UNIVERSIDADE FEDERAL DE SANTA CATARINA

# Uso de MDA em um Framework para Seleção de Práticas Ágeis

Guilherme Aguiar

Florianópolis - SC

2012/1

UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

# Uso de MDA em um Framework para Seleção de Práticas Ágeis

Guilherme Aguiar

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação

Florianópolis - SC

2012/1

Guilherme Aguiar

# Uso de MDA em um Framework para Seleção de Práticas Ágeis

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação

Orientadora: Patrícia Vilain

Banca examinadora

Ricardo Pereira e Silva

Vitório Bruno Mazzola

## Agradecimentos


- Agradeço principalmente ao meu pai (João Aguiar Filho) e à minha mãe (Lourdes Maria Aguiar), que nunca mediram esforços para me proporcionar as melhores oportunidades possíveis, e sempre buscam a minha felicidade.
- Ao meu irmão por estar do meu lado e sempre me apoiar.
- Aos meus familiares e amigos, que me proporcionam momentos de descontração, mesmo estando longe em alguns casos.
- E agradeço o desenvolvedor de uma das ferramentas utilizadas nesse trabalho, Rafael Chaves, que esteve sempre disponível para esclarecer dúvidas e apoiar a minha pesquisa.

## Sumário

**1 INTRODUÇÃO ........................................................................................ 14**

**2 FUNDAMENTAÇÃO TEÓRICA .................................................................. 18**

2.1 Métodos Ágeis ........................................................................................ 18

2.2 Framework de Práticas Ágeis ................................................................ 21

2.2.1 Atividade de Definição dos requisitos ............................................... 22

2.2.2 Atividade de Projeto de Arquitetura do Sistema .............................. 23

2.2.3 Atividade Atribuir Requisitos às Iterações......................................... 23

2.2.4 Atividade Desenvolvimento do Incremento do Sistema .................... 24

2.2.5 Atividade validação do incremento .................................................. 27

2.2.6 Atividade Integração do Incremento ................................................. 27

2.2.7 Atividade Entrega Final...................................................................... 27

2.2.8 Práticas excludentes entre si ............................................................ 28

2.3 Model Driven Architecture e Model Driven Development........................ 30

2.3.1 Ferramentas MDA............................................................................... 36

2.3.1.1 Enterprise Architect ...................................................................... 38

2.3.1.2 Moderne ........................................................................................ 39

2.3.1.3 MagicDraw..................................................................................... 42

2.3.1.4 AlphaSimple................................................................................... 43

**3 TRABALHOS RELACIONADOS .............................................................. 45**

3.1 Agile Model-Driven Development in Practice......................................... 45

3.2 Um Relato de Experiência no Desenvolvimento Ágil de Sistemas com a MDA ........ 46

3.3 Agile Model Driven Development Is Good Enough ........ 47

3.4 Test-Driven Modeling for Model-Driven Development ........ 49

**4 ADEQUAÇÃO DO FRAMEWORK E DE FERRAMENTAS AO MDD ........ 52**

4.1 Processos do Framework ........ 52

**4.2 FERRAMENTAS ........ 57**

4.2.1 Moderne ........ 57

4.2.2 AlphaSimple ........ 64

4.2.3 Enterprise Architect ........ 68

4.2.4 MagicDraw ........ 71

**5 EXEMPLOS DE USO E DISCUSSÃO ........ 73**

5.1 Exemplos ........ 77

5.1.1 Aplicação para Restaurante ........ 77

5.1.1.1 Primeira Iteração ........ 81

5.1.1.2 Segunda Iteração ........ 84

5.1.1.3 Terceira Iteração ........ 87

5.1.2 Contador de Caracteres ........ 90

5.2 Considerações Finais ........ 94

**6 CONCLUSÃO ........ 97**

6.1 Trabalhos Futuros ........ 100

**7 REFERÊNCIAS BIBLIOGRÁFICAS ........................................................ 102**

**Lista de Figuras**

FIGURA 1 – DIAGRAMA DE FLUXO DO FRAMEWORK ............................... 22

FIGURA 2 - EXEMPLO PIM PSM ............................................................... 32

FIGURA 3– PASSOS DA ESPECIFICAÇÃO MDA. ....................................... 33

FIGURA 4 - TRASFORMAÇÕES MDA E SUAS PONTES ............................ 33

FIGURA 5 - CADEIA DE TRANFORMAÇÕES MDA..................................... 35

FIGURA 6 - METAMODELO CORE ............................................................. 40

FIGURA 7 - METAMODELO DE TESTE...................................................... 40

FIGURA 8 - DIAGRAMA DE CLASSES DISCIPLINA DEFINIÇÃO DE REQUISITOS ........................................................................................... 59

FIGURA 9 - DIAGRAMA DE CLASSES DISCIPLINA PROJETO ARQUITETURA DO SISTEMA......................................................................... 59

FIGURA 10 - DIAGRAMA DE CLASSES DISCIPLINA ATRIBUIÇÃO DOS REQUISITOS ÀS ITERAÇÕES......................................................................... 60

FIGURA11 - DIAGRAMA DE CLASSES DISCIPLINA DESENVOLVIMENTO INCREMENTAL DO SISTEMA I......................................................................... 60

FIGURA 12 - DIAGRAMA DE CLASSES DISCIPLINA DESENVOLVIMENTO INCREMENTAL DO SISTEMA II........................................................................ 61

FIGURA 13 - DIAGRAMA DE CLASSES DISCIPLINA DESENVOLVIMENTO INCREMENTAL DO SISTEMA III....................................................................... 61

FIGURA 14 - DIAGRAMA DE CLASSES DISCIPLINA VALIDAÇÃO DO INCREMENTO ........................................................................................... 62

FIGURA 15 - DIAGRAMA DE CLASSES DISCIPLINA INTEGRAÇÃO DO INCREMENTO ........ 62

FIGURA 16 - DIAGRAMA DE CLASSES DISCIPLINA VALIDAÇÃO DO SISTEMA........ 63

FIGURA 17 - DIAGRAMA DE CLASSES DISCIPLINA ENTREGA FINAL....... 63

FIGURA 18 – DIAGRAMA DE CLASSES ALPHASIMPLE ........ 65

FIGURA 19 – TEXTUML ALPHASIMPLE ........ 65

FIGURA 20 – CÓDIGO GERADO ALPHASIMPLE ........ 68

FIGURA 21 – MODELO ENTERPRISE ........ 69

FIGURA 22 – CÓDIGO GERADO ENTEPRISE........ 70

FIGURA 23 – MODELO MAGICDRAW........ 71

FIGURA 24 – CÓDIGO GERADO MAGICDRAW ........ 72

FIGURA 25 – REQUISITOS SISTEMA RESTAURANTE ........ 79

FIGURA 26 – MODELO PIM PARA O SISTEMA DO RESTAURANTE........ 80

FIGURA 27 – DISTRIBUIÇÃO DOS REQUISITOS NAS ITERAÇÕES........ 81

FIGURA 28 – PSM PRIMEIRA ITERAÇÃO........ 82

FIGURA 29 – PSM PRIMEIRA ITERAÇÃO REFINADO ........ 83

FIGURA 30 – PSM SEGUNDA ITERAÇÃO ........ 86

FIGURA 31- PSM TERCEIRA ITERAÇÃO........ 89

FIGURA 32 – PIM CONTADORDECARACTERES........ 91

FIGURA 33 – PSM CONTADORDECARACTERES ........ 91

FIGURA 34 – CÓDIGO GERADO CONTADORDECARACTERES ................. 92

FIGURA 35 – PSM FINAL CONTADORDECARACTERES ............................. 93

FIGURA 36 – CÓDIGO FINAL CONTADORDECARACTERES ...................... 94

FIGURA 37 – FRAMEWORK PARA SELEÇÃO DE PRÁTICAS ÁGEIS COM MDA ........................................................................................................ 99

## Lista de Tabelas

TABELA 1 – PRÁTICAS ÁGEIS EXCLUDENTES. .......................................... 28

TABELA 2 – COMBINAÇÃO 1 ........................................................................... 53

TABELA 3 – COMBINAÇÃO 2 ........................................................................... 54

TABELA 4 – COMBINAÇÃO 3 ........................................................................... 54

TABELA 5 – COMBINAÇÃO 4 ........................................................................... 55

TABELA 6 – COMBINAÇÃO 5 ........................................................................... 56

TABELA 7 – CORRESPONDÊNCIAS ENTRE O FRAMEWORK E A FERRAMENTA.................................................................................... 57

TABELA 8 – PRÁTICAS DA SIMULAÇÃO....................................................... 77

**Lista de Reduções**

MDA - Model Driven Architecture.

MDD - Model-Driven Development.

OMG – Object Management Group

CIM - Computation Independent Model.

PIM - Plataform Independent Model.

PSM - Plataform Specific Model.

UML – Unified Modeling Language

XP - Extreme Programming.

FDD - Feature Driven Development.

DSDM - Dynamic Systems Development Method

LSD - Lean Systems Development

ASD - Adaptive Software Development.

AM - Agile Modeling.

SPEM - Software Process Engineering Metamodel specification

SLAP – System-Level Agile Process

M2M – Model-to-Model

M2C – Model-to-Code

AMDD – Agile Model-Driven Development

XML – Extensible Markup Language

XMI - XML Metadata Interchange

TDD – Test-Driven Modeling

SDL – Specification and Description Language.

DFS – Diagramas de Fluxo do Sistema

## RESUMO

O desenvolvimento ágil vem sendo amplamente utilizado pelos engenheiros de software da atualidade. Outra técnica que vem sendo usada no desenvolvimento de software é a abordagem conhecida como *Model-Driven Architecture* (MDA), que prevê a construção de três modelos em diferentes níveis de abstração e a transformação direta entre eles.

Este trabalho pretende analisar as transformações presentes entre os modelos previstos no MDA no desenvolvimento ágil, utilizando para isto um framework para a seleção de práticas e definição de processos ágeis.

Serão analisadas as possíveis combinações de práticas ágeis geradas na definição de processos ágeis a partir da seleção das práticas oferecidas no framework, e para estas possíveis combinações de práticas ágeis, será analisada a viabilidade de sua utilização de acordo com os princípios de MDA.

Contudo, será necessário um estudo das ferramentas de transformação de modelos disponíveis, analisando sua adequação aos conceitos de MDA e aos processos gerados pelo framework para seleção de práticas ágeis.

**Palavras-chave:** MDA, Métodos Ágeis, Framework, Ferramentas.

# 1 INTRODUÇÃO

Com o passar dos anos, os sistemas computacionais foram tomando grandes proporções e sendo aplicados em larga escala, estando cada vez mais presentes no cotidiano da sociedade. Paralelamente, sua complexidade de desenvolvimento cresceu de maneira proporcional. Diante de métodos de desenvolvimento de software que eram vistos como burocráticos, lentos e contraditórios, alguns engenheiros de software se reuniram para criar o que eles chamariam de "métodos ágeis" (FAGUNDES, 2005).

O principal objetivo dos métodos ágeis é a criação de softwares de maneira mais rápida e incremental, utilizando um grupo de princípios e práticas ágeis definidos por alguns engenheiros de software no chamado manifesto ágil (BECK, COCKBURN, JEFFRIES, HIGHSMITH, 2001). Outro ponto importante a ser destacado na evolução da construção de software, é que o foco em seu processo de desenvolvimento também sofreu transformações. A construção do código-fonte deixa de ser o principal foco dos métodos, enquanto os modelos construídos passaram de artefatos de documentação para uma forma de tornar a escrita do código mais fácil, além de permitir a geração automática parcial de código-fonte (KOCH, ZHANG, ESCALONA, 2006).

Uma das técnicas que utiliza modelos como o centro do processo de desenvolvimento de software é chamada de Model-Driven Development (MDD). O MDD traz a idéia simples de que podemos construir modelos para um sistema e então construí-lo a partir desses modelos (MELLOR, CLARK, FUTAGAMI 2003). Considerando os conceitos do MDD, a *Object Management Group* (OMG) criou um conjunto de normas que permite a especificação de

modelos e suas transformações em outros modelos e no sistema final (MELLOR, CLARK, FUTAGAMI 2003). O MDD propõe que sejam construídos modelos de diferentes níveis de abstração, e através de transformações, seja construído o sistema final. Para isso, a especificação MDA utiliza a criação de três modelos, os quais oferecem visões em diferentes níveis de abstração de um mesmo sistema - o Computation Independent Model (CIM), o Plataform Independent Model (PIM), o Plataform Specific Model (PSM) - e a transformação direta entre esses modelos. Um desafio atual é conciliar essas duas técnicas, os métodos ágeis e a abordagem MDA.

Um primeiro problema, segundo Vilain (2007), é que a decisão de qual método ágil aplicar no desenvolvimento de software pode não ser simples. Além disso, pode-se, até mesmo, criar um processo com a utilização de práticas oriundas de diferentes métodos ágeis. Para auxiliar nesta tarefa, foi criado um framework, que contribui na comparação das práticas de cada método ágil e nas dependências entre elas, o que torna possível a escolha daquelas mais adequadas a um processo de desenvolvimento de software.

Para ZHANG e PATEL (2010), a chave para realizar uma abordagem MDD ágil, é conciliar os dois processos de uma forma que as vantagens de ambos sejam aproveitadas, e ao mesmo tempo evitar os problemas de cada um deles.

O framework proposto em FAGUNDES (2005) auxilia na construção de processos ágeis, no entanto, grande parte dos possíveis processos que podem ser gerados por esse framework, não se adéquam às necessidades do MDA. Para que os processos sejam utilizados com MDA, devem voltar grandes esforços à atividade de modelagem, e produzir modelos com qualidade

suficiente para compor a cadeia de transformações e gerarem resultados satisfatórios. Muitos dos processos criados pelo Framework de Práticas Ágeis[1] podem ser extremamente simples e com o foco na produção direta de código fonte.

Este trabalho irá mostrar os resultados de uma tentativa de conciliar as técnicas das transformações MDA nos processos de software definidos a partir do framework para seleção de práticas ágeis, analisando a viabilidade e possíveis vantagens desta combinação.

## 1.1 Objetivos

### *1.1.1 Objetivo Geral*

O objetivo principal deste trabalho é analisar o uso de técnicas de MDA (Model Driven Architecture) em processos ágeis, definidos a partir do Framework de Práticas Ágeis.

### *1.1.2 Objetivos Específicos*

- Estudo dos Métodos Ágeis e do Framework de Práticas Ágeis, do MDD, MDA e de Ferramentas MDA.
- Proposta de uso do MDA em algumas etapas dos processos ágeis definidos a partir do Framework de Práticas Ágeis.
- Utilização de uma ferramenta existente para transformação de modelos gerados a partir da utilização do Framework de Práticas Ágeis.

[1] Neste trabalho, o termo "Framework de Práticas Ágeis" referencia o framework para seleção de práticas ágeis proposto em [Fagundes, 2005].

- Experimento utilizando possíveis transformações entre modelos usados em um processo ágil definido.

# 2 Fundamentação teórica

## 2.1 Métodos Ágeis

O desenvolvimento de *software* possui como um de seus grandes desafios, entregar um algo que atenda às necessidades do cliente, dentro do prazo estipulado, sem que para isso o orçamento do projeto seja excedido (FAGUNDES 2005). Para ajudar os desenvolvedores a vencer este desafio, a Engenharia de Software oferece vários métodos a serem seguidos durante o processo de desenvolvimento do *software* (FAGUNDES 2005).

Os métodos mais tradicionais eram baseados em um conjunto de atividades predefinidas, descritas como processos prescritivos (AMBLER 2004) que normalmente, iniciavam os trabalhos com o levantamento total do conjunto de requisitos, seguido por um projeto de alto-nível, a implementação do sistema, sua validação e finalmente a manutenção do mesmo (SOMMERVILLE 2003). No entanto, esse tipo de método é muito burocrático e mais adequado para projetos grandes com poucas mudanças nos requisitos (Fowler 2001).

Em direção oposta a estes métodos, surgiram métodos com uma abordagem ágil de desenvolvimento de *software*. Os processos adeptos a essa tentam adaptar-se às mudanças apoiando a equipe de desenvolvimento abordagem, surgiram na década de 90 e ganharam muitos adeptos com o passar dos anos (FAGUNDES 2005).

Para que melhores meios de produzir software fossem encorajados, no ano de 2001 um grupo com inicialmente 17 metodologistas formou a Aliança para o Desenvolvimento Ágil de Software, também conhecida como Aliança Ágil (FAGUNDES 2005). A Aliança Ágil formulou um manifesto contendo um

conjunto de princípios que definem critérios para os processos de desenvolvimento ágil de software (AMBLER 2004).

A seguir serão apresentados os valores do Manifesto Ágil de acordo com Beck et. al.(2001). Estes quatro valores são a base do Manifesto Ágil, definem preferências e encorajam o enfoque em certas áreas (FAGUNDES 2005).

• Indivíduos e interações valem mais que processos e ferramentas.

• Um software funcionando vale mais que uma documentação extensa.

• A colaboração do cliente vale mais que a negociação de contrato.

• Responder as mudanças vale mais que seguir um plano.

Segundo FAGUNES (2005), para que o enfoque do desenvolvimento ágil fosse mais bem compreendido pelas pessoas, os membros da Aliança Ágil refinaram as filosofias presentes em seu manifesto em uma coleção de doze princípios, os métodos ágeis de desenvolvimento de software devem se adequar. Estes princípios são listados a seguir.

1. A prioridade é satisfazer ao cliente através de entregas de software de valor contínuas e frequentes;

2. Entregar software sem funcionamento com frequência de algumas semanas ou meses, sempre na menor escala de tempo;

3. Ter o software funcionando é a melhor medida de progresso;

4. Receber bem as mudanças de requisitos, mesmo em uma fase avançada, dando aos clientes vantagens competitivas;

5. As equipes de negócio e de desenvolvimento devem trabalhar juntas diariamente durante todo o projeto;

6. Manter uma equipe motivada fornecendo ambiente, apoio e confiança necessário para a realização do trabalho;

7. A maneira mais eficiente da informação circular dentro da equipe é através de uma conversa face a face;

8. As melhores arquiteturas, requisitos e projetos provêm de equipes organizadas;

9. Atenção contínua a excelência técnica e um bom projeto aumentam a agilidade;

10. Processos ágeis promovem o desenvolvimento sustentável. Todos envolvidos devem ser capazes de manter um ritmo de desenvolvimento constante;

11. Simplicidade é essencial;

12. Em intervalos regulares, a equipe deve refletir sobre como se tornarem mais eficazes e então se ajustar e adaptar seu comportamento.

A comunidade de desenvolvimento de software vem demonstrando grande interesse nos métodos ágeis. Admiti-se que devido a esta demanda que, têm surgido nos últimos anos uma considerável quantidade de métodos que apresentam características ágeis, juntamente com livros técnicos abordando o assunto (FAGUNDES 205).

Como os diversos métodos ágeis criados seguem os mesmos princípios, vários destes métodos apresentam práticas ágeis comuns (ALVAREZ 2010). Para que a tarefa de escolher um dos métodos ágeis, ou um conjunto de atividades retiradas de diferentes métodos para um projeto específico, seja

facilitada, o framework definido em (FAGUNDES 2005) pode ser utilizado. A próxima seção deste capítulo é dedicada em apresentar esse framework.

### 2.2 Framework de Práticas Ágeis

O framework utilizado neste trabalho foi criado em FAGUNDES(2005). Ele apresenta as atividades dos métodos ágeis: *Extreme Programming* (XP), *Scrum*, *Feature Driven Development* (FDD), *Adaptive Software Development* (ASD) e *Agile Modeling* (AM) e pode ser utilizado durante o desenvolvimento de softwares, onde as atividades sugeridas por cada um dos métodos poderão ser selecionadas pela equipe de desenvolvimento de acordo com suas necessidades e experiências.

FAGUNDES(2005) fez uma pesquisa sobre cada um dos métodos e analisou suas atividades, fazendo um estudo comparativo entre elas, podendo assim agrupar as atividades semelhantes, o que possibilitou a definição da estrutura do framework proposto em seu trabalho.

Para uma equipe de desenvolvimento utilizar um método ágil, é de extrema importância que todos os envolvidos no projeto conheçam o método de maneira profunda, para não comprometer o sucesso do projeto. Uma das vantagens da utilização do framework criado em FAGUNDES (2005) é que, através dele, a equipe pode selecionar as atividades de alguns métodos para serem utilizadas em seu processo de software, sem a necessidade de se conhecer nenhum dos métodos específicos de maneira profunda.

Portanto, o framework desenvolvido demonstra ser de fácil aplicação por equipes que possuam conhecimento sobre as práticas contidas nele, proporcionando aos interessados na utilização de um processo ágil uma boa

oportunidade de fazê-lo de maneira simples, sem exigir grandes especializações.

A seguir um diagrama de atividades do Framework de Práticas Ágeis, proposto em FAGUNDES (2005), é mostrado.

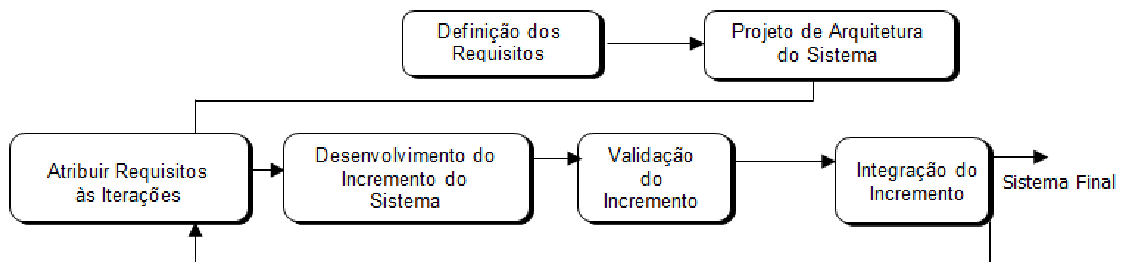



**Figura 1** – Diagrama de Fluxo do Framework

A seguir um diagrama de atividades do Framework de Práticas Ágeis, proposto em FAGUNDES (2005), é mostrado.

*2.2.1 Atividade de Definição dos requisitos*

**Lista de Requisitos** (Obrigatória): Consiste de um documento contendo os requisitos do sistema, agrupados por prioridades. Cada um dos requisitos terá especificado neste documento uma breve descrição, o tempo estimado e os papéis envolvidos no seu desenvolvimento. Os papéis relacionados com cada requisito serão anexados a este. O papel envolvido na realização desta tarefa é a do gerente de projeto. Fontes: Scrum, FDD, DSDM, Crystal Clear, e LSD.

**Modelagem Geral**: Para entender e documentar os requisitos do sistema, o processo pode incluir um diagrama de classe. Os papéis envolvidos são basicamente o gerente de projeto e o cliente, que é considerado o especialista de domínio no Framework de Práticas Ágeis, o qual irá ajudar a equipe a entender o problema. Fonte: FDD.

**Documentação Inicial:** Documentação que contém uma visão geral do sistema, junto com um esboço da estimativa do custo total do projeto, além dos benefícios esperados e estimativas de recursos humanos. Pode conter também que contém um resumo com informações como a descrição do sistema, contatos dos principais utilizadores, além de tecnologias e ferramentas utilizadas durante o desenvolvimento do projeto. Esta documentação tem o intuito de manter a documentação do projeto organizada. Esta prática também deve ser executada.

### *2.2.2 Atividade de Projeto de Arquitetura do Sistema*

**Projeto de Arquitetura do Sistema**: Consiste de uma visão geral das técnicas de concepção arquitetônica. Quaisquer artefatos de design podem ser usados no projeto de arquitetura, tais como diagramas de classe e sequência; porém, os diagramas de pacotes e *deployment* costumam ser mais utilizados. Os papéis relacionados são programadores e gerente de projeto. A decisão de executar essa prática é até programadores e gerente de projeto. Fontes: XP, Scrum, FDD, DSDM, Crystal Clear, LSD e AM.

### *2.2.3 Atividade Atribuir Requisitos às Iterações*

**Planejamento da Iteração (Obrigatório)** - Consiste em distribuir os requisitos para as iterações de acordo com suas prioridades, dependências e riscos. Uma ou mais reuniões no início de cada iteração são sugeridas para planejar o desenvolvimento. Estas reuniões podem contar com a participação do cliente também. Dentro desta prática também estão inclusas o planejamento da duração das iterações e a distribuição dos requisitos para responsáveis. A

duração de uma iteração varia de uma a oito semanas, e as equipes envolvidas no desenvolvimento, caso um requisito não seja atribuído diretamente a uma pessoa, não devem passar de deis membros e um responsável. Os papéis sociais associados a esta prática são programadores, gerente de projeto e clientes (se participar). Fontes: XP, Scrum, FDD, ASD, DSDM, Crystal Clear, e LSD.

### *2.2.4 Atividade Desenvolvimento do Incremento do Sistema*

**Estórias do Usuário:** As estórias de usuário são escritas pelo cliente, que deve construir uma estória para cada funcionalidade do sistema, apresentando informações sobre elas e estabelecer suas prioridades. Os papéis associados a esta prática são clientes, programadores e o projetista, os quais devem auxiliar os clientes a escrever estórias do usuário. Fontes: XP e Crystal Clear.

**Casos de Uso:** Para detalhar cada um dos requisitos, casos de uso podem ser utilizados. Um diagrama de casos de uso pode ser construído nesta etapa. Para cada funcionalidade do sistema a ser desenvolvido na iteração, um caso de uso pode ser definido. Também possui como papéis envolvidos os clientes, os programadores e o projetista. Fonte: FDD.

**Projeto da Iteração**: Consiste em um projeto de sistema com base nos requisitos conhecidos na iteração atual. Pode-se utilizar um diagrama UML, como por exemplo, um diagrama de classes, para representar toda a iteração, ou até mesmo um diagrama que seja específico para cada estória de usuário

ou funcionalidade, como um diagrama de sequência. O papel envolvido nesta atividade é o projetista. Sources: FDD.

**Escrita dos testes de unidade:** Essa prática sugere que os testes de unidade são escritos antes do desenvolvimento, e executados posteriormente. O papel relacionado com esta função é o do programador. Fonte: XP.

**Escrita sobre os testes de aceitação:** Esta prática sugere que os testes de aceitação são escritos antes do desenvolvimento, e executados posteriormente. Os testes de aceitação podem ser escritos por clientes ou programadores que, além do projetista, são os papéis relacionados com esta prática. Fonte: XP.

**Desenvolvimento** (Obrigatório): Esta prática consiste basicamente de codificação dos requisitos que fazem parte da iteração atual. É recomendado que algumas medidas sejam tomadas, como a utilização de padrões para a estrutura do código e geração, assim como utilizar um sistema de para fazer o controle de versões, a fim de manter o código gerado organizado. O papel envolvido nesta prática é o de programador Fontes: XP, Scrum, FDD, ASD, DSDM, Crystal Clear, e LSD.

**Programação em Pares:** Esta prática sugere que o código seja escrito por um par de programadores. Um deles é responsável por pensar qual maneira seria a mais adequada para a implementação do método, enquanto o outro analisa se esta abordagem funcionará, se existe algum teste que ainda não está

funcionando ou se existe algum modo de tornar o código mais simples. Os papéis envolvidos são basicamente os programadores. Fonte: XP.

**Programação lado-a-lado:** Esta prática sugere que a dupla de programadores utilize diferentes computadores, porém, sejam capazes de enxergar os monitores um do outro. O objetivo desta prática é obter as vantagens da programação em par ao mesmo tempo em que permite aos programadores trabalhar em paralelo. Os papéis envolvidos são programadores. Fonte: Crystal Clear.

**Refatoração:** Esta prática sugere a reconstrução do código sempre que necessário. A refatoração no código deve ser realizada apenas quando é realmente necessário. As funções envolvidas são programadores. Fonte: XP.

**Reuniões Diárias:** Esta prática sugere reuniões diárias para informar os participantes sobre progressos realizados e dificuldades encontradas. As reuniões devem durar entre 15 e 30 minutos, com os participantes de pé. Os papéis relacionados com esta prática são programadores e gerente de projetos. Fontes: XP, Scrum e Crystal Clear.

