, que faz uso de expressões ou anotações textuais (MIGNON, 2007, p. 24). Incorpora restrições aos objetos modelados. É capaz de definir uma linguagem que especifica consultas, restrições e operações aos objetos. Pelo fato da UML não possuir um recurso que proponha um nível elevado de expressividade de forma simples, o OMG incorporou a OCL na UML desde a versão 1.1. Os benefícios propiciaram a essa linguagem de restrição, a facilidade durante o processo de desenvolvimento, oferecendo recursos que aumentam o grau de expressividade das modelagens durante fases de análise e projeto. A linguagem de restrição de objetos é muito dependente dos tipos presentes na UML, já que as expressões restritivas são montadas com base em uma modelagem, devendo usar algum aspecto da UML.

A linguagem de restrição possui características que fazem com que se torne um instrumento de grande valor agregado (MIGNON, 2007, p. 24):

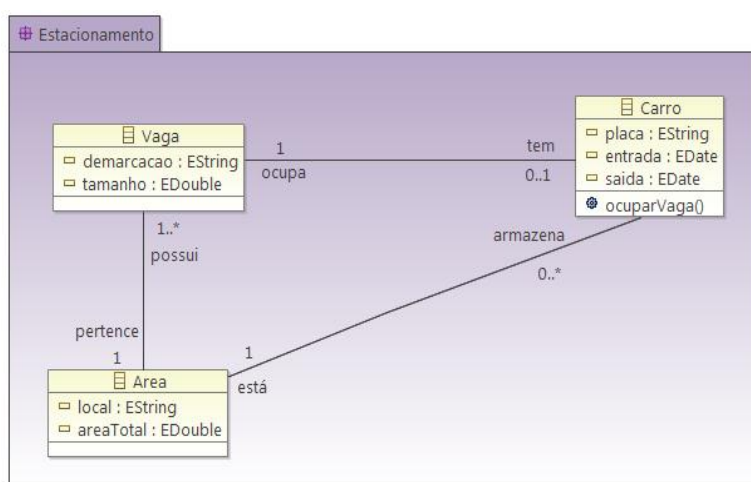
- Por ser uma linguagem puramente declarativa, suas expressões não impactam na codificação quando avaliadas, pois essas retornam apenas um valor;
- Trata-se de uma linguagem fortemente tipada, ou seja, cada expressão possui um tipo e as construções dessas devem respeitar as regras da linguagem;
- Os tipos tratados numa expressão são vinculados aos tipos presentes na modelagem UML, sendo que cada expressão é escrita num contexto, com classificadores, associações e generalizações;
- A OCL não se trata de uma linguagem de programação, mas sim de uma linguagem textual, não sendo possível assim incluir lógica de programação ou fluxo de controle;
- Oferece a disponibilidade de navegar entre os relacionamentos, acessando atributos de outras classes, ou ainda acessar relacionamentos que estão presentes em outras classes relacionadas, podendo também acessar ou criar expressões num auto relacionamento;

- E o ponto mais importante, a OCL consegue expressar ações restritivas que veem a oferecer um complemento no comportamento dos modelos.

#### 4.1 Restrições na modelagem

Para entender a aplicação da OCL, a seguir será descrito um caso a ser analisado diante de uma simples modelagem UML, posteriormente os casos restritivos serão retratados com o auxílio da linguagem.

Considere um estacionamento numa determinada área, esse local deve oferecer serviços para acomodar os veículos de seus clientes, inicialmente deve haver um levantamento de quantas vagas poderão ser oferecidas diante da área. Os carros são identificados de acordo com a numeração da placa, horários de entrada e saída são registrados para que haja o controle no momento do pagamento. Assim sendo, confecciona-se um diagrama conforme a **Figura 2**:



**Figura 2** - Diagrama de modelagem.

Através do modelo pode-se perceber que regras mais restritivas não estão presentes no diagrama, pois um diagrama de classes por si só não consegue reunir artefatos que venham a limitar e representar um objeto de forma específica, nesse momento é necessária a abordagem por OCL.

Analisando o diagrama é identificada a falta de certas regras, por exemplo, a modelagem transparece que é possível comportar um número ilimitado de vagas numa dada área. Com isso, é necessário acrescentar uma regra que restrinja o tamanho total das vagas, não podendo exceder a área total disponível do estabelecimento. Consequentemente a relação “possui” entre as classes “Área” e “Vaga” é afetada diretamente, visto que o número de carros a serem comportados está relacionado à quantidade de vagas

disponíveis. Outros aspectos importantes são: um carro não possuirá um horário de saída inferior ao horário de entrada e existe a obrigatoriedade de se fazer a identificação do veículo através de uma placa de identificação. Todas essas regras não podem ser expressas simplesmente através de um diagrama UML, para isso é necessário construir expressões ou consultas que realizem tais restrições, conforme pode ser constatado através da **Figura 3**.

```
Context Area inv:
self.areaTotal > self.possui.tamanho->sum()

Context Carro inv:
self.saida > self.entrada

Context Carro inv:
self.placa->notEmpty()
```

**Figura 3** - Expressões em OCL.

As expressões acima são anexas ao diagrama de modelagem da **Figura 2**, e essas são responsáveis por incluir restrições que influenciam no comportamento dos objetos das classes “Area” e “Vaga”. A primeira delas expressa que no contexto da classe “Area”, o atributo “areaTotal” deve ser maior que a soma de todos atributos “tamanho”, identificados através do relacionamento “possui”.

A expressão “self” incorpora que o atributo ou operação a ser referenciado se trata de uma instância do contexto (*context*) (OMG, 2006). No primeiro exemplo, nota-se que o contexto está no nível da classe “Area”. O atributo “areaTotal” está incorporado a uma instância da classe do contexto, sendo que o atributo deve ser maior que o “tamanho”, presente no relacionamento “possui”.

A linguagem OCL possui a propriedade de oferecer navegabilidade entre as classes através dos relacionamentos existentes (OMG, 2006). No primeiro exemplo da **Figura 3** a evidência pode ser constatada, já que através do relacionamento “possui” é possível acessar as instâncias relacionadas com os seus devidos atributos.

Um relacionamento pode resultar em um objeto ou numa coleção de objetos dependendo da cardinalidade de cada relação, sendo que relações do tipo 0..1 (monovaloradas) devolvem uma instância da classe. As multivaloradas, como 0..\* ou 1..\*, devolvem uma coleção de itens que estão integrados ao relacionamento.

As operações podem ser atribuídas às coleções, sendo indicadas pelo operador seta “->” (OMG, 2006). Nos exemplos anteriores a operação *sum()* e *notEmpty()* devolvem valores que serão utilizados para avaliar cada expressão.

No segundo exemplo há uma validação nos critérios de permanência do veículo, pois é identificada a necessidade de validação dos dados, onde o momento de entrada do

veículo não poderá ser superior ao momento de saída. Para o tratamento dessa validação, a expressão identifica os atributos “entrada” e “saida”, que fazem parte da classe (contexto) “Carro”.

O terceiro simplesmente impõe uma segunda restrição aos objetos da classe “Carro”, visto que a identificação dos veículos no estacionamento é feita através da placa veicular. Os dados da placa não podem estar ausentes no sistema, a operação *notEmpty()* é aplicada ao atributo “placa” validando se o atributo está presente.

## 4.2 Abordagem OCL e o padrão MDA

O OMG adota a OCL como um dos padrões utilizados pelo MDA, por ser uma linguagem bem definida e declarativa, capaz de integrar modelos UML e MOF.

Em Mignon (2007, p. 23), o processo de desenvolvimento do MDA deve estar vinculado a diretrizes capazes de construir um modelo que seja legível ao computador, ou seja, fazer com que o modelo seja claro na exposição de restrições e conceitos sistêmicos, para que assim possa auxiliar na automatização do processo de transformação de modelos. Por esses motivos acredita-se que a OCL é um ponto chave no desenvolvimento, conseguindo expor características e comportamentos próprios a cada entidade do contexto.

A linguagem de restrição, na MDA, não reside somente na aplicação a entidades restritas de um modelo específico, mas também está inserida na construção de meta-modelos, para auxiliar na construção de regras de cada elemento da meta-modelagem. Por exemplo, uma restrição pode ser usada para estabelecer a condição de que uma interface não possua atributos declarados em sua estrutura (MIGNON, 2007, p. 26), tal restrição pode ser codificada conforme **Figura 4**.

```
-- Interface não pode conter atributos
Context Interface inv:
features->select(f | f.ocllsKindOf(Attribute))->isEmpty()
```

**Figura 4** - Restrição para construção de interface (MIGNON, 2007, p. 26).

Transformações possibilitam que um modelo possa ser convertido num outro modelo em outra linguagem, no entanto, é necessário que haja um mapeamento entre o modelo fonte e o modelo alvo. Conceitos das linguagens dos dois modelos, em seus respectivos meta-modelos, são usados para que se possa aplicar as transformações.

