UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

REÚSO DE FRAMEWORKS TRANSVERSAIS COM APOIO DE MODELOS

THIAGO GOTTARDI

ORIENTADOR: PROF. DR. VALTER VIEIRA DE CAMARGO

São Carlos – SP

Julho/2012

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

REÚSO DE FRAMEWORKS TRANSVERSAIS COM APOIO DE MODELOS

THIAGO GOTTARDI

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Engenharia de Software
Orientador: Prof. Dr. Valter Vieira de Camargo

São Carlos – SP

Julho/2012

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

G685rf

Gottardi, Thiago.
Reúso de frameworks transversais com apoio de modelos / Thiago Gottardi. -- São Carlos : UFSCar, 2012.
152 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2012.

1. Engenharia de software. 2. Reuso. 3. Orientação a aspectos. 4. Metamodelo. 5. Transformação de modelos. I. Título.

CDD: 005.1 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Reúso de Frameworks Transversais com o Apoio de Modelos"

Thiago Gottardi

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

Membros da Banca:

Prof. Dr. Valter Vieira de Camargo
(Orientador - DC/UFSCar)

Prof. Dr. Paulo Cesar Masiero
(ICMC/USP)

Prof. Dr. Marco Túlio de Oliveira Valente
(DCC/UFMG)

São Carlos
Julho/2012

AGRADECIMENTOS

Agradeço a participação de Rafael Serapilha Durelli na elaboração da outra parte da ferramenta implementada, ao orientador Valter Vieira de Camargo, às agências de fomento:

- Bolsa FAPESP (Processo 2011/04064-8);
- Bolsa CNPq (Processo 132996/2010-3);
- Apoio financeiro a projeto CNPq (560241/2010-0);
- Projeto Universal CNPq (483106/2009-7).

Também agradeço ao departamento e a disponibilidade de suas instalações. Agradeço aos membros da banca: Marco Tulio de Oliveira Valente e Paulo Cesar Masiero que são especialistas em assuntos relevantes ao que foi tratado neste trabalho, com pontos de vista diferentes, isso permitiu que suas sugestões fossem mais proveitosas.

Agraceço aos ouvintes da defesa porque nem sempre é possível deixar seu trabalho de lado para ver o trabalho de outra pessoa. Agradeço muito aos professores que se disponibilizaram para escrever cartas de recomendação para a FAPESP. Agradeço à administradora da rede do deparamento que permitiu a execução do sistema de coleta de dados estatísticos. Também agradeço a qualquer outra pessoa ou entidade que foram importantes para a elaboração deste projeto de forma direta ou indireta, incluindo os leitores que lerem esta página.

Em especial, agradeço a Deus e aos meus pais, que me deram oportunidade de chegar a este momento.

Muito obrigado a todos.

Thiago Gottardi.

RESUMO

A programação orientada a aspectos foi criada para permitir a modularização de um tipo de interesse de software denominado de “interesse transversal”, que não pode ser completamente modularizado com paradigmas como o orientado a objetos. Com o uso do paradigma orientado a aspectos, vários pesquisadores começaram a pesquisar como determinados interesses transversais poderiam ser modularizados de formas genéricas para aumentar suas possibilidades de reúso, fazendo surgir Frameworks Orientados a Aspectos e também o termo Frameworks Transversais. Framework Transversal é um tipo de framework orientado a aspectos que tem o objetivo de permitir o reúso de código de interesses transversais, como persistência, distribuição, concorrência ou regras de negócio. Em geral, esses frameworks são disponibilizados na forma de código-fonte e devem ser reusados por meio da edição de código. Realizar o reúso neste nível faz com que engenheiros de aplicação tenham que se preocupar com detalhes da implementação do framework, afetando o entendimento, a produtividade e a qualidade do software final. Neste trabalho, o objetivo é elevar o nível de abstração do processo de reúso de frameworks transversais, propondo um processo dirigido por modelos que permite iniciar o processo de reúso nas fases antecedentes à implementação. Experimentos foram realizados para comparar o tempo de aplicar no novo processo com o processo de edição de código-fonte. Foi identificado que o novo processo possui vantagens em diminuir o tempo para reusar os frameworks, porém, não foram encontradas vantagens durante a manutenção de aplicações acopladas a frameworks transversais.

Palavras-chave: Reúso de Frameworks, Orientado a Aspectos, Modelo de Software, Composição de Modelos, Transformação de Modelos

ABSTRACT

Aspect-Oriented programming was created to modularize the so-called “crosscutting concerns”. Crosscutting concerns have some properties that cannot be fully modularized with the object-oriented paradigm. After that, aspect-oriented frameworks were created in order to make reuse of different concern codes easier. Among these frameworks, Crosscutting Frameworks are aspect-oriented frameworks specifically created for crosscutting concern code modularization, for instance, persistence, distribution, concurrency and business rules. Currently, these frameworks are usually distributed as source code and must be reused by extending classes, aspects and methods. Reusing these frameworks in code-level require application developers to worry about implementation issues, that affects understandability, productivity and quality of final software. In this thesis, the objective is to raise abstraction level by applying a new model-driven approach for crosscutting framework reuse, which also allows reusability during earlier development phases. Experiments were conducted in order to compare the productivity of the proposed process with the conventional reuse technique, which is based on source-code edition. It was identified that the proposed process has advantages during the reuse activity, however, no advantages were detected while maintaining an application coupled to a crosscutting framework.

Keywords: Framework Reuse, Aspect-Oriented, Software Model, Model Composition, Model Transformation

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	–	Application Programming Interfaces
DSOA	–	Desenvolvimento de Software Orientado a Aspectos
DSOO	–	Desenvolvimento de Software Orientado a Objetos
DSL	–	Domain Specific Language
EMF	–	Eclipse Modeling Framework
FOA	–	Framework Orientado a Aspectos
FOO	–	Framework Orientado a Objetos
FFT	–	Família de Framework Transversal
FSML	–	Framework Specification Modeling Language
FT	–	Framework Transversal
GMF	–	Graphical Modeling Framework
MDA	–	Model-Driven Architecture
MDD	–	Model-Driven Development
MOF	–	Metamodel Object Facility
MR	–	Modelo de Reúso
MRR	–	Modelo de Requisitos de Reúso
OA	–	Orientação a Aspectos
OBS	–	On Board Software
OMG	–	Object Management Group
OO	–	Orientação a Objetos
PDM	–	Platform Definition Model
PIM	–	Platform-Independent Model
POA	–	Programação Orientada a Aspectos
POO	–	Programação Orientada a Objetos
PSM	–	Platform-Specific Model
RDL	–	Reuse Definition Language
RR	–	Requisitos de Reúso
SGBD	–	Sistema Gerenciador de Banco de Dados
UML	–	Unified Modeling Language
UML-FI	–	Unified Modeling Language – Framework Instantiation
WPI	–	Workbench Part Interactions

SUMÁRIO

Capítulo 1

INTRODUÇÃO

1.1 Contexto

A Programação Orientada a Aspectos (POA) foi criada com o objetivo de melhorar a modularização de software por meio do fornecimento de construções específicas para separar os interesses “base” dos “transversais”. Na terminologia da POA, “interesses base” são aqueles que dizem respeito ao objetivo principal do software e os “transversais” são aqueles requisitos que não são modularizáveis quando implementados com a orientação a objetos (KICZALES et al., 1997).