**Desenvolvimento Coletivo de Código:** Essa prática torna o desenvolvimento de responsabilidade de toda a equipe. Desta maneira, é importante que a equipe de desenvolvimento tenha pleno conhecimento e entendimento do que

deve ser desenvolvido. O papel envolvido nessa prática é basicamente o de programador. Fonte: XP.

### *2.2.5 Atividade validação do incremento*

***Inspeção de código:*** *Esta prática possui o objetivo de encontrar defeitos no código, além de analisar se o mesmo foi escrito de maneira compreensível. Para isso, é sugerido que o programador que realiza a inspeção seja diferente do que construiu o código. As funções são basicamente programadores. Fontes: FDD e ASD.*

### *2.2.6 Atividade Integração do Incremento*

**Reunião de Revisão da Iteração:** Esta prática sugere um encontro no final da iteração, para que a equipe de desenvolvimento faça uma avaliação sobre ela. Inicialmente, apenas o gerente e os programadores participam da reunião. Após o incremento resultante ser apresentado e os testes de integração serem executados, os clientes também estão convidados a participar da reunião. Os papéis envolvidos são os clientes, programadores e gerente de projetos. Fontes: Scrum, FDD e LSD.

### *2.2.7 Atividade Entrega Final*

**Breve Documentação:** Caso uma documentação detalhada do sistema não tenha sido gerada durante o processo de desenvolvimento, é aconselhável que após a entrega final do sistema seja gerada uma documentação curta, entre 5

e 10 páginas, incluindo uma Documentação do Usuário. Nesta prática, a engenharia reversa pode ser aplicada sobre o código construído para gerar diagramas UML. Os papéis envolvidos são programadores e gerente de projeto. Fontes: XP e Scrum, AM e Crystal Clear.

**Entrega do Sistema:** Prática executada apenas quando todos os requisitos do sistema tenham sido desenvolvidos e o cliente está totalmente satisfeito com o sistema. Uma reunião com todos os interessados pode ser realizada para que eles reconheçam a entrega final. Apesar de todos os interessados participarem, quem executa a tarefa são basicamente os programadores. Fontes: XP, Scrum, FDD, DSDM, ASD e Crystal Clear.

### *2.2.8 Práticas excludentes entre si*

Abaixo é apresentada a tabela de dependências entre as práticas do Framework de Práticas Ágeis. As dependências entre as práticas obrigatórias não são apresentadas. Por exemplo, várias práticas dependem do desenvolvimento da iteração (codificação), mas isto não é mostrado pois esta prática é obrigatória. Práticas obrigatórias são marcadas com a letra “o” e a letra “E” aparece na tabela marcando as práticas excludentes entre si.

**Tabela 1** – Práticas ágeis excludentes.

| | Lista de Requisitos(O) | Modelagem Geral | Documentação Inicial | Projeto de Arquitetura do Sistema | Planejamento da Iteração (O) | Reuniões Diárias | Estórias do Usuário | Casos de Uso | Projeto da Iteração | Desenvolvimento (O) | Escrita dos testes de unidade | Escrita sobre os testes de aceitação | Desenvolvimento Coletivo de Código | Programação em Pares | Refatoração | Integração Paralela | Controle de Versões | Programação lado-a-lado | Inspeção de código | Reunião de Revisão da Iteração | Breve Documentação | Entrega do Sistema |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Lista de Requisitos (O) | | | | | | | | | | | | | | | | | | | | | | |
| Modelagem Geral | | | | | | | | | | | | | | | | | | | | | | |
| Documentação Inicial | | | | | | | | | | | | | | | | | | | | | | |
| Projeto de Arquitetura do Sistema | | | | | | | | | | | | | | | | | | | | | | |
| Planejamento da Iteração (O) | | | | | | | | | | | | | | | | | | | | | | |
| Reuniões Diárias | | | | | | | | | | | | | | | | | | | | | | |
| Estórias do Usuário | | | | | | | | E | | | | | | | | | | | | | | |
| Casos de Uso | | | | | | | E | | | | | | | | | | | | | | | |
| Projeto da Iteração | | | | | | | | | | | | | | | | | | | | | | |
| Desenvolvimento (O) | | | | | | | | | | | | | | | | | | | | | | |
| Desenvolvimento Coletivo de Código | | | | | | | | | | | | | | | | | | | | | | |
| Programação em Pares | | | | | | | | | | | | | | | | | | E | | | | |
| Refatoração | | | | | | | | | | | | | | | | | | | | | | |
| Integração Paralela | | | | | | | | | | | | | | | | | | | | | | |
| Controle de Versões | | | | | | | | | | | | | | | | | | | | | | |
| Programação lado-a-lado | | | | | | | | | | | | | | E | | | | | | | | |
| Inspeção de código | | | | | | | | | | | | | | | | | | | | | | |
| Reunião de Revisão da Iteração | | | | | | | | | | | | | | | | | | | | | | |
| Breve Documentação | | | | E | | | | | E | | | | | | | | | | | | | |
| Entrega do Sistema | | | | | | | | | | | | | | | | | | | | | | |

## 2.3 Model Driven Architecture e Model Driven Development

O desenvolvimento de software envolve, muitas vezes, um grande número de tecnologias, gerando assim um de seus problemas habituais. Com cada nova tecnologia que surge, muito trabalho deve ser refeito, já que um software nunca utiliza uma tecnologia apenas e deve na maioria das vezes comunicar-se com outros softwares. (KLEPPE, WARMER, BAST, 2003)

Normalmente os processos de software possuem muita documentação produzida nas etapas iniciais, até mesmo alguns métodos ágeis focam suas primeiras iterações no desenvolvimento de modelos e documentação. (SOUZA, 2011). Contudo, esses modelos vão sendo deixados de lado pelas equipes de desenvolvimento à medida que as fases do processo evoluem e o código fonte é escrito. Além disso, é muito comum que manutenções sejam executadas no software sem que a documentação seja atualizada, diminuindo ainda mais o valor destes artefatos (SOUZA, 2011).

Uma arquitetura desenvolvida pela Object Management Group (OMG) vai em direção oposta a esses fatos. Chamada de Model Driven Architecture (MDA), esta arquitetura reconhece o valor dos artefatos de documentação dentro do processo de software, tornando-os o ponto-chave no desenvolvimento. A abordagem proposta pela MDA, define que o processo de software seja direcionado pela atividade de modelagem do sistema, no nível conceitual, desconsiderando implementação ou qualquer tipo de plataforma, e que através de transformações, este modelo conceitual evolua para níveis cada vez mais específicos e ligados à implementação automaticamente, a partir do modelo gerado inicialmente (SOUZA, 2011). A MDA, na verdade, é uma

especificação para apoiar o desenvolvimento dirigido por modelos, chamado de Model-Driven Development (MDD). O MDD não é algo recente, mas ganhou notoriedade após o OMG lançar a especificação MDA, em 2001. A arquitetura MDA compreende três passos principais (SOUZA, 2011):

O primeiro passo é a construção de um modelo independente do ponto de vista computacional, o Computacional Independent Model (CIM). Este modelo deve representar requisitos do sistema sem abordar detalhes de sua estrutura.

O passo posterior é a geração de um modelo independente da plataforma, o Plataform Independent Model (PIM), a partir do CIM. O PIM é um modelo com alto nível de abstração, independente de qualquer tecnologia e descreve o sistema de software a partir de uma perspectiva que melhor represente o negócio sendo projetado.

O terceiro passo é a construção de um ou mais Plataform Specific Models (PSM), a partir do PIM. O PSM leva em consideração detalhes específicos de uma determinada tecnologia a ser utilizada na implementação, sendo este o motivo que leva a possível geração de mais de um PSM a partir de um PIM.

Um modelo é sempre um PIM ou um PSM, no entanto, esta diferença nem sempre é clara (KLEPPE, WARMER, BAST, 2003). Por exemplo, um modelo escrito em UML que possui um dos seus diagramas de classe definindo uma ou mais interfaces pode ser considerado específico da plataforma Java (KLEPPE, WARMER, BAST, 2003)? Portanto, conceitos de PIM e PSM podem ser relativos. Porém, têm-se a certeza de que um PSM é mais específico a uma plataforma que o PIM, e que o este, ao ser transformado, passa de um modelo

menos específico para um mais específico à plataforma. A próxima figura mostra de forma simplória um modelo PIM e um PSM gerado a partir dele. (KLEPPE, WARMER, BAST, 2003).

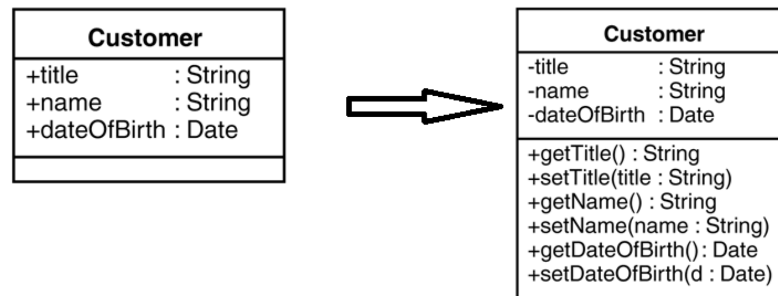



**Figura 2 -** Exemplo PIM PSM

Fonte : KLEPPE, WARMER, BAST (2003)

Existe ainda um último passo que é a geração de código a partir de cada um dos PSM. Esse código fonte não deve conter apenas estruturas básicas e *templates*, como o código gerado por muitas ferramentas case. O código gerado em MDA deve ser o mais próximo possível do software final, incluindo as regras de negócio (SOUZA, 2011). A seguir, a Figura 3 irá ilustrar os passos da arquitetura MDA, descritos anteriormente.

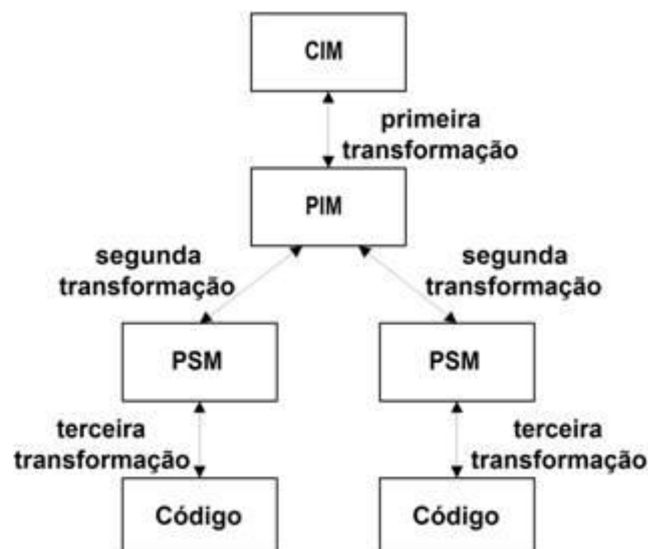


**Figura 3**– Passos da especificação MDA.

Fonte: SOUZA (2011)

A figura mostra uma transformação direta entre o CIM e o PIM. Todavia, não é possível fazê-lo de forma automática (KLEPPE, WARMER, BAST, 2003). Por se caracterizar como independente de computação, nem sempre o CIM fala algo especificamente sobre o software, o que torna a tarefa de decidir o que é ou não relativo a ele, inerentemente humana (KLEPPE, WARMER, BAST, 2003). Além do mais, muitas vezes o CIM utilizado no processo de software acaba sendo na forma textual. A Figura 4 mostra o processo MDA de maneira mais apropriada.

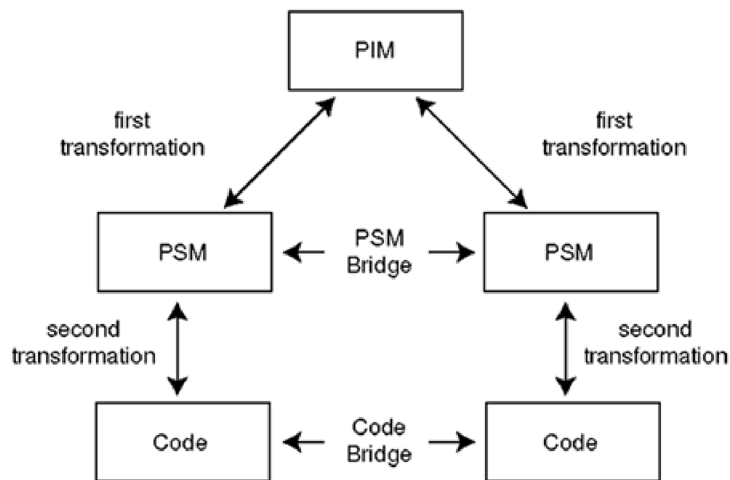



**Figura 4 -** Trasformações MDA e suas pontes

Fonte: KLEPPE, WARMER, BAST (2003)

A figura mostra dois elementos presentes na arquitetura dirigida a modelos, chamados de PSM bridge e Code bridge. Esses elementos são

“pontes” entre o código e os modelos de diferentes arquiteturas. Como podemos definir claramente as regras de transformações entre modelos independentes de arquitetura para modelos de diferentes arquiteturas, pode-se também criar correspondências entre estes modelos. Por exemplo, para um elemento Cliente em um diagrama de classes PIM, podemos mapeá-lo para uma classe Java e para uma tabela em um diagrama Entidade-Relacionamento (ER). Logo, é possível criar uma correspondência entre a classe Java e uma tabela no modelo ER (KLEPPE, WARMER, BAST, 2003).

O processo MDA pode parecer como os processos de software tradicionais, mas a principal diferença está na maneira como suas transformações são tratadas. Elas devem ser totalmente realizadas por ferramentas. Muitas delas, como as do PSM para o código, não trazem muitas novidades, já que o PSM, às vezes, é muito próximo da plataforma alvo. Ressalta-se que, tradicionalmente, ferramentas não produzem código com qualidade muito superior a *templates* e devem ser completados à mão. A novidade principal do processo está na total automatização da transformação do PIM para os PSMs alvos (KLEPPE, WARMER, BAST, 2003).

Estas transformações também são definidas pela MDA, além dos três modelos vistos anteriormente. Elas são gerações automáticas de um modelo para outro, de acordo com a especificação criada com um conjunto de regras e especificações de como um modelo deve ser criado a partir de outro (SOUZA, 2011). Se, de alguma maneira, olharmos dentro da ferramenta de transformação, veremos regras de como um modelo deve ser transformado em outro, isto é chamado de definições de transformações (KLEPPE, WARMER, BAST, 2003). Podemos, então, dizer que uma definição de transformação é um

conjunto de regras de transformações que mapeia as partes de um modelo para outro, criando assim um novo modelo (KLEPPE, WARMER, BAST, 2003).

MDA define que devem ser criados modelos, no entanto não define como estes modelos devem ser escritos. Modelos são escritos nas mais diversas linguagens, como UML, alguma linguagem de programação ou outras formas de expressão. A única restrição é que, para o modelo ser utilizado com MDA, é essencial que ele seja escrito em uma linguagem bem definida, pois apenas desta maneira uma geração automática de outros modelos a partir dele seria possível (KLEPPE, WARMER, BAST, 2003).

Considera-se uma linguagem bem formada, e utilizável em MDA, uma linguagem que possua sua sintaxe e semântica bem definidas e computáveis. Assim sendo, linguagens naturais não são utilizáveis na cadeia de transformações da MDA, já que não são computáveis (KLEPPE, WARMER, BAST, 2003). A Figura 5 ilustra essa cadeia de transformações, com as ferramentas e as definições de transformações.

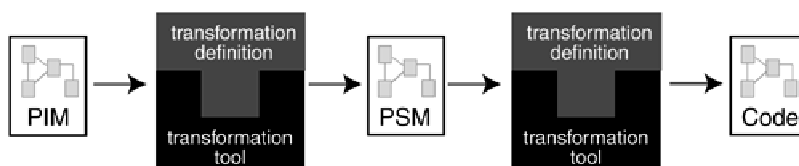



**Figura 5** - Cadeia de tranformações MDA.

Fonte: KLEPPE, WARMER, BAST (2003)

Tendo visto estes conceitos, é possível destacar as principais vantagens do uso da MDA: (Souza, 2011)

- Produtividade – apesar de a arquitetura MDA exigir a criação das regras de transformação entre os modelos PIM, PSM e de código, esses modelos devem

ser gerados uma única vez para cada plataforma. O foco do desenvolvedor muda diretamente para o desenvolvimento das transformações entre o PIM e o PSM, assim que as regras de transformação forem construídas, apenas os modelos independentes de plataforma devem ser atualizados, enquanto os outros serão atualizados automaticamente, proporcionando um desenvolvimento mais ágil e com um produto final de mais qualidade.

- Portabilidade – Já que a arquitetura produz modelos independentes de plataforma, eles podem ser transformados automaticamente para diversos PSMs, que são específicos às plataformas. Essa vantagem está diretamente ligada à qualidade das ferramentas utilizadas e seu suporte às diversas tecnologias no mercado.

- Manutenção e documentação – A manutenção na arquitetura MDA deve ser feita no modelo conceitual (o modelo PIM). Assim, a manutenção é facilitada, e a documentação deverá permanecer sempre atualizada.

Contudo, a arquitetura MDA ainda não contempla todos estes benefícios em sua totalidade. Para que isso seja possível, todas as transformações entre modelos devem ser automáticas, o que dificilmente é obtido com a atual fase da tecnologia. Muitos aspectos como ferramentas e processos MDA devem ser aprimoradas, para que ela seja utilizada a fim de se obter suas plenas vantagens (SOUZA, 2011).

### *2.3.1 Ferramentas MDA*

Para apoiar o desenvolvimento dirigido a modelos, tentativas de desenvolvimento de ferramentas apropriadas vêm sendo feitas, além da

adaptação de outras já existentes. Algumas são ferramentas comerciais de grande porte já conhecidas pela maioria dos desenvolvedores de software mundo afora, outras, porém, podem ser resultado de esforços individuais ou de grupos de pesquisas de construir meios apropriados para utilizar o MDA em seus processos de software.

A seguir, algumas ferramentas são mostradas. Elas foram encontradas através de referências mencionadas em artigos, ou eram previamente conhecidas e tiveram sua adequação ao MDA verificado. Foi dado foco àquelas tradicionalmente conhecidas pelos desenvolvedores, e ferramentas que possam englobar o processo de produção do software como um todo. Além disso, algumas das referências de ferramentas não puderam ser verificadas. No caso da ArcStyler, o antigo site da empresa está fora do ar e um novo site foi encontrado, no entanto, com uma outra solução disponível. Um cadastro foi efetuado na tentativa de obter o link que disponibiliza uma versão gratuita da ferramenta. Entretanto, este link não foi disponibilizado. No caso da Mastercraft, o link da empresa também encontra-se indisponível, e nenhuma fonte segura e/ou legal foi encontrada.

As ferramentas são o elemento principal da cadeia de transformações que compõe o MDA. Para que o Framework de Práticas Ágeis seja considerado compatível ou não com MDA, deve ser verificada sua adequação às ferramentas MDA disponíveis.

As ferramentas encontradas serão testadas, a fim de verificar à quais aspectos de MDA elas fornecem apoio. Após essa análise, será possível verificar a adaptabilidade das características da ferramenta com relação às características do Framework de Práticas Ágeis.

Sem esta análise, identificar a adequação das ferramentas ao uso do Framework de Práticas Ágeis, juntamente com o MDA, seria uma difícil de ser realizada com precisão.

#### 2.3.1.1 Enterprise Architect

Uma das ferramentas mais conhecidas na especificação de modelos UML, a Enterprise Architect, também tentou se adaptar a nova arquitetura dirigida a modelos, oferecendo transformações diretas entre modelos de diferentes níveis de abstração.

A Enterprise Architect disponibiliza uma versão gratuita de trinta dias para testes, que foi utilizada no trabalho. A referida versão permite ao usuário escolher a edição que quiser da ferramenta, proporcionando, desta forma, totais condições de escolher qual delas oferece as funcionalidades mais adequadas às necessidades do usuário. Sua versão completa é uma das mais robustas encontradas no mercado.

Nela encontram-se as transformações que apoiam MDA, e tornam seu uso viável em processos desse tipo. A Enterprise Architect permite que sejam criados modelos independentes de plataforma, e que estes sejam mapeados para uma grande coleção de tecnologias. Por exemplo, cria-se um diagrama UML, e nele especifica-se que o modelo não pertence a nenhuma linguagem, então a partir dele pode-se gerar diagramas para a plataforma Java, onde aspectos inerentes a ela são adicionados ao modelo, dentre outras, e posteriormente seu código fonte.

#### 2.3.1.2 Moderne

A MODERNE demonstrou ser uma ferramenta adequada para modelar o framework proposto em FAGUNDES (2005) e os processos de softwares construídos a partir dele (Maciel, 2011). A Moderne apóia de maneira satisfatória a especificação e execução de modelos que usam a abordagem chamada MDA.

A ferramenta oferece dois módulos: o editor, que permite a especificação de modelos; e o executor, que apóia a execução dos mesmos. Entre esses dois módulos fica um repositório (MySql), que armazena os dados do processo especificado no editor e permite ao executor acessá-los e executá-los. Esses módulos foram projetados como um produto sobre a plataforma Eclipse, tornando possível o uso de seu arcabouço gráfico e de diversos *plugins* (SILVA, MAGALHÃES, MACIEL, MANTINS, NOGUEIRA, QUEIROZ, 2009).

Esta ferramenta utiliza dois metamodelos como base para instanciar os modelos e processos. Esses metamodelos utilizam conceitos do SPEM, um meta-modelo que define estereótipos UML para a modelagem de processos de software. Estes conceitos são usados para representar processos de software, que têm seus conceitos estendidos para atender ao contexto da MDA e de testes. Os metamodelos são o Metamodelo Core e o Metamodelo de Testes, apresentados a seguir pelas Figuras 6 e 7.

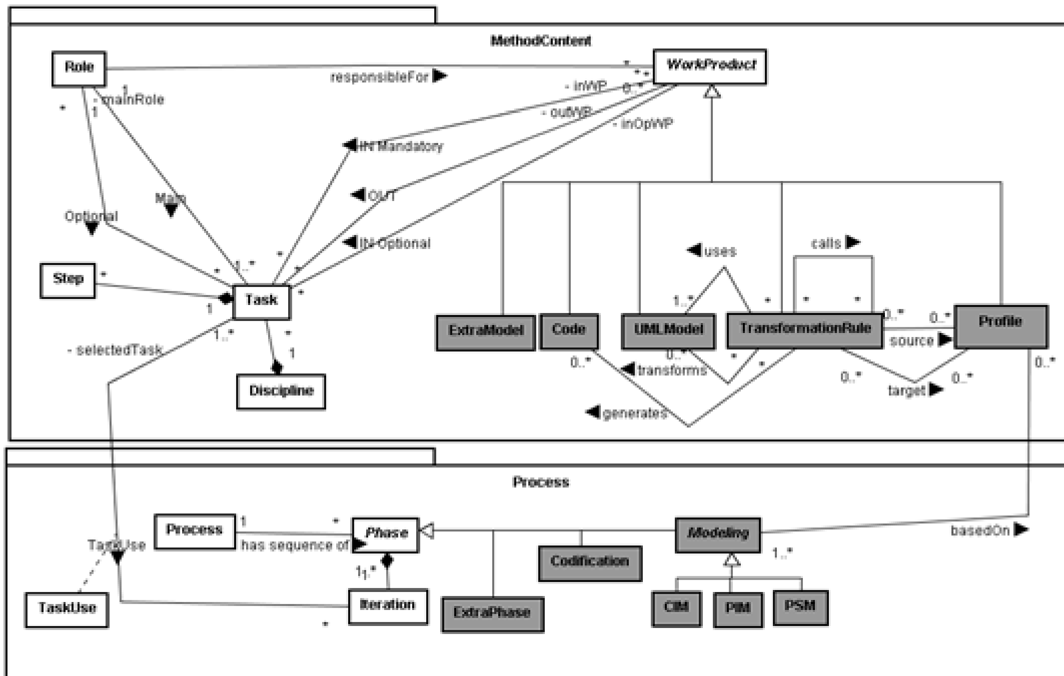



**Figura 6** - Metamodelo Core

Fonte: Maciel(2011).

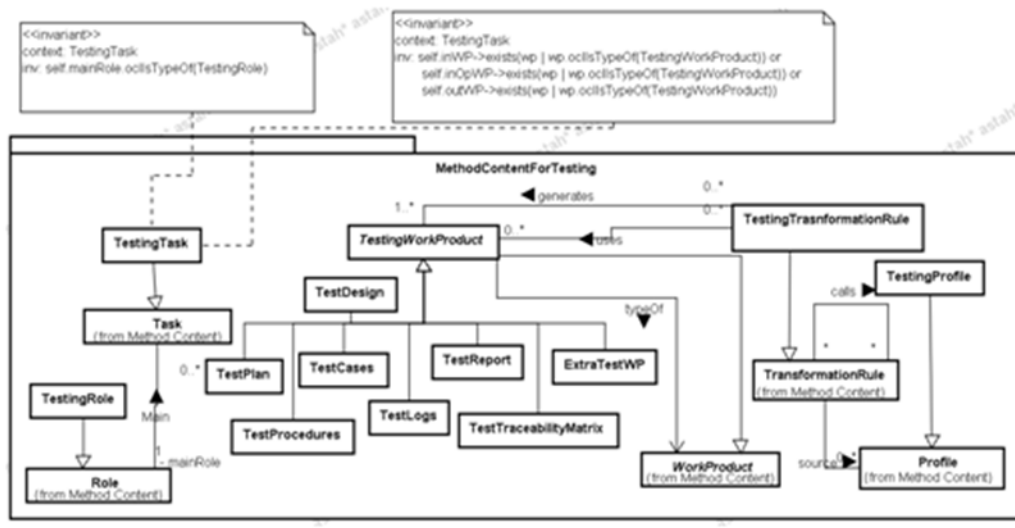



**Figura 7** - Metamodelo de Teste

Fonte: Maciel(2011).

O uso de MDA requer a definição de elementos de processo associados com atividades de modelagem e regras de transformação para compor a cadeia de transformação.

Esses elementos geralmente não são explicitamente encontrados em processos de desenvolvimento. Desta forma, alguns conceitos do SPEM 2.0 foram selecionados e especializados para cobrir aspectos específicos da MDA. Segundo o SPEM, um processo de software deve ser especificado em uma dimensão estática e outra dinâmica. A parte estática do processo contém os conceitos de disciplines, tasks, steps, roles e workproducts. Estes conceitos agrupados formam o que é chamado de Method Content. A parte dinâmica do processo possui agregada a ela os conceitos *phases*, *iterations* e *taskUses*, chamada de *Process* (MACIEL, SILVA, MAGALHÃES, ALVES, GOMES, 2010). Portanto, a ferramenta comporta a construção de dois modelos: um referente aos Method Contents e outro referente aos Processes.

O conceito de Method Content envolve elementos correlacionados que podem ser usados em diversos processos. Já os processos, podem se basear nas definições estáticas dos Methods Content, podendo ser modelados com uma variedade destes (Maciel, Silva, Magalhães, Alves, Gomes, 2010).

O metamodelo de teste foi construído para permitir a definição explícita de elementos relativos a processos de teste dirigido a modelos, dentro do contexto de um processo MDA. Este metamodelo estende o metamodelo Core para atender as questões referentes ao teste dirigido por modelos (Maciel, Silva, Magalhães, Alves, Gomes, 2010).

#### 2.3.1.3 MagicDraw

Outra ferramenta comercial de grande porte e sucesso, a MagicDraw, segundo seus criadores, é uma ferramenta de desenvolvimento dinâmico e versátil que facilita a análise e o projeto de sistemas e bases de dados orientados a objetos.

Assim como a Enterprise Architect, MagicDraw possui uma versão gratuita para testes e averiguações. Esta versão permanece funcionando por 90 dias e impõe limitações também no número de elementos dos modelos criados nela. Um usuário que experimenta a MagicDraw não pode construir modelos com mais de 25 casos de uso, 20 classes, 25 nodos, dentre outras limitações. No entanto, oferece a geração de código para mais de uma plataforma alvo, além de alguns dos tipos de transformações revistas na versão completa da ferramenta.