As restrições favorecem o processo de mapeamento, estabelecendo através da OCL com que haja uma abordagem precisa e não ambígua, pois assim é possível tratar elementos do modelo fonte com os do modelo alvo.

## 5 TRANSFORMAÇÃO COM ACCELEO

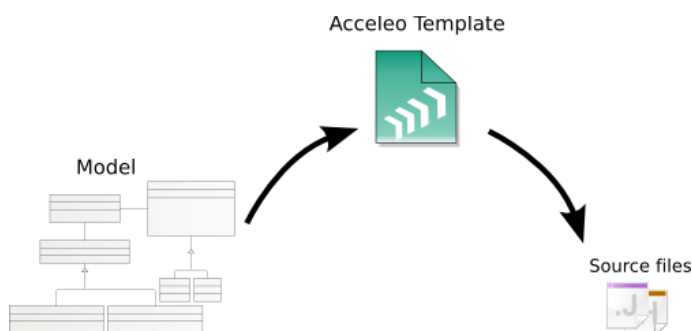
Em busca de seguir os objetivos propostos pelo MDA, organizações de software decidiram propor ferramentas que sigam os padrões do OMG. Com isso, a instituição francesa Obeo idealizou o projeto *open source* Acceleo, que combina propostas da MDA e destaca-se pela eficiência em transformações *model to text*, pois é uma ferramenta que consegue fazer geração de código a partir de *templates* pré-definidos.

O Acceleo por meio da transformação *model to text*, consegue transformar os conceitos presentes no modelo, sendo que os *templates* definem a estrutura de código a ser gerado. A ferramenta de geração de código é adotada por processos MDD e MDE (*Model Driven Engineering*), visto que facilita o processo de desenvolvimento baseado em modelo e possui vantagens como: customização na confecção de *templates*, interoperabilidade, facilidade de aprendizagem, gestão de rastreabilidade e processo de transformação incremental.

### 5.1 Objetivo

O principal objetivo do Acceleo é reunir as boas práticas de modelagem propostas pelo OMG e facilitar a produção de softwares orientados a modelos, proporcionando assim a automatização nos processos de transformação e, a nível de desenvolvimento, oferecer a geração automatizada de código.

Os processos para geração de código seguem uma linha de desenvolvimento, com a presença de etapas que oferecem: prototipação do *template*, validação através dos critérios de qualidade adotados, conversão em um módulo gerador que encapsula os *templates* de transformação. Partindo do pressuposto que o modelo é claro e bem definido, a confecção do *template* considera o modelo a ser transformado, onde se deve definir um algoritmo capaz de vincular os recursos do gerador de código com as especificações do modelo. O fluxo de geração do Acceleo pode ser considerado conforme **Figura 5**.



**Figura 5** - Fluxo de geração (Acceleo, 2011).

## 5.2 Recursos

Um dos destaques do Acceleo é, sem dúvida, a geração de código, em que a partir de um modelo UML2, *Ecore* ou qualquer outro meta-modelo que esteja de acordo com os padrões MDA, seja possível transformá-lo a ponto de auxiliar no processo de desenvolvimento. O principal fator é possibilitar a construção de uma documentação, que venha a automatizar o processo de criação ou atualização do software.

Os principais recursos do Acceleo estão vinculados aos formatos de modelos que são suportados e integrados, obtendo assim diagramas que podem ser confeccionados em outras ferramentas de modelagem como: EMF, GMF (*Graphical Modeling Framework*), Topcased, Reuseware entre outros. Essa integração é resultado dos padrões adotados a partir do MDA, onde o XMI (*XML Metadata Interchange*) é o maior responsável pela universalização entre modelagens.

O XMI é um padrão de serialização de modelos adotado pelo MDA, que utiliza o formato XML para o intercâmbio de dados entre ferramentas de modelagem (LUCREDIO, 2009, p. 44). Possibilita a definição de documento capaz de elencar dados e seus metadados. Esse padrão de serialização oferece benefícios que fazem com que a modelagem ofereça um processo de unificação para interpretação de dados, são eles:

- Possibilitar a existência de um intercâmbio de objetos através da serialização, utilizando uma linguagem universalmente conhecida, o XML;
- Proporcionar a conversão de modelos para XML e vice-versa, desde que estejam de acordo com os padrões da meta-modelagem MOF;
- Confeccionar documento XML que pode ser facilmente exportado ou importado por diferentes ferramentas;
- Disponibilizar o manuseio de dados e meta-dados;
- Permitir a interoperabilidade dos modelos entre diversos tipos de ferramentas e repositórios.

O critério de interoperabilidade é reflexo da padronização na serialização de modelos. Um modelo pode ser especificado em UML a partir de uma ferramenta de modelagem, em seguida ser exportado para XMI e posteriormente utilizado numa ferramenta de transformação de modelos. Com isso, é possível aplicar transformações e gerar o código fonte do sistema.

Para exemplificar os recursos, tome como base o diagrama de classes apresentado na **Figura 2**, essa modelagem foi construída em um projeto EMF, mas é possível de ser incorporado a um projeto Acceleo e confeccionar um *template* que faça a geração de código. Como esclarecido anteriormente, o XMI é o padrão adotado para serialização de

modelos no MDA, sendo que o diagrama de classes possui uma versão serializável conforme pode ser constatado na **Figura 6**.

```
<?xml version="1.0" encoding="UTF-8"?>
<notation:Diagram xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
  xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
  xmlns:notation="http://www.eclipse.org/gmf/runtime/1.0.2/notation"
  xmi:id="_OxO_QAGvEeGFa-1kacRRbg" type="EcoreTools" name="EstacionDiagram">
  <children xmi:type="notation:Node" xmi:id="_fSLCAAGvEeGFa-1kacRRbg" type="1002">
    <children xmi:type="notation:Node" xmi:id="_fSNeQAGvEeGFa-1kacRRbg" type="4006"/>
    <children xmi:type="notation:Node" xmi:id="_fSOFUAGvEeGFa-1kacRRbg" type="5003">
      <children xmi:type="notation:Node" xmi:id="_jKEgoAGvEeGFa-1kacRRbg" type="2003">
        <children xmi:type="notation:Node" xmi:id="_jKG84AGvEeGFa-1kacRRbg" type="4002"/>
        <children xmi:type="notation:Node" xmi:id="_jKIyEAGvEeGFa-1kacRRbg" type="5004">
          <children xmi:type="notation:Node" xmi:id="_oo2dgAGvEeGFa-1kacRRbg" type="2001">
            <element xmi:type="ecore:EAttribute" href="EstacionDiagram.ecore#//Estacionamento/Vaga/demarcacao"/>
            <layoutConstraint xmi:type="notation:Location" xmi:id="_oo2dgQGvEeGFa-1kacRRbg"/>
          </children>
        <children xmi:type="notation:Node" xmi:id="_8HmHMAJJEEGuUZTnpVie6g" type="2001">
          <element xmi:type="ecore:EAttribute" href="EstacionDiagram.ecore#//Estacionamento/Vaga/tamanho"/>
          <layoutConstraint xmi:type="notation:Location" xmi:id="_8HmHMQJJEEGuUZTnpVie6g"/>
        </children>
      </children>
    </children>
  </children>
```

**Figura 6** - Representação de um diagrama em XML.

O código XML é gerado a partir da elaboração do modelo de sistema construído pelo usuário, ou seja, a partir do módulo gráfico é gerado o código que será interpretado por outras ferramentas que fazem uso dessa mesma modelagem, nesse quesito pode-se afirmar que há uma abordagem GUI. A metodologia que adota uma interação diante do usuário com diagramas, proporcionando a geração de código de forma automatizada, pode ser considerada um instrumento de aplicação dos princípios do MDA.

Um passo importante para a geração de código no Acceleo é a definição do *template*, que servirá para executar a transformação *model to text*. Os *templates* são scripts que reúnem a especificação do modelo alvo (código a ser gerado), com códigos de acesso ao modelo de origem que geram a estrutura dinâmica.