Depois do surgimento da POA, vários autores começaram a investigar como as novas abstrações propostas poderiam ser usadas para melhorar o reúso de interesses transversais. Assim, muitos autores propuseram implementações abstratas de determinados interesses transversais com o objetivo de facilitar seu reúso em diferentes contextos (CONSTANTINIDES; ELRAD, 2000). Vários termos surgiram para designar esse tipo de implementação, como Frameworks Orientados a Aspectos (SHAH; HILL, 2004), Framework de Aspectos, Frameworks baseados em aspectos, aspectos reusáveis (MORTENSEN; GHOSH, 2006a, 2006b; BYNENS et al., 2010; SAKENOU et al., 2006; CUNHA; SOBRAL; MONTEIRO, 2006; SOUDARAJAN; KHATCHADOURIAN, 2009), biblioteca de aspectos, e outros nomes (SOARES; LAUREANO; BORBA, 2006; KULESZA et al., 2006; CAMARGO; MASIERO, 2005; HUANG; WANG; ZHANG, 2004; ZANON; CAMARGO; PENTEADO, 2010). Com o objetivo de fornecer uma terminologia mais adequada nesse contexto, Camargo e Masiero (2005) propuseram o termo Framework Transversal (FT) para representar frameworks orientados a aspectos que encapsulam apenas um determinado interesse transversal, como persistência, concorrência ou segurança. Em geral, como a maior parte dos FTs encontrados na literatura são caixa-branca, seu processo de reúso é totalmente condicionado à edição manual de código-fonte com o auxílio de documentações textuais (“Cookbooks”) não padronizadas.

Outro conceito importante e pertinente a este trabalho é “Família de Frameworks

Transversais" (FFT). Uma FFT é um conjunto de características (em inglês, *features*) que podem ser compostas para a criação de um determinado FT. Esse conjunto de características provê várias funções que podem ser selecionadas durante o reúso de forma a melhor se adequar às necessidades da aplicação final, evitando assim, carregar código não utilizado (CAMARGO; MASIERO, 2008).

Uma linha de pesquisa que pode auxiliar no processo de reúso de frameworks é a técnica de Desenvolvimento Dirigido a Modelos (Model Driven Development - MDD). Ela permite aumentar o nível de abstração ao desenvolver software, pois, modelos de software são utilizados para gerar outros modelos ou o próprio software final (SCHMIDT, 2006; PASTOR; MOLINA, 2007).

1.2 Materiais e Métodos

Este projeto de mestrado é parte de um projeto maior, apoiado pelo CNPq, denominado "Infraestrutura de Apoio ao Reúso e Gerenciamento de Famílias de Frameworks Transversais". O objetivo desse projeto é a criação de um ambiente em que o processo de reúso de famílias de FTs seja totalmente apoiado por modelos. Assim, o engenheiro de aplicação pode montar um membro da família de FTs e realizar sua instanciação e composição com as aplicações usando os modelos, sem precisar escrever código-fonte. Também é objetivo neste projeto o desenvolvimento de ferramentas que forneçam apoio ao gerenciamento e manutenção das aplicações desenvolvidas com essa infraestrutura.

Neste projeto também é avaliado se o uso de modelos para documentar e apoiar o processo de reúso é benéfico no contexto de reúso de frameworks transversais.

1.3 Motivação

Em geral, como a maior parte dos FTs encontrados são caixa-branca, seu processo de reúso consiste em realizar sua instanciação e a composição em nível de código-fonte na mesma linguagem de programação em que o framework foi desenvolvido (MORTENSEN; GHOSH, 2006a, 2006b; SHAH; HILL, 2004; SOARES; LAUREANO; BORBA, 2006; KULESZA et al., 2006; CAMARGO; MASIERO, 2005; HUANG; WANG; ZHANG, 2004; ZANON; CAMARGO; PENTEADO, 2010; LAZANHA et al., 2010). Isso faz com que engenheiros de aplicação tenham que se preocupar com detalhes de baixo nível que afetam a concentração no nível de abstração adequado ao processo de reúso, gerando os seguintes problemas:

- O engenheiro de aplicação precisa conhecer detalhes da linguagem de programação orientada a aspectos usada no desenvolvimento do framework, aumentando a curva de aprendizado para reusar o FT;
- Trabalhar em baixo nível pode acarretar erros de implementação;
- Muitas linhas de código precisam ser escritas para informar poucos detalhes durante o processo de reúso; afetando a produtividade da equipe de desenvolvimento;
- O reúso só pode ser feito quando o desenvolvimento atinge a etapa de implementação. Muitas vezes é interessante que o processo de reúso seja iniciado em etapas iniciais do processo de desenvolvimento, por exemplo, na análise e no projeto.

Além disso, não há na literatura um estudo que analise quantitativamente as vantagens e desvantagens de um processo de reúso de FTs baseado em modelos comparado com um processo de reúso tradicional baseado em código-fonte.

1.4 Objetivos

O objetivo geral do trabalho descrito nesta dissertação de pesquisa é investigar as vantagens e as desvantagens de se elevar o nível de abstração do processo de reúso de FTs de forma que ele seja feito inteiramente com o apoio de modelos. Como objetivos específicos têm-se:

- Criar um modelo para representar “Cookbooks”, que é uma documentação para reúso de frameworks (JOHNSON, 1992);
- Criar um modelo para realizar o processo de reúso propriamente dito em nível de projeto;
- Desenvolver uma ferramenta para reúso de FTs com base no segundo modelo proposto;
- Avaliar a ferramenta comparando tempo necessário para reusar um framework com a ferramenta e com o processo convencional de edição de código-fonte;
- Avaliar a ferramenta comparando tempo necessário para realizar um processo de manutenção de uma aplicação acoplada a um framework com a ferramenta e com o processo convencional de edição de código-fonte.

1.5 Estrutura

No Capítulo 2 são apresentadas referências importantes para a elaboração e compreensão do projeto proposto, incluindo definições sobre o Paradigma Orientado a Aspectos, Abordagens de Modelagem e Desenvolvimento Dirigido a Modelos. No Capítulo 4 é apresentada a abordagem elaborada e avaliada dentro deste projeto de pesquisa. No Capítulo 5 é exemplificado um uso da abordagem com um framework transversal real. No Capítulo 6 são apresentados dois experimentos que avaliam a abordagem proposta. No Capítulo 3 são apresentadas outras abordagens que envolvem o uso de ferramentas para o reúso de frameworks. Por fim, no Capítulo 7 estão as conclusões.