Nas versões pagas da ferramenta, a geração de código, funcionalidade mais importante para a abordagem MDD, é adicionada na ferramenta apenas na terceira, em uma escala de cinco versões, ou seja, para poder obtê-la de maneira definitiva, o usuário deve comprar a ferramenta, pagando US$ 1.124,00 para obtê-la e US$ 360,00 anuais para mantê-la, além de optar para qual linguagem ele deseja este suporte (Java, C++ ou C#). No entanto, apenas a geração de código não é suficiente. Para ter o apoio total, com transformações em duas vias entre diferentes níveis de modelos, deve-se obter a versão completa da ferramenta, que custa US$ 1.999,00, além de US$ 640,00[2] extras para mantê-la.

[2] Valores referentes a 31 de maio de 2012

As transformações incorporadas pela MagicDraw são parecidas com as da Enterprise Architect. Ambas oferecem transformações entre modelos de diferentes níveis de abstração e tecnologias. A principal diferença, é que a MagicDraw, não exige que uma transformação que não gera código fonte seja vinculada a uma plataforma específica. Por exemplo, quando um modelo genérico descrito em uma linguagem de definição de dados for transformado em um modelo UML, não é necessário que seja definida uma linguagem para este modelo, esta será definida apenas quando for solicitada a geração automática do código a partir dele.

#### 2.3.1.4 AlphaSimple

AlphaSimple é uma ferramenta web centrada em modelos que permite ao usuário, uma vez que tenha modelado sua aplicação, criar os testes de aceitação, gerar um protótipo para uma aceitação rápida e o código para a plataforma alvo (CHAVES, 2010).

A modelagem na AlphaSimple dá-se em uma forma textual para UML, chamada de textUML. A prototipação é feita de maneira automática, e permite ao usuário executar as funções descritas no modelo, tudo através do browser. Os testes são efetuados através de modelos criados com o propósito de testar outros modelos. Estes modelos devem aplicar estereótipos escritos em perfis de teste, um deles disponibilizado no site da ferramenta. Então, quando um modelo for compilado, os testes de aceitação serão executados automaticamente, e caso algum destes testes falhe, ele será reportado como um erro de construção no editor.

A geração de código é a principal função da AlphaSimple. Ela utiliza um template de código aberto, chamado de stringTemplate para gerar código para as plataformas alvo. Atualmente, a geração de código embutida na ferramenta é apenas Java POJOs e JPA DAOs. No entanto, a utilização do stringTemplate oferece a liberdade ao usuário de criar templates para as plataformas alvo que ele desejar.

Outra funcionalidade, porém considerada secundária pelo criador da ferramenta, é a criação de um diagrama de classe UML automaticamente a partir do modelo escrito em textUML, e também visualizado no browser de maneira simultânea.

# 3 Trabalhos relacionados

Com a vasta adoção dos métodos ágeis e popularização do MDA na indústria de software, naturalmente foram surgindo esforços para conciliar essas duas maneiras de se produzir tecnologia.

Para tentar encontrar relatos do uso de MDA juntamente com métodos ágeis, pesquisas foram realizadas nas bases de dados oferecidas pela universidade e em outras ferramentas de busca como o Google Acadêmico. Algumas delas utilizaram os termos “MDD”, “MDA”, “a*gile*” e “ágil” combinados, mas outras buscam fora mais específicas, utilizando o nome da obra, por exemplo, conhecida previamente através de referências em outros trabalhos e artigos.

Foram encontrados relatos do uso de MDD, e também de MDA, junto com métodos ágeis, e documentos que discorriam ou definiam o chamado *Agile MDD*.

Alguns dos trabalhos têm o objetivo de conciliar MDA com um processo de software específico e bem formado. Este trabalho difere dos demais pelo fato da tentativa de conciliação dar-se entre MDA e um framework que pode gerar os mais variados processos ágeis. Os trabalhos encontrados foram:

## 3.1 Agile Model-Driven Development in Practice

ZHANG e PATEL mencionam dois trabalhos que demonstram como adicionar práticas de *Extreme Programming* (XP) ao MDD, e mostram como o MDD pode ser combinado com um processo ágil chamado *System-Level Agile*

*Process* (SLAP), para acelerar a taxa de desenvolvimento, melhorar a qualidade e encurtar o tempo dos ciclos de entrega (ZHANG, PATEL, 2010).

SLAP é um método ágil adotado pela Motorola, e tem sua base definida no *Scrum,* além de utilizar práticas de *Exteme Programming.* O trabalho apresentado por ZHANG e PATEL cria correspondências e relacionamentos entre as práticas de um processo MDD e as práticas ágeis do SLAP, para criar um único processo classificado como *Agile* MDD. Neste processo criam práticas como "modelagem paralela", "modelagem iterativa e incremental" entre outras, a fim de miscigenar MDA com desenvolvimento ágil e verificar como seus conceitos podem ser utilizados juntos de maneira adequada.

Os modelos utilizados pela equipe de desenvolvimento são definidos em UML, e transformados em código C++ por um compilador UML.

Eles conseguem cumprir com a proposta do trabalho, desenvolvendo o sistema proposto com sucesso e destacam a importância da cadeia de transformações e das ferramentas disponíveis para isso dentro do processo.

### 3.2 Um Relato de Experiência no Desenvolvimento Ágil de Sistemas com a MDA

BASSO e PILLA trazem um relato do uso de MDD com métodos ágeis no contexto de uma empresa privada com seus próprios processos e ferramentas. O trabalho menciona como o processo de software e a ferramenta foram sofrendo modificações para adaptar-se ao contexto MDA.

O relato dá grande visibilidade à ferramenta, e destaca os perfis dos utilizadores e a importância deles dentro do processo de maneira geral. O

trabalho relaciona diretamente a qualidade das definições de transformação com a qualidade do produto final e classifica as transformações de acordo com os modelos produzidos. Chama-se de *Model-to-Model* (M2M) as transformações cujo objetivo é gerar modelos específicos de plataforma (PSM) através dos modelos de entrada (PIM); enquanto as transformações que possuem um PSM como entrada e geram código fonte são chamadas de Model-to-Code (M2C).

BASSO e PILLA publicaram seu relato em 2010 e afirmam não ter encontrado referências do uso de MDA com práticas ágeis. Apresentam resultados referentes ao uso destas práticas com a ferramenta utilizada durante a produção do artigo, e explicitam o revés gerado pela adaptação da equipe à ferramenta, modificada para moldar-se aos conceitos MDA.

### 3.3 Agile Model Driven Development Is Good Enough

Em pouco mais de duas páginas, AMBLER dá sua opinião e discute alguns aspectos de MDD. O artigo datado de 2003, traz segundo o autor, uma visão do atual estado do MDD naquela data e não leva em consideração o que se promete para o futuro.

Para AMBLER, é preciso distinguir o MDD definido pela OMG, baseado em conceitos que envolvem o fato de que as pessoas irão utilizar ferramentas sofisticadas para criar modelos também sofisticados que irão compor transformações automáticas, do que ele intitula de Agile MDD (AMDD). Segundo o autor, apenas o AMDD teria alguma chance de dar certo.

AMDD utilizaria apenas “o suficientemente bom”. Modelos seriam apenas suficientemente bons para cumprir seus objetivos e nada mais. De acordo com AMBLER, este conceito funcionaria, pois, é relativo. Muitas vezes um esboço de diagrama UML pode ser considerado suficiente, mas em outros casos modelos mais complexos e detalhados são necessários. Contudo, para os modelos mais simples, as ferramentas mais simples seriam suficientes, e ferramentas sofisticadas apenas seriam necessárias para casos em que os diagramas fossem tão sofisticados tanto.

AMBLER defende o uso de ferramentas simples, e afirma que apenas se a geração de código fosse o objetivo, ferramentas de modelagem sofisticadas seriam necessárias, no entanto, o AMDD prega um desenvolvimento iterativo e incremental, fazendo com que os programadores desenvolvessem os modelos em sintonia com o código.

Segundo AMBLER, MDD está baseado em conceitos extremamente instáveis. Afirma que não se tem uma linguagem de modelagem padrão que englobe todos os aspectos do mundo real de forma efetiva. E afirma que UML não trata de questões cruciais como interfaces e bases de dados, presentes na maioria dos softwares implementados. Aponta também a falta de habilidade das pessoas na construção de modelos. E por fim afirma que as ferramentas não apoiam o MDD como deveriam.

AMBLER ainda cita algumas ferramentas durante o texto, mas diz que essas não são usadas em larga escala e que os padrões definidos pela *Object Managemet Group* (OMG) para que se possa interligar ferramentas diferentes, como XML e XMI, na prática não cumprem seus papéis, devido principalmente a motivos comerciais.

### 3.4 Test-Driven Modeling for Model-Driven Development

Este artigo com autoria de Yuefeng Zhang traz um processo chamado por ele de *test-driven modeling* (TDD). A base do trabalho é um projeto desenvolvido dentro da empresa Motorola, que utiliza este processo. No entanto, os modelos apresentados, são apenas exemplos para ilustrar o processo, pois os diagramas que fizeram parte do projeto são confidencias.

O processo une conceitos evolvidos no método ágil conhecido como *Extreme Programming* (XP), onde os desenvolvedores criam primeiro os testes e depois o código fonte, com alguns conceitos e transformações do MDD, criando assim o TDD.

O processo mostrado no artigo é dividido em seis fases: Requisitos do Sistema, Modelagem de Alto Nível, Modelagem de Baixo Nível, Controle de Qualidade da Modelagem, Geração Automática do Código e o Teste na Plataforma Alvo. Contudo, o artigo dá foco nas fases de modelagem de alto nível e modelagem de baixo nível, e então discute o contole de qualidade da modelagem e a geração automática do código.

Para o projeto, foram utilizados diagramas baseados na *Specification and Description Language* (SDL) e a ferramenta comercial *Telelogic TAU SDL Suite*. Durante a modelagem de alto nível deste processo, são criados diagramas de fluxo do sistema (DFS), e também é representada a arquitetura do sistema. Definir se os objetos são uma instância passiva ou ativa é uma tarefa característica da fase de modelagem de baixo nível.

Segundo ZHANG, a modelagem de alto nível é dirigida a testes, pois os DFSs são criados primeiro, e o projeto da arquitetura do sistema é desenvolvido a partir deles. Então os DFS são executados na ferramenta, para verificar se a arquitetura desenvolvida incorpora os requisitos do sistema definidos pelos DFSs. Só depois disso os DFSs são refinados e são executados os testes de unidade.

Já na modelagem de baixo nível deste processo, o principal objetivo é definir o comportamento da parte dinâmica e estática dos subsistemas que compõem o sistema geral. Para a definição do comportamento dinâmico dos subsistemas, também são utilizados DFS. Já o comportamento estático dos subsistemas segundo ZHANG, são definidos pelo comportamento estático dos sub-subsistemas e/ou pelo comportamento estático dos objetos passivos ou ativos contidos no subsistema. Um objeto ativo tem seu comportamento estático no sistema definido por um diagrama de transição de estados.

A fase de modelagem de baixo nível é iterativa e incremental, já que segundo o autor, ela pode exigir que uma quantidade muito grande de trabalho seja realizado durante sua execução. Assim cada iteração implementa algum subconjunto dos DFSs dos subsistemas, incrementando o sistema final.

Os modelos utilizados nestas fases são os artefatos de entrada para a geração direta do código. Segundo o autor, esta fase é dirigida a testes já que de testes são realizados na plataforma alvo, produzem resultados que afetam diretamente na geração do código final após serem resolvidas rastreadas e modificadas nos modelos.

ZHANG ainda justifica a escolha da ferramenta, com uma análise dos aspectos que considerou importantes para que a mesma fosse adequada ao

seu projeto. Ele revela que a Motorola teve a produtividade e a qualidade do código desenvolvido aumentadas, com uma redução de até vinte por cento de erros encontrados no código gerado.

# 4 Adequação do Framework e de Ferramentas ao MDD

Depois de analisar o Framework de Práticas Ágeis, MDA e as ferramentas disponíveis para a sua utilização, é necessário verificar a possibilidade de conjugar estes elementos em um único processo de software. Este capítulo verifica a conjugação das ferramentas e dos processos aos conceitos do MDA para que possa ser possível juntá-los em um único processo.

## 4.1 Processos do Framework

Dos vários processos possíveis de serem construídos a partir do Framework de Práticas Ágeis, todos aqueles que envolverem atividades de modelagem em suas fases iniciais são bons candidatos a se conjugarem ao MDA. Contudo, os processos devem focar os esforços nas práticas de modelagem em suas primeiras fases, e especializar seus modelos nas fases seguintes, não permitindo que o esforço inicial seja perdido.

Os seguintes critérios foram utilizados na definição de alguns exemplos de processos de software gerados através do Framework de Práticas Ágeis: todas as combinações possuem pelo menos quatro práticas que geram artefatos, distribuídas em pelo menos três atividades diferentes desse framework. Também foi incluída no mínimo uma atividade que possibilita a geração de diversos artefatos diferentes, dando liberdade ao processo para se adaptar a uma abordagem MDA e produzir resultados mais satisfatórios de acordo com as ferramentas utilizadas dentro desse processo.

A seguir, as combinações geradas serão mostradas na forma de tabelas, seguidas por comentários sobre sua adequação aos conceitos MDA. Estas tabelas contêm todas as práticas do Framework de Práticas Ágeis, onde as destacadas em negrito se referem àquelas escolhidas para serem usadas no processo.

Combinação 1:

**Tabela 2** – Combinação 1

| **Lista de Requisitos** | **Casos de Uso** | ~~Desenvolvimento Lado a Lado~~ | **Breve Documentação** |
|---|---|---|---|
| **Modelagem Geral** | **Projeto da Iteração** | ~~Refatoração~~ | **Entrega do Sistema** |
| **Documentação Incial** | **Escrita dos Testes de Unidade** | **Reuniões Diárias** | |
| ~~Projeto da Arquitetura do Sistema~~ | ~~Escrita dos Testes de Aceitação~~ | ~~Desenvolvimento Coletivo de Código~~ | |
| **Planejamento da Iteração** | **Desenvolvimento** | ~~Inspeção de Código~~ | |
| ~~Estórias do Usuário~~ | ~~Programação em Pares~~ | **Reunião de Revisão da Iteração** | |

A Combinação 1 utiliza a prática Modelagem Geral, que possibilita outra prática, Projeto da Iteração, utilizar os artefatos produzidos nela para construir seus próprios artefatos, através de transformações MDA. Além disso, a combinação foi feita para viabilizar a utilização dos artefatos como documentação, realizando a prática Breve Documentação de forma automática, analisando a eficiência do MDD neste caso.

Combinação 2:

**Tabela 3** – Combinação 2

| **Lista de Requisitos** | ~~Casos de Uso~~ | **Desenvolvimento Lado a Lado** | **Breve Documentação** |
|---|---|---|---|
| ~~Modelagem Geral~~ | **Projeto da Iteração** | ~~Refatoração~~ | **Entrega do Sistema** |
| **Documentação Inicial** | ~~Escrita dos Testes de Unidade~~ | **Reuniões Diárias** | |
| **Projeto da Arquitetura do Sistema** | **Escrita dos Testes de Aceitação** | **Desenvolvimento Coletivo de Código** | |
| **Planejamento da Iteração** | **Desenvolvimento** | ~~Inspeção de Código~~ | |
| ~~Estórias do Usuário~~ | ~~Programação em Pares~~ | **Reunião de Revisão da Iteração** | |

A segunda combinação foi construída para trazer uma possibilidade de maior de utilização das transformações MDA, já que a prática de Projeto da Arquitetura do Sistema permite a geração de diversos artefatos. Portanto, a equipe de desenvolvimento poderá escolher produzi-los adequadamente para que as transformações gerem os artefatos de práticas que serão realizadas em atividades posteriores de forma automática.

Combinação 3:

**Tabela 4** – Combinação 3

| **Lista de Requisitos** | ~~Casos de Uso~~ | ~~Desenvolvimento Lado a Lado~~ | ~~Breve Documentação~~ |
|---|---|---|---|

| **Modelagem Geral** | **Projeto da Iteração** | **Refatoração** | **Entrega do Sistema** |
|---|---|---|---|
| **Documentação Inicial** | **Escrita dos Testes de Unidade** | ~~Reuniões Diárias~~ | |
| **Projeto da Arquitetura do Sistema** | ~~Escrita dos Testes de Aceitação~~ | **Desenvolvimento Coletivo de Código** | |
| **Planejamento da Iteração** | **Desenvolvimento** | **Inspeção de Código** | |
| **Estórias do Usuário** | ~~Programação em Pares~~ | **Reunião de Revisão da Iteração** | |

A terceira combinação traz um adicional com relação à segunda. A prática Modelagem Geral, presente nesta combinação, irá produzir um diagrama de classe, que permitirá a construção de uma cadeia de transformações a partir dele. Assim, a Combinação 3 traz a possibilidade da análise de adaptação dos processos à cadeia de transformações MDA e vice-versa. Caso o diagrama de classe gerado não cumpra com este papel, as práticas seguintes continuam tendo a possibilidade de gerar seus artefatos e iniciar uma cadeia de transformações.

Combinação 4:

**Tabela 5** – Combinação 4

| **Lista de Requisitos** | **Casos de Uso** | ~~Desenvolvimento Lado a Lado~~ | **Breve Documentação** |
|---|---|---|---|
| **Modelagem Geral** | **Projeto da Iteração** | ~~Refatoração~~ | **Entrega do Sistema** |

| **Documentação Inicial** | ~~Escrita dos Testes de Unidade~~ | **Reuniões Diárias** | |
|---|---|---|---|
| ~~Projeto da Arquitetura do Sistema~~ | **Escrita dos Testes de Aceitação** | ~~Desenvolvimento Coletivo de Código~~ | |
| **Planejamento da Iteração** | **Desenvolvimento** | ~~Inspeção de Código~~ | |
| ~~Estórias do Usuário~~ | **Programação em Pares** | **Reunião de Revisão da Iteração** | |

A quarta combinação também deverá possuir uma cadeia de transformações a partir de um diagrama de classes. No entanto, não possui tanta flexibilidade quanto a Combinação 3, já que o projeto da iteração, que irá utilizar este diagrama para tentar gerar seus artefatos de maneira automática é realizada em um estágio mais avançado do processo, e não tem outra opção de artefato de entrada para sua cadeia de transformações.

Combinação 5:

**Tabela 6** – Combinação 5

| **Lista de Requisitos** | ~~Casos de Uso~~ | **Desenvolvimento Lado a Lado** | **Breve Documentação** |
|---|---|---|---|
| **Modelagem Geral** | ~~Projeto da Iteração~~ | ~~Refatoração~~ | **Entrega do Sistema** |
| **Documentação Inicial** | ~~Escrita dos Testes de Unidade~~ | **Reuniões Diárias** | |
| **Projeto da Arquitetura do Sistema** | ~~Escrita dos Testes de Aceitação~~ | **Desenvolvimento Coletivo de Código** | |
| **Planejamento da** | **Desenvolvimento** | ~~Inspeção de Código~~ | |

| **Iteração** | | | |
|---|---|---|---|
| **Estórias do Usuário** | ~~Programação em Pares~~ | **Reunião de Revisão da Iteração** | |

A quinta combinação feita não possui a prática do Projeto da Iteração. Sendo assim, o processo utiliza os artefatos para uma visão geral do sistema em um estágio inicial do processo e também para a documentação. Esta combinação é apropriada para a análise do MDA em processos que têm os artefatos um pouco mais voltados para a documentação do desenvolvimento de software.

Após Após um olhar crítico sobre algumas das potenciais combinações do Framework de Práticas Ágeis para o uso com MDA, a seção seguinte trará testes com as ferramentas alvo do trabalho. Estes testes serão apresentados juntamente com alguns comentários dos resultados obtidos.

## 4.2 Ferramentas

### *4.2.1 Moderne*

O teste na MODERNE foi iniciado pela tentativa de se modelar o Framework de Práticas Ágeis de maneira geral. Para adequar a modelagem desse framework, e seus processos, aos metamodelos que abordam os conceitos de MDA, foi criado um modelo Method Content, que engloba o framework como um todo. Para que o modelo se adequasse à ferramenta, utilizando os conceitos MDA nela inseridos, e ao mesmo tempo fosse próprio ao Framework de Práticas Ágeis, algumas correspondências entre conceitos foram inseridas.

As atividades do Framework de Práticas Ágeis foram modeladas como disciplinas na ferramenta. Sendo assim, as práticas contidas em cada uma das atividades desse framework são correspondidas pelas tarefas relacionadas a cada disciplina na ferramenta. A ferramenta também permite que as tarefas sejam divididas em passos. Contudo, a modelagem destas tarefas foi feita de forma que todas elas são executadas em um único passo. Os artefatos de saída das tarefas foram modelados como workproducts, que são especializados de acordo com o seu tipo.

A Tabela 7, mostrada a seguir, mostra estas correspondências. Levando em consideração que nem todos os elementos presentes nos metamodelos da ferramenta possuem um correspondente no Framework de Práticas Ágeis, a tabela mostra apenas os elementos desse framework com seus devidos correspondentes na ferramenta.

**Tabela 7** – Correspondências entre o Framework e a Ferramenta.

| **Framework** | **Ferramenta** |
|---|---|
| Atividade | Discipline |
| Prática | Task |
| Papel | Role |
| Artefatos de saída | Workproducts |

Os diagramas de classes da modelagem da cada uma das disciplinas do Framework de Práticas Ágeis são mostrados a seguir, frisando que, para uma melhor visualização, serão mostrados três diagramas para a disciplina

Desenvolvimento Incremental do Sistema. As Figuras 8 a 17 mostram os diagramas de classe gerados durante a modelagem das atividades do framework.