Os apêndices B e C ilustram o código de um *template* que faz a geração de classes Java *bean's* e de classes da camada de acesso a dados. Algumas definições do Acceleo são apresentadas para a identificação do artefato a ser gerado. Os delimitadores `<%` e `%>` são utilizados para declararem inicializações ou atributos do *template*, e também comportam a parte dinâmica da programação com fluxos de decisão, repetição, entre outros. A *tag metamodel* sinaliza qual padrão de modelagem é utilizado pelo modelo fonte. Os atributos do documento estão presentes na *tag script*, sendo que *type* determina o tipo de diagrama fonte e *file* fornece o nome e extensão do arquivo a ser gerado.

Aplicado o gerador são obtidas classes na extensão Java, que de acordo com o modelo fonte, apresentarão os métodos *getters/setters* de forma automatizada.

## 6 MODELAGEM COM EMF

O projeto EMF é um *framework* de modelagem incorporado ao Eclipse, que oferece facilidades na geração de código e favorece construções para aplicações baseadas em modelos estruturados. A partir de uma especificação de modelo escrita em XMI, o EMF fornece ferramentas e suporte para produção de um conjunto de classes Java, juntamente com um conjunto de classes que permitem a visualização baseada em comandos de edição do modelo (ECLIPSE, 2011).

O *framework* oferece uma unificação entre três importantes tecnologias que o compõem: Java, XML e UML. A linguagem Java oferece os mecanismos para edição e customização dos modelos a serem gerados, visto que as gerações de código produzem as chamadas *class adapters*. O XML compõe o formato de serialização de modelos para oferecer interoperabilidade entre as ferramentas que suportam modelagens construídas em EMF. A UML é a linguagem de modelagem padrão, recomendada pelo MDA, para atender as especificações dos domínios em níveis de abstração, favorecendo o desenvolvimento do modelo.

### 6.1 Objetivo

O principal objetivo está em proporcionar com que haja a especificação de modelos e geração de código, para que possa propiciar a interoperabilidade entre os módulos que compõem o EMF. A ideia principal é facilitar a modelagem em um aspecto menos geral e num nível de abstração menor que o oferecido pelo UML, fazendo com que o desenvolvedor consiga integrar o modelo de especificação diante de outras ferramentas do Eclipse.

A interoperabilidade vem da necessidade em oferecer um nível de automatização que favoreça a construção de artefatos (diagramas ou códigos), que venham a ser utilizados durante a fase de especificação do domínio. Esses mesmos artefatos oferecem durante a fase de desenvolvimento a possibilidade de geração de código, por exemplo, extrair as classes Java que compõem o negócio.

Mais do que oferecer condições para a construção de modelos, o EMF visa facilitar a integração de modelos diante de outras ferramentas, pois a ferramenta oferece integrações com a maioria das ferramentas de modelagem do Eclipse. Os modelos construídos através do EMF podem ser refinados por outros mecanismos de modelagem, por exemplo, serem incorporados ao Acceleo e aplicar transformações *model to text*.

## 6.2 Recursos

O EMF se destaca por oferecer diversos mecanismos que possam gerenciar a criação de uma modelagem estruturada, que pode ser incorporada a outros recursos do Eclipse. Esses recursos estão ligados às estruturas construídas para suportar os ciclos de modelagem e favorecer a integração de soluções, diante dos formatos oferecidos pelo *framework*. Os recursos mais importantes abordados durante esse trabalho são:

- Unificação das principais tecnologias de modelagem: Java, XML e UML;
- Oferecer mecanismos de edição de modelos através do EMF.Edit;
- Oferecer uma serialização no padrão XMI, para se adequar às ferramentas que seguem o padrão do MDA.

Um dos pontos cruciais para o embasamento do EMF é a unificação que é apresentada para o gerenciamento dos modelos, onde as tecnologias Java, XML e UML propiciam uma integração para a geração dos artefatos de modelagem.

Uma das formas mais usuais na definição de modelos é a elaboração de representações gráficas, ou seja, construir diagramas UML que fazem referência a um domínio. A partir da especificação formal é possível aplicar a geração de código, criando classes Java que refletem a especificação presente no diagrama. No entanto, a unificação proporciona com que a partir de uma especificação construída em uma das três tecnologias, seja possível alcançar as demais. Caso o usuário decida por uma especificação em XML *Schema*, pode-se dizer que facilmente obterá a mesma especificação em Java e UML.

O suporte a arquivos XML em formato XSD (*XML Schema Definition*) faz com que se possa construir uma especificação numa estrutura textual, que permita a elaboração de modelos. Para ilustrar esse recurso, atente à seguinte especificação presente na **Figura 7**. Ela contém a representação estrutural de um estabelecimento comercial, onde há relações existentes entre fornecedores, produtos e clientes.

```

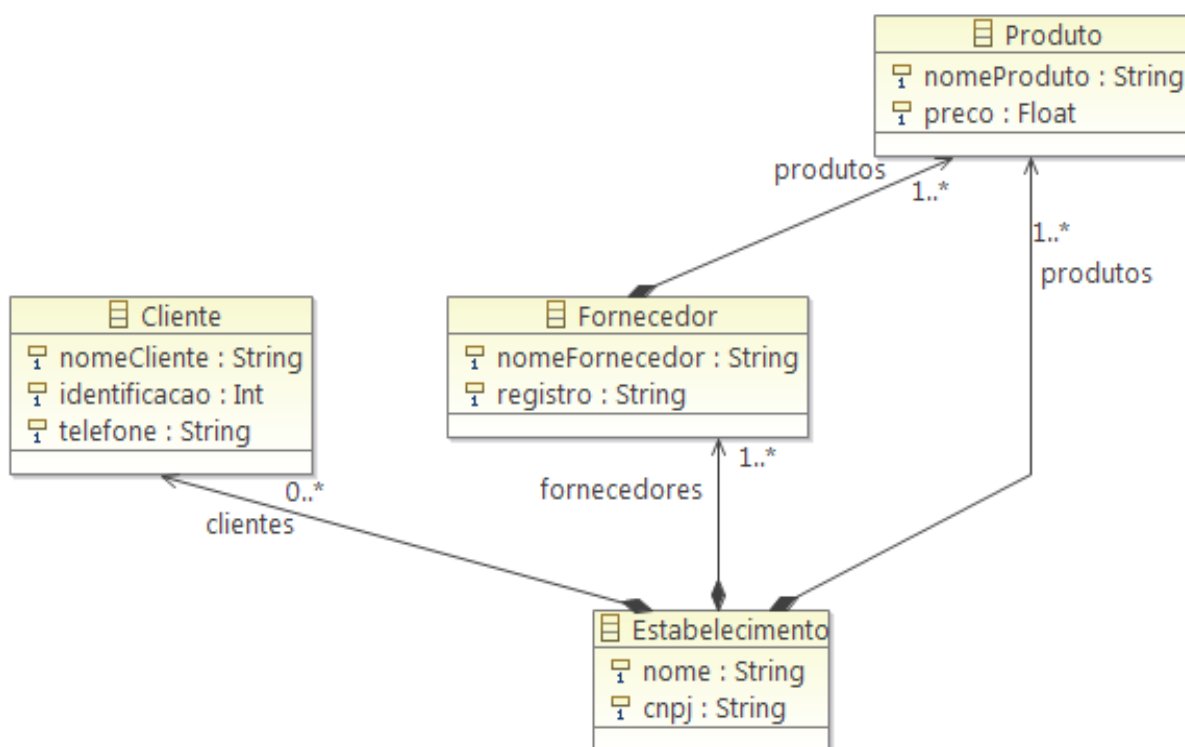
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.example.com/Estabelecimento"
  xmlns:EST="http://www.example.com/Estabelecimento">
  <xsd:complexType name="Estabelecimento">
    <xsd:sequence>
      <xsd:element name="nome" type="xsd:string"/>
      <xsd:element name="cnpj" type="xsd:string"/>
      <xsd:element name="clientes" type="EST:Cliente" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="fornecedores" type="EST:Fornecedor" minOccurs="1" maxOccurs="unbounded"/>
      <xsd:element name="produtos" type="EST:Produto" minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="Fornecedor">
    <xsd:sequence>
      <xsd:element name="nomeFornecedor" type="xsd:string"/>
      <xsd:element name="registro" type="xsd:string"/>
      <xsd:element name="produtos" type="EST:Produto" minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="Produto">
    <xsd:sequence>
      <xsd:element name="nomeProduto" type="xsd:string"/>
      <xsd:element name="preco" type="xsd:float"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="Cliente">
    <xsd:sequence>
      <xsd:element name="nomeCliente" type="xsd:string"/>
      <xsd:element name="identificacao" type="xsd:int"/>
      <xsd:element name="telefone" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

```

**Figura 7** - Especificação em formato XSD.

A esquematização retrata um estabelecimento que possui clientes e fornecedores vinculados ao seu fluxo de operação. Os fornecedores oferecem produtos que são de consumo do estabelecimento, onde esse necessita catalogar os produtos de cada fornecedor. Mas também é necessário ter o conhecimento de quais produtos estão presentes no estoque.