Capítulo 2

FUNDAMENTAÇÃO TEÓRICA

2.1 Considerações Iniciais

Neste capítulo são apresentados trabalhos que dão embasamento teórico à proposta apresentada neste documento.

Na Seção 2.2 é descrita a metodologia de desenvolvimento dirigido por modelos, que é empregada na abordagem proposta neste trabalho. Na Seção 2.3 é apresentado o paradigma no qual os FTs foram desenvolvidos. Na Seção 2.4 são apresentadas abordagens de modelagem de softwares orientados a objetos que foram consideradas para criar modelos para a abordagem deste trabalho. Na Seção 2.5 é apresentada uma metodologia para derivar softwares para usos específicos evitando a transferência de códigos não utilizados, o que poderia ocorrer ao reusar um framework com mais características do que necessário para uma aplicação. Essa metodologia é tratada nesta abordagem e também é empregada ao criar a família de frameworks transversais da Seção 2.6 em que é apresentado um tipo de framework orientado a aspectos que é considerado na abordagem de reúso proposta neste trabalho.

2.2 Desenvolvimento Dirigido a Modelos

O Desenvolvimento Dirigido a Modelos (Model Driven Development - MDD) é uma abordagem de desenvolvimento de software apoiada em modelos. No MDD, modelos não são apenas artefatos que servem para ilustrar ou comunicar ideias; eles são efetivamente utilizados como entrada para geradores de código que produzem aplicações completas. MDD é a combinação de programação generativa, linguagens específicas de domínio e transformações de software. Seu objetivo é reduzir a distância semântica entre o problema e a implementação/solução, por meio de modelos de alto nível que protegem os desenvolvedores das complexidades da plataforma de implementação

(FRANCE; RUMPE, 2007).

No MDD, modelos são usados para expressar conceitos do domínio de forma mais efetiva, enquanto transformações geram automaticamente os artefatos que refletem a solução expressa nos modelos (SCHMIDT, 2006; PASTOR; MOLINA, 2007).

Para permitir essa metodologia, são criadas ferramentas capazes de transformar modelos em outros artefatos. Essa característica permite que alguns sistemas sejam construídos inteiramente por meio da especificação de modelos, não exigindo que desenvolvedores necessitem trabalhar em baixo nível de abstração, ou seja, no código-fonte. Nessa transformação podem ser gerados modelos intermediários antes de atingir o nível de implementação, de modo a reduzir o nível de abstração gradativamente.

Por exemplo, Model-Driven Architecture (MDA) (OMG, 2010a) é um padrão arquitetural para desenvolvimento dirigido a modelos com três tipos de modelagem: Platform-Independent Model (PIM), Platform-Specific Model (PSM) e Platform-Definition Model (PDM). O PIM é o modelo livre de plataforma e de maior nível de abstração enquanto que o PSM é específico de plataforma. O último, o PDM, tem o objetivo de mapear os elementos do PSM com relação à uma plataforma.

Conforme representados na Figura 1, a composição dos modelos PSM e PDM pode ser transformada em código-fonte com a finalidade de gerar o software final. O benefício dessas divisões é a reusabilidade, assim, para gerar sistemas em um domínio comum ou transformar um mesmo sistema para uma plataforma diferente, basta substituir um ou mais modelos.

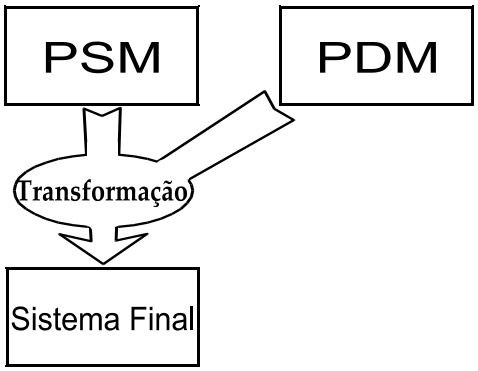

Figura 1: Transformação de modelos
Fonte: (OMG, 2010a)

Os modelos independentes de plataforma (PIM) podem ser utilizados em vários níveis de abstração. Cada um desses níveis pode ser transformado entre si. Portanto, um processo de desenvolvimento dirigido a modelos pode ter “N” níveis de abstração, utilizados conforme as etapas de desenvolvimento. Na Figura 2 está ilustrado

um processo com “N” níveis de abstração de modelos independentes de plataforma (PIM), que são processados sequencialmente. Cada novo modelo PIM com um nível inferior de abstração abrange maiores detalhes do processo de desenvolvimento. Essas transformações sucessivas são realizadas até atingir o nível zero, que é o modelo específico de plataforma (PSM), que por sua vez, pode ser utilizado para geração do código fonte, como ilustrado na Figura 1.

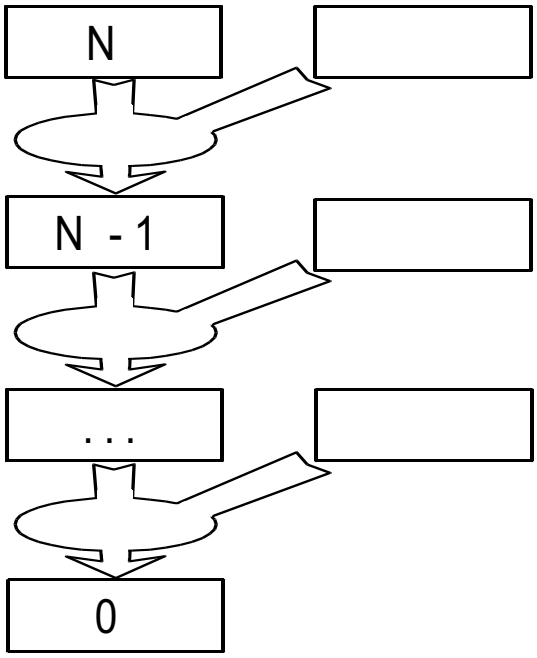

Figura 2: Processo com vários níveis de Abstração
Fonte: (OMG, 2010a)

O Desenvolvimento Dirigido a Modelos pode ser adaptado para várias finalidades. No caso da Figura 2, os modelos têm base no mesmo metamodelo, porém, em alguns casos pode ser interessante uma tradução entre modelos de linguagens diferentes. A definição formal de uma linguagem de modelagem também pode ser descrita por modelos, ou seja, metamodelos.

Na Figura 3 é ilustrado um processo de tradução do “Modelo 1”, que utiliza a linguagem do “Metamodelo A”, para o “Modelo 2”, que utiliza a linguagem do “Metamodelo B”. Para realizar a tradução, há um modelo de transformação que provê mapeamento entre elementos de modelo de cada linguagem. Esse processo permite uma tradução automatizada e pode ser empregado para traduzir modelos entre linguagens de mesmo domínio.