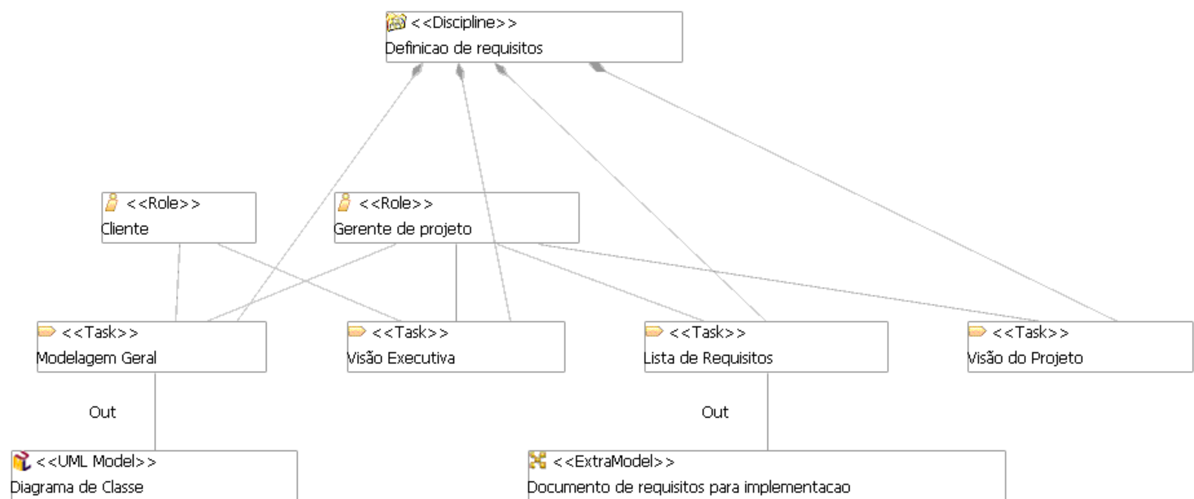



**Figura 8** - Diagrama de Classes Disciplina Definição de requisitos

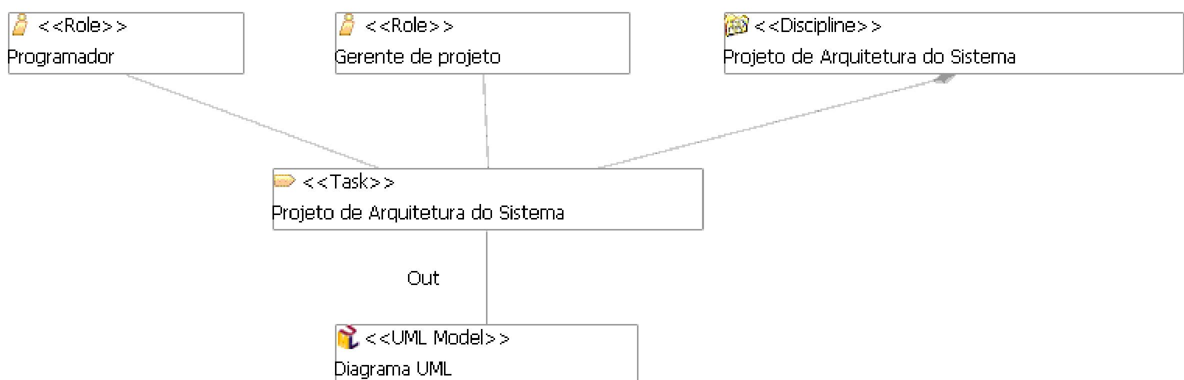



**Figura 9** - Diagrama de Classes Disciplina Projeto Arquitetura do Sistema

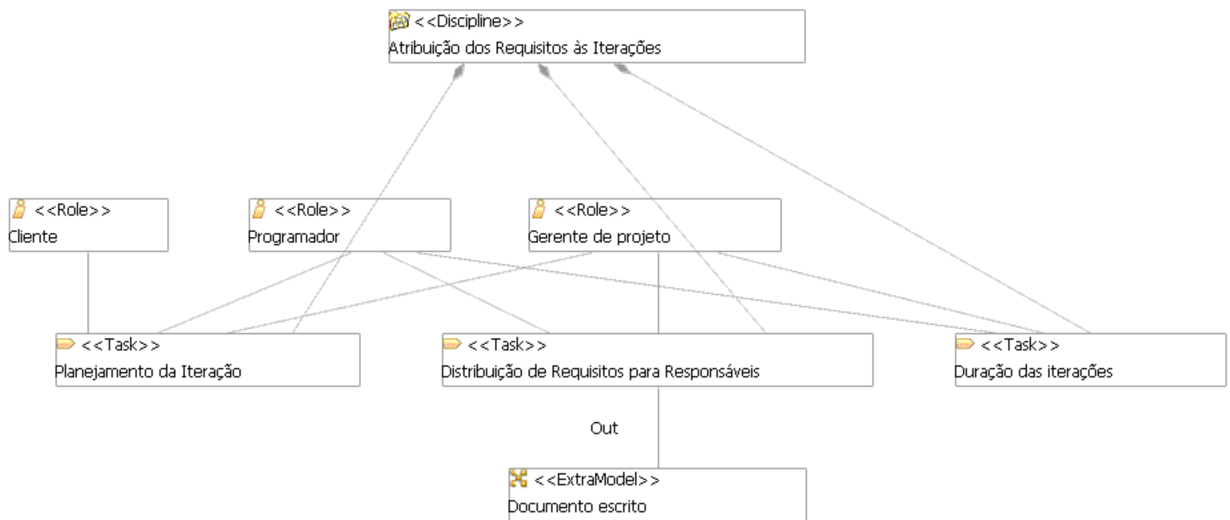



**Figura 10** - Diagrama de Classes Disciplina Atribuição dos Requisitos às Iterações

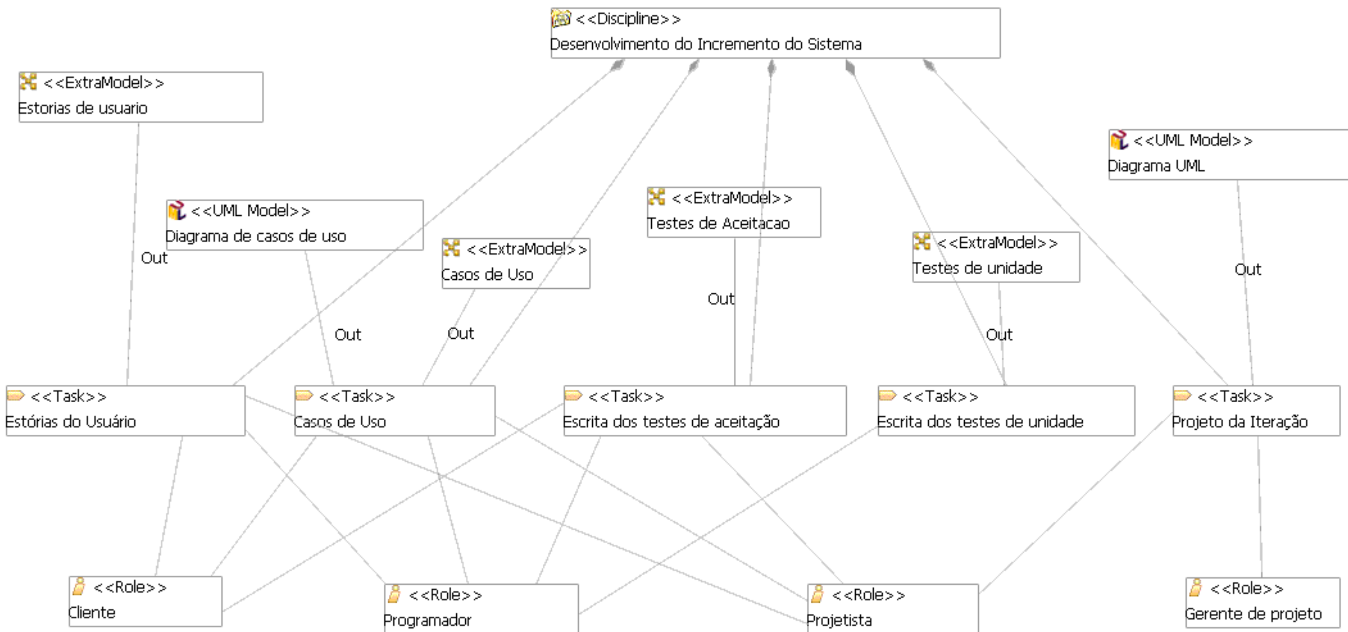



**Figura11** - Diagrama de Classes Disciplina Desenvolvimento Incremental do Sistema I

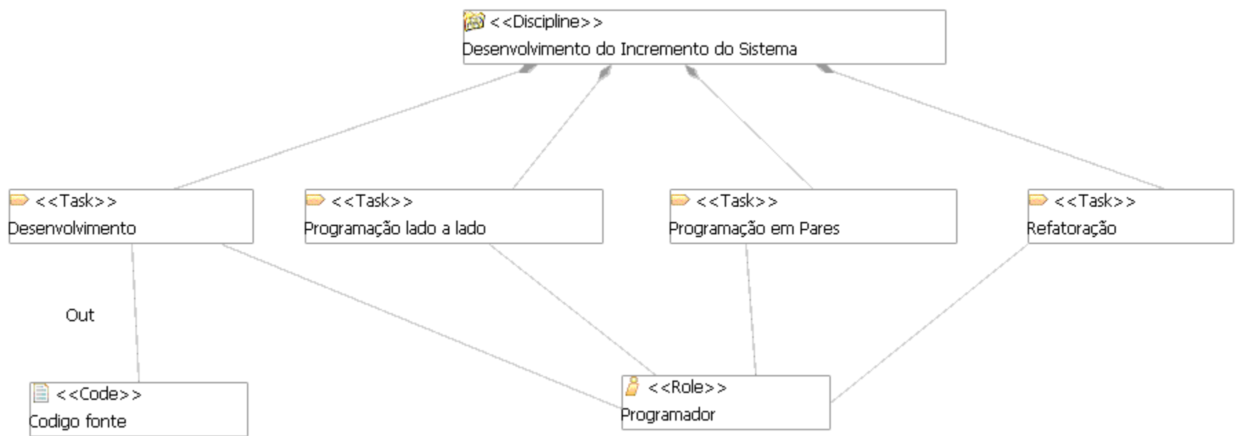



**Figura 12** - Diagrama de Classes Disciplina Desenvolvimento Incremental do Sistema II

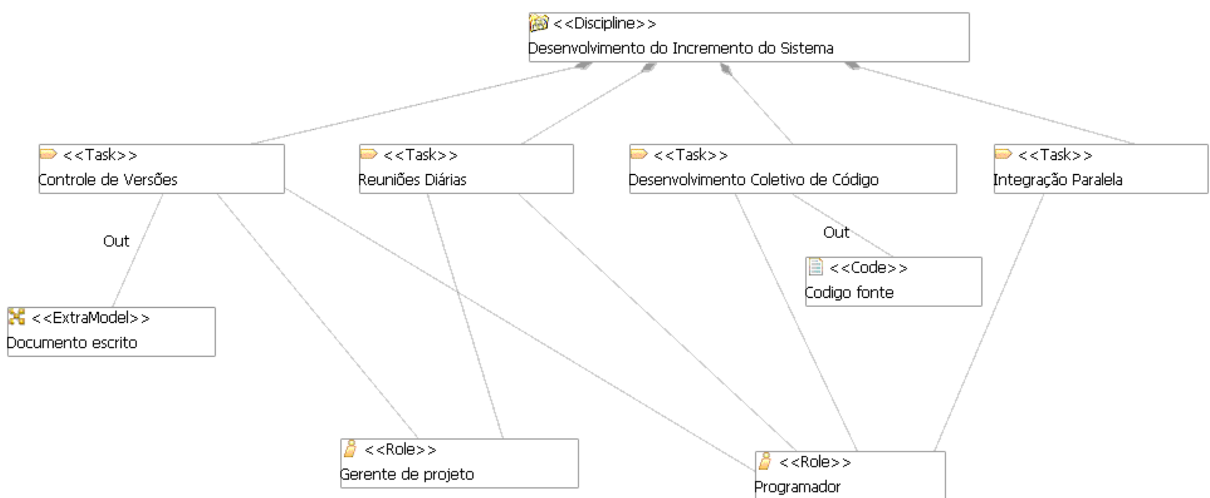



**Figura 13** - Diagrama de Classes Disciplina Desenvolvimento Incremental do Sistema III

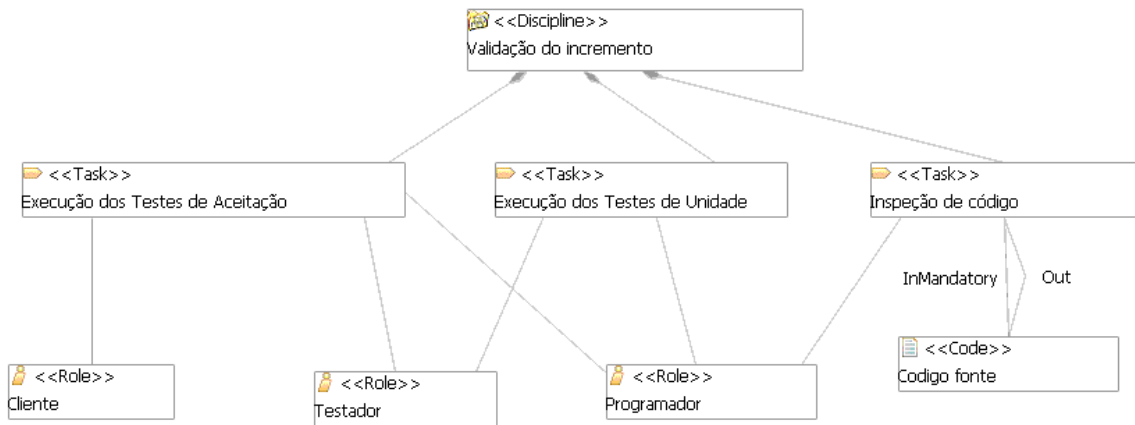



**Figura 14** - Diagrama de Classes Disciplina Validação do Incremento

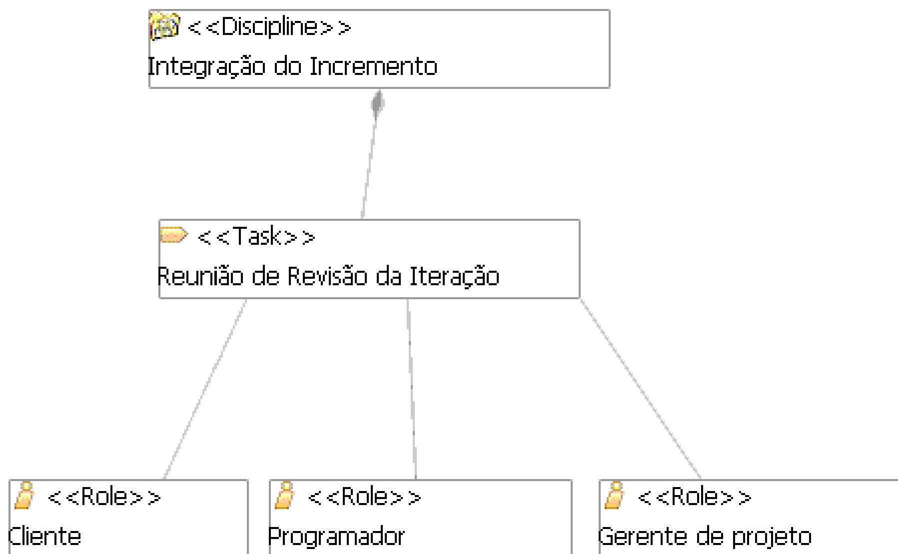



**Figura 15** - Diagrama de Classes Disciplina Integração do Incremento

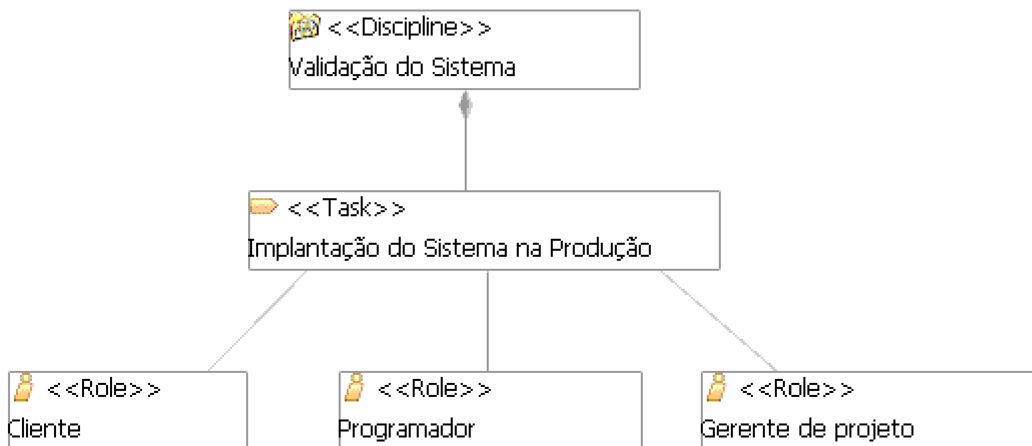



**Figura 16** - Diagrama de Classes Disciplina Validação do Sistema

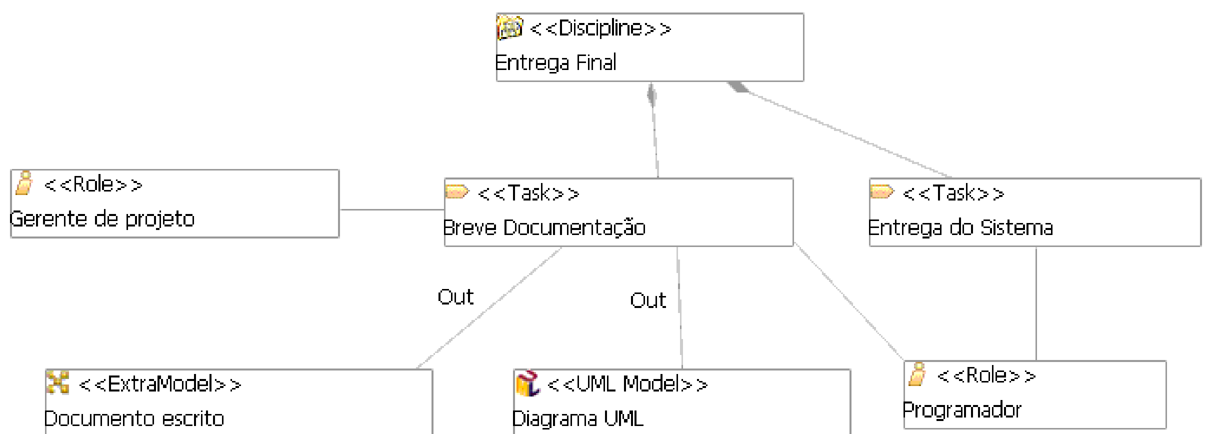



**Figura 17** - Diagrama de Classes Disciplina Entrega Final

A ferramenta também permite a criação de fases no processo de software criado. Para os processos criados durante o trabalho, será considerada apenas uma fase, chamada codificação. Esta decisão foi tomada levando em consideração que um processo ágil possui tarefas tanto de modelagem quanto de codificação em todas as suas iterações, e que cada uma destas iterações têm como objetivo gerar um artefato de entrega para o cliente,

agregando funcionalidades ao sistema. Logo, o processo ágil foi interpretado como uma grande fase de codificação contínua.

Infelizmente, as atividades utilizando a MODERNE não avançaram além da modelagem. A ferramenta apresentou inúmeros erros durante a tentativa de comunicação entre seus módulos de execução e de edição. Vários e-mails foram trocados com os envolvidos no projeto de implementação da ferramenta. Porém, o problema não foi solucionado e não foi possível instanciar um processo específico do Framework de Práticas Ágeis para que fosse utilizado dentro do executor da ferramenta.

### *4.2.2 AlphaSimple*

Para que as vantagens da AlphaSimple fossem verificadas em sua plenitude, um exemplo disponibilizado no site da ferramenta, disponível no endereço eletrônico http://alphasimple.com/project/show/548, e de autoria de Rafael Chaves, idealizador da ferramenta, foi utilizado para analisar seu potencial. O projeto escolhido chama-se Codgem – POJO Templates, por envolver templates de geração de código ricos, facilitando a visualização da qualidade produzida por eles.

O exemplo traz um sistema bancário com operações básicas, com uma classe chamada Person, representando um cliente possuidor de uma conta (representada pela classe Account), que executa duas operações, uma de retirada e outra de depósito na conta. Essa funcionalidade é representada por uma especialização da classe Operation em Withdrawal e Deposit. Além disso, operações de manipulação dos objetos contidos nas classes também são apresentadas. A figura 18 mostrará o diagrama de classes gerado

automaticamente pela AlphaSimple através da especificação textual do programa.

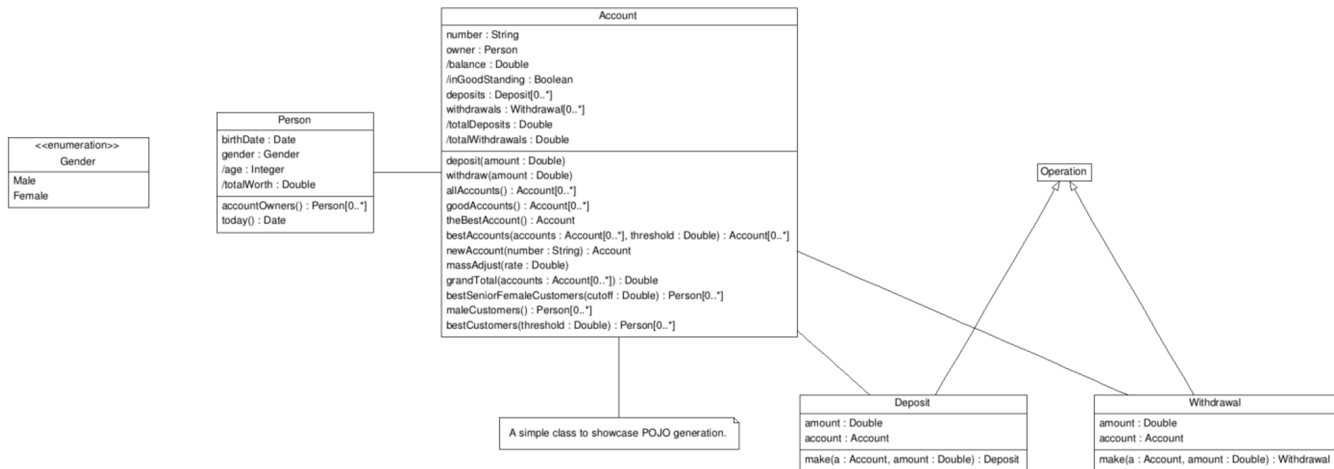



**Figura 18** – Diagrama de classes AlphaSimple

Fonte: Chaves (2010)

Um trecho desta descrição feita em textUML está presente na figura a seguir (Figura 19). Anexo ao trabalho está todo o documento citado.

```
1  /* Copyright Abstratt Technologies 2011. All rights reserved. */
2
3  package banking;
4
5  enumeration Gender
6    Male, Female
7  end;
8
9  class Person
10     attribute birthDate : Date;
11     attribute gender : Gender;
12
13     derived attribute age : Integer := () : Integer {
14         return Date#today().differenceInDays(self.birthDate);
15     };
16
17     derived attribute totalWorth : Double := () : Double {
18         return (self<-PersonAccounts->accounts.reduce(
19             (a : Account, partial : Double) : Double { return partial + a.balance
20         ) as Double);
21     };
```

**Figura 19** – textUML AlphaSimple

Fonte: Chaves (2010)

A descrição do sistema é o suficiente para a geração automática de um protótipo totalmente funcional, que roda diretamente no browser, permitindo ao usuário inserir dados no sistema e testar suas funcionalidades de maneira concreta e satisfatória. Todas as regras de mapeamento e do modelo escrito para os elementos do protótipo estão disponíveis em http://alphasimple.com/doc/prototyping/prototyping-basics/.

Além da descrição do sistema, o projeto conta com templates de geração de código criados pelo autor do projeto, que produzem um código fonte de alta qualidade automaticamente, podendo ser visualizado diretamente no browser e exportado para arquivos externos.

A descrição do sistema na forma textual traz suas vantagens. Por mais que um diagrama seja rico em detalhes, muitos detalhes da implementação não podem ser representados na maioria desses modelos (CHAVES, 2010). Já na forma textual utilizada na ferramenta, os modelos tornam-se muito ricos, permitindo à geração de código ter uma qualidade superior. Um revés pode ser o esforço utilizado para construir estes modelos na forma textual, assim como os templates de geração de código. Entretanto, como MDA volta seus esforços para a construção de modelos, partes dos modelos escritos podem ser reutilizadas, assim como os templates para a geração de código, trazendo o foco do processo tanto no desenvolvimento quanto no reaproveitamento para os modelos, e não mais para o código. Neste aspecto, a AlphSimple mostra uma adequação aos conceitos de MDA, modelos ricos e processos com foco voltado à eles.

A Figura 20 abaixo demonstra a visualização de trecho do código gerado na ferramenta. Ressalta-se que todo o código está anexado ao trabalho,

juntamente com o arquivo de configuração da ferramenta. Percebe-se que o código não traz apenas assinaturas de métodos, com atributos e relações entre as classes. O corpo dos métodos é implementado de maneira eficiente, tornando o código fonte o código objeto do sistema, sem a necessidade de ser editado pelos desenvolvedores que já gastaram seu tempo no desenvolvimento dos modelos.

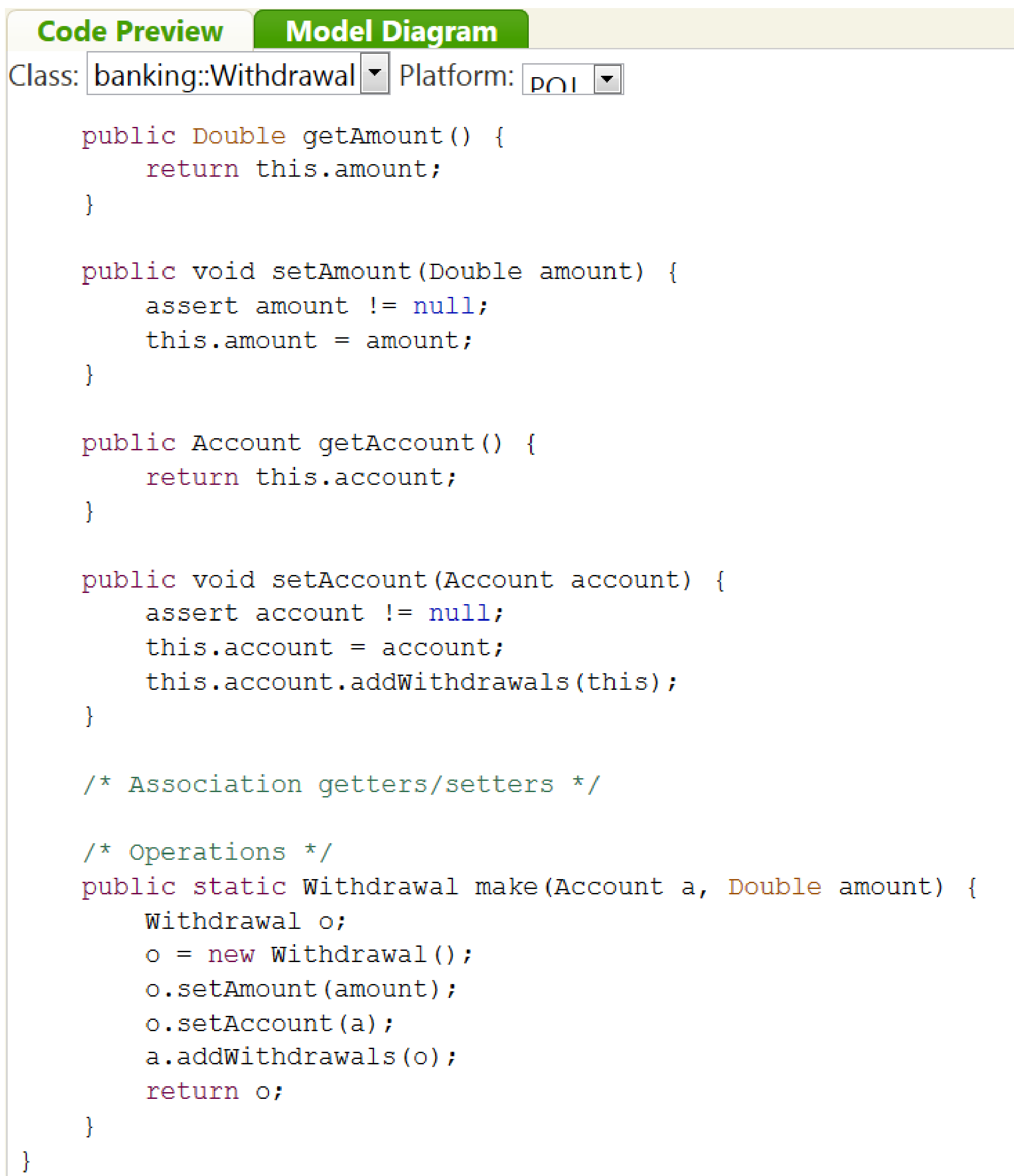


**Figura 20** – Código gerado AlphaSimple

Fonte: Chaves (2010)

Desta maneira, verifica-se que as transformações realizadas estão de acordo com os conceitos introduzidos de MDA. Suas transformações são de qualidade quando programadas de maneira correta. Contudo dentro do contexto do Framework de Práticas Ágeis, a ferramenta impossibilitaria qualquer processo que exigisse uma documentação na forma gramatical mais rica. Além disso, muita documentação gerada pelos processos poderia acabar sendo inútil na cadeia de transformações MDA.

Para fins de comparação entre as ferramentas, o mesmo diagrama de classes produzido automaticamente pela AlphaSimple será reproduzido nas próximas duas ferramentas a serem testadas: Enterprise Architect e MagicDraw.

### *4.2.3 Enterprise Architect*

A Enterprise Architect é uma ferramenta com muitas funcionalidades e oferece inúmeras possibilidades de modelagem aos seus usuários. Durante o teste, a versão utilizada foi a 9.3, com sua edição Ultimate. Apesar de ser conhecida como uma ferramenta para a modelagem UML, através de funcionalidades embutidas nela, é possível construir seus próprios templates de modelagem, com regras de mapeamento específicas, e utilizar outras tecnologias, inclusive criadas pelo usuário.

A modelagem do diagrama de classes na Enterprise Architect permitiu perceber o quanto a ferramenta é robusta de maneira geral, e traz algumas

características importantes para a abordagem MDA. A transformação direta entre modelos é uma delas; todavia, para este primeiro teste, o modelo gerado já era considerado um PSM para o desenvolvimento de código Java. Para a geração de código na Enterprise Architect, as definições de transformações já estão previamente definidas dentro da ferramenta para a maioria das tecnologias comerciais conhecidas. Porém, todas estas definições podem ser editadas, permitindo assim que desenvolvedores MDA tenham total controle sobre as transformações utilizadas nos seus processos, algo muito importante dentro da arquitetura dirigida a modelos.

O modelo criado para testar esta ferramenta é mostrado na Figura 21.

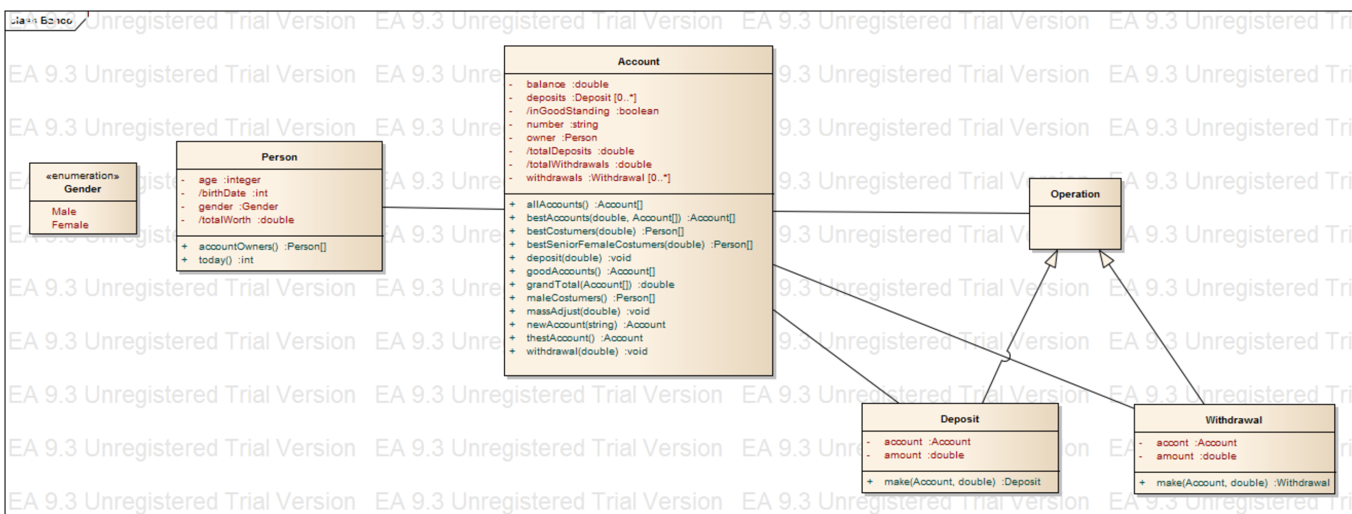



**Figura 21** – Modelo Enterprise

Através do modelo mostrado, foi gerado o código para a plataforma alvo do projeto, no caso Java. Por mais que o modelo tenha sido construído com a maior riqueza de detalhes possíveis, o código produzido não demonstra mais que assinaturas de métodos, relacionamento entre classes e atributos, tendo o

desenvolvedor que dedicar seu tempo para enriquecê-lo de forma manual, posteriormente.

A ferramenta demonstra corresponder às expectativas, apoiando MDA de forma adequada, além de não restringir nenhuma atividade do Framework de Práticas Ágeis. As atividades que geram diagramas UML podem ser contempladas pela ferramenta, além do apoio às transformações e a muitas plataformas diferentes.