O intuito dessa análise é fazer com que a partir da definição do domínio em XML, obtenha o diagrama de classes UML que ilustra bem todos relacionamentos e interações que compõem o cenário do estabelecimento. Por meio de um projeto “EMF Project”, consegue-se informar que o tipo de modelo a ser gerado se baseia num XML Schema, referenciando o arquivo da **Figura 7**. Após a criação do projeto é criada uma estrutura de arquivos, que apresenta visões de modelagem nos formatos *ecore* e *genmodel*, sendo que o formato *ecore* será o responsável por construir o diagrama de classes UML. Através da criação de um arquivo “Ecore Diagram” pode-se apontar para o “Ecore Model”, obtendo assim um diagrama de classes conforme a **Figura 8**. Essa representação conclui a construção do diagrama de classes por meio de uma especificação em XML/XSD.



**Figura 8** - Diagrama de classes UML com base em um XML.

O EMF.Edit é um recurso que confere maior dinamismo ao EMF na elaboração de modelos. Trata-se de um *framework* estendido a partir do EMF Core, que propicia a construção de modelos mais adequados e elaborados, adicionando suporte à visualização e edição do modelo. A capacidade de re-gerar o modelo, após alguma modificação, faz com que a construção inicial seja mantida e mudanças ao longo do desenvolvimento possam ser aplicadas de maneira contínua. Se a inclusão de código ao modelo criado alterar alguma estrutura ou refletir nos relacionamentos existentes, é necessário aplicar a geração de código novamente, para que reflita nos demais itens de código e no modelo estruturado (SCHOEPPING, 2007, p. 41).

Diante das possibilidades de modelagem do EMF, são oferecidos recursos para proporcionar flexibilidade aos usuários no momento da definição de modelos. A utilização do XMI faz com que a definição do modelo possa ser serializada e incorporada a outras abordagens UML junto ao EMF. O *framework* possibilita a definição do XMI por meio do modelo, diante das seguintes circunstâncias (SCHOEPPING, 2007, p. 42):

- Com o auxílio de um editor XML, elaborando um documento XMI diretamente;
- Anotações em interfaces Java para registrar as propriedades do modelo;

- Exportação de um documento XML através de ferramentas de modelagem, como o *Rational Rose*;
- Descrição de um modelo com o uso de um documento XML *Schema*, concebendo assim a forma serializada da abstração.

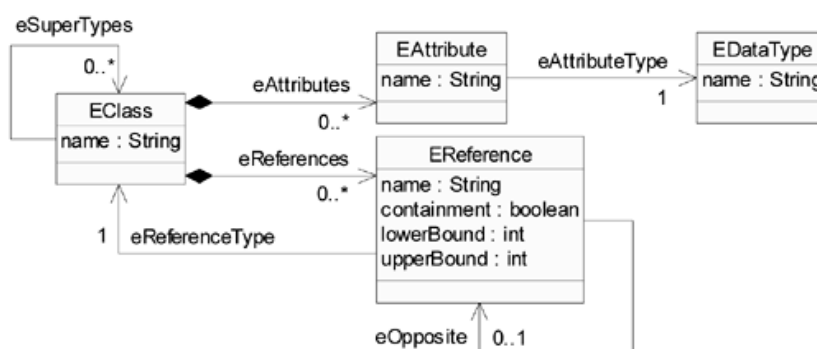
Com a especialização do XML, a incorporação de modelos a qualquer outra ferramenta, que adote os padrões do MDA, faz com que a integração possa ser contínua e rotineiramente atualizada. O benefício desse recurso é constatado ao favorecer a extração do modelo após qualquer alteração feita na estrutura. Sendo assim, esse padrão possibilita com que o modelo possua interoperabilidade diante das ferramentas, propiciando uma maior produtividade durante a construção ou manutenção do modelo.

### 6.3 O meta-modelo *ECORE*

O meta-modelo em EMF é designado por *Ecore*. Os princípios do *Ecore* estão ligados à meta-modelagem MOF e UML, visando facilitar as implementações Java, já que toda estrutura do meta-modelo é construída a partir de conceitos de orientação a objeto.

Os modelos podem ser definidos por meio da programação executada com o auxílio do ambiente de desenvolvimento, ou ao carregar arquivos serializados que estejam de acordo com o meta-modelo (BUDINSKY, 2004, p. 120). A primeira abordagem é tratada por meio da ferramenta Eclipse, usando os itens nativos do EMF. A segunda é proporcionada ao carregar um arquivo serializado no formato *Ecore*, sendo que há a possibilidade de converter *Ecore* em outros formatos de meta-modelagem. Os recursos do EMF possuem a capacidade de gerenciar o meta-modelo, podendo converter um modelo que está definido em um meta-modelo para outro.

A estrutura do meta-modelo é representada pela **Figura 9**, que destaca os itens principais presentes na modelagem e os relacionamentos responsáveis por comportar cada elemento para as abstrações.

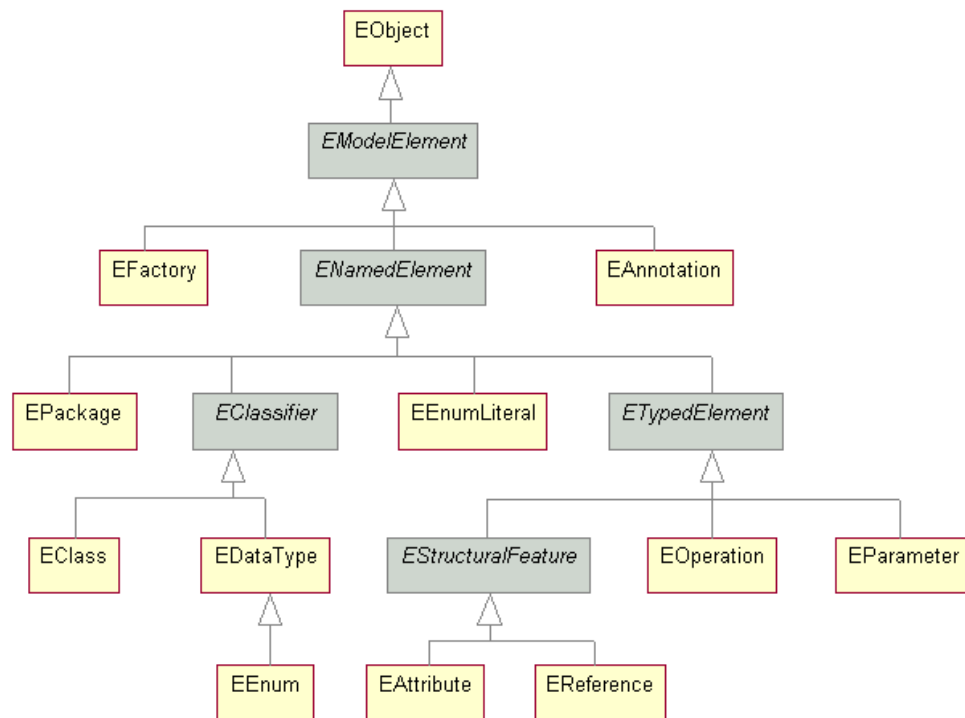


**Figura 9** - Relacionamento dos itens do *Ecore* (BUDINSKY, 2004, p. 123).

A figura destaca quatro objetos principais, por meio das classes:

- *EClass*: define as classes dos objetos. Possuem o atributo “name” e podem conter um número variado de relacionamentos ou atributos. O suporte a hierarquias é confeccionado por meio do relacionamento “eSuperTypes”, que pode referenciar outras classes, compondo uma hierarquia estruturada do *Ecore*;
- *EAttribute*: define os atributos que são comportados pela estrutura de modelagem. São identificados por nome e possuem um tipo de dados;
- *EDatatype*: define os tipos de atributos, representando tipos primitivos ou tipos de objeto de dados definidos em Java. Os tipos de dados são identificados por nomes;
- *EReference*: define as referências existentes entre classes. As referências são identificadas por nome e possuem um tipo. Porém, esse tipo deve ser um objeto “*EClass*” ao fim da associação. Caso a associação propicie a navegabilidade, a relação oposta terá uma referência correspondente. Os atributos “lowerBound” e “upperBound” especificam a multiplicidade no relacionamento. Por fim, uma referência pode ser usada para representar um tipo de associação mais restrita: “containment”.