Outra possibilidade, dentre várias outras, é a união de modelos. Nesse processo, dois ou mais modelos são combinados no processo de geração de um único modelo final. Esse processo é ilustrado na Figura 4, em que o “Modelo 1” e o “Modelo 2” são utilizados na geração do “Modelo 3”. Dessa forma, os dois primeiros modelos

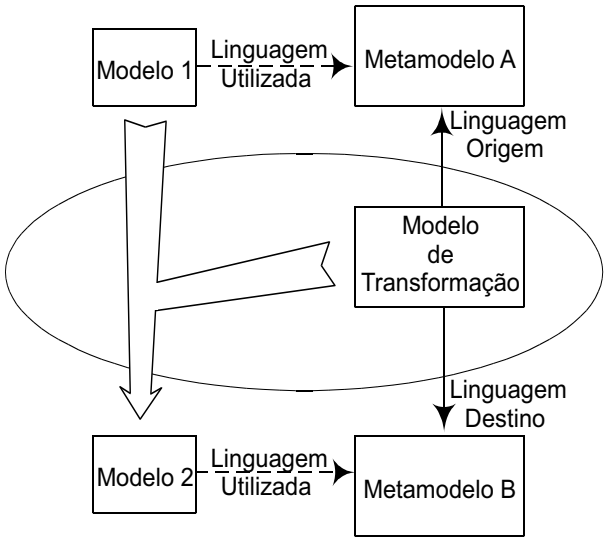

Figura 3: Processo de Tradução de Modelos
Fonte: (OMG, 2010a)

podem representar partes modularizadas de um problema, que será combinado no último modelo.

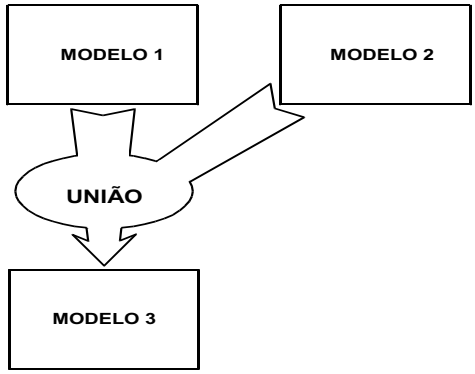

Figura 4: Processo que envolve União de Modelos
Fonte: (OMG, 2010a)

A união de modelos é importante para a abordagem proposta deste trabalho, em que são utilizados três tipos de modelos. Dois desses modelos são combinados para geração do terceiro, que é específico de plataforma. O terceiro modelo, por sua vez, é utilizado para geração do código final.

2.3 Orientação a Aspectos

A Programação Orientada a Aspectos (POA) (KICZALES et al., 1997) é usada desde o final da década de 90 para modularizar de forma mais adequada os “interesses transversais” de um sistema, que até então ficavam misturados e espalhados pelo código

dos interesses-base sem que fosse possível organizá-los em módulos independentes.

O entrelaçamento e espalhamento de código acabavam culminando em problemas de modularização e consequentemente, manutenção e reúso. Na terminologia da POA, os “interesses-base” referem-se à funcionalidade principal do sistema e os “transversais” referem-se a restrições globais e a requisitos não funcionais, como por exemplo, persistência, distribuição, autenticação, controle de acesso, criptografia e concorrência (KICZALES et al., 1997).

Com as novas abstrações fornecidas pela POA é possível implementar separadamente os interesses-base e os interesses transversais, o que até então era difícil somente com a programação orientada a objetos. Na Figura 5, há uma ilustração do processo de combinação aplicado durante a construção de um software orientado a aspectos. A caixa “Base” representa a parte base do software, que são os comportamentos principais a serem tratados pela aplicação. A caixa “Transversal” representa comportamentos que dependem e afetam a parte base. O triângulo “Combinação”, representa o processo também conhecido como Weaving, que irá entrelaçar os interesses apenas no momento da geração do sistema final. Dessa forma, no código-fonte, os interesses permanecem separados.

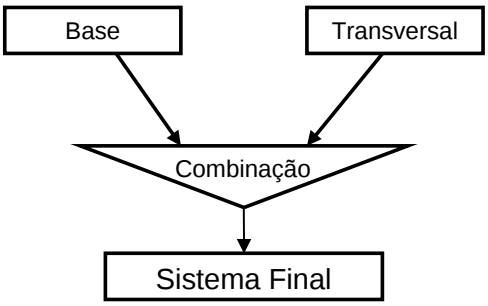

Figura 5: Construção de um Sistema OA
Fonte: AspectJ Team (2003).

Abaixo, encontram-se conceitos do paradigma pertinentes a este trabalho, conforme descrito na ontologia sobre DSOA da Comunidade Europeia (BERG; CONEJERO; CHITCHYAN, 2005):

- Interesse (*Concern*): algo no processo de desenvolvimento que é de interesse a ser feito, definido como um requisito de software qualquer.
- Interesse Transversal (*Crosscutting-Concern*): uma responsabilidade que tem ou terá implementação espalhada entre vários pontos do código-fonte, visto que não pode ser representada separadamente das outras. Isto ocasiona entrelaçamento de código.

- Separação de Interesses (*Separation of Concerns*): Separação de interesses é um estudo para isolar cada interesse, graças a uma densa análise que tem como objetivo simplificar o software;
- Aspecto: uma unidade modular que implementa um interesse transversal sem causar entrelaçamento de código;
- Composição (*Composition*): composição de software é um conceito presente em vários paradigmas, que significa unir partes de softwares que foram criadas separadamente. No caso de uso OA, representa a união entre aspectos com os pontos de junção presentes nas classes base.
- Combinação (*Weaving*): um caso especifico de composição de software para gerar o sistema final a partir de um código fonte orientado a aspectos. Neste processo, os interesses base e transversais são novamente reunidos no software final;
- Ponto de Junção (*Join Point*): ponto de junção é um ponto de interesse em algum artefato no ciclo de vida do software em que interesses transversais podem ser aplicados;
- Adendo (*Advice*): Um adendo é um comportamento a ser executado em um ou mais pontos de junção. Todo adendo pertence a um aspecto, que o define em conjunto com uma expressão de conjunto de junção, ou seja, os pontos de junção nos quais ele deve ser executado.
- Conjunto de Junção (*PointCut*): Conjunto de junção é uma expressão lógica que seleciona um conjunto de pontos de junção.

Outra abstração importante, não presente na ontologia de DSOA, mas comum a várias linguagens do paradigma, é possibilitar declarações intertipo, que permitem realizar declarações que afetem o código base via aspectos, por exemplo, inserir atributos, métodos e heranças de classes, mantendo-as inconscientes, ou seja, sem dependência alguma com relação ao que foi inserido.