Um trecho do código gerado é mostrado na Figura 22, e o conteúdo completo do código gerado pela ferramenta está anexado ao trabalho.

```
package Banco;

/**
 * @author Toco
 * @version 1.0
 * @created 23-mai-2012 00:54:55
 */
public class Withdrawal extends Operation {

        private Account accont;
        private double amount;

        public Withdrawal(){

        }

        public void finalize() throws Throwable {
                super.finalize();
        }
        /**
         *
         * @param a
         * @param amount
         */
        public Withdrawal make(Account a, double amount){
                return null;
        }
}//end Withdrawal
```

**Figura 22** – Código Gerado Enterprise

### *4.2.4 MagicDraw*

Outra ferramenta de modelagem muito completa é a MagicDraw. Assim como a Enterprise Architect, possui várias funcionalidades e uma documentação online muito rica, tornando a modelagem para os usuários menos experientes e desfamiliarizados com a ferramenta mais amena.

Para o teste realizado, as limitações impostas pela versão gratuita da versão 17.0.1 não surtiram efeito, já que o modelo é relativamente simples, e a geração de código é contemplada na mesma. Possui características importantes para a abordagem MDA, assim como a Enterprise Architect, como as transformações entre modelos com níveis de abstração e plataformas diferentes, a atualização de alguns modelos automaticamente a partir de mudanças que foram realizadas em outros modelos, e o controle entre as definições das transformações.

O modelo produzido utilizando-se a MagicDraw é mostrado na Figura 23.

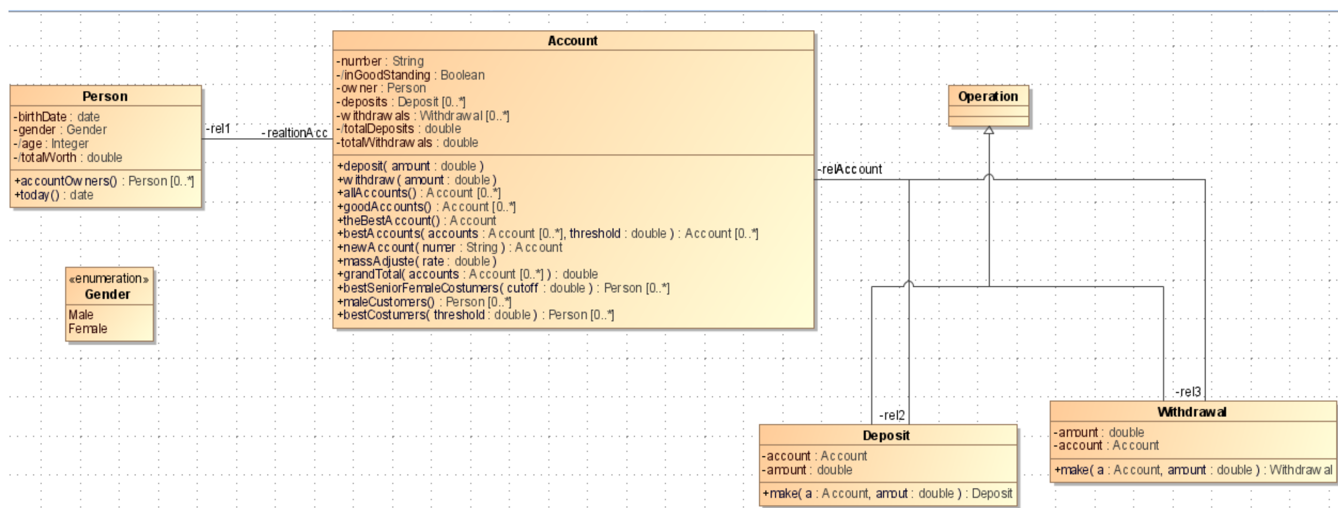



**Figura 23** – Modelo MagicDraw

Tanto o modelo produzido quanto o código gerado na MagicDraw, não possuem divergências destacáveis se forem comparados com os mesmos gerados na Enterprise Architect. Assim sendo, uma escolha entre as ferramentas teria seu foco voltado para funcionalidades específicas relacionadas a cada projeto e aos aspectos financeiros e de usabilidade relacionados ao versionamento de cada ferramenta para as necessidades de cada equipe e seus projetos.

MagicDraw também mostra ter capacidade de atender o Framework de Práticas Ágeis em sua plenitude. Apesar de transformar modelos entre as tecnologias mais utilizadas, não restringe as transformações apenas a elas, assim como a Enterprise Architect. As atividades de modelagem do Framework de Práticas Ágeis podem ser totalmente absorvidas pela ferramenta, assim como a cadeia de transformações criada por elas.

Parte do código gerado esta representado na figura 24, e seu conteúdo completo também esta anexado ao trabalho.

```
/**
 * @(#) Withdrawal.java
 */

public class Withdrawal extends Operation
{
        private double amount;

        private Account account;

        private Account relAccount;

        public Withdrawal make( Account a, double amount )
        {
                return null;
        }


}
```

**Figura 24** – Código gerado MagicDraw

# 5 Exemplos de Uso e Discussão

A questão fundamental para conciliar o Framework de Práticas Ágeis apresentado no capítulo dois com MDA é a adequação da ferramenta utilizada aos processos produzidos por ele e aos conceitos cruciais envolvendo MDA e métodos ágeis. Só assim MDA poderá ser utilizada com o Framework de Práticas Ágeis com total sucesso.

Uma das características pertencentes aos métodos ágeis que dão origem aos processos criados a partir do Framework de Práticas Ágeis é o fato de serem iterativos e incrementais. A ferramenta utilizada deve permitir que os modelos criados em fases iniciais do processo possam ser editados, de maneira que o trabalho realizado anteriormente não se torne perda de tempo, por produzir meros modelos sem contribuição direta à criação do produto final.

Além disso, o Framework de Práticas Ágeis permite extrema flexibilidade às equipes de desenvolvimento que o utilizarão. A equipe deve ter em mãos uma ferramenta que se adapte às suas necessidades, e não o contrário. Uma ferramenta com funcionalidades limitadas poderia comprometer o sucesso da utilização desse framework de maneira abrangente, limitando-o a processos com práticas pertinentes a ferramenta.

Algo importante a ser salientado é que, como MDA volta seus esforços à criação de modelos bem definidos e à geração de outros modelos e código a partir deles, essas transformações deveriam produzir artefatos de qualidade, para que o esforço gasto nessas fases não se torne um fardo para a equipe. Ferramentas que exigem muitas mudanças e aperfeiçoamento dos artefatos gerados automaticamente não se enquadram no uso de MDA de maneira ideal.

Dentre as ferramentas analisadas, a AlphaSimple permite o desenvolvimento incremental, já que o usuário pode descrever o modelo de forma incremental na forma textual empregada por ela, e seu diagrama de classes será incrementado automaticamente. Como o código produzido por ela deverá ser o código objeto da aplicação, sem que haja a necessidade de se editá-lo, a incrementação do modelo refletirá diretamente no código produzido incrementando-o da mesma forma. No entanto, a ferramenta impossibilita combinações com uma documentação rica, já que limita muito os tipos de modelos que podem ser utilizados. O único modelo gráfico contemplado é o diagrama de classes gerado por ela, e apesar da forma textual ter suas vantagens na geração de código, torna o processo limitado.

A MagicDraw esbarra no desenvolvimento incremental e na limitação financeira que oferece aos usuários. Além de produzir um código que exige a edição manual dos desenvolvedores, limita as transformações dentre os modelos em sua versão gratuita. Ela permite que transformações no código sejam incorporadas nos modelos, como a adição e exclusão de atributos e métodos, prática divergente dos conceitos trazidos por MDA, onde os modelos devem ser editados e o código gerado após isso, e não incorpora de maneira automática a implementação dos corpos dos métodos gerados, exigindo um re-trabalho em iterações posteriores. Apenas um dos tipos de transformações que copia os elementos dos modelos é permitido na versão gratuita. A MagicDraw exige dos usuários a aquisição de uma ferramenta cara e capaz de muitas coisas para que o MDA seja utilizado em sua plenitude dentro dela.

A Enterprise Architect possui vantagens e limitações muito parecidas com a MagicDraw. O código fonte gerado por ela precisa de edição e não

incorpora de forma automática mudanças importantes e relevantes. A crucial diferença entre estas duas ferramentas é que a Enterprise Architect disponibiliza todas as suas funcionalidades ao usuário na versão gratuita, enquanto a MagicDraw impõe limitações. A Enterprise Architect traz transformações satisfatórias entre modelos, gerando mais de um PSM a partir de um PIM, além de criar modelos de qualidade partindo de um PSM a outro. O limite de tempo em que a versão permanece gratuita para o usuário é de apenas trinta dias, inferior aos noventa da MagicDraw, o que pode ser prejudicial a um projeto que não tenha previsto a aquisição da ferramenta. Por outro lado, possui um preço inferior para sua aquisição, tendo o preço unitário máximo de 799,00 dólares, sem a necessidade de um pagamento anual para a manutenção da ferramenta.

A seguir será mostrada uma pequena aplicação desenvolvida usando o framework para seleção de práticas ágeis, com MDA e uma das ferramentas analisadas durante o trabalho, para que as afirmações sobre a conjugação desses elementos possam ser visualizadas de maneira mais clara. Para isto, será utilizada a Enterprise Architect, por fornecer uma quantidade maior de funcionalidades utilizáveis, englobando as combinações do Framework de Práticas Ágeis de forma mais abrangente e permitindo que a simulação da utilização do processo em uma situação real traga resultados com uma visualização mais satisfatória.

O sistema desenvolvido simula uma aplicação construída para satisfazer algumas necessidades de um restaurante. O funcionamento do restaurante dá-se da seguinte maneira:

Quando um cliente chega ao restaurante, um dos garçons aborda-o e o posiciona em uma das mesas vagas do restaurante, dessa maneira ele passa a ser o responsável pelo atendimento daquela mesa. Os clientes fazem um pedido ao garçom, que deve saber seu valor, os clientes naturalmente devem ter a oportunidade de modificar o pedido quando desejarem, modificando assim seu valor automaticamente. Assim que os clientes fecham o pedido com o garçom, devem informar o valor da gorjeta, e o garçom fica responsável em deixar a mesa pronta para receber novos clientes.

Quando o garçom finaliza o seu período de trabalho, o sistema exibe um pequeno registro de sua jornada. Nele consta as datas e os horários que ele iniciou e terminou seu trabalho, além da quantidade de pedidos atendidos, o valor total vendido por ele, além do valor total arrecadado com as gorjetas e a soma desses dois valores.

Além de todos os garçons e mesas, o sistema deve possuir também um registro com todos os pedidos atendidos no restaurante e todos os registros de trabalho dos garçons. Assim pode-se verificar o total de pedidos atendidos pelo restaurante, o valor total dos serviços prestados, além da soma das gorjetas e o valor somado dos serviços e gorjetas.

Além destas verificações, o utilizador do sistema poderá indicar uma faixa de valores, para que seja verificada a frequência de pedidos atendidos com dentro do valor indicado. Por exemplo, podemos consultar o número de pedidos com valor entre 50 e 100 reais, e o sistema irá exibir na tela a quantidade total de pedidos e a frequência relativa ao total de pedidos já atendidos pelo restaurante.

## 5.1 Exemplos

### *5.1.1 Aplicação para Restaurante*

Primeiramente deve-se construir um processo de software utilizando o framework do capítulo dois. A combinação de práticas escolhida para compor esse processo, seguindo as exigências relacionadas à obrigatoriedade e dependência entre elas, é mostrada na tabela abaixo, que apresenta todas as práticas e destaca em negrito as selecionadas.

**Tabela 8** – Práticas da simulação

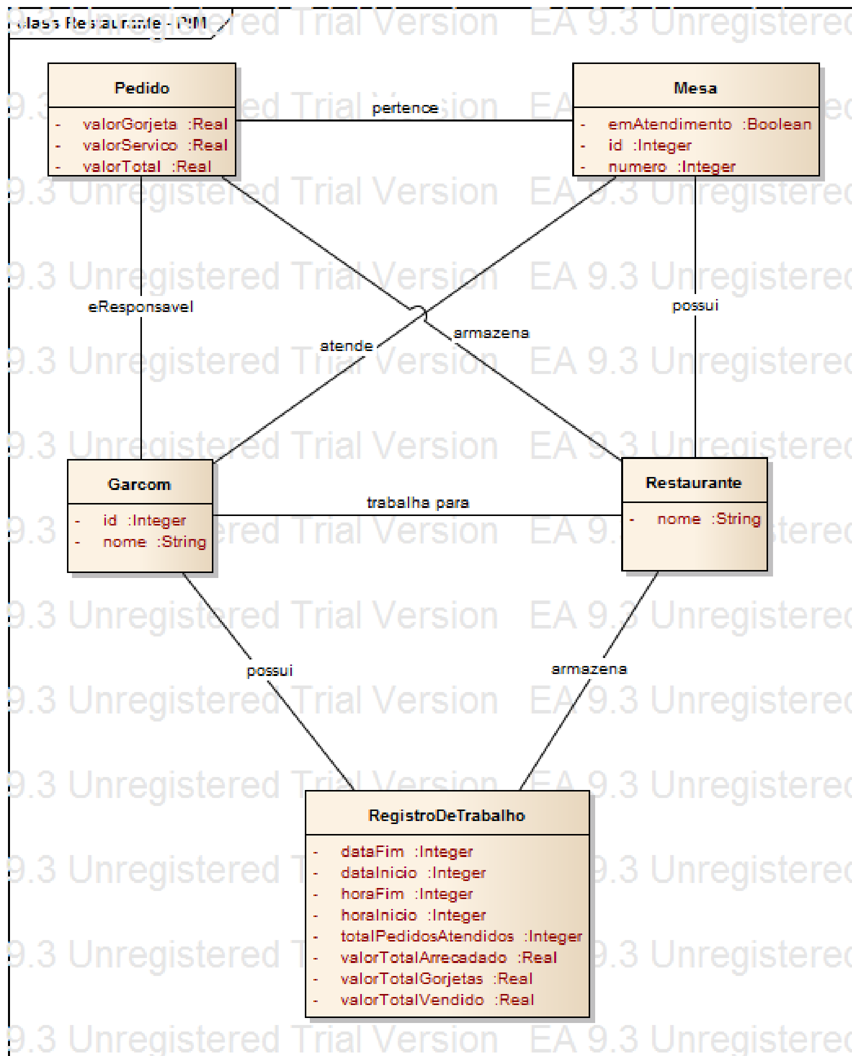
| **Lista de Requisitos** | ~~Casos de Uso~~ | ~~Desenvolvimento Lado a Lado~~ | ~~Breve Documentação~~ |
|---|---|---|---|
| **Modelagem Geral** | **Projeto da Iteração** | **Refatoração** | ~~Entrega do Sistema~~ |
| ~~Documentação Inicial~~ | ~~Escrita dos Testes de Unidade~~ | ~~Reuniões Diárias~~ | |
| ~~Projeto da Arquitetura do Sistema~~ | ~~Escrita dos Testes de Aceitação~~ | ~~Desenvolvimento Coletivo de Código~~ | |
| **Planejamento da Iteração** | **Desenvolvimento** | **Inspeção de Código** | |
| ~~Estórias do Usuário~~ | ~~Programação em Pares~~ | **Reunião de Revisão da Iteração** | |

Tendo em mãos a combinação das atividades, a execução do processo deve ser iniciada. A primeira atividade é a de definição dos requisitos, e engloba a prática de definição dos requisitos do sistema. Para isso, os envolvidos devem se reunir com os clientes a fim de identificar as funcionalidades do sistema juntamente com suas prioridades. Uma lista de requisitos que representa o sistema a ser desenvolvido é apresentada abaixo.

| Requisitos do Sistema | | |
|---|---|---|
| **Prioridade** | **Item** | **Descrição** |
| **Muito Alta** | | |
| | 1 | Cadastrar Mesa |
| | 2 | Cadastrar Garçom |
| | 3 | Criar o Registro de Trabalho do garçom |
| **Alta** | | |
| | 4 | Ocupar Mesa |
| | 5 | Criar Pedido |
| | 6 | Atualizar Pedido |
| | 7 | Fechar Pedido |
| | 8 | Desocupar Mesa |
| | 9 | Exibir o Registro de Trabalho do Garçom |
| | 10 | Computar o total de pedidos atendidos pelo restaurante |
| | 11 | Exibir valor total dos serviços |
| | 12 | Exibir valor total de gorjetas arrecadadas |
| | 13 | Exibir arrecadação total do restaurante |
| | 14 | Exibir frequência de valores dos pedidos |

**Figura 25** – Requisitos Sistema Restaurante

Com os requisitos em mãos, a execução do processo prossegue com a atividade de modelagem geral. A partir daí, os conceitos de MDD podem ser mesclados com as práticas ágeis. Pode-se utilizar a Enterprise Architect para a geração de um PIM que englobe o sistema de forma geral e a partir dele prosseguir com o processo de software, criando os PSMs de acordo com a necessidade e evolução da construção do sistema. O modelo criado está representado na figura abaixo. Nesse ponto a ferramenta empregada na simulação apoia o processo de maneira eficiente, pois permite a criação dos mais variados modelos, dando liberdade à equipe de desenvolvimento como propõe o Framework de Práticas Ágeis.



EA 9.3 Unregistered Trial Version

**Figura 26** – Modelo PIM para o Sistema do Restaurante

Logo, o processo segue com a atividade de atribuir requisitos às iterações, que contém uma única atividade obrigatória: planejamento da iteração. A figura a seguir exemplifica o resultado obtido após a realização desta atividade.

| **Iteração** | **Itens Selecionados** |
|---|---|
| Primeira Iteração | 1, 2, 3 |

| | |
|---|---|
| Segunda Iteração | 4, 5, 6, 7, 8 |
| Terceira Iteração | 9, 10, 11, 12,13, 14 |

**Figura 27** – Distribuição dos Requisitos nas Iterações

Os requisitos foram distribuídos em três iterações, onde os que possuem maior prioridade foram selecionados para serem desenvolvidos primeiro.

Logo, o processo chega à atividade de desenvolvimento do incremento do sistema. As práticas desta atividade são o projeto da iteração, desenvolvimento do sistema, a inspeção e a refatoração do código.

5.1.1.1 Primeira Iteração

Para a primeira iteração, a prática de refatoração de código não é possível, já que nenhum código fonte foi produzido ainda. Durante a primeira iteração, a ferramenta utilizada (Enterprise Architect) apoia de maneira satisfatória os conceitos de MDA, no que diz respeito à cadeia de transformações, pois permite a especialização do PIM produzido anteriormente em um PSM, que pode ser referente às plataformas mais utilizadas comercialmente. A Enterprise Architect também permite que o usuário selecione os elementos do PIM que farão parte da transformação, tornando possível uma transformação incremental concordante com métodos ágeis. Assim, a prática de projeto da iteração é realizada, construindo um PSM através do PIM criado anteriormente. A seguir, é mostrado o modelo referente aos requisitos da primeira iteração, gerado automaticamente através de uma transformação automática da Enterprise Architect.

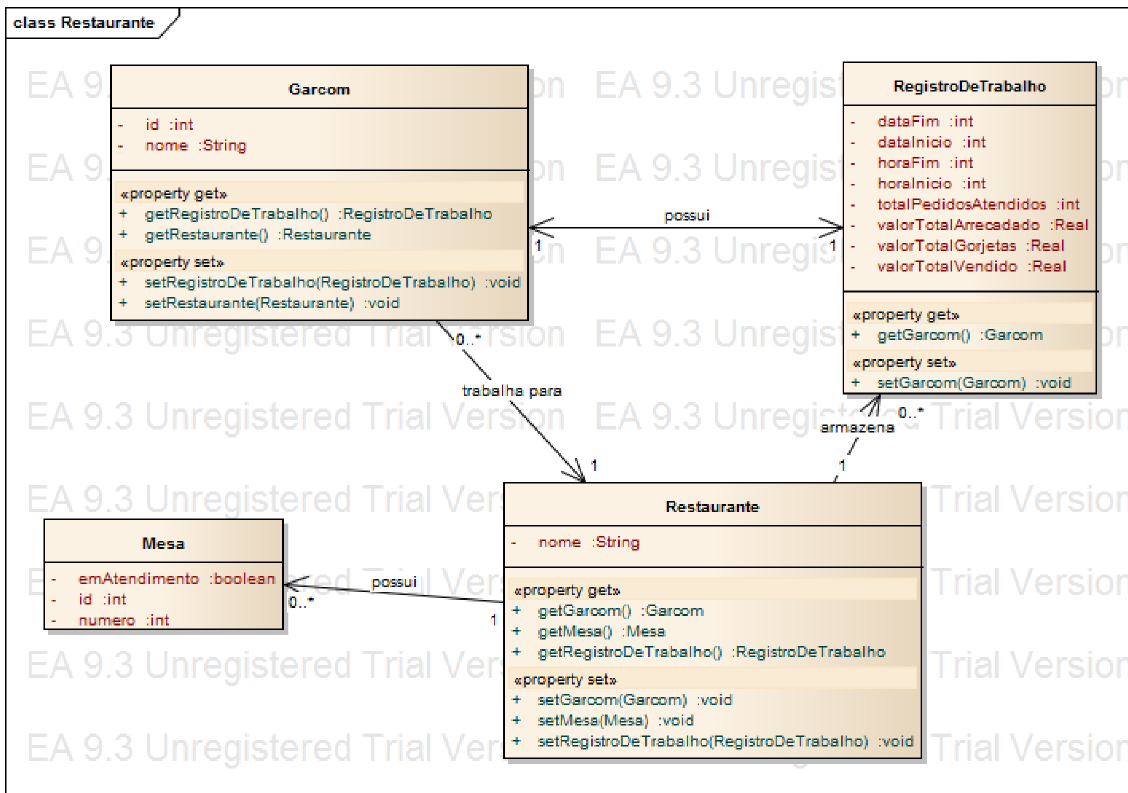



**Figura 28** – PSM Primeira Iteração

Para adequar o diagrama à plataforma Java, a Enterprise Architect cria *getters* e *setters* para as propriedades das classes envolvidas no PIM, entretanto, apenas os atributos criados a partir das relações entre as classes têm estes métodos gerados. A partir deste modelo, foi criado o modelo utilizado na transformação que irá gerar o código-fonte. Este novo modelo conta com os métodos necessários para que os requisitos da iteração sejam implementados. Para que a qualidade da ferramenta seja testada de maneira mais concreta, analisando alguns dos padrões envolvendo nomes de variáveis e outros aspectos, outras mudanças no modelo foram deixadas de lado. No entanto vale lembrar que essa prática é contrária aos conceitos de MDA. Segundo eles, o correto seria refinar o modelo para que a transformação seguinte produza os melhores resultados possíveis (KLEPPE WARMER BAST, 2003). O modelo com os métodos adicionais é mostrado na figura a seguir.

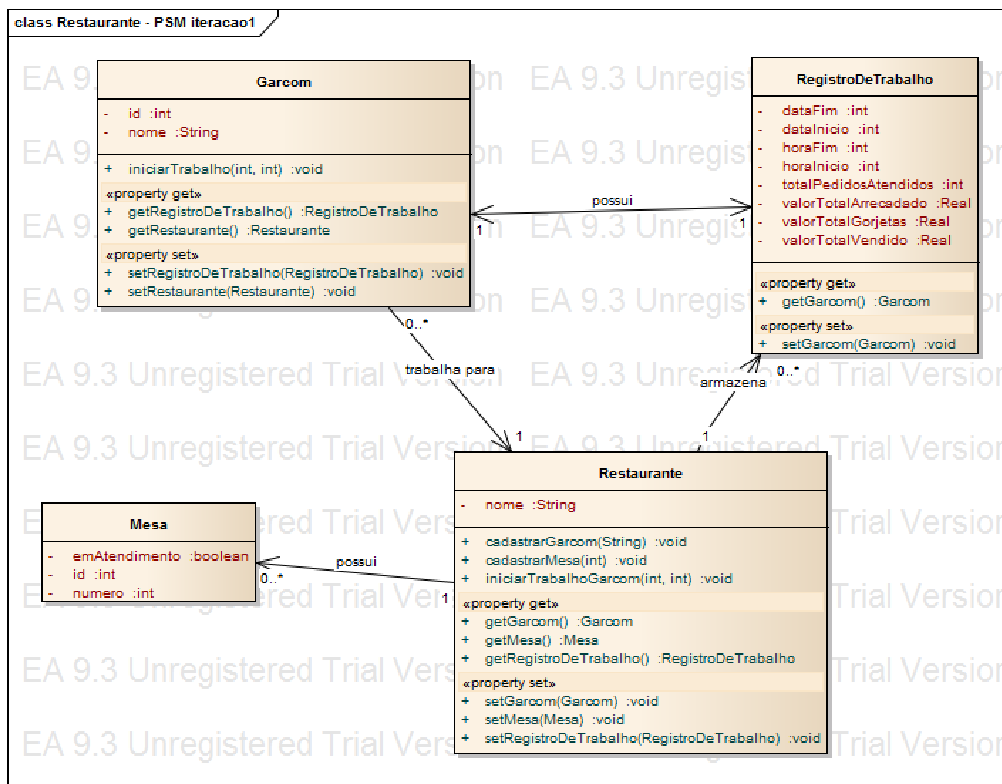



**Figura 29** – PSM Primeira Iteração Refinado

Como visto anteriormente, a Enterprise Architect não apoia da melhor maneira possível a geração automática de código a partir dos modelos criados e torna a intervenção manual no código produzido inevitável. O código gerado de forma automática traz os métodos de *get* e *set* para o garçom na classe Restaurante, mas não cria o atributo correspondente a estes métodos. Além disso, quando um modelo é independente de qualquer plataforma, apenas alguns tipos são disponibilizados para as variáveis do sistema, os valores tipados como Real não foram convertidos para seu correspondente na plataforma Java. Fica claro que a inspeção do código gerado é indispensável.

Após a implementação do código referente aos requisitos da primeira iteração, e de sua devida inspeção, o processo passa para a atividade de integração do incremento. Portanto, é realizada a prática de reunião de revisão da iteração. Assim, os envolvidos no projeto podem discutir a evolução dos

trabalhos, fazendo uma avaliação dos benefícios da utilização do MDA com o processo, discutir melhorias e integrar o código produzido ao sistema final. Desta maneira, a primeira iteração do processo chega ao fim.

5.1.1.2 Segunda Iteração

A segunda iteração, assim como as seguintes, pode durante o projeto da iteração, despender esforços, a fim de aperfeiçoar o PIM produzido anteriormente. Assim, a qualidade dos PSMs gerados é incrementada, como sugerido pelo MDA, garantindo a adaptação às mudanças de requisitos, prática concordante com os preceitos dos métodos ágeis. Os PSMs gerados, também podem passar por uma fase de aperfeiçoamento.

A partir da segunda iteração, a atividade de desenvolvimento do incremento do sistema pode contar com a refatoração do código como previsto no processo, se ela for necessária. Dessa vez, novos requisitos serão envolvidos na transformação que irá gerar um novo PSM da mesma maneira que na iteração anterior. Agora, uma das vantagens do MDA fica mais visível no processo: caso o desenvolvimento envolva uma nova plataforma alvo, como, por exemplo, a construção de uma base de dados que deverá ser representada através de um modelo ER, basta utilizar a Enterprise.

O processo segue com a plataforma Java como alvo, e com os novos requisitos, todos os elementos do PIM gerado na atividade de modelagem geral continuam sendo utilizados.

Uma funcionalidade útil da ferramenta é a sincronização do código gerado com os modelos. Após a edição do código, se sincronizarmos o modelo com ele, automaticamente os modelos gerados incorporam as mudanças feitas

no código. Sendo assim, o PSM criado para a segunda iteração sincronizado com o código já produzido na primeira iteração, incrementando assim o modelo com seus resultados. Após a sincronização, o modelo é aperfeiçoado, para suportar os requisitos da segunda iteração.

Vale salientar que esta funcionalidade da ferramenta é utilizada principalmente para que uma das práticas comuns entre os programadores não ocorra. É comum que, assim que mudanças surgem durante a implementação do código, elas são realizadas diretamente no código, tornando os modelos desatualizados e sem funcionalidade. Entretanto, MDA sugere que a equipe as faça diretamente no modelo, e assim, através das transformações automáticas, as mudanças são incorporadas diretamente ao código. Sendo assim, a Enterprise Architect demonstra oferecer apoio às equipes que pretendem utilizar o Framework de Práticas Ágeis com as práticas do MDA, mas ainda não estão habituados com suas práticas, ou pecaram durante a modelagem de seu sistema. Além disso, como a geração de código deixa a desejar, muito do esforço despendido no aperfeiçoamento do modelo é em vão, tornando a sincronização a maneira mais fácil de inserir certos aspectos, como *getters* e *setters*.