A hierarquia de classes da **Figura 10** completa a definição do meta-modelo *Ecore*. A interface principal responsável por comportar todas definições dos objetos é o *EObject*, sendo conceitualmente equivalente a classe *object* em Java. Essa interface se torna um item obrigatório no momento da implementação, pois todos objetos de modelagem devem implementar seus métodos. A hierarquia também contém o *EPackage*, que possui detalhes sobre classes (*EClass*) e os tipos de dados (*EDatatype*). O elemento *EClass* é uma abstração que representa as classes modeladas, especificando atributos e referências para os objetos. O *EAttribute* reúne simplesmente a representação do dado, especificado por meio de um *EDatatype*. As referências são representadas pelo *EReference*, tratando de associações entre classes. Finalmente, o *EFactory* reúne os métodos que são necessários para a criação de elementos do modelo (BUDINSKY, 2004, p. 135).



**Figura 10** - Hierarquia de classes no *Ecore* (Eclipse, 2006).

## 7 METODOLOGIA

A metodologia adotada segue diretrizes do MDA e MDD para a construção da modelagem. Os padrões de especificação do OMG no MDA fornecem meios para que a construção de modelos seja tratada em níveis, ou seja, partir de um modelo inicial alto nível e, progressivamente adaptá-lo aos aspectos técnicos. Com isso, é gerado ao fim do processo um artefato com nível menor de abstração.

Para que a construção dos artefatos de modelagem siga as diretrizes e métodos estabelecidos pelo MDA, as seguintes etapas são adotadas no desenvolvimento orientado a modelos:

1. Uma especificação do domínio deve ser produzida, para que detalhe cenários na visão do usuário. A produção de uma especificação alto nível favorece o entendimento do sistema em uma primeira abordagem, e fornece os insumos de modelagem às etapas posteriores do processo. Esses procedimentos de especificação fazem parte do CIM.
2. Para a fase do PIM utiliza-se a especificação construída pelo CIM, sendo que um modelo livre de critérios de plataforma é confeccionado, ilustrando as interações que cada objeto do domínio pratica. No próximo estágio, o PIM passa a dar origem a um ou vários PSM's, que complementam o nível de abstração do sistema.
3. A próxima etapa deve construir artefatos técnicos, que exponham características tecnológicas, aproximando o requisito funcional da fase de desenvolvimento. A visão PSM tende a melhorar a modelagem da fase anterior, oferecendo um nível menor de abstração comparado ao PIM. Nessa fase os levantamentos técnicos devem tornar o modelo eficaz, construindo uma abordagem clara ao nível da plataforma.
4. O estágio seguinte possui um aspecto prático para a geração de código. Propicia o uso de ferramentas que automatizem a geração de código, produzindo o código que irá compor o sistema. A partir desse estágio, os artefatos produzidos possuem um menor nível de abstração, quando comparados às demais fases.
5. O uso de ferramentas de modelagem proporciona a automatização de processos e enriquece a produção de modelos. O EMF ajuda na elaboração de modelos serializáveis, para integrá-los a outros mecanismos ao decorrer do processo.
6. O Acceleo pode ser um aliado na confecção de *templates*, onde o modelo de especificação é a entrada e, por meio dos *templates* obtém-se o código fonte. Os *templates* são confeccionados de modo a atingir o domínio específico,



## 8 ESTUDO DE CASO

O estudo de caso visa expor as aplicações das ferramentas e conceitos apresentados nesse trabalho. As abstrações referentes a cada cenário e modelo são apresentadas de forma a situar o entendimento, oferecendo diretrizes para o bom entendimento diante das abordagens de modelagem.

A abordagem diante do desenvolvimento orientado a modelo é ilustrada para que se possa identificar e detalhar cada etapa da modelagem. A construção de artefatos nas fases CIM, PIM, PSM e ISM, detalham o processo de desenvolvimento e fornecem fundamentos para análises do domínio.

O uso de ferramentas de modelagem auxilia na construção do domínio, oferece subsídios para a automatização de processos e, faz com que os conceitos do MDA sejam concretizados.

### 8.1 Construção do CIM

A título de ilustração sobre as etapas de modelagem, o estudo de caso apresenta um domínio que fornecerá subsídios para as análises de modelagem. Tal domínio trata-se de uma biofábrica, onde manuseios são executados para a confecção de plantas melhoradas geneticamente.

Para a construção do CIM desse cenário, deve-se reunir detalhes a nível de usuário, ou seja, preparar um modelo que reflita os requisitos sistêmicos. A especificação a seguir é um artefato do CIM:

“Uma Biofábrica necessita automatizar os processos de mapeamento de produção de suas mudas. A companhia consegue multiplicar a produção das mudas aplicando diversas medidas, que fazem o manuseamento genótipo de cada espécie. Deve-se identificar durante o processo a qual espécie pertence o item genético, onde e quando foi colhida, sendo que essas informações irão compor o lote. O processo é executado em diversos recipientes, com isso deve-se identificar cada recipiente nas execuções, juntamente deve-se ter o conhecimento dos funcionários que manipulam cada item genético. Os recipientes estão presentes em algumas fases que compõem o processo. Na fase pode ser necessário o manuseamento do organismo, de um recipiente para outro(s) conteúdo(s). Ao longo da produção um recipiente pode ser descartado por diversos motivos: quebra, contaminação por fungos e até mesmo necessidade de higienização”.

Com a especificação sistêmica, a construção de modelos diagramáticos é facilitada, visto que a visão do usuário fora concretizada em uma etapa alto nível. A especificação acima é um dos artefatos produzidos pelo CIM. O nível CIM não aborda detalhes técnicos, mas trata de quesitos da fase de concepção, oferecendo uma visão geral do sistema a ser construído.

## 8.2 Construção do PIM

O modelo independente de plataforma possui a especificação do domínio, considerando quesitos que não são específicos a uma tecnologia. O nível de abstração oferece detalhamento suficiente para que o domínio esteja contextualizado, e exponha as particularidades que serão utilizadas em modelos específicos.

Após a construção do CIM, aplica-se uma transformação considerando a especificação do cenário. O resultado obtido é um modelo em um nível de abstração menor, o PIM, conforme exposto pela **Figura 12**. Esse PIM reúne informações relevantes sobre o sistema, onde considera alguns detalhes técnicos que não são abordados em um modelo de abstração mais alto e próximo do usuário.

O modelo é construído em um meta-modelo que propicia bases para a definição de objetos. Esse nível de modelagem se diferencia pelo fato de apresentar algum grau de tecnologia, pois as classes apresentam seus atributos, os tipos primitivos de cada atributo e os construtores. A modelagem trata apenas de detalhes que são independentes de plataforma, visto que a orientação a objetos e os tipos primitivos compõem uma vasta área de plataformas, não ficando restrita a uma específica.

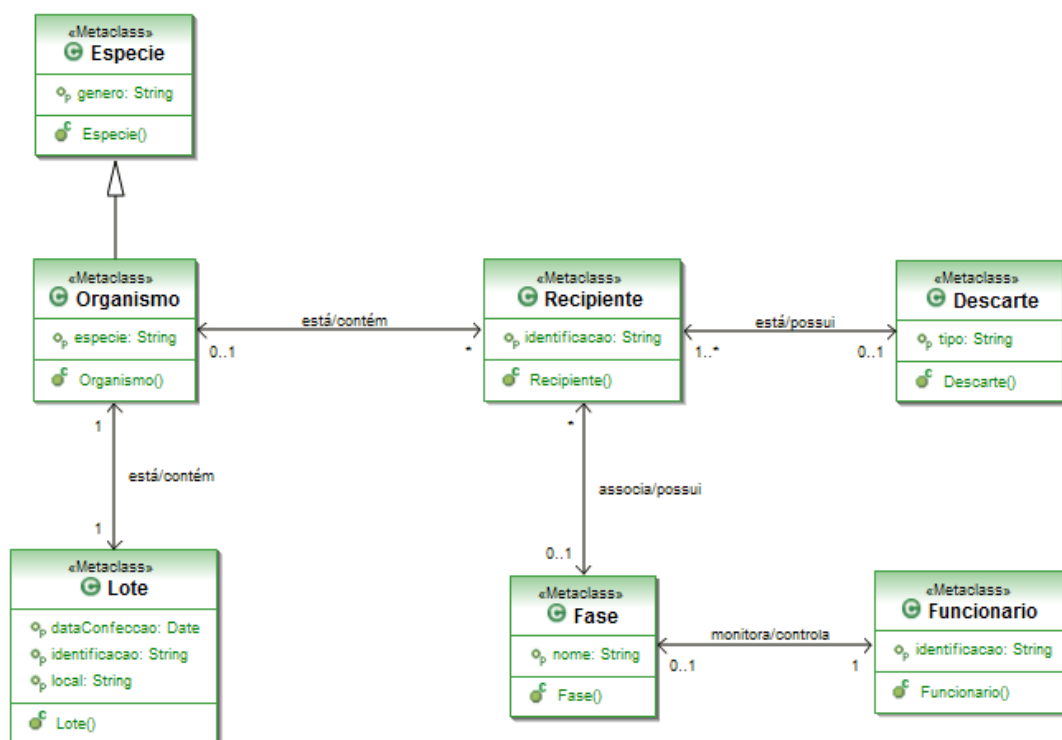


Figura 12 - Modelo PIM de especificação.