Linguagens de programação orientadas a aspectos estão disponíveis, como AspectJ (KICZALES et al., 2001; ASPECTJ TEAM, 2003; KISELEV, 2002; LADDAD, 2003), AspectC++ (SPINCZYK; GAL; SCHRöDER-PREIKSCHAT, 2002), HyperJ (TARR; OSSHER; SUTTON, 2002), CaesarJ (ARACIC et al., 2006), AspectH (ANDRADE; SANTOS; BORBA, 2004), Ptolemy (RAJAN; LEAVENS, 2008), AspeCtC (GONG; JACOBSEN., 2008) e ComposeJ (WICHMAN, 1999).

2.3.1 POA com Linguagem AspectJ

AspectJ (KICZALES et al., 2001) é uma extensão orientada a aspectos para a linguagem Java. Assim, códigos-fonte e classes compiladas Java são compatíveis também com AspectJ. Além disso, esta linguagem é a mais difundida, no momento deste trabalho, para implementar softwares orientados a aspectos. Por ser condizente com os conceitos da POA (Programação Orientada a Aspectos), com AspectJ é possível separar interesses transversais do base, além de possuir vários conceitos como pontos de junção, conjuntos de junção, adendos, declarações inter-tipos e aspectos. Para exemplificar o uso de AspectJ, na Figura 6, há um exemplo comum no ensino da linguagem, embasado no padrão "Observer" (GAMMA et al., 1994), em que um observador, ou um monitor de saída (Display), representa figuras que devem ser atualizadas na imagem após a ocorrência de qualquer alteração.

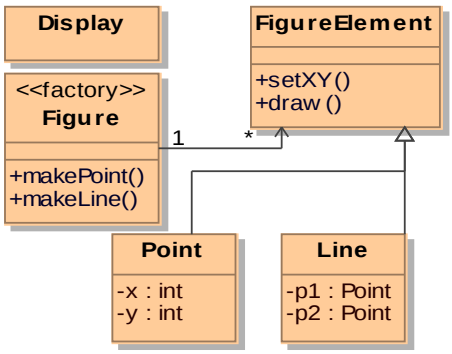

Figura 6: Diagrama de um editor de Figuras
Fonte: AspectJ Team (2003).

Desse modo, a atualização do Display no instante que figuras são alteradas ou criadas é um interesse transversal. Para capturar essas ocorrências, Display deve selecionar pontos de junção que representem a criação e alteração de figuras.

Pontos de Junção são pontos dentro da execução de um código, como por exemplo: criação de objetos, invocação de métodos, acesso a campos e lançamento de exceções. Um conjunto de junção é uma expressão lógica que é capaz de selecionar um conjunto de pontos de junção.

A sintaxe dos conjuntos de junção se dá pela palavra reservada de sua categoria seguida pelo elemento a ser considerado entre parênteses, como por exemplo, chamadas à "setXY()" pode ser descrito pela expressão "call(public void FigureElement.setXY(int,int)), ou seja, como "call" captura chamada de métodos, a assinatura do método a ser capturado deve ser inserida entre parênteses. Conjuntos de junção mais utilizados em AspectJ:

- Conjuntos de Junção por tipos usados:

  - Argumentos: “args” para selecionar uso de parâmetros com tais tipos;
  - Origem deste Objeto: “this” para selecionar objetos que iniciaram uma mensagem interobjeto com tais tipos;
  - Objeto Alvo: “target” para selecionar um objeto que recebeu uma mensagem interobjeto com tais tipos.

- Conjuntos de Junção por métodos usados:
  - Chamada: “call” para selecionar a ocorrência de uma chamada ao método descrito (o contexto está no objeto que chamou o método especificado);
  - Execução: “execution” para selecionar a ocorrência de uma execução do método descrito (o contexto está sempre no objeto que executa o método especificado);

- Conjuntos de Junção para Atributos:
  - Leitura de valor: “get” para selecionar ocorrência de leitura do atributo especificado;
  - Alteração de valor: “set” para selecionar a ocorrência de alteração do atributo especificado.

Além da lista acima, é possível criar composições de conjuntos de junção, utilizando operadores “&&” para conjunção, “||” para disjunção e “!” para negação, seguindo a estrutura de expressões lógicas em linguagens como Java.

Além disso, conjuntos de junção podem ser declarados com um nome para facilitar seu uso, como exemplo na Figura 7. O conjunto de junção do exemplo é formado pela composição de disjunção entre conjuntos de junção de chamadas.

```
pointcut SetXorY() :
        call(void Point.setX(int)) ||
        call(void Point.setY(int));
```

Figura 7: Declaração de um Conjunto de Junção
Fonte: AspectJ Team (2003).

Um adendo é um trecho de código que pode ser declarado para ser executado quando a execução de pontos de junção ocorre. Em AspectJ, um adendo é classificado quanto ao momento que é executado em comparação à ocorrência do ponto de junção. São, portanto, três tipos básicos:

- Before (Antes): Adendo é executado logo antes da ocorrência da execução do ponto de junção;
- Around (Durante): Adendo é executado no lugar da ocorrência da execução do ponto de junção;
- After (Após): Adendo é executado logo após a ocorrência da execução do ponto de junção.

O adendo Before é o mais simples entre os três; quando usado, este adendo é executado logo antes das linhas de código que o conjunto de junção capturou.

O After executa após o código capturado. O After em sua forma pura é executado independente do que ocorreu com o código capturado, também é possível utilizar After Throwing (após lançamento de exceção) e After Returning (após retorno normal) que executam apenas se o código capturado executou de forma normal ou teve uma exceção.

Já o Around, o mais complexo, envolve toda a execução do ponto de junção: inicia antes da ocorrência e termina depois. Deste modo o Around pode possuir a chamada especial “proceed()” que especifica a execução do conjunto de linhas de código capturadas. Se a linha de execução do adendo não passar por um “proceed(), o trecho capturado não é executado. Isso permite que o fluxo de execução seja completamente desviado e a execução capturada pelo conjunto de junção pode ser evitada.

Um exemplo de declaração de adendos em AspectJ está na Figura 8, em que dois adendos, um antes e um após, são executados a partir do conjunto de junção “SetXorY()”, exibindo mensagens correspondentes as suas execuções.

```
before SetXorY() {
        System.out.println("Antes da ocorrência");
}
after SetXorY() {
        System.out.println("Após ocorrência");
}
```

Figura 8: Declaração de adendos em AspectJ
Fonte: AspectJ Team (2003).

Um aspecto de AspectJ é uma especialização de classes do Java. Dessa forma, um aspecto possui as mesmas propriedades que uma classe com a adição de outras propriedades. Um aspecto pode possuir conjuntos de junção, adendos e declarações

intertipo. Com esses conceitos, aspectos podem modificar a estrutura de um programa de duas formas: estática e dinâmica.