Como recomendado pela MDA, durante esta iteração, os esforços foram gastos no aperfeiçoamento do modelo, para que o código gerado fosse produzido com maior qualidade. O modelo resultante foi o seguinte:

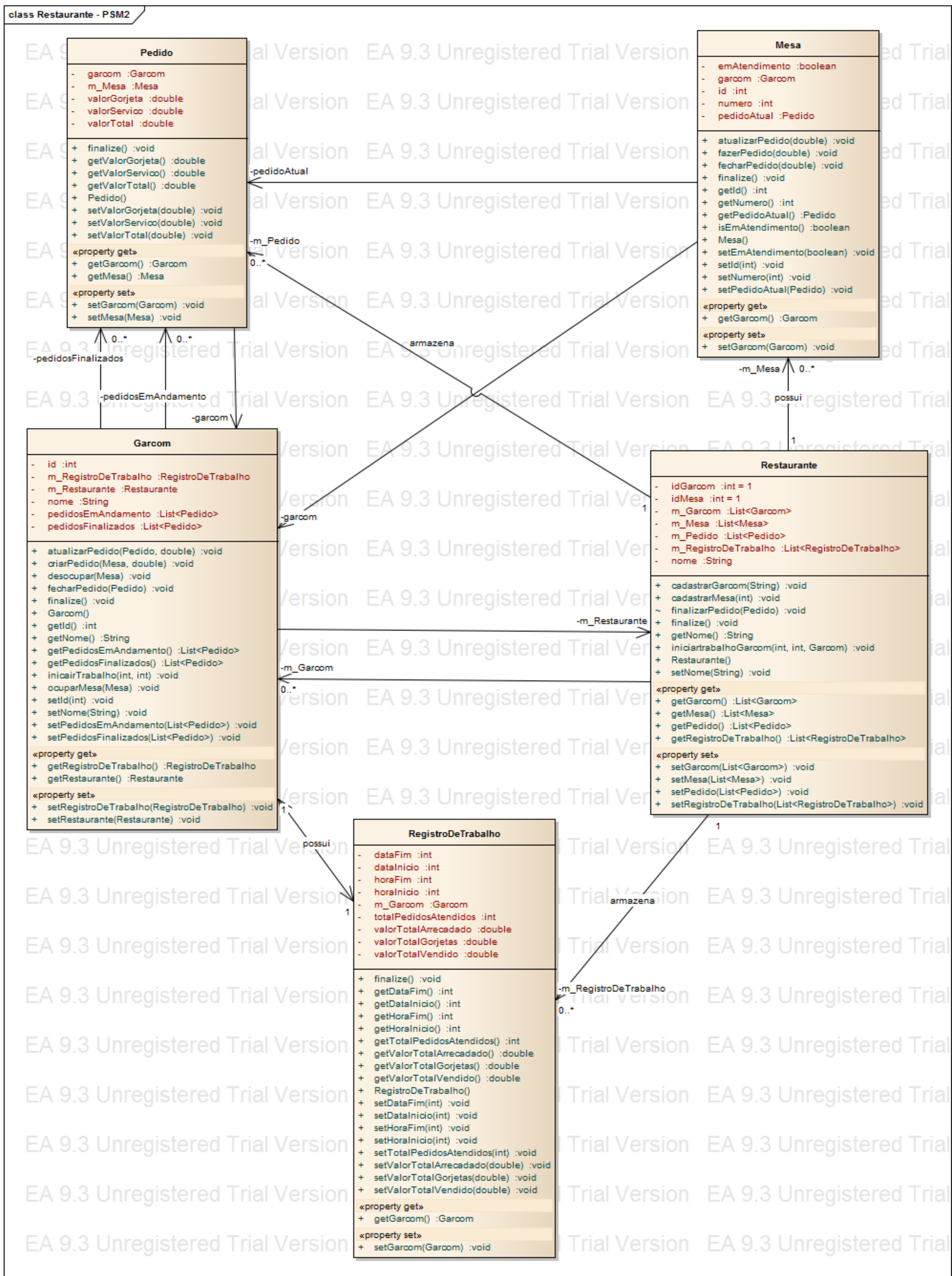



**Figura 30** – PSM Segunda Iteração

A partir dele, novamente é gerado o código fonte da aplicação, que após a realização da prática de inspeção de código, deverá ser implementado a fim de se adequar às reais necessidades do sistema. A prática da inspeção de código torna-se comum antes da própria implementação manual, já que os desenvolvedores deverão lidar com código produzido automaticamente por ferramentas. No entanto este fato não descarta sua possível realização depois da implementação manual. A refatoração do código novamente não é necessária, e o processo novamente chega à atividade de integração do incremento. Logo, a atividade de reunião de revisão da iteração é realizada e o código produzido na segunda iteração é integrado ao da primeira

#### 5.1.1.3 Terceira Iteração

Por fim chega-se a terceira e última iteração, nela todos os requisitos restantes serão implementados no sistema. Esses requisitos são os com maior complexidade de desenvolvimento dentro da aplicação. Eles exigem que as listas contendo os dados dos pedidos já atendidos pelo restaurante sejam acessados. No caso do requisito que verifica a frequência dos valores totais dos pedidos já atendidos pelo restaurante, as entradas do método deverão ser utilizadas durante as verificações necessárias para produzir a saída adequada.

Contudo, durante a execução do processo é muito comum a mudança dos requisitos. Para a terceira iteração, uma mudança nos requisitos anteriores será adicionada. Caso um dos garçons deseje terminar seu turno sem ter finalizado todos os seus pedidos, ou por algum motivo não possa continuar atendendo uma das mesas de sua responsabilidade, ele deve ser capaz de atribuí-las a algum outro garçom. Para implementar o método, o garçom passará a possuir uma lista com todas as mesas em atendimento por ele.

Apesar de poder alcançar as mesas que estão sendo atendidas através dos pedidos, pode ocorrer o caso em que o garçom inicia o atendimento de uma mesa onde o pedido ainda não tinha sido efetuado. Neste caso, o garçom poderia pensar que não possui nenhuma mesa não finalizada erroneamente.

Assim, um novo PSM é construído durante o planejamento da iteração, que irá englobar todos os novos requisitos da terceira iteração, além de todos os já envolvidos anteriormente, e é mostrado a seguir.

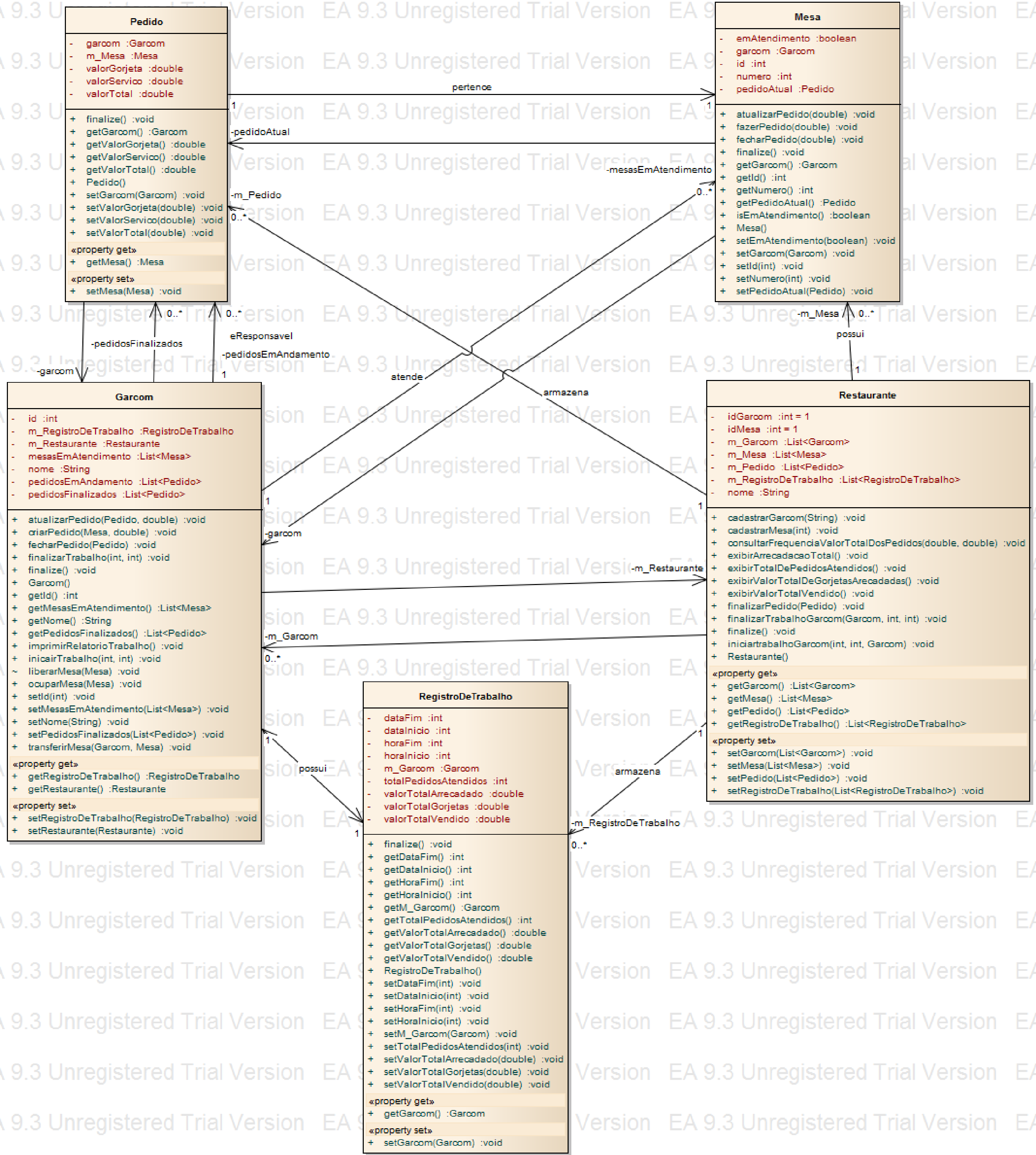



**Figura 31**- PSM Terceira Iteração

Novamente, o código é gerado automaticamente e, após uma inspeção, é implementado, desta vez de maneira definitiva. A prática de refatoração não foi necessária em nenhuma das iterações, no entanto, foi despendido um grande esforço durante as inspeções realizadas no código gerado de forma automática pela ferramenta. Note também que o diagrama que representa o modelo PSM para a terceira iteração já possui um tamanho consideravelmente grande. Isso demonstra como os modelos utilizados em MDA podem atingir grandes proporções em aplicações de maior porte.

Ao final da terceira iteração, a reunião integra todo o código já produzido e o processo de desenvolvimento da aplicação chega ao fim. O código fonte final está anexado ao trabalho.

### *5.1.2 Contador de Caracteres*

Apesar de ilustrar uma clara tentativa da utilização do framework para seleção de práticas ágeis com o MDA, o primeiro exemplo de aplicação tem como métodos mais complexos, aqueles que utilizam as listas que armazenam dados da aplicação. Para exemplificar melhor os casos em que os métodos podem ter uma complexidade maior, será mostrado o desenvolvimento uma classe que tem como objetivo implementar os métodos necessários para que dada uma entrada de texto, sejam impressos cada um dos diferentes caracteres pertencentes a ele, seguidos de sua frequência relativa.

Para isso será considerado que a classe pertence a uma aplicação maior, mas que durante essa etapa do desenvolvimento, apenas ela deverá ser implementada. O modelo desenvolvido para representar o PIM dessa classe é mostrado na figura a seguir (Figura 32).

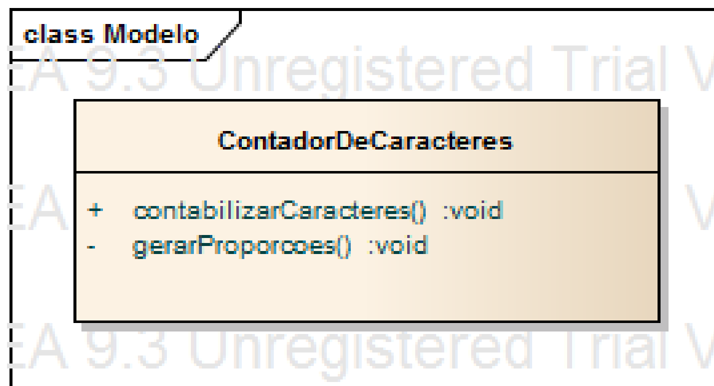



**Figura 32** – PIM ContadorDeCaracteres

O modelo apresentado na figura acima é pobre em detalhes devido à simplicidade da aplicação. Agora será apresentado o PSM que representa a classe correspondente ao PIM mostrado acima.

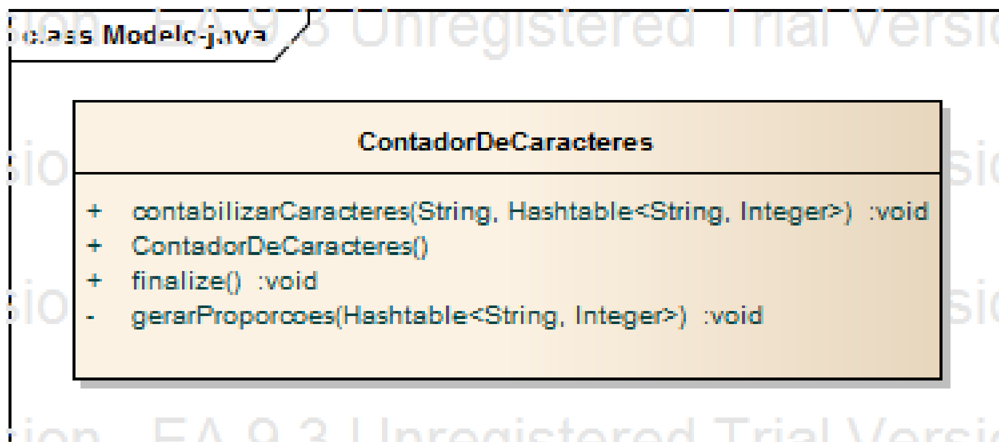



**Figura 33** – PSM ContadorDeCaracteres

Desta forma, pode-se verificar que apesar de em alguns casos os modelos utilizados na cadeia de transformações MDA tomarem grandes proporções, modelos muito pequenos, com poucos detalhes, não trazem boas vantagens para o processo que utiliza MDA.

O exemplo demonstra claramente a falta de detalhes, e a deficiência de diagramas para modelar sistemas em alguns casos. Apesar de a ferramenta utilizada permitir a adição de detalhes que às vezes não são visualizados

diretamente no diagrama, a fim de tornar os modelos mais ricos e detalhados para os utilizadores da ferramenta, o modelo na forma de diagramas demonstra ser ineficiente para a cadeia de transformações MDA em alguns casos.

Como já foi verificado no exemplo apresentado na seção anterior, o código produzido por esse tipo de modelo, é extremamente pobre. O código gerado por este modelo para a plataforma Java é mostrado na Figura 34.

```
package Modelo;
public class ContadorDeCaracteres {
        public ContadorDeCaracteres(){
        }
        public void finalize() throws Throwable {
        }
        /**
         *
         * @param tabela
         * @param texto
         */
        public void contabilizarCaracteres(Hashtable<String, Integer> tabela, String texto){
        }
        /**
         *
         * @param tabela
         */
        public void gerarProporcoes(Hashtable<String, Integer> tabela){
        }

}//end ConadorDeCaracteres
```

**Figura 34** – código gerado ContadorDeCaracteres

Para que resultados satisfatórios fossem atingidos em casos como este, onde as aplicações necessitam de código com complexidade relativamente alta para ser gerado automaticamente, os modelos que compõem as transformações MDA devem ser mais detalhados que os diagramas UML geralmente utilizados. A forma textual utilizada na ferramenta AlphaSimple é uma alternativa, no entanto com uma complexidade que impede desenvolvedores desfamiliarizados com a tecnologia de utilizá-la, e exige um esforço de modelagem parecido com o de desenvolvimento de código.

Então a partir do código gerado pela Enterpirse Architect, mostrado na figura anterior, foi desenvolvido o código referente aos métodos da classe em questão. Durante o desenvolvimento, foi notada a necessidade da criação de mais um método dentro da classe. Assim sendo, o modelo final da classe desenvolvida e o código fonte da mesma, serão respectivamente mostrados nas figuras a seguir (Figuras 35 e 36).

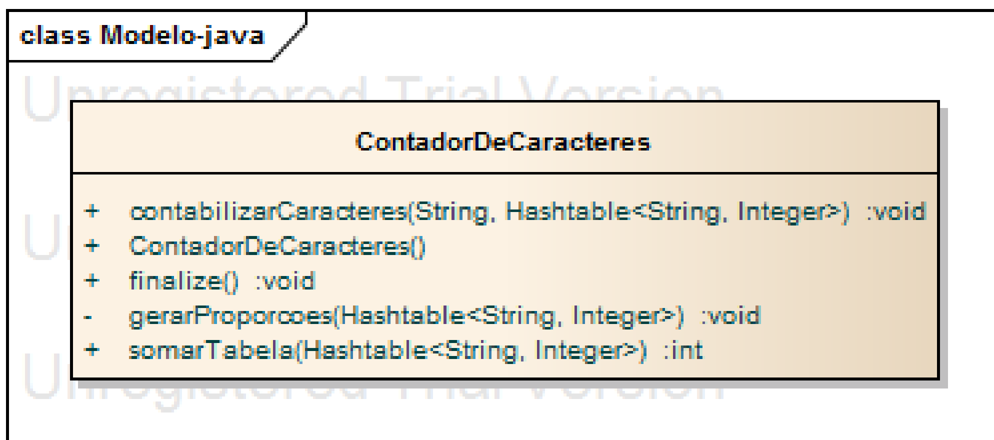



**Figura 35** – PSM Final ContadorDeCaracteres

```
package Modelo;

import java.util.Hashtable;
import java.util.Iterator;
import java.util.Set;

public class ContadorDeCaracteres {

        public ContadorDeCaracteres(){

        }
        public void finalize()
          throws Throwable{

        }

        public void contabilizarCaracteres(String texto, Hashtable<String, Integer> tabela){
            int soma;
            for (int i = 0; i < texto.length(); i++) {//percorro o texto
                if (texto.substring(i, i + 1).compareTo(" ") != 0) { //se o caracter nao for " "
                    if (tabela.get(texto.substring(i, i + 1)) != null) {// verifico se essa entrada ja esta na tabela
                        soma = tabela.get(texto.substring(i, i + 1)) + 1; //se stiver, como 1 ao numero de ocorrencias
                        tabela.put(texto.substring(i, i + 1), soma); // atualizao a tabela
                        soma = 0; // zero a variavel
                    } else {//senao
                        tabela.put(texto.substring(i, i + 1), 1);//senao, crio a entrada na tabela com o valor 1
                    }
                }
            }
        }

        private void gerarProporcoes(Hashtable<String, Integer> tabela){
            int somaDaTabela = this.somarTabela(tabela);
            String chave;
            float valorDaChave = 0;
            float frequencia =0;
            Set set = tabela.keySet();//recupero um Set com todas as chaves da tabela
            Iterator i = set.iterator(); // um iterador é criado
            while(i.hasNext()){ //enquanto o iterador encontra proximo elemento no set de chaves
                 chave = (String)i.next();
                 valorDaChave = tabela.get(chave); // recupera o valor daquela chave
                 frequencia = (valorDaChave*100)/somaDaTabela; // faz o calculo da frequencia relativa daquela chave
                 System.out.println(chave + " -> " + frequencia + "%"); //imprime os valores
            }
        }

        public int somarTabela(Hashtable<String, Integer> tabela){
          int soma = 0;
          int x=0;
          Set set = tabela.keySet(); //recupero um Set com todas as chaves da tabela
          Iterator i = set.iterator(); // um iterador é criado
          while(i.hasNext()){ //enquanto o iterador encontra proximo elemento no set de chaves
                String chave = (String)i.next();
                x = tabela.get(chave); //recupera o valor daquela chave
                soma +=x; // soma ao montante total
            }
               return soma;
        }
}//end ContadorDeCaracteres
```

**Figura 36** – Código Final ContadorDeCaracteres

### 5.2 Considerações Finais

Durante o processo de desenvolvimento da aplicação, alguns aspectos da utilização do Framework de Práticas Ágeis com MDA foram notados.

A ferramenta utilizada é o fator com maior influência dentro dessa conciliação. Através dela muitas modificações no processo podem ocorrer. Enquanto a Enterprise Architect era utilizada no desenvolvimento da aplicação,

pode-se perceber que a adaptação à ferramenta torna o processo um pouco mais custoso do que deveria ser. Apesar de possuir todas as suas regras de mapeamento editáveis, leva algum tempo para que os aspectos das transformações automáticas, que influenciam diretamente na forma de modelagem, sejam notados. A ferramenta também traz um revés para os seguidores fiéis dos preceitos de MDA, onde os modelos devem ser definidos de maneira extremamente rica, já que muito do tempo gasto para que o modelo chegue a este nível de detalhamento acaba sendo desperdiçado, pois não se reflete diretamente na qualidade do código.

A Enterpise Architect apoia fortemente os preceitos ágeis, e o desenvolvimento iterativo e incremental pode ser realizado em sua plenitude. No entanto, apesar da edição do código gerado não ser perdida quando um novo código é gerado automaticamente, a inclusão de novos métodos continua sendo custosa para o programador. Também é importante destacar que algumas equipes podem esbarrar nas barreiras economicamente impostas pela ferramenta.

O exemplo implementado para testar o uso de MDA com o Framework de Práticas Ágeis é pequeno e mostra funcionalidades simples. Dentro delas, aquelas com maior complexidade exigem apenas que listas sejam percorridas e algumas verificações feitas. Mesmo com um exemplo desse tipo, é possível observar que aplicações ou algoritmos com complexidade maior teriam seus códigos gerados de maneira insatisfatória.

O segundo exemplo tem como objetivo ilustrar esse caso de forma isolada. Através dele, é possível notar que mesmo uma pequena classe, se possuir métodos com alguma complexidade, deverá ser desenvolvida

obrigatoriamente por um programador, caso a ferramenta utilizada seja a mesma do exemplo.

Como observado no exemplo, a ferramenta mantém o código implementado no corpo do método para as próximas transformações, mas novos métodos surgem apenas com suas assinaturas nas gerações de código, com exceção de alguns métodos *getters* e *setters.*

Isto não significa que o uso de MDA seja totalmente inútil em aplicações complexas. Como alternativa para os casos em que a modelagem deve ser muito detalhada, a fim de produzir um código complexo, existem ferramentas de geração de código, como a utilizada dentro da AlphaSimple,que possuem a capacidade de criar código objeto até mesmo para esse tipo de aplicação. No entanto, para que este código seja gerado com eficiência, o desenvolvedor deverá gastar uma grande quantidade de esforço e tempo, para modelar o método de forma detalhada, tornando discutível a vantagem desse tipo de atividade.

# 6 Conclusão

A aplicação de MDA em um processo de software pode trazer vantagens, dependendo da forma como os processos são definidos, da experiência e capacidade da equipe, além das ferramentas disponíveis.

Métodos ágeis e MDA podem fazer parte de um processo de software. Entretanto, nem todos os processos permitem esta combinação. Desta maneira, a utilização do framework para seleção de práticas ágeis com MDA varia de caso em caso. Logo, a extrema flexibilidade desse framework torna difícil afirmar que os dois possam ser utilizados juntos de maneira geral.

No atual estado em que se encontram as tecnologias, a utilização de uma única ferramenta que contemple MDA em sua plenitude é inviável. Para o sucesso da integração entre MDA e o framework para seleção de práticas ágeis, seriam necessários esforços para que mais ferramentas fossem analisadas, a fim de encontrar uma combinação ideal entre funcionalidades e custo, que permita a exploração do desenvolvimento dirigido a modelos de forma adequada.

O mais importante é que o uso de novas tecnologias e metodologias dentro de um processo de software seja inserido de maneira gradativa, eficiente, e que os benefícios trazidos pela sua adoção sejam maiores que os custos de adaptação e conhecimento dos mesmos.

Algumas das atividades do framework proposto em FAGUNDES (2005) devem estar presentes nos processos que pretendem utilizar MDA, como por exemplo, a inspeção do código. Com o atual estado da tecnologia, é necessário que o código gerado automaticamente seja inspecionado pelos

desenvolvedores. Outras características necessárias para os processos podem ser destacas, como a necessidade de artefatos para a cadeia de transformações MDA. Isto obriga os processos a possuírem pelo menos uma atividade que produza um artefato de entrada para a cadeia de transformações do MDA. Levando esse fato em consideração, tornar a prática de modelagem geral obrigatória, faria o framework para seleção de práticas ágeis se tornar razoavelmente adequado ao MDA de forma simples, no entanto, o problema não seria totalmente resolvido.

Para a combinação utilizada durante a implementação do sistema do restaurante, caso a atividade de projeto da iteração não estivesse presente, o processo não funcionaria com MDA, ou então, para que seus conceitos fossem aplicados, os desenvolvedores teriam que fugir do processo proposto. Já que o modelo criado durante a modelagem geral não possui detalhes suficientes para uma geração de código eficiente, construir modelos mais detalhados seria inevitável para a continuação da cadeia de transformações de maneira satisfatória.

Sendo assim, uma nova proposta de framework para seleção de práticas ágeis, é mostrada a seguir:

| | Lista de Requisitos(O) | Modelagem Geral (O) | Documentação Inicial | Projeto de Arquitetura do Sistema | Planejamento da Iteração (O) | Reuniões Diárias | Estórias do Usuário | Casos de Uso | Projeto da Iteração (O) | Desenvolvimento (O) | Escrita dos testes de unidade | Escrita sobre os testes de aceitação | Desenvolvimento Coletivo de Código | Programação em Pares | Refatoração | Integração Paralela | Controle de Versões | Programação lado-a-lado | Inspeção de código (O) | Reunião de Revisão da Iteração | Entrega do Sistema |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Lista de Requisitos (O) | | | | | | | | | | | | | | | | | | | | | |
| Modelagem Geral (O) | | | | | | | | | | | | | | | | | | | | | |

| Documentação Inicial | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Projeto de Arquitetura do Sistema | | | | | | | | | | | | | | | | | | | | | |
| Planejamento da Iteração (O) | | | | | | | | | | | | | | | | | | | | | |
| Reuniões Diárias | | | | | | | | | | | | | | | | | | | | | |
| Estórias do Usuário | | | | | | | | E | | | | | | | | | | | | | |
| Casos de Uso | | | | | | | E | | | | | | | | | | | | | | |
| Projeto da Iteração (O) | | | | | | | | | | | | | | | | | | | | | |
| Desenvolvimento (O) | | | | | | | | | | | | | | | | | | | | | |
| Desenvolvimento Coletivo de Código | | | | | | | | | | | | | | | | | | | | | |
| Programação em Pares | | | | | | | | | | | | | | | | | | E | | | |
| Refatoração | | | | | | | | | | | | | | | | | | | | | |
| Integração Paralela | | | | | | | | | | | | | | | | | | | | | |
| Controle de Versões | | | | | | | | | | | | | | | | | | | | | |
| Programação lado-a-lado | | | | | | | | | | | | | | E | | | | | | | |
| Inspeção de código (O) | | | | | | | | | | | | | | | | | | | | | |
| Reunião de Revisão da Iteração | | | | | | | | | | | | | | | | | | | | | |
| Entrega do Sistema | | | | | | | | | | | | | | | | | | | | | |

**Figura 37** – Framework para seleção de práticas ágeis com MDA

Esta extensão do Framework de Práticas Ágeis traz as práticas de modelagem geral, projeto da iteração e inspeção de código como obrigatórias, além de excluir a prática de breve documentação. A razão da exclusão da breve documentação dá-se pelo fato que, caso ela fosse selecionada para compor um processo de software criado pelo novo framework, os artefatos utilizados na cadeia de transformações estariam automaticamente promovendo essa documentação. Sendo assim, um processo que contemple as práticas do

MDA, acaba realizando a prática de breve documentação de maneira inevitável.