### 8.3 Construção do PSM

A modelagem no nível PSM aborda aspectos específicos a uma plataforma, detalhando características relevantes e particulares, que são minuciosamente tratadas na fase de desenvolvimento. Dependendo do nível do sistema, um refinamento a mais pode ser exigido, pois se o sistema possui inúmeros cenários ou arquiteturas, esses podem vir a ser prejudicados por um único modelo que talvez não detalhe suficientemente o aspecto técnico.

Com o PIM construído, a transformação para o PSM deve extrair as características alto nível que são imprescindíveis a um modelo de visão técnica.

A **Figura 13** é um bom exemplo de transformação PSM, onde as características principais da especificação da **Figura 12** são mantidas. Para a construção do modelo fora usado o *plug-in* Azzurri do Eclipse, que facilita a construção diagramática de modelos relacionais.

Esse modelo PSM aborda uma visão entidade-relacionamento, para uma aplicação em banco de dados, sendo assim uma visão específica de plataforma. Nessa modelagem existem duas marcações que reforçam a visão técnica: <<PK>> e <<FK>>. A primeira marcação especifica a coluna que é a chave primária (*primary key*) da tabela. A marcação

<<FK>> indica a coluna que é a chave estrangeira (*foreign key*) entre as entidades participantes do relacionamento.

A arquitetura apresentada reforça o intuito da modelagem diante de aspectos técnicos, oferecendo uma visão detalhista durante a fase de desenvolvimento. O modelo PSM tem como objetivo, aproximar a especificação sistêmica alto nível da plataforma escolhida. Essa abordagem facilita a geração de *scripts* de banco de dados durante a fase de desenvolvimento.

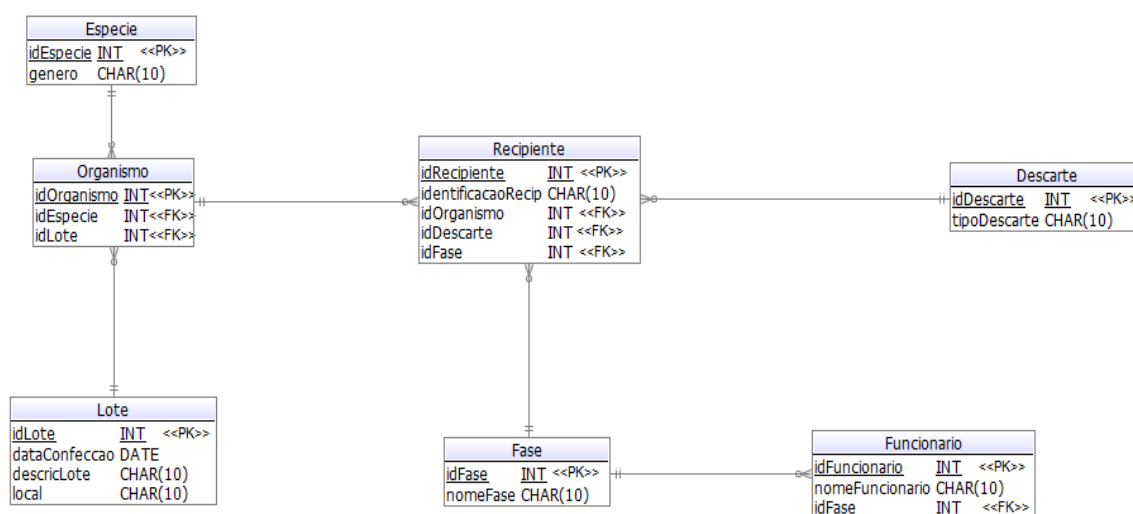


Figura 13 - Modelo PSM de especificação.

## 8.4 Construção do ISM

A fase de concepção do ISM tem como objetivo fazer a geração de código a partir do modelo PSM, sendo que algum mecanismo de geração de código pode auxiliar durante o processo. Visto que o intuito do presente trabalho é o desenvolvimento e arquitetura orientada a modelos, seguindo os padrões do MDA, a geração de código é tratada de forma automatizada, utilizando as ferramentas Acceleo e EMF.

A etapa ISM disponibiliza os artefatos com menor nível de abstração: o código fonte. Através de ferramentas de modelagem e de geração de código, o modelo PSM é utilizado como parâmetro de entrada durante o processo de transformação. Essa se trata de uma transformação interlinguagem, pois há uma transição entre níveis de linguagens diferentes, partindo da modelagem e, finalmente obtendo a linguagem de código.

Inicialmente o modelo PIM de especificação deve ser incluído em um projeto EMF, para que através do modelo se possa obter a representação XMI. Durante essa abordagem, a definição do modelo se encarrega de gerar a estrutura XMI, que será incorporada ao Acceleo.

No Acceleo, um projeto de geração por *templates* é criado para que se possa gerar um padrão de código específico. O estudo caso apresenta a automatização na geração da camada de acesso a dados e as classes *bean's*, sendo orientado à linguagem Java. A geração demonstra claramente que através de uma modelagem pode-se obter artefatos de código, beneficiando o processo de desenvolvimento por meio da automatização oferecida pelas ferramentas.

O apêndice A ilustra a representação XML que fora obtida a partir do modelo PIM da **Figura 12**, para em seguida ser incorporada ao Acceleo. A construção dos *templates*, que constam nos apêndices B e C proporciona a obtenção de classes específicas com base na definição do PIM. Algumas classes, obtidas através do gerador de código, são apresentadas no apêndice D.

Com a construção do PSM da **Figura 13**, a automatização na produção de código pode ser praticada por meio do *plug-in* Azzurri, produzindo *scripts* que criam a estrutura de banco de dados do esquema biofábrica.

A construção do ISM se encerra com a obtenção dos artefatos de código ou de modelagem, pois esse é o último estágio do processo adotado pelo MDA. A abordagem adotada nesse estudo de caso é tipicamente *top-down*, onde se parte do nível de maior abstração para o de menor abstração. Por ser uma “via de mão dupla”, existem abordagens do tipo *bottom-up*, que de posse de um modelo baixo nível, chega-se a um artefato alto nível.

## 9 CONCLUSÃO

Essa seção apresenta os resultados obtidos com o presente trabalho, como cada assunto contribuiu com a apresentação dos temas de modelagem e o que foi alcançado com cada abordagem.

O trabalho apresentou aspectos importantes e relevantes para a arquitetura e o desenvolvimento orientado a modelos, não atentando unicamente a essas abordagens, mas expondo outras abordagens e ferramentas que venham a contribuir com o assunto. Através da confecção de modelos consegue-se abstrair o cenário a ser tratado, a metodologia adotada faz com que ao longo dos estágios da modelagem haja a produção de artefatos. Esses artefatos forneceram mecanismos, para que assim alcance o objetivo do MDA e MDD.

A abordagem de assuntos como DSL e OCL, contribui e complementa a exposição e elaboração das perspectivas de modelagem, fazendo com que os conceitos MDA estejam estruturados de forma a melhorar o entendimento ao usuário sobre o assunto.

Os modelos são os principais elementos produzidos pelo MDA, propiciando com que o modelo esteja alinhado aos padrões de modelagem, aos quesitos de especificação do sistema e contribua com a automatização de processos no desenvolvimento sistêmico.