A modificação estática (também chamada de entrecorte estático) é feita via declarações intertipo. Declarações intertipo permitem adicionar atributos, métodos e definições de herança em classes e interfaces a partir de um aspecto externo. A modificação dinâmica é feita via adendos que executam na ocorrência de pontos de junção, capturados por conjuntos de junção. A única diferença de declaração de um aspecto para uma classe é a substituição de “class” por “aspect”.

Aspectos também podem possuir adendos e instanciação diferente, o que não é necessário de ser introduzido por este texto. Um aspecto concreto não é instanciado por programação imperativa, e sim automaticamente pela ocorrência de uma regra de sua instanciação. Deste modo é comum exigir que aspectos concretos possuam construtores sem nenhum parâmetro.

A linguagem AspectJ foi muitas vezes utilizada como base para propostas de modelagem de sistemas orientados a aspectos (EVERMANN, 2007; HAN; KNIESEL; CREMERS, 2005), que estão descritas com maiores detalhes na Seção 2.4.

2.4 Abordagens de Modelagem

Para melhor compreensão deste trabalho, é importante o conhecimento de linguagens de modelagem e a UML é constantemente referenciada, pois, a definição do metamodelo da UML foi considerada durante a elaboração dos modelos propostos neste trabalho.

UML (Unified Modeling Language) é uma linguagem de modelagem para software orientado a objetos, que dá apoio a todas as fases de desenvolvimento presentes em processos de desenvolvimento de software orientado a objeto, como o RUP (Rational Unified Process), provendo diagramas que documentam e especificam informações pertinentes para facilitar o desenvolvimento destes sistemas (BOOCH; RUMBAUGH; JACOBSON, 2006).

O modelo que define a semântica da UML é chamado de metamodelo, incluindo todos os tipos de elementos possíveis a modelos e suas relações. Tais definições de elementos são feitas em metaclasses, que são classes presentes em nível de metamodelo. As definições formais e o metamodelo da UML são especificados na linguagem chamada de MOF (Meta Object Facility) e estão disponíveis no site da OMG (Object Management Group) (OMG, 2010c). Na Figura 9 é ilustrada parte do metamodelo da UML 2, com as metaclasses “Class”, “Property” e “Operation”, que definem em nível de

modelo, propriedades de *Class* (Classe), *Property* (Atributo) e *Operation* (Operação).

É visível na figura as associações e atributos das metaclasses. A metaclasse "Class", por exemplo, é associada com a metaclasse "Property" por meio de um relacionamento de agregação. Essa associação indica que uma classe pode possuir vários atributos sendo que um atributo pode pertencer a no máximo uma (visível pelo modificador de multiplicidade "0..1") classe. Entre a metaclasse "Class" e a metaclasse "Operation" há outro relacionamento. Esse relacionamento indica que uma classe pode possuir várias operações sendo que uma operação pode pertencer a no máximo uma classe. Outra associação ocorre no metamodelo, dessa vez entre a metaclasse "Operation" e a metaclasse "Constraint". Esse relacionamento indica que uma operação pode possuir várias restrições sendo que uma restrição pode pertencer a no máximo uma operação.

Também são importantes as generalizações presentes no metamodelo. Na Figura 9, algumas generalizações são visíveis (representadas pela seta com centro transparente) entre as metaclasses. São visíveis: a metaclasse "Class" herda da metaclasse "Classifier", a metaclasse "Property" herda da metaclasse "StructuralFeature" e a metaclasse "Operation" herda de "BehavioralFeature". As metaclasses "Class", "Property" e "Operation" também possuem atributos e papéis nos relacionamentos. Em nível de modelo essas características são representadas de maneiras diferentes. Por exemplo, a metaclasse "Class" em nível de modelo se torna uma classe que pode conter atributos e métodos.

Entre chaves estão restrições, que são códigos anexados para aumentar a semântica do modelo. Essas regras não são possíveis de definir somente com uso de modelagem.

Em alguns casos, estas metaclasses do metamodelo UML são limitadas para representar alguns conceitos de um domínio específico, linguagem, tecnologia, ou até outro paradigma. Para sanar este problema, a UML pode ser estendida com perfis. Caso isso não seja suficiente, também é possível criar novos metamodelos, similares ou não ao da UML.

2.4.1 Metamodelagem por Perfis

A definição de perfis de modelagem permite adicionar conceitos a uma linguagem de modelagem sem alterar o metamodelo. Por exemplo, a UML permite criação de perfis para adicionar, se necessário, restrições e conceitos via estereótipos, etiquetas valoradas de estereótipos e restrições em código. O uso de estereótipos é comum e a própria UML já trás alguns estereótipos como "create" e "destroy", utilizados para