Esta extensão do Framework de Práticas Ágeis proposta constrói processos que integram MDA com métodos ágeis, mas possui limitações. Ela foi proposta levando em consideração que o usuário irá utilizar uma ferramenta como a Enterprise Architect ou MagicDraw, que possuem alto custo. Além disso, o framework já não possui tanta flexibilidade como antes, limitando assim os processos criados e consequentemente as equipes que podem utilizá-lo.

Desta forma, pode-se afirmar que o custo de adaptação do Framework de Práticas Ágeis ao MDA é grande, já que sua aplicabilidade diminui muito, já que o número de processos que podem ser criados decresce, e o sucesso da utilização do framework está diretamente vinculado a ferramentas de alto custo.

## 6.1 Trabalhos Futuros

Alguns trabalhos futuros são citados a seguir.

Analisar um conjunto maior de tecnologias e ferramentas relacionadas ao MDA para tentar encontrar uma combinação que atenda aos requisitos da arquitetura e dos processos do Framework de Práticas Ágeis de maneira integral.

Verificar as práticas do framework mostrado no capítulo dois de acordo com os conceitos de MDA, criando novas obrigatoriedades, dependências e talvez gerar exclusões, produzindo uma nova versão desse framework que

construa apenas processos que podem ser utilizados de maneira conjugada ao MDA.

# 7 REFERÊNCIAS BIBLIOGRÁFICAS

ALVAREZ, Diego Perez; *Proposta de um Processo Ágil para Projetos com um único Desenvolvedor*

AMBLER, S.W. Agile model driven development is good enough. **Ieee Software**, Arlington Heights, Il, v.20, n.5, p.71-73, out. 2003.

AMBLER, Scott W. **Modelagem ágil: práticas eficazes para a Programação Extrema e o Processo Unificado**.Trad. Acauan Fernandes. Porto Alegre: Bookman, 2004.

BASSO, Fabio Paulo; PILLAT, Raquel Mainardi. *Um Relato de Experiência no Desenvolvimento Ágil de Sistemas MDA.*

BECK, K.; COCKBURN, A.; JEFFRIES, R.; HIGHSMITH, J., *Agile Manifesto.* Disponível em <http://www.agilemanifesto.org>, Ano: 2001. Acesso em 18 de abril de 2011.

*Desenvolvimento ágil de software.* Disponível em <http://pt.wikipedia.org/wiki/Desenvolvimento_%C3%A1gil_de_software#Hist.C3.B3ria>. Acesso em 14 dez. 2010.

CHAVES, Rafael. **AlphaSimple.** 2010. Disponível em: <http://alphasimple.com/>. Acesso em: 21 maio 2012.

CHAVES, Rafael. **Codgen - POJO templates.** Disponível em: <http://alphasimple.com/project/show/548>. Acesso em: 15 maio 2010.

CHAVES, Rafael. **On code and diagrams.** Disponível em: <http://abstratt.com/blog/2008/05/05/on-code-and-diagrams/>. Acesso em: 21 maio 2012.

COCKBURN, Alistair. **Agile Software Development**.Adisson-Wesley, 2001.

FAGUNDES, P. B. *Framework Para Comparação e Análise de Métodos Ágeis.* 2005. 134f. Dissertação (Mestrado em Ciência da Computação) – Departamento de Informática e Estatística, Universidade Federal de Santa Catarina, Florianópolis, 2005.

FOWLER, Martin. The New Methodology. 2005. Disponível em: <http://www.martinfowler.com/articles/newMethodology.html>. Acessado em: 22 maio 2012.

HIGHSMITH, Jim. **Agile Software Development Ecosystems.Adisson-Wesley**, 2002.

HILMAN. Metodologias Ágeis. 2004. Disponível em: <http://www.redes.unb.br/material/ESOO/Metodologias%20%c1geis.pdf>. Acessado em: 21 maio 2012.

KLEPPE, Anneke G.; WARMER, Jos B.; BAST, Wim. **MDA Explained: The Model Driven Architecture : Practice and Promise.** Boston: Addison-wesley Professional, 2003.

KOCH, Nora. *Transformation Techniques in the Model-Driven Development Process of UWE.*

KOCH, Nora; ZHANG, Gefei; ESCALONA; Maria José. Model Transformations from Requirements to Web System Design.

MACIEL, Rita Suzana P.. Moderne**.** Disponível em: <http://homes.dcc.ufba.br/~ritasuzana/moderne>. Acesso em: 15 maio 2011.

MACIEL, Rita Suzana P.; SILVA, Bruno C. da; MAGALHÃES, Ana Patrícia F.; ALVES ,Fabrício; GOMES, Ramon. *Towards a Process-Centered Software Engineering Environment for Model-Driven Development.*

*MDA Specificatios.* Disponível em <http://www.omg.org/mda/specs.htm>. Acesso em 14 dez. 2010.

MELLOR, S. J.; CLARK, A. N.; FUTAGAMI, T. Model-driven development. **IEEE Software,**v. 20, n. 5, p. 14–18, 2003.

SILVA, Bruno C. da; MAGALHÃES, Ana Patrícia F.; MACIEL, Rita Suzana P.; MARTINS; Narciso; NOGUEIRA, Leandro; QUEIROZ, João C. *Transforms: Um Ambiente de Apoio a Modelagem e Execução de Processos de Software Dirigido por Modelos*

*SOARES, Michel dos Santos. Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software. 2004. Disponível em:* <http://www.dcc.ufla.br/infocomp/artigos/v3.2/art02.pdf>

SOMMERVILLE, Ian, **Engenharia de Software**, São Paulo: Addison-Wesley, 2003.

SOUZA, Thiago Silva de. Model Driven Architecture – Conceitos Fundamentais. Disponível em: <http://www.linhadecodigo.com.br/ArtigoImpressao.aspx?id=1953>. Acesso em: 02 jul. 2011

VILAIN, Patrícia; FAGUNDES, Priscila Bastos; MACHADO, Thiago Leão; *A Framework for Selecting Agile Practices and Defining Agile Software Processes.*.

ZHANG, Yuefeng; PATEL, Shailesh; *Agile Model-Driven Development in Practice*

ZHANG, Yuefeng. Test-driven modeling for model-driven development. **Ieee Software**, v.21, n.5, p.80-86, out. 2004

*MDA Specificatios.* Disponível em <http://www.omg.org/mda/specs.htm>. Acesso em 14 dez. 2010.

MELLOR, S. J.; CLARK, A. N.; FUTAGAMI, T. Model-driven development. **IEEE Software,**v. 20, n. 5, p. 14–18, 2003.

SILVA, Bruno C. da; MAGALHÃES, Ana Patrícia F.; MACIEL, Rita Suzana P.; MARTINS; Narciso; NOGUEIRA, Leandro; QUEIROZ, João C. *Transforms: Um Ambiente de Apoio a Modelagem e Execução de Processos de Software Dirigido por Modelos*

*SOARES, Michel dos Santos. Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software. 2004. Disponível em:* <http://www.dcc.ufla.br/infocomp/artigos/v3.2/art02.pdf>

SOMMERVILLE, Ian, **Engenharia de Software**, São Paulo: Addison-Wesley, 2003.

SOUZA, Thiago Silva de. Model Driven Architecture – Conceitos Fundamentais. Disponível em: <http://www.linhadecodigo.com.br/ArtigoImpressao.aspx?id=1953>. Acesso em: 02 jul. 2011

VILAIN, Patrícia; FAGUNDES, Priscila Bastos; MACHADO, Thiago Leão; *A Framework for Selecting Agile Practices and Defining Agile Software Processes.*.

ZHANG, Yuefeng; PATEL, Shailesh; *Agile Model-Driven Development in Practice*

ZHANG, Yuefeng. Test-driven modeling for model-driven development. **Ieee Software**, v.21, n.5, p.80-86, out. 2004

# ANEXO 1 – Arquivos Projeto AlphaSimple

**1 – banking**

```
/* Copyright Abstratt Technologies 2011. All rights reserved. */

package banking;

enumeration Gender
  Male, Female
end;

class Person
    attribute birthDate : Date;
    attribute gender : Gender;

    derived attribute age : Integer := () : Integer {
        return Date#today().differenceInDays(self.birthDate);
    };

    derived attribute totalWorth : Double := () : Double {
        return (self<-PersonAccounts->accounts.reduce(
            (a : Account, partial : Double) : Double { return partial + a.balance}, 0.0
        ) as Double);
    };

    static operation accountOwners() : Person[*];
    begin
        return (Account extent.collect((a : Account) : Person { return a.owner }) as
Person)
    end;
    static operation today() : Date;
    begin
        return Date#today();
    end;
end;

association PersonAccounts
```

```
    role Account.owner;
    navigable role accounts : Account[*];
end;

abstract class Operation end;

class Deposit specializes Operation
    attribute amount : Double;
    attribute account : Account;
    static operation make(a : Account, amount : Double) : Deposit;
    begin
      var d : Deposit;
      d := new Deposit;
      d.amount := amount;
      link AccountDeposits(deposits:=d, account:= a);
      return d;
    end;
end;

class Withdrawal specializes Operation
    attribute amount : Double;
    attribute account : Account;
    static operation make(a : Account, amount : Double) : Withdrawal;
    begin
      var o : Withdrawal;
      o := new Withdrawal;
      o.amount := amount;
      link AccountWithdrawals(withdrawals:=o, account:= a);
      return o;
    end;
end;

association AccountWithdrawals
    role Account.withdrawals;
    role Withdrawal.account;
```

```
end;

association AccountDeposits
   role Account.deposits;
   role Deposit.account;
end;



(* A simple class to showcase POJO generation. *)
class Account
   (* The account number. *)
   attribute number : String;
   attribute owner : Person;
   (* The account balance. *)
   derived attribute balance : Double := () : Double {
      return self.totalDeposits - self.totalWithdrawals
   };
   derived attribute inGoodStanding : Boolean := () : Boolean { return
self.balance >= 0 };

   attribute deposits : Deposit[*];
   attribute withdrawals : Withdrawal[*];

   derived attribute totalDeposits : Double := () : Double {
      return (self.deposits.reduce(
         (a : Deposit, total : Double) : Double {
            return total + a.amount
         },
         0.0
      ) as Double);
   };

   derived attribute totalWithdrawals : Double := () : Double {
      return (self.withdrawals.reduce(
         (a : Withdrawal, total : Double) : Double {
```

```
            return total + a.amount
        },
        0.0
    ) as Double);
};
```

```
(* Deposits the given amount into the account, increasing the balance accordingly. *)
operation deposit(amount : Double);
precondition (amount) { return amount > 0 }
begin
    Deposit#make(self, amount);
end;
```

```
(* Withdraws the given amount from the account, decreasing the balance accordingly. *)
operation withdraw(amount : Double);
precondition INSUFFICIENT\ FUNDS (amount) { return amount <= self.balance }
precondition INVALID\ AMOUNT (amount) { return amount > 0 }
begin
    Withdrawal#make(self, amount);
end;
static operation allAccounts() : Account[*];
begin
    return Account extent;
end;
static operation goodAccounts() : Account[*];
begin
    return Account extent.select((a:Account):Boolean { return a.balance > 0});
end;
static operation theBestAccount() : Account;
begin
    var best : Account;
    best := null;
```

```
    Account extent.forEach((a:Account) { if ((best == null) or (a.balance > best.balance)) then best := a });
    return best;
  end;
  static operation bestAccounts(accounts : Account[*], threshold : Double) : Account[*];
  begin
    return accounts.select((a:Account):Boolean { return a.balance > threshold });
  end;
  static operation newAccount(number : String) : Account;
  begin
    var newAccount : Account;
    newAccount := new Account;
    newAccount.number := number;
    return newAccount;
  end;
  static operation massAdjust(rate : Double);
  begin
    Account extent.forEach(
      (a : Account) {
        a.deposit(a.balance*rate)
      }
    )
  end;
  static operation grandTotal(accounts : Account[*]) : Double;
  begin
    return (
      accounts.reduce(
        (a : Account, partial : Double) : Double {
          return partial + a.balance
        }, 0.0
      ) as Double
    );
  end;
```

```
    static operation bestSeniorFemaleCustomers(cutoff : Double) : Person[*];
    begin
        return
            (Account extent
                .select((a:Account) : Boolean { return a.balance > cutoff })
                .collect((a:Account) : Person { return a->owner }) as Person)
                .select((p:Person):Boolean { return (p.age > 65) and (p.gender =
Gender#Female) });
    end;

    static operation maleCustomers() : Person[*];
    begin
        return
            (Account extent
                .collect((a:Account) : Person { return a->owner }) as Person)
                .select((p:Person):Boolean { return p.gender = Gender#Male });
    end;

    static operation bestCustomers(threshold : Double) : Person[*];
    begin
        return
            (Account extent
                .select((a:Account) : Boolean { return a.balance > threshold })
                .collect((a:Account) : Person { return a->owner }) as Person);
    end;
end;

end.
```

**2 – basic_pojo.stg**

```
/* Copyright Abstratt Technologies 2011. All rights reserved. */

/*
This shows basic POJO generation by disabling some of the
```

```
advanced features such as behavior generation.
*/

group basic_pojo : pojo_struct;

outputPath(class) ::= "<class.nearestPackage.name>/<class.name>.java"

shouldMatch ::= ["Class" : "true", "Enumeration" : "true", default: "false"]

match(class) ::= "<shouldMatch.(class.metaClass)>"

initialization(attribute) ::= ""

derivedGetter(attribute) ::= ""

contents(class) ::= <<
<javaSource(class)>
>>
```

**3 - java_basic.stg**

```
/* Copyright Abstratt Technologies 2011. All rights reserved. */

group java_basic : basic;

javadoc(prefix, element) ::= <<
<if(element.ownedComments)>
/**
<if(prefix)>
  <prefix> <first(element.ownedComments).body:trim():uncapitalize()>
<else>
  <first(element.ownedComments).body:trim()>
<endif>

*/
```

```
<endif>
>>

javaSource(classifier) ::= <<
// Generated at <outputPath(classifier)>
<packageDecl(package=classifier.nearestPackage)>

<@imports()>

<@classifier()>
>>

visibility(namedElement) ::= <<
<javaVisibility.(namedElement.visibility)>
>>

javaVisibility ::= [
   "package" : "",
   "private" : "private ",
   "protected" : "protected ",
   "public" : "public "
]

packageDecl(package) ::= <<
package <package.qualifiedName>;
>>

typeName(typedElement) ::= <<
<if(typedElement.multivalued)><typedElement:collectionType()>\<<typedEleme
nt:simpleTypeName()>\><else><typedElement:simpleTypeName()><endif>
>>

simpleTypeName(typedElement) ::= <<
<typedElement.type.name>
>>
```

```
mapQualifiedName(qualifiedName) ::= "<qualifiedName>"

collectionType(multiplicityElement) ::= <<
<if(multiplicityElement.ordered)>
List
<else>
<if(multiplicityElement.unique)>Set<else>Collection<endif>
<endif>
>>

concreteCollectionType(multiplicityElement) ::= <<
<if(multiplicityElement.ordered)>
ArrayList
<else>
<if(multiplicityElement.unique)>HashSet<else>ArrayList<endif>
<endif>
\<<multiplicityElement:simpleTypeName()>\>
>>

parameterList(operation) ::= <<
<operation.ownedParameters:parameter(); separator=", ">
>>

parameter(parameter) ::= <<
<if(!parameter.return)><parameter:typeName()> <parameter.name><endif>
>>

returnType(operation) ::=
"<if(operation.type)><last(operation.ownedParameters):typeName()><else>void
<endif>"

closureReturnType(inputPin) ::= <<
<last(first(inputPin.incomings).source.owner.value.behavior.ownedParameters):
simpleTypeName()>
>>
```

```
EnumerationLiteral(literal) ::= "<literal.enumeration.name>.<literal.name>"

ValueSpecification(valueSpec) ::= "/* ValueSpecification: <valueSpec.metaClass> <valueSpec>*/"

LiteralNull(literal) ::= "null"


unaryOperator ::= [
	"not":"!",
	default :
]

binaryOperator ::= [
	"equals":"==",
	"same":"==",
	"add":"+",
	"subtract":"-",
	"and":"&&",
	"or":"||",
	"divide":"/",
	"multiply":"*",
	"greaterThan":"\>",
	"lowerThan":"\<",
	"greaterOrEquals":"\>=",
	"lowerOrEquals":"\<=",
	default :
]
```

**4 - java_enum.stg**

```
/* Copyright Abstratt Technologies 2011. All rights reserved. */

group java_enum : pojo_struct;
```

```
outputPath(class) ::= "<class.nearestPackage.name>/<class.name>.java"

shouldMatch ::= ["Enumeration" : "true", default: "false"]

match(class) ::= "<shouldMatch.(class.metaClass)>"

contents(class) ::= <<
<javaSource(class)>
>>
```

## 5 - mdd.properties

```
mdd.target.engine=stringtemplate
mdd.importedProjects=http://alphasimple.com/mdd/publisher/rafael-800/
mdd.target.POJO.template=pojo.stg
mdd.target.pojo_tests.template=pojo_tests.stg
mdd.target.pojo_tests.testing=true
mdd.enableTests=true
```

### 6 – pojo.stg

```
/* Copyright Abstratt Technologies 2011. All rights reserved. */

group pojo : pojo_struct;

outputPath(class) ::= "<class.nearestPackage.name>/<class.name>.java"

shouldMatch ::= ["Class" : "true", "Enumeration" : "true", default: "false"]

match(class) ::= "<shouldMatch.(class.metaClass)>"

contents(class) ::= <<
<javaSource(class)>
>>
```

```
@pojoClass.preamble() ::= <<
<if(class.abstract)>
<else>

/** Simulates the concept of a class extent for the (non-persistent) POJO target. */
private static Set\<<class.name>\> allInstances = new HashSet\<<class.name>\>();

public static Set\<<class.name>\> allInstances() {
   return Collections.unmodifiableSet(allInstances);
}
<endif>
>>

@pojoClass.constructorBody() ::= <<
<if(class.abstract)>
<else>
allInstances.add(this);
<endif>
>>

ReadExtentAction(action) ::= "<pre><action.classifier.name>.allInstances()<post>"
```

**7 - pojo_behavior.stg**

```
/* Copyright Abstratt Technologies 2011. All rights reserved. */

/*
   This group cannot be used directly as a target platform. It is intended to be used by other templates as a base group.
*/
group pojo_behavior : java_basic;
```

```
outputPath(class) ::= "<class.nearestPackage.name>/<class.name>.java"

/* An operation can declare one or more methods (we only support 1). */
method(operation) ::= <<
<generateMethod(specification = operation, generator = "activity")>
>>

/* A derived attribute can declare a derivation method. */
derivation(attribute) ::= <<
<generateBody(activity = attribute.derivation, generator = "body")>
>>

activity(activityToRender) ::= <<
<generateBody(activity = activityToRender, generator = "body")>
>>

body(bodyToRender) ::= <<
<blockWithLocalVars(action =
first(bodyToRender.nodes),beforeblock=activity.specification:preconditions())>
>>

preconditions(operation) ::= <<
<operation.preconditions:{p|<p:precondition()>};separator="\n">
>>

statement(action) ::= "<weaveAction(action=action)>;"

expression(action) ::= "<weaveAction(action=action)>"

/* TODO: remove prefix and things break down - why?*/
blockWithLocalVars(action, beforeblock,prefix="") ::= <<
{
   <if(beforeblock)><beforeblock><endif>
   <action.variables:{v|<v:typeName()> <v.suggestedName>;};separator="\n">
   <action.nodes:{a|<if(a.terminal)><a:statement()><endif>};separator="\n">
```

```
}
>>

blockAsExpression(action) ::= <<
<first(action.nodes:{a|<if(a.terminal)><a:expression()><endif>})>
>>

/*assert <constraint.specification:value()>;*/
precondition(constraint) ::= <<
assert <constraint.specification:value()>;
>>

CallOperationAction(action) ::= <<
<generateCallOperationAction(action=action, prefix="")>
>>

/**
 * Produces the code in the closure that is parameter
 * to a collection-based CallOperationAction.
 */
unrollClosure(action) ::= <<
<first(action.arguments):composeSourceAction()>
>>

/**
 * Generic rendering of call operation actions.
 * If the source action is a CallOperationAction (that produces a collection), render
 * it as a collection operation as well (and embeds the current operation rendering).
 * If not, render a for each loop that iterates over the result of the action.
 *
 * @param action the CallOperationAction representing the collection operation invocation
 * @param toEmbed this is the rendering of the collection operation that goes within the loop
```

```
 */
renderUpstream(action,toEmbed,resultRet="") ::= <<
<if(action.target.sourceAction.operation)>
<doGenerateCollectionOpCallOperationAction(action=action.target.sourceActio
n,embed=(toEmbed))>
<else>
<action:resultDeclaration()>for (<action.target:simpleTypeName()>
<first(action.parameterVariables)> : <action.target:composeSourceAction()>) {
   <toEmbed>
}<resultRet>
<endif>
>>

size(action) ::= "<pre>/*.size()*/<post>"

/* for-each has no downstream */
forEach(action,implicitVariable={<(first(action.parameterVariables))>}) ::= <<
<renderUpstream(action=action,toEmbed={<unrollClosure(action=action)>;})>
>>

resultDeclaration(action) ::= <<
<if(action.outputs.empty)>
<first(action.inputs):resultDeclarationKernel()>
<else>
<if(first(action.outputs).targetAction)>
<first(action.outputs).targetAction:resultDeclaration()>
<else>
<first(action.outputs):resultDeclarationKernel()>
<endif>
<endif>
>>

resultDeclarationKernel(objectNode) ::= <<
<if(objectNode.multivalued)>
<objectNode:typeName()> result = new
<action.value:concreteCollectionType()>();
```

```
<endif>
>>

/* Collect declares a new local variable. Implicit variable is replaced with a new one. */
collect(action,collectVariable={<(action.resultVariable)>},implicitVariable={<(first(action.parameterVariables))>}) ::= <<
<renderUpstream(action=action, resultRet={<\n><pre>result<post>}, toEmbed=collectEmbeddable(action=action,collectVariable=collectVariable,implicitVariable=implicitVariable))>
>>

collectEmbeddable(action,collectVariable, implicitVariable) ::= <<
<first(action.arguments):closureReturnType()> <collectVariable> = <unrollClosure(action=action)>;
<embed>
>>

/**
 * Renders a select as a for-each loop that will add
 * elements that satisfy a condition to the result
 * collection.
 *
 * @param action the CallOperationAction that invokes #select on the input set
 * @param implicitVariable
 */
select(action,implicitVariable={<(first(action.parameterVariables))>}) ::= <<
<renderUpstream(action=action,resultRet={<\n><pre>result<post>},toEmbed={if (<unrollClosure(action=action)>) \{
   <embed>
\}})>
>>

/**
 * Renders a reduce operation.
 */
```

```
reduce(action,implicitVariable={<(first(action.parameterVariables))>}) ::= <<
<generateOutputType(node=first(action.results),generator="typeName")>
<last(action.parameterVariables)>;
<last(action.parameterVariables)> =
<first(rest(action.arguments)).sourceAction:expression()>;
<renderUpstream(action=action,toEmbed={<last(action.parameterVariables)> =
<unrollClosure(action=action)>;})>
<pre><last(action.parameterVariables)><post>
>>

/**
 * Renders the action as a collection operation call operation action.
 */
doGenerateCollectionOpCallOperationAction(
    action,
    prefix,
    embed={result.add(<action.resultVariable>);}) ::= <<
<(action.operation.name)(action=action)>
>>

/**
 * Renders a call operation action as a normal Java method invocation.
 *
 * @param action a CallOperationAction to be rendered
 */
generateOrdinaryCallOperationAction(action) ::= <<
<pre><generateOrdinaryCallOperationAction_kernel(action=action)><post>
>>

generateOrdinaryCallOperationAction_kernel(action,pre="",post="") ::= <<
<if(action.operation.static)>
<action:(static_CallOperationAction_mapper.(action.operation.qualifiedName))()
>
<else>
<action:(instance_CallOperationAction_mapper.(action.operation.qualifiedName
))()>
<endif>
```

```
>>

java_argumentList(action) ::= <<
<action.arguments:{arg|<arg:composeSourceAction()>};separator=", ">
>>

generateStaticCallOperationAction(action) ::= <<
<action.operation.class_.name>.<action.operation.name>(<action:java_argume
ntList()>)
>>

generateInstanceCallOperationAction(action) ::= <<
<action.target:composeSourceAction()>.<action.operation.name>(<action:java_
argumentList()>)
>>

static_CallOperationAction_mapper ::= [
   "mdd_types::Date::today" : "static_Date_today",
   default: "generateStaticCallOperationAction"
]

instance_CallOperationAction_mapper ::= [
   "mdd_types::Date::differenceInDays" : "instance_Date_differenceInDays",
   "mdd_types::Date::differenceInYears" : "instance_Date_differenceInYears",
   "mdd_types::Boolean::not" : "instance_Boolean_not",
   default: "generateInstanceCallOperationAction"
]

static_Date_today(action) ::= "new Date()"

instance_Date_differenceInDays(action) ::= <<
(int)          ((<action.target:composeSourceAction()>).getTime()          -
(<first(action.arguments):composeSourceAction()>).getTime()) / (1000 * 60 * 60
* 24)
>>
```

```
instance_Date_differenceInYears(action) ::= <<
(<action:instance_Date_differenceInDays()>) / 365
>>

generateUnaryOpCallOperationAction(action, prefix) ::= <<
<pre><generateUnaryOpCallOperationAction_kernel(action=action)><post>
>>

generateUnaryOpCallOperationAction_kernel(action, pre="", post="") ::= <<
<unaryOperator.(action.operation.name)>(<action.target.sourceAction:prefixed_
action()>)
>>

generateBinaryOpCallOperationAction(action, prefix) ::= <<
<pre><generateBinaryOpCallOperationAction_kernel(action=action)><post>
>>

generateBinaryOpCallOperationAction_kernel(action, pre="", post="") ::= <<
<action.target.sourceAction:prefixed_action()>
<binaryOperator.(action.operation.name)>
<first(action.arguments).sourceAction:prefixed_action()>
>>



ConditionalNode(action) ::= <<
<pre>if                    (<first(first(action.clauses).tests):blockAsExpression()>)
<blockWithLocalVars(action=first(first(action.clauses).bodies))><post>
>>

AddStructuralFeatureValueAction(action) ::= <<
<pre><if(action.structuralFeature.public)>
<action.object:composeSourceAction()>.set<action.structuralFeature.name;for
mat="capitalize">(<action.value:composeSourceAction()>)
<else>
<action.object:composeSourceAction()>.<action.structuralFeature.name>        =
<action.value:composeSourceAction()>
<endif><post>
```

```
>>

AddVariableValueAction(action) ::= <<
<if(emptyString.(action.variable.name))>
<action:returnStatement()>
<else>
<action:assignToVariable()>
<endif>
>>

assignToVariable(action) ::= <<
<weaveAction(action=action.value.sourceAction,pre=action:assignToVariable_p
re())>
>>

assignToVariable_pre(action) ::= <<
<action.variable.suggestedName> =
>>



/**
 * A return in a closure just produces the value of the closure as an expression.
 */
returnInExpression(action) ::= "<action.value:composeSourceAction()>"

/**
 * A return in a method will just return with the value.
 * In some cases we can't do that though - say, the expression involves
collection operations and closures.
 */
returnInMethod(action) ::= <<
<weaveAction(action=action.value.sourceAction,pre=action:returnInMethod_pre
())>
>>

returnInMethod_pre(action) ::= <<
```

```
return
>>

/**
 * Handles a AddVariableValueAction on a anonymous variable (return value).
 * For an operation method, generates a "return" statement with the value expression.
 * For a closure, produces the value expression only.
 * For a derivation method, generates a "return" statement with the value expression.
 * For a precondition expression, generates the value expression only.
 */
returnStatement(action) ::= <<
<if(action.owner.owner.activity.specification.methods)>
<action:returnInMethod()><elseif(action.owner.owner.activity.asClosure)>
<action:returnInExpression()><elseif(action.owner.owner.activity.asConstraintBehavior)>
<action:returnInExpression()><else>
<action:returnInMethod()><endif>
>>

ReadSelfAction(action) ::= "<pre>this<post>"

ReadVariableAction(action) ::= "<pre><action.variable.suggestedName><post>"

ReadStructuralFeatureAction(action) ::= "<pre><action.object:composeSourceAction()>.<if(action.structuralFeature.public)>get<action.structuralFeature.name:capitalize()>()<else><action.structuralFeature.name><endif><post>"

ValueSpecificationAction(action) ::= "<pre><action.value:value()><post>"

CreateObjectAction(action) ::= "<pre>new <action.classifier.name>()<post>"

DestroyObjectAction(action) ::= "<pre>allInstances.remove(this)<post>"

DestroyLinkAction(action) ::= <<
```

```
<pre><DestroyLinkActionHelper(sourceEndData=first(action.endData),targetEndData=last(action.endData))><if(first(action.endData).end.navigable)><if(last(action.endData).end.navigable)>;<\n><endif><endif><DestroyLinkActionHelper(targetEndData=first(action.endData),sourceEndData=last(action.endData))><post>

>>


DestroyLinkActionHelper(sourceEndData, targetEndData, insertSemiColon) ::= <<

<pre><if(targetEndData.end.navigable)>

<sourceEndData.value:composeSourceAction()>.<if(sourceEndData.end.public)>

<if(targetEndData.end.multivalued)>remove<targetEndData.end.name;format="capitalize">(<targetEndData.value:composeSourceAction()>)<else>set<targetEndData.end.name;format="capitalize">(null)<endif>

<else>

<targetEndData.end.name><if(targetEndData.end.multivalued)>.remove(<targetEndData.value:composeSourceAction()>)<else> = null<endif>

<endif>

<endif><post>

>>


RaiseExceptionAction(action) ::= <<

<pre>throw <action.exception:composeSourceAction()><post>

>>



ReadLinkAction(action) ::= <<

<pre><ReadLinkActionHelper(sourceEndData=last(action.endData))><post>

>>


ReadLinkActionHelper(sourceEndData) ::= <<

<pre><sourceEndData.value:composeSourceAction()>.<if(sourceEndData.end.otherEnd.public)>

get<sourceEndData.end.otherEnd.name;format="capitalize">()

<else>

<sourceEndData.end.otherEnd.name>

<endif><post>
```

```
>>

CreateLinkAction(action) ::= <<
<pre><CreateLinkActionHelper(sourceEndData=first(action.endData),targetEndData=last(action.endData))><if(first(action.endData).end.navigable)><if(last(action.endData).end.navigable)>;<\n><endif><endif><CreateLinkActionHelper(targetEndData=first(action.endData),sourceEndData=last(action.endData))><post>
>>

CreateLinkActionHelper(sourceEndData, targetEndData, insertSemiColon) ::= <<
<if(targetEndData.end.navigable)>
<sourceEndData.value:composeSourceAction()>.<if(sourceEndData.end.public)>
<if(targetEndData.end.multivalued)>add<else>set<endif><targetEndData.end.name;format="capitalize">(<targetEndData.value:composeSourceAction()>)
<else>
<targetEndData.end.name><if(targetEndData.end.multivalued)>.add(<targetEndData.value:composeSourceAction()>)<else> = <targetEndData.value:composeSourceAction()><endif>
<endif>
<endif>
>>

OpaqueExpression(expression) ::= <<
<generateExpression(activity=expression.behavior,generator="expression")>
>>

LiteralString(literalString) ::= "<if(literalString.asBasicValue)><literalString:basicValue()><else><literalString.value:simpleValue()><endif>"

basicValue(value) ::= "<delimitedValue(value=value.value, type=value.asBasicValue.basicType.name)>"

simpleValue(value) ::= "<delimitedValue(value=value, type=value.class.simpleName)>"
```

```
delimitedValue(value, type) ::= "<valueDelimiters.(type)><value><valueDelimiters.(type)>"

valueDelimiters ::= ["String":"\"",default:""]
```