Ao longo do trabalho, a abordagem de especificações diagramáticas reforçou o contexto de geração de código numa perspectiva modelo – código, unindo o modelo de especificação com a prática que circunda o processo de desenvolvimento. A produção automatizada de código complementa a visão de modelagem, pois o principal objetivo é propor com que a modelagem e o código fonte estejam unidos desde a fase de especificação até o desenvolvimento sistêmico.

As ferramentas de modelagem e de geração de código complementam os objetivos a serem alcançados pela metodologia do MDA, fazem com que a construção de modelos seja facilitada e oferecem contribuições consideráveis na elaboração de artefatos. Em conjunto com as ferramentas, um dos artefatos produzidos foi a geração de código fonte, que concretizou todos conceitos e aspectos da metodologia adotada. Os apêndices expõem o resultado das abordagens e demonstram que modelo e código estão engajados numa mesma perspectiva.

O estudo de caso forneceu informações relevantes numa visão prática, expondo como cada fase contribui para o processo. Essas fases são responsáveis por gerar modelos, que passam a explorar gradualmente os aspectos do domínio e como cada integrante desse domínio irá interagir perante a plataforma.

## 10 REFERÊNCIAS

ACCELEO. **Acceleo Architect Tutorial**. [S.I.], 2011. Disponível em: <<http://www.acceleo.org/pages/introduction/en>>. Acessado em: 22 ago. 2011.

BELIX, J. E. **Um estudo sobre MDA: Suporte fornecido pela UML e reuso de soluções pré-definidas**. São Paulo, 2006. Tese (Mestrado em Engenharia).

BUDINSKY, F. et. al. **Eclipse Modeling Framework: A Developer's Guide**. Boston, 2004.

CALIARI, G. L. P. **Transformações e mapeamentos da MDA e sua implementação em três ferramentas**. São Paulo, 2007. Tese (Mestrado em Engenharia).

ECLIPSE. **Acceleo - transforming models into code**. [S.I.], 2011. Disponível em: <<http://eclipse.org/acceleo/>>. Acessado em: 02 nov. 2011.

ECLIPSE. **XText Documentation**. [S.I.], 2011. Disponível em: <[http://www.eclipse.org/Xtext/documentation/2\\_0\\_0/Xtext-Documentation.pdf](http://www.eclipse.org/Xtext/documentation/2_0_0/Xtext-Documentation.pdf)>. Acessado em: 20 ago. 2011.

ECLIPSE. **Ecore Components**. [S.I.], 2006. Disponível em: <<http://eclipse.org/modeling/emf/>>. Acessado em: 12 nov. 2011.

LUCREDIO, D. **Uma abordagem orientada a modelos para reutilização de software**. São Carlos, 2009. Tese (Doutorado em Ciências).

MIGNON, A. S. **Aplicação da técnica de tecelagem de modelos na transformação de modelos na MDA**. São Paulo, 2007. Tese (Mestrado em Engenharia).

NALON, F. R. **Adequação de um perfil UML para modelagem conceitual de bancos de dados geográficos aos padrões ISO e OGC usando MDA**. Viçosa, 2010. Trabalho de pós-graduação em Ciências da Computação.

OMG. **Meta-Object Facility Specification Version 2.1.1**. [S.I.], 2007. Disponível em: <<http://www.omg.org>>. Acessado em 24 out. 2011.

OMG. **Object Constraint Language OMG Available Specification Version 2.0**. [S.I.], 2006. Disponível em: <<http://www.omg.org>>. Acessado em 24 out. 2011.

OMG. **Model Driven Architecture Specification**. [S.I.], 2000. Disponível em: <<http://www.omg.org>>. Acessado em 26 out. 2011.

SCHOEPPING, G. **Projeto e implementação de uma ferramenta gráfica para UID**. Florianópolis, 2007. Trabalho de conclusão de curso em Ciências da Computação.

## 11 APÊNDICES

### Apêndice A – Representação XMI do esquema biofábrica

Este apêndice apresenta a representação XMI do esquema biofábrica tratado no estudo de caso. O XMI é um documento XML que apresenta o modelo de forma serializada.

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
  xmlns:xmi="http://www.omg.org/XMI"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="biofabrica"
  nsURI="http://biofabrica/1.0" nsPrefix="biofabrica">
  <eClassifiers xsi:type="ecore:EClass" name="Especie">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="genero"
      eType="ecore:EDatatype http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    </eClassifiers>
    <eClassifiers xsi:type="ecore:EClass" name="Organismo">
      eSuperTypes="#//Especie">
        <eStructuralFeatures xsi:type="ecore:EAttribute" name="especie"
          eType="ecore:EDatatype http://www.eclipse.org/emf/2002/Ecore#//EString"/>
        <eStructuralFeatures xsi:type="ecore:EReference" name="está"
          lowerBound="1" eType="#//Lote"
            eOpposite="#//Lote/contém"/>
        <eStructuralFeatures xsi:type="ecore:EReference" name="está"
          upperBound="-1" eType="#//Recipiente"
            eOpposite="#//Recipiente/contém"/>
        </eClassifiers>
        <eClassifiers xsi:type="ecore:EClass" name="Lote">
          <eStructuralFeatures xsi:type="ecore:EAttribute" name="dataConfeccao"
            eType="ecore:EDatatype http://www.eclipse.org/emf/2002/Ecore#//EDate"/>
          <eStructuralFeatures xsi:type="ecore:EAttribute" name="identificacao"
            eType="ecore:EDatatype http://www.eclipse.org/emf/2002/Ecore#//EString"/>
          <eStructuralFeatures xsi:type="ecore:EAttribute" name="loca"
            eType="ecore:EDatatype http://www.eclipse.org/emf/2002/Ecore#//EString"/>
          <eStructuralFeatures xsi:type="ecore:EReference" name="contém"
            lowerBound="1"
              eType="#//Organismo" eOpposite="#//Organismo/está"/>
          </eClassifiers>
          <eClassifiers xsi:type="ecore:EClass" name="Recipiente">
            <eStructuralFeatures xsi:type="ecore:EAttribute" name="identificacao"
              eType="ecore:EDatatype http://www.eclipse.org/emf/2002/Ecore#//EString"/>
            <eStructuralFeatures xsi:type="ecore:EReference" name="contém"
              eType="#//Organismo"
                eOpposite="#//Organismo/está.1"/>
            <eStructuralFeatures xsi:type="ecore:EReference" name="associa"
              eType="#//Fase"
                eOpposite="#//Fase/contém"/>
            <eStructuralFeatures xsi:type="ecore:EReference" name="está"
              eType="#//Descarte"
                eOpposite="#//Descarte/possui"/>
            </eClassifiers>
            <eClassifiers xsi:type="ecore:EClass" name="Fase">
              <eStructuralFeatures xsi:type="ecore:EAttribute" name="nome"
                eType="ecore:EDatatype http://www.eclipse.org/emf/2002/Ecore#//EString"/>
```

```

    <eStructuralFeatures xsi:type="ecore:EReference" name="contém"
upperBound="-1"
    eType="#//Recipiente" eOpposite="#//Recipiente/associa"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="monitorada"
lowerBound="1"
    eType="#//Funcionario" eOpposite="#//Funcionario/controla"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Descarte">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="tipo"
eType="ecore:EDatatype http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="possui"
lowerBound="1"
    upperBound="-1" eType="#//Recipiente"
eOpposite="#//Recipiente/está"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Funcionario">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="nome"
eType="ecore:EDatatype http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="controla"
eType="#//Fase"
    eOpposite="#//Fase/monitorada"/>
  </eClassifiers>
</ecore:EPackage>

```

## Apêndice B – *Template* para geração das classes Java *bean's*

O *template* abaixo é usado juntamente com a ferramenta Acceleo, auxiliando na geração das classes Java *bean's*.

```
<%  
metamodel http://www.eclipse.org/uml2/2.0.0/UML  
%>  
  
<%script type="uml.Class" name="beanTemplate" file="<%name%>.java"%>  
  
package br.acceleio.projectgenerator.java.bean;  
  
public class <%name%>{  
  
    <%for (attribute) {%>  
    private <%type.name%> <%name%>;  
  
    public void set<%name%>(<%type.name%> entity) {  
        <%name%> = entity;  
    }  
  
    public <%type.name%> get<%name%>() {  
        return <%name%>;  
    }  
    <%}%>  
}
```

## Apêndice C – *Template* para geração da camada de acesso a dados

O estudo de caso durante a fase ISM automatiza a geração de classes que são responsáveis pelo acesso aos dados. Com isso, esse *template* é incorporado à ferramenta Aceleo, oferecendo diretrizes para que o gerador de código forneça os artefatos de código.