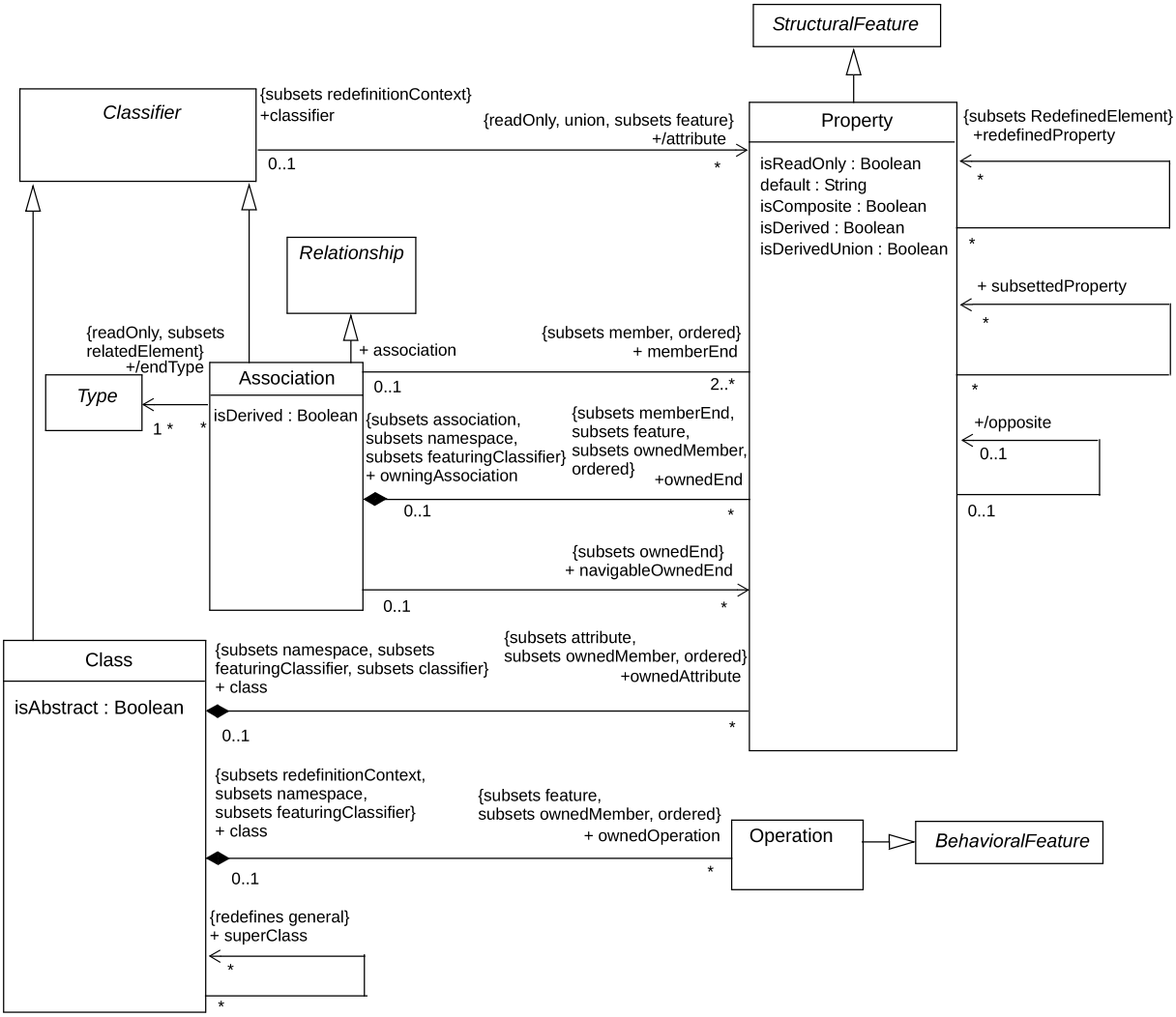

Figura 9: Parte do Metamodelo UML – MOF (Meta Object Facility)
Fonte: OMG (2010c)

denotar operações que definem construtores e destrutores, respectivamente.

Em nível de modelo, estereótipos são aplicados a instâncias das metaclasses. Por exemplo, classes podem receber estereótipos da metaclasse “Class” ou de qualquer uma de suas generalizações, como “Classifier” e “Element”.

O elemento que recebe estereótipos apresenta no diagrama em nível de modelo o nome do estereótipo entre os símbolos de “maior maior” (“≪”) e “menor menor” (“≫”) (como por exemplo, ≪create≫).

Estereótipos podem possuir etiquetas valoradas e restrições. Restrições são expressões lógicas que devem ser mantidas verdadeiras para que a semântica do perfil se mantenha correta (BOOCH; RUMBAUGH; JACOBSON, 2006) e são codificados em OCL (Object Constraint Language) (OMG, 2010b) para aumentar a consistência semântica do perfil, evitando uso errôneo que não foi possível de evitar usando-se apenas de definição gráfica.

Na Figura 10 é mostrado um exemplo de um perfil UML, em que na parte esquerda da figura, há o estereótipo “Aspect”, que estende a metaclasse “Class” da UML (extensão é representada pela seta de centro preto). O estereótipo apresenta três etiquetas no diagrama: “isPrivileged”, “declaredParents” e “declaredImplements” e uma associação “precedes”. O atributo “isPrivileged” é booleano e define se o aspecto é privilegiado ou não. Já o atributo “declaredParents” permite a declaração de generalizações pelo aspecto. O atributo “declaredImplements” permite que o aspecto implemente uma classe em uma interface. A associação “precedes” equivale a um atributo do tipo “Aspect” de multiplicidade zero a um e define qual é o aspecto que precede o próprio. Na parte direita da Figura é mostrado a aplicação do estereótipo “Aspect”. Os atributos criados no metamodelo se tornam etiquetas valoradas no nível de modelo. No exemplo, “isPrivileged” recebe o valor “false” indicando que o aspecto não é privilegiado. O atributo “declareImplements” recebe o valor “RealizaçãoExemplo” indicando que o aspecto implementa essa classe em uma interface. A definição de todas as etiquetas valoradas não é obrigatória.

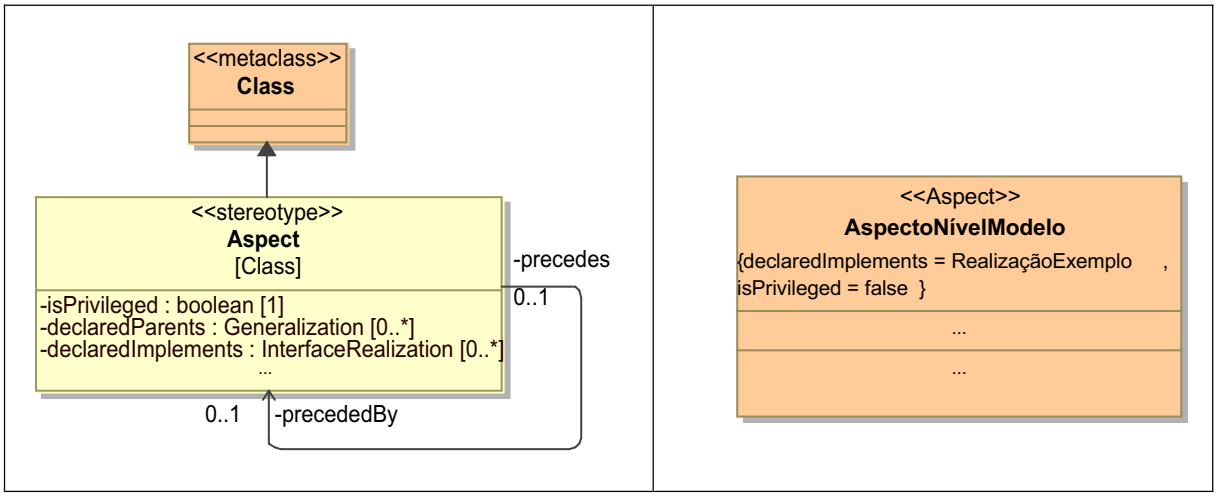

Figura 10: Exemplo de um Perfil e sua aplicação
Fonte: Evermann (2007)

Os perfis permitem extensões do metamodelo, tratando uma determinada necessidade sem alterar o metamodelo UML. O problema em alterar ou criar um metamodelo é a dificuldade de utilizá-lo em ferramentas pré-existentes. Já perfis UML são mais fáceis de serem concebidos e em muitas ocasiões substituem a necessidade de manipulação complexa do metamodelo. Além disso, várias aplicações de modelagem permitem sua importação e algumas até sua criação e edição. Com o perfil integrado a uma ferramenta de modelagem é possível utilizá-lo e aplicá-lo a modelos de software com grande facilidade.