**8 - pojo_struct.stg**

```
/* Copyright Abstratt Technologies 2011 */

/*
  POJO generation.
*/

group pojo_struct : pojo_behavior;

@javaSource.imports() ::= <<
import java.util.*;
>>

/* Delegates to pojoClass, pojoEnumeration, pojo<...> depending on the metaclass. */
@javaSource.classifier() ::= <<
<classifier:(concat(v1="pojo",v2=classifier.metaClass))()>
>>

pojoEnumeration(enumeration) ::= <<
<javadoc(element=enumeration)>
<@javaAnnotations()>
<visibility(enumeration)>enum <enumeration.name> {
   <enumeration.ownedLiterals:simpleName(); separator=", ">
}
>>

pojoClass(class) ::= <<
<javadoc(element=class)>
<@javaAnnotations()>
```

```
<visibility(class)>class <class.name> {
   <@preamble()>

   /* Attributes */
   <class.ownedAttributes:attribute(); separator="\n\n">

   /* Relationships */
   <class.navigableAssociationEnds:attribute(); separator="\n">

   /* Constructor */
   public <class.name>() {
      <@constructorBody()>
   }

   /* Attribute setters/getters */
   <class.ownedAttributes:setterAndGetter(); separator="\n">

   /* Association getters/setters */
   <class.navigableAssociationEnds:setterAndGetter(); separator="\n">

   /* Operations */
   <class.ownedOperations:operation(); separator="\n\n">
}
>>

annotation(annotation) ::= <<
@<annotation.name><if(annotation.valueMap)>(<annotation.valueMap.keys:{k|
<k>=<annotation.valueMap.(k)>}; separator=", ">)<endif>
>>

annotationAttribute(key,value) ::= <<
<key>=<value>
>>

packageImport(importedType) ::= <<
```

```
import <importedType>;
>>

attribute(attribute) ::= <<
<if(!attribute.derived)>
<javadoc(element=attribute)>
private <attribute:typeName()> <attribute.name><attribute:initialization()>;
<endif>
>>

initialization(attribute) ::= <<
<if(attribute.defaultValue)>
 = <attribute.defaultValue:value()>
<endif>
>>

setterAndGetter(attribute) ::= <<
<if(attribute.public)>
<if(!attribute.derived)>
<attribute:getter()>
<if(!attribute.readOnly)>
<if(attribute.multivalued)>

<attribute:adder()>

<attribute:remover()>
<else>

<attribute:setter()>
<endif>
<endif>
<else>

<attribute:derivedGetter()>
<endif>
```

```
<endif>
>>


getter(attribute) ::= <<

<javadoc(element=attribute,prefix="Returns")>
<attribute.javaGetterAnnotations:annotation(); separator="\n">
public <attribute:typeName()> <getterPrefix.(attribute:typeName())><attribute.name:capitalize()>() {
   return this.<attribute.name>;
}>>

getterPrefix ::= [ "Boolean" : "is", "boolean" : "is", default : "get" ]

setter(attribute) ::= <<
<javadoc(element=attribute,prefix="Sets")>
public void set<attribute.name;format="capitalize">(<attribute:typeName()> <attribute.name>) {
<if(attribute.required)>
   assert <attribute.name> != null;

<endif>
   this.<attribute.name> = <attribute.name>;
   <setOtherEnd(attribute)>
}
>>

setOtherEnd(attribute) ::= <<
<if(attribute.association)>
<if(attribute.otherEnd.navigable)>
this.<attribute.name>.<if(attribute.otherEnd.public)>
<if(attribute.otherEnd.multivalued)>add<else>set<endif><attribute.otherEnd.name;format="capitalize">(this)
<else>
```

```
<attribute.otherEnd.name><if(attribute.otherEnd.multivalued)>.add(this)<else>
= this<endif>

<endif>;

<endif>

<endif>

>>


adder(attribute) ::= <<

<javadoc(element=attribute,prefix="Adds")>

public void
add<attribute.name;format="capitalize">(<attribute:simpleTypeName()>...
toAdd) {

this.<attribute.name>.addAll(Arrays.\<<attribute:simpleTypeName()>\>asList(to
Add));

   <setOtherEnd(attribute)>

}


<javadoc(element=attribute,prefix="Adds")>

public void add<attribute.name;format="capitalize">(<attribute:typeName()>
toAdd) {

   this.<attribute.name>.addAll(toAdd);

   <setOtherEnd(attribute)>

}>>


remover(attribute) ::= <<

<javadoc(element=attribute,prefix="Removes")>

public void
remove<attribute.name;format="capitalize">(<attribute:simpleTypeName()>...
toRemove) {

<if(attribute.required)>

   if ((this.<attribute.name>.size() - toRemove.length) \< attribute.lower)

      throw new IllegalStateException();


<endif>


this.<attribute.name>.removeAll(Arrays.\<<attribute:simpleTypeName()>\>asLis
t(toRemove));
```

```
}

<javadoc(element=attribute,prefix="Removes")>
public void remove<attribute.name;format="capitalize">(<attribute:typeName()>
toRemove) {
<if(attribute.required)>
   if ((this.<attribute.name>.size() - toRemove.size()) \< attribute.lower)
      throw new IllegalStateException();

<endif>
   this.<attribute.name>.removeAll(toRemove);
}
>>

toString(value) ::= "<value>"

operation(operation) ::= <<
<if(operation.asQuery)>
<operation:queryOperation()><else><operation:basicOperation()>
<endif>
>>

/* By default we map queries as basic operations. */
queryOperation(operation) ::= "<operation:basicOperation()>"

basicOperation(operation) ::= <<
<javadoc(element=operation)>
<operation:signature()> <operation:method()>
>>

signature(operation) ::= <<
<operation:visibility()><if(operation.static)>static
<endif><operation:returnType()>
<operation.name>(<operation:parameterList()>)
>>
```

```
derivedGetter(attribute) ::= <<
<javadoc(element=attribute,prefix="Returns the")>
public <attribute:typeName()> <getterPrefix.(attribute:typeName())><attribute.name;format="capitalize">() <attribute:derivation()>
>>
```

## 9 - pojo_tests.stg

```
group pojo_tests : pojo;

/* Enumerations */
actual_pojo_enumeration(element, elementName = "banking::Gender") ::= "<element:pojoEnumeration()>"
expected_pojo_enumeration() ::= <<
public enum Gender {
   Male, Female
}
>>

/* POJO getter */
actual_attribute_getter(element, elementName = "banking::Account::number") ::= "<element:getter()>"
expected_attribute_getter() ::= <<
/**
  Returns the account number.
*/
public String getNumber() {
   return this.number;
}
>>

/* POJO setter */
actual_attribute_setter(element, elementName = "banking::Account::number") ::= "<element:setter()>"
expected_attribute_setter() ::= <<
```

```
/**
  Sets the account number.
*/
public void setNumber(String number) {
    assert number != null;
    this.number = number;
}
>>

/* Operation signatures */
actual_query_signature(element, elementName = "banking::Account::bestAccounts") ::= "<element:signature()>"
expected_query_signature() ::= <<
public static Set\<Account\> bestAccounts(Set\<Account\> accounts, Double threshold)
>>

actual_action_signature(element, elementName = "banking::Account::deposit") ::= "<element:signature()>"
expected_action_signature() ::= <<
public void deposit(Double amount)
>>

/* Method definitions */
actual_select(element, elementName = "banking::Account::bestAccounts") ::= "<element:method()>"
expected_select() ::= <<
{
    Set\<Account\> result = new HashSet\<Account\>();
    for (Account a : accounts) {
        if (a.getBalance() > threshold) {
            result.add(a);
        }
    }
    return result;
}
```

```
>>

actual_collect(element, elementName = "banking::Person::accountOwners") ::=
"<element:method()>"
expected_collect() ::= <<
{
   Set\<Person\> result = new HashSet\<Person\>();
   for (Account a : Account.allInstances()) {
      Person mapped = a.getOwner();
      result.add(mapped);
   }
   return result;
}
>>

actual_forEach(element, elementName = "banking::Account::massAdjust") ::=
"<element:method()>"
expected_forEach() ::= <<
{
   for (Account a : Account.allInstances()) {
      a.deposit(a.getBalance() * rate);
   };
}
>>

actual_reduce(element, elementName = "banking::Account::grandTotal") ::=
"<element:method()>"
expected_reduce() ::= <<
{
   Double partial;
   partial = 0.0;
   for (Account a : accounts) {
      partial = partial + a.getBalance();
   }
   return partial;
}
```

```
>>

actual_select_collect(element, elementName = "banking::Account::bestCustomers") ::= "<element:method()>"
expected_select_collect() ::= <<
{
   Set\<Person\> result = new HashSet\<Person\>();
   for (Account a : Account.allInstances()) {
      if (a.getBalance() > threshold) {
         Person mapped = a.getOwner();
         result.add(mapped);
      }
   }
   return result;
}
>>

actual_read_extent(element, elementName = "banking::Account::allAccounts") ::= "<element:method()>"
expected_read_extent() ::= <<
{
   return Account.allInstances();
}
>>

actual_action_method(element, elementName = "banking::Account::deposit") ::= "<element:method()>"
expected_action_method() ::= <<
{
   assert amount \> 0;
   Deposit.make(this, amount);
}
>>

actual_derived_attribute_method(element, elementName = "banking::Account::balance") ::= "<element:derivation()>"
```

```
expected_derived_attribute_method() ::= <<
{
   return this.getTotalDeposits() - this.getTotalWithdrawals();
}
>>

actual_date_today(element, elementName = "banking::Person::today") ::=
"<element:method()>"
expected_date_today() ::= <<
{
   return new Date();
}
>>

actual_date_difference(element, elementName = "banking::Person::age") ::=
"<element:derivation()>"
expected_date_difference() ::= <<
{
   return (int) ((new Date()).getTime() - (this.getBirthDate()).getTime()) / (1000 *
60 * 60 * 24);
}
>>
```

## ANEXO 2 – Código Aplicação Restaurante

### 1 – Garcom.java

```
package Restaurante;

import java.util.List;

/**
 * @author Toco
 * @version 1.0
 * @created 28-mai-2012 21:03:46
 */
```

```java
public class Garcom {

    private int id;
    private String nome;
    private Restaurante m_Restaurante;
    private RegistroDeTrabalho m_RegistroDeTrabalho;
    private List<Mesa> mesasEmAtendimento;
    private List<Pedido> pedidosEmAndamento;
  private List<Pedido> pedidosFinalizados;

    public Garcom(){

    }

  public int getId() {
     return id;
  }

  public void setId(int id) {
     this.id = id;
  }

  public String getNome() {
     return nome;
  }

  public void setNome(String nome) {
     this.nome = nome;
  }


    public void finalize() throws Throwable {

    }
    public Restaurante getRestaurante(){
```

```
            return m_Restaurante;
    }

    /**
     *
     * @param newVal
     */
    public void setRestaurante(Restaurante newVal){
            m_Restaurante = newVal;
    }

    public RegistroDeTrabalho getRegistroDeTrabalho(){
            return m_RegistroDeTrabalho;
    }

    /**
     *
     * @param newVal
     */
    public void setRegistroDeTrabalho(RegistroDeTrabalho newVal){
            m_RegistroDeTrabalho = newVal;
    }
 public void ocuparMesa(Mesa m){
     m.setGarcom(this);
     m.setEmAtendimento(true);
     mesasEmAtendimento.add(m);
 }

 void liberarMesa(Mesa m){
     m.setEmAtendimento(false);
     m.setGarcom(null);
 }
    /**
     * @param horarioInicio
     *
```

```
  * @param dataInicio    dataInicio
  * @param horarioInicial
  */
public void inicairTrabalho(int dataInicio, int horarioInicial){
  m_Restaurante.iniciartrabalhoGarcom(dataInicio , horarioInicial, this);
}

 /**
  *
  * @param pedido
  * @param valor
  */
 public void atualizarPedido(Pedido pedido, double valor){
  pedido.setValorServico(valor);
 }

 /**
  *
  * @param mesa
  * @param valor
  */
 public void criarPedido(Mesa mesa , double valor){
  Pedido p = new Pedido();
  p.setGarcom(this);
  p.setValorServico(valor);
  p.setMesa(mesa);
  pedidosEmAndamento.add(p);
  mesa.setPedidoAtual(p);
 }

 /**
  *
  * @param pedido
  */
 public void fecharPedido(Pedido pedido){
```

```
            pedidosEmAndamento.remove(pedido);
            pedidosFinalizados.add(pedido);
            m_Restaurante.finalizarPedido(pedido);
            pedido.getMesa().setPedidoAtual(null);
            mesasEmAtendimento.remove(pedido.getMesa());
        }

        /**
         * @param horarioInicio
         *
         * @param dataFim    dataInicio
         * @param horarioFinal
         */
        public void finalizarTrabalho(int dataFim, int horarioFinal){
         if(mesasEmAtendimento.isEmpty())
            m_Restaurante.finalizarTrabalhoGarcom( this,dataFim, horarioFinal);
         else
            System.out.println("Finalizar atendimento das mesas, ou transferí-
las");
        }

    public List<Pedido> getPedidosFinalizados() {
        return pedidosFinalizados;
    }

    public void setPedidosFinalizados(List<Pedido> pedidosFinalizados) {
        this.pedidosFinalizados = pedidosFinalizados;
    }

    public List<Mesa> getMesasEmAtendimento() {
        return mesasEmAtendimento;
    }

    public void setMesasEmAtendimento(List<Mesa> mesasEmAtendimento) {
        this.mesasEmAtendimento = mesasEmAtendimento;
```

```
    }



    public void imprimirRelatorioTrabalho(){
     System.out.println("----------------------- Dados do Turno ----------------------
");
     System.out.println("Data Início:" +
m_RegistroDeTrabalho.getDataInicio());
     System.out.println("Horário Inicial:" +
m_RegistroDeTrabalho.getHoraInicio());
     System.out.println("Data Final:" + m_RegistroDeTrabalho.getDataFim());
     System.out.println("Horário final:" +
m_RegistroDeTrabalho.getHoraFim());
     System.out.println("Total Pedidos Atendidos" +
m_RegistroDeTrabalho.getTotalPedidosAtendidos());
     System.out.println("Valor Total Serviços" +
m_RegistroDeTrabalho.getValorTotalVendido());
     System.out.println("Total Gorjetas" +
m_RegistroDeTrabalho.getValorTotalGorjetas());
     System.out.println("Total Arrecadado" +
m_RegistroDeTrabalho.getValorTotalArrecadado());
    }

    /**
     *
     * @param g
     * @param m
     */
    public void transferirMesa(Garcom g, Mesa m){
     m.setGarcom(g);
     mesasEmAtendimento.remove(m);
     g.getMesasEmAtendimento().add(m);

    }
}//end Garcom
```

**2 – Mesa.java**

```
package Restaurante;

/**
 * @author Toco
 * @version 1.0
* @created 28-mai-2012 20:53:08
 */
public class Mesa {

        private boolean emAtendimento;
        private int id;
        private int numero;
        private Garcom garcom;
        private Pedido pedidoAtual;

        public Mesa(){

        }

      public boolean isEmAtendimento() {
         return emAtendimento;
      }

      public void setEmAtendimento(boolean emAtendimento) {
         this.emAtendimento = emAtendimento;
      }

      public int getId() {
         return id;
      }

      public void setId(int id) {
         this.id = id;
      }
```

```
public int getNumero() {
   return numero;
}

public void setNumero(int numero) {
   this.numero = numero;
}

  public void finalize() throws Throwable {

  }

public Garcom getGarcom() {
   return garcom;
}

public void setGarcom(Garcom garcom) {
   this.garcom = garcom;
}

public Pedido getPedidoAtual() {
   return pedidoAtual;
}

public void setPedidoAtual(Pedido pedidoAtual) {
   this.pedidoAtual = pedidoAtual;
}


  /**
   *
   * @param valor
   */
  public void atualizarPedido(double valor){
```

```
        garcom.atualizarPedido(pedidoAtual, valor);
    }

    /**
     *
     * @param valor
     */
    public void fazerPedido(double valor){
        garcom.criarPedido(this, valor);
    }

    /**
     *
     * @param gorjeta
     */
    public void fecharPedido(double gorjeta){
        pedidoAtual.setValorGorjeta(gorjeta);
        pedidoAtual.setValorTotal(gorjeta + pedidoAtual.getValorServico());
        garcom.fecharPedido(pedidoAtual);
    }
}//end Mesa
```

**3 – Pedido.java**

```
package Restaurante;

/**
 * @author Toco
 * @version 1.0
 * @created 29-mai-2012 02:46:22
 */
public class Pedido {

    private double valorGorjeta;
    private double valorServico;
```

```
  private double valorTotal;
  private Mesa m_Mesa;
  private Garcom garcom;

  public Pedido(){

  }

  public void finalize() throws Throwable {

  }
  public Mesa getMesa(){
          return m_Mesa;
  }

  /**
   *
   * @param newVal
   */
  public void setMesa(Mesa newVal){
          m_Mesa = newVal;
  }

public Garcom getGarcom() {
   return garcom;
}

public void setGarcom(Garcom garcom) {
   this.garcom = garcom;
}

public double getValorGorjeta() {
   return valorGorjeta;
}
```

```
        public void setValorGorjeta(double valorGorjeta) {
            this.valorGorjeta = valorGorjeta;
        }

        public double getValorServico() {
            return valorServico;
        }

        public void setValorServico(double valorServico) {
            this.valorServico = valorServico;
        }

        public double getValorTotal() {
            return valorTotal;
        }

        public void setValorTotal(double valorTotal) {
            this.valorTotal = valorTotal;
        }


}//end Pedido
```

**4 – RegistroDeTrabalho.java**

```
package Restaurante;

/**
 * @author Toco
 * @version 1.0
 * @created 28-mai-2012 21:18:59
 */
public class RegistroDeTrabalho {

        private int dataFim;
```

```
  private int dataInicio;
  private int horaFim;
  private int horaInicio;
  private int totalPedidosAtendidos;
  private double valorTotalArrecadado;
  private double valorTotalGorjetas;
  private double valorTotalVendido;
  private Garcom m_Garcom;

  public RegistroDeTrabalho(){

  }

  public void finalize() throws Throwable {

  }

public int getDataFim() {
   return dataFim;
}

public void setDataFim(int dataFim) {
   this.dataFim = dataFim;
}

public int getDataInicio() {
   return dataInicio;
}

public void setDataInicio(int dataInicio) {
   this.dataInicio = dataInicio;
}

public int getHoraFim() {
   return horaFim;
```

```
}

public void setHoraFim(int horaFim) {
    this.horaFim = horaFim;
}

public int getHoraInicio() {
    return horaInicio;
}

public void setHoraInicio(int horaInicio) {
    this.horaInicio = horaInicio;
}

public int getTotalPedidosAtendidos() {
    return totalPedidosAtendidos;
}

public void setTotalPedidosAtendidos(int totalPedidosAtendidos) {
    this.totalPedidosAtendidos = totalPedidosAtendidos;
}

public double getValorTotalArrecadado() {
    return valorTotalArrecadado;
}

public void setValorTotalArrecadado(double valorTotalArrecadado) {
    this.valorTotalArrecadado = valorTotalArrecadado;
}

public double getValorTotalGorjetas() {
    return valorTotalGorjetas;
}

public void setValorTotalGorjetas(double valorTotalGorjetas) {
```

```
        this.valorTotalGorjetas = valorTotalGorjetas;
    }

    public double getValorTotalVendido() {
        return valorTotalVendido;
    }

    public void setValorTotalVendido(double valorTotalVendido) {
        this.valorTotalVendido = valorTotalVendido;
    }

     public Garcom getGarcom(){
            return m_Garcom;
     }

     /**
      *
      * @param newVal
      */
     public void setGarcom(Garcom newVal){
            m_Garcom = newVal;
     }
}//end RegistroDeTrabalho
```

**5 – Rstaurante.java**

```
package Restaurante;

import java.util.ArrayList;
import java.util.List;

/**
 * @author Toco
 * @version 1.0
 * @created 28-mai-2012 21:08:09
 */
```

```
public class Restaurante {

        private String nome;
        private List<Mesa> m_Mesa;
        private List<RegistroDeTrabalho> m_RegistroDeTrabalho;
      private List<Garcom> m_Garcom;
      private int idGarcom = 1;
      private int idMesa = 1;
        private List<Pedido> m_Pedido;

        public Restaurante(){

        }

        public void finalize() throws Throwable {

        }
        public List<Garcom> getGarcom(){
                return m_Garcom;
        }

        public List<Mesa> getMesa(){
                return m_Mesa;
        }

        /**
         *
         * @param newVal
         */
        public void setGarcom(List<Garcom> newVal){
                m_Garcom = newVal;
        }

        /**
         *
```

```
     * @param newVal
     */
    public void setMesa(List<Mesa> newVal){
            m_Mesa = newVal;
    }

    public List<RegistroDeTrabalho> getRegistroDeTrabalho(){
            return m_RegistroDeTrabalho;
    }

    /**
     *
     * @param newVal
     */
    public void setRegistroDeTrabalho(List<RegistroDeTrabalho> newVal){
            m_RegistroDeTrabalho = newVal;
    }

    /**
     *
     * @param nome
     */
   public void cadastrarGarcom(String nome){
     Garcom g = new Garcom();
     List<Mesa> m = new ArrayList();
     g.setId(idGarcom);
     idGarcom += 1;
     g.setNome(nome);
     g.setRestaurante(this);
     m_Garcom.add(g);
}

    /**
     *
     * @param numero
```

```
 */
public void cadastrarMesa(int numero){
 Mesa m = new Mesa();
 m.setId(idMesa);
 idMesa += 1;
 m.setNumero(numero);
 m.setEmAtendimento(false);
 m_Mesa.add(m);
}

/**
 *
 * @param horarioInicio
 * @param dataInicio
 */
public void iniciartrabalhoGarcom(int dataInicio, int horarioIncial, Garcom g)
{
   RegistroDeTrabalho r = new RegistroDeTrabalho();
   r.setDataInicio(dataInicio);
   r.setHoraInicio(horarioIncial);
   g.setRegistroDeTrabalho(r);

}


public List<Pedido> getPedido(){
        return m_Pedido;
}

/**
 *
 * @param newVal
 */
public void setPedido(List<Pedido> newVal){
        m_Pedido = newVal;
}
```

```
public void exibirTotalDePedidosAtendidos(){
 System.out.println("Total Pedidos Atendidos pelo Restaurante: " +m_Pedido.size());
}

public void exibirValorTotalDeGorjetasArecadadas(){
 double gorjetas = 0;
 for(Pedido p : m_Pedido){
    gorjetas += p.getValorGorjeta();
 }
 System.out.println("Total Gorjetas Arrecadas pelos Garcçons: " + gorjetas);
}

public void exibirValorTotalVendido(){
 double valorServicos = 0;
 for(Pedido p : m_Pedido){
    valorServicos += p.getValorGorjeta();
 }
 System.out.println("Total Valor Serviços: " + valorServicos);
}

public void exibirArrecadacaoTotal(){
 double valorServicos = 0;
 for(Pedido p : m_Pedido){
    valorServicos += p.getValorGorjeta();
 }
 System.out.println("Total Valor Serviços: " + valorServicos);
}

/**
 *
 * @param g
 * @param horarioFinal
 * @param dataFim
```

```
        */
       public void finalizarTrabalhoGarcom(Garcom g, int horarioFinal, int dataFim){
        RegistroDeTrabalho r = g.getRegistroDeTrabalho();
        double gorjetas = 0;
        double totalServicos = 0;
        double valorTotal = 0;
          for(Pedido p : g.getPedidosFinalizados()){
             gorjetas += p.getValorGorjeta();
             totalServicos += p.getValorServico();
          }
        valorTotal = gorjetas + totalServicos;
        r.setDataFim(dataFim);
        r.setHoraFim(horarioFinal);
        r.setValorTotalGorjetas(gorjetas);
        r.setValorTotalVendido(totalServicos);
        r.setValorTotalArrecadado(valorTotal);
        r.setTotalPedidosAtendidos(g.getPedidosFinalizados().size());
        m_RegistroDeTrabalho.add(r);
        g.imprimirRelatorioTrabalho();
        g.getPedidosFinalizados().clear();
        g.setRegistroDeTrabalho(null);

       }

     public void finalizarPedido(Pedido pedido) {
        m_Pedido.add(pedido);
     }

}//end Restaurante
```