```
<%
metamodel http://www.eclipse.org/uml2/2.0.0/UML
%>

<%script type="uml.Class" name="default" file="<%name%>DaoImp.java"%>

package br.acceleoprojectgenerator.java.dao;

import br.acceleoprojectgenerator.java.bean.<%name%>;
import br.acceleoprojectgenerator.java.util.FabricaConexao;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class <%name%>DaoImp {

    public int inserir (Connection conn, <%name%> entity){
        final String query = "INSERT INTO <%name%> (" +
            "<%for (attribute) {%><%name%><%if (i != length()-1){%>,<%}%><%}%> " +
            ")VALUES(<%for (attribute) {%>?<%if (i != length()-1){%>,<%}%><%}%>);";

        PreparedStatement pstmt = null;
        ResultSet res = null;
        int idInserido = -1;

        if (conn != null){
            try{
                pstmt = conn.prepareStatement(query,
                    PreparedStatement.RETURN_GENERATED_KEYS);

                <%for (attribute) {%>
                pstmt.set<%if (type.name ==
                    "Integer") {%>Int<%}%><%else{%><%type.name%><%}%>(<%i+1%>,
                    entity.get<%name%>());
                <%}%>

                pstmt.execute();
                res = pstmt.getGeneratedKeys();
                if (res.next())
                    idInserido = res.getInt(1);
            }
            catch(SQLException ex){
                System.out.println("Erro ao inserir <%name%>: " +
                    ex.getMessage());
            }
            finally{
```

```

        FabricaConexao.fechar(pstmt);
    }
}

return idInserido;
}

public List<<%name%>> selecionarTds() {
    final String query = "SELECT <%for (attribute)
{%><%name%><%if (i != length() - 1){%>, <%}%><%}%> " +
        "FROM <%name%>;";

    List<<%name%>> lst<%name.substring(0, 3)%> = new
ArrayList<<%name%>>();
    <%name%> entity = null;

    Connection conn = FabricaConexao.getConexao();
    PreparedStatement pstmt = null;
    ResultSet res = null;

    if (conn != null){
        try{
            pstmt = conn.prepareStatement(query);
            res = pstmt.executeQuery();

            while(res.next()){
                entity = new <%name%>();
                <%for (attribute) {%>
                entity.set<%name%>(res.get<%if (type.name ==
"Integer") {%>Int<%}else{%><%type.name%><%}%>(<%i+1%>));
                <%}%>
                lst<%name.substring(0, 3)%>.add(entity);
            }
        }
        catch(SQLException ex){
            System.out.println("Problemas ao consultar <%name%> " +
ex.getMessage());
        }
        finally{
            FabricaConexao.fechar(conn, pstmt);
        }
    }

    return lst<%name.substring(0, 3)%>;
}
}

```

## Apêndice D – Classes geradas

As classes abaixo foram geradas pelo gerador de código do Acceleio. Essas compõem algumas das classes da camada de acesso a dados da aplicação biofábrica.

```
package br.acceleio.projectgenerator.java.dao;

import br.acceleio.projectgenerator.java.bean.Especie;
import br.acceleio.projectgenerator.java.bean.Descarte;
import br.acceleio.projectgenerator.java.bean.Lote;
import br.acceleio.projectgenerator.java.util.FabricaConexao;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class EspecieDaoImp {

    public int inserir (Connection conn, Especie entity){
        final String query = "INSERT INTO Especie (" +
            "genero " +
            ")VALUES(?);";

        PreparedStatement pstmt = null;
        ResultSet res = null;
        int idInserido = -1;

        if (conn != null){
            try{
                pstmt = conn.prepareStatement(query,
PreparedStatement.RETURN_GENERATED_KEYS);

                pstmt.setString(1, entity.getgenero());

                pstmt.execute();
                res = pstmt.getGeneratedKeys();
                if (res.next())
                    idInserido = res.getInt(1);
            }
            catch(SQLException ex){
                System.out.println("Erro ao inserir Especie:
" + ex.getMessage());
            }
            finally{
                FabricaConexao.fechar(pstmt);
            }
        }

        return idInserido;
    }

    public List<Especie> selecionarTds () {
        final String query = "SELECT genero, " +
            "FROM Especie;";

        List<Especie> lstEsp = new ArrayList<Especie>();
        Especie entity = null;
    }
}
```

```

Connection conn = FabricaConexao.getConexao();
PreparedStatement pstmt = null;
ResultSet res = null;

if (conn != null){
    try{
        pstmt = conn.prepareStatement(query);
        res = pstmt.executeQuery();

        while(res.next()){
            entity = new Especie();
            entity.setgenero(res.getString(1));
            lstEsp.add(entity);
        }
    }
    catch(SQLException ex){
        System.out.println("Problemas ao consultar
Especie " + ex.getMessage());
    }
    finally{
        FabricaConexao.fechar(conn, pstmt);
    }
}

return lstEsp;
}
}

```

```

public class DescarteDaoImp {

    public int inserir (Connection conn, Descarte entity){
        final String query = "INSERT INTO Descarte (" +
            "tipo " +
            ")VALUES(?);";

        PreparedStatement pstmt = null;
        ResultSet res = null;
        int idInserido = -1;

        if (conn != null){
            try{
                pstmt = conn.prepareStatement(query,
PreparedStatement.RETURN_GENERATED_KEYS);

                pstmt.setString(1, entity.gettipo());

                pstmt.execute();
                res = pstmt.getGeneratedKeys();
                if (res.next())
                    idInserido = res.getInt(1);
            }
            catch(SQLException ex){
                System.out.println("Erro ao inserir
Descarte: " + ex.getMessage());
            }
            finally{
                FabricaConexao.fechar(pstmt);
            }
        }
    }
}

```

```

    }

    return idInserido;
}

public List<Descarte> selecionarTds() {
    final String query = "SELECT tipo, " +
        "FROM Descarte;";

    List<Descarte> lstDes = new ArrayList<Descarte>();
    Descarte entity = null;

    Connection conn = FabricaConexao.getConexao();
    PreparedStatement pstmt = null;
    ResultSet res = null;

    if (conn != null) {
        try {
            pstmt = conn.prepareStatement(query);
            res = pstmt.executeQuery();

            while (res.next()) {
                entity = new Descarte();
                entity.settipo(res.getString(1));
                lstDes.add(entity);
            }
        } catch (SQLException ex) {
            System.out.println("Problemas ao consultar
Descarte " + ex.getMessage());
        } finally {
            FabricaConexao.fechar(conn, pstmt);
        }
    }

    return lstDes;
}

}

public class LoteDaoImp {

    public int inserir (Connection conn, Lote entity) {
        final String query = "INSERT INTO Lote (" +
            "dataConfeccao, identificacao, local" +
            ") VALUES(?,?,?);";

        PreparedStatement pstmt = null;
        ResultSet res = null;
        int idInserido = -1;

        if (conn != null) {
            try {
                pstmt = conn.prepareStatement(query,
PreparedStatement.RETURN_GENERATED_KEYS);

                pstmt.setDate(1, entity.getdataConfeccao());
                pstmt.setString(2,
entity.getidentificacao());

```

```

        pstmt.setString(3, entity.getLocal());

        pstmt.execute();
        res = pstmt.getGeneratedKeys();
        if (res.next())
            idInserido = res.getInt(1);
    }
    catch(SQLException ex) {
        System.out.println("Erro ao inserir Lote: "
+ ex.getMessage());
    }
    finally{
        FabricaConexao.fechar(pstmt);
    }
}

return idInserido;
}

public List<Lote> selecionarTds() {
    final String query = "SELECT dataConfeccao, identificacao,
local, " +
        "FROM Lote;";

    List<Lote> lstLot = new ArrayList<Lote>();
    Lote entity = null;

    Connection conn = FabricaConexao.getConexao();
    PreparedStatement pstmt = null;
    ResultSet res = null;

    if (conn != null){
        try{
            pstmt = conn.prepareStatement(query);
            res = pstmt.executeQuery();

            while(res.next()){
                entity = new Lote();

                entity.setDataConfeccao(res.getDate(1));
                entity.setIdentificacao(res.getString(2));
                entity.setLocal(res.getString(3));
                lstLot.add(entity);
            }
            catch(SQLException ex) {
                System.out.println("Problemas ao consultar
Lote " + ex.getMessage());
            }
            finally{
                FabricaConexao.fechar(conn, pstmt);
            }
        }

        return lstLot;
    }
}

```