Entretanto, a criação de perfis também possui dificuldades, por exemplo (EVERMANN, 2007):

1. Perfis são obrigados a seguir o metamodelo UML, não é possível alterar metaclasses e propriedades das metaclasses;

2. Assim como criar metamodelos, perfis podem ter erros que podem causar inconsistências nos modelos que os utilizem;

3. Perfis devem ser portáveis entre ferramentas que suportem a mesma versão da UML;

4. Estereótipos são presos a instâncias de metaclasses, sendo assim dependentes de suas representações gráficas, e não podendo alterá-las;

5. Vários perfis podem ser aplicados a um mesmo modelo, portanto, algumas combinações podem ser contraditórias.

2.4.2 Criação e Edição de Metamodelos

Como descrito no início desta seção, a UML é definida por um metamodelo. Qualquer extensão definida pela alteração ou criação de um novo metamodelo envolve a definição de uma nova linguagem de modelagem.

A criação de uma nova notação de modelagem editando metamodelos não depende de metaclasses da UML como ocorre com criação de perfis. Desse modo, será possível definir uma linguagem de modelagem com elementos de modelo com maior nível semântico e com menor risco de gerar inconsistências (HAN; KNIESEL; CREMERS, 2005).

Para utilizar uma abordagem de edição de metamodelos, é necessária a criação de uma ferramenta de modelagem específica, que pode não ser compatível com modelos UML ou outro metamodelo. Por exemplo, o Graphical Modeling Framework fornece um conjunto de ferramentas para facilitar a criação de ferramentas com especificação pesada (ECLIPSE CONSORTIUM, 2011).

Na Figura 11 há um exemplo de extensão pesada à UML. A proposta de Han et at. (2005) foi criada com o intuito de permitir modelagem de projeto de software orientado a aspectos com base na linguagem de programação AspectJ.

À esquerda da Figura 11 está um subconjunto do metamodelo com o intuito de representar aspectos (*Aspect*). Esse metamodelo é uma extensão da UML em conjunto com um metamodelo específico para Java, como uma forma de manter melhor consistência com a linguagem AspectJ. Dessa forma “Aspect” estende a classe do metamodelo Java e possui atributos “isPrivileged” e “perClause”, se relaciona com “Generalization” para permitir declaração intertipo de herança. “PointCut” se relaciona

com *Class*, pois no AspectJ, classes também podem ter conjuntos de junção. Aspect se relaciona com “Advice” para representar seus adendos, com “Feature” para representar elementos inseridos por declaração intertipo.

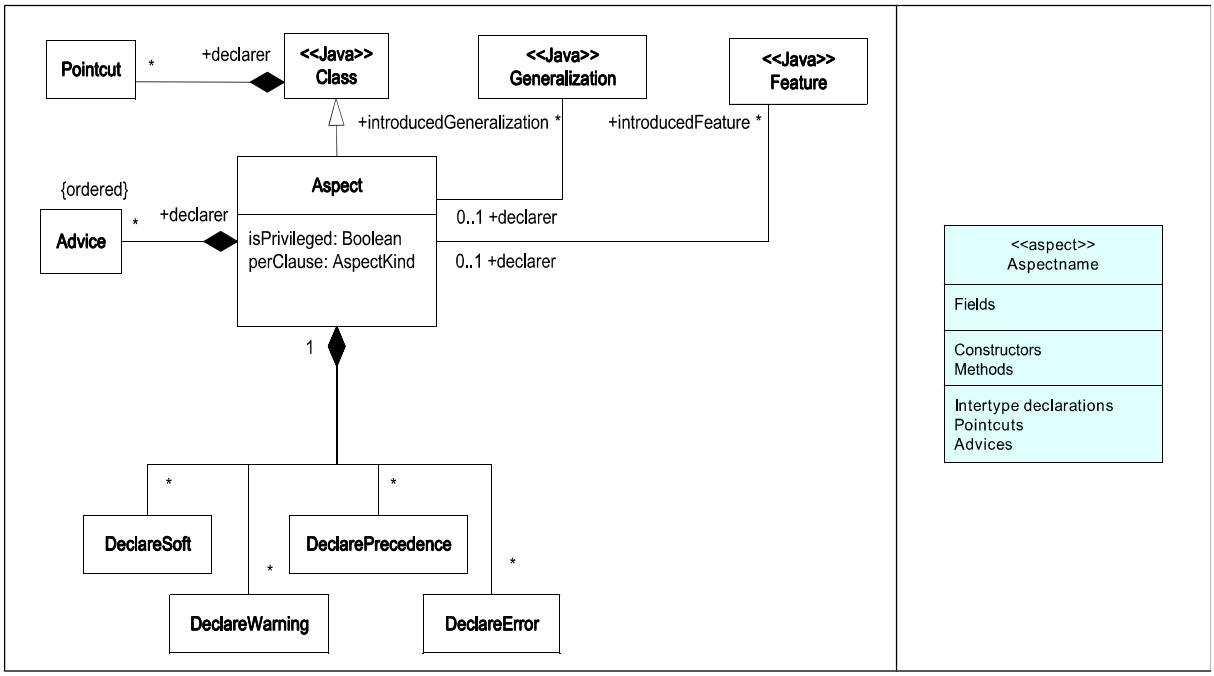

Figura 11: Metamodelo para AspectJ e um modelo com um aspecto
Han, Kniesel e Cremers (2005)

Na direita da Figura 11 há um exemplo gráfico de um aspecto em nível de modelo. Neste exemplo é possível identificar maior liberdade ao definir notações de modelagem, quando comparadas às extensões leves: novos elementos de modelos não precisam seguir a forma gráfica dos estendidos. Assim, o aspecto de Han, Kniesel e Cremers (2005) pode possuir mais divisões na sua representação gráfica do que uma classe da UML.

2.5 Linha de Produtos de Software

O conceito de Linhas de Produtos de Software foi criado para permitir derivações de um único software para ser otimizado para uma aplicação específica (CLEMENTS; NORTHROP, 2002).

Dessa forma, o software a ser derivado contém um conjunto de características. As características de um software são estruturadas em um modelo de características, que tem uma estrutura de árvore. Cada nó da arvore que é disposto como filho de um nó pai indica uma relação de dependência entre as características que, neste caso, indica que o nó filho só pode estar presente no software final se o nó pai estiver presente.

Estas características podem ser distribuídas em:

- Obrigatórias;
- Opcionais;
- Alternativas.

Características obrigatórias são aquelas que devem estar presente no software final, exceto no caso de um ancestral opcional ou alternativo não estar selecionado. Características opcionais são aquelas que são selecionáveis e não necessariamente precisam estar presentes no software final, Alternativas são características mutualmente exclusivas entre suas irmãs de um mesmo nó imediatamente ancestral, portanto, apenas um nó do ramo pode ser selecionado. Por exemplo, na Figura 12 há um exemplo de um modelo de características modelado na ferramenta FeatureIDE (KASTNER et al., 2009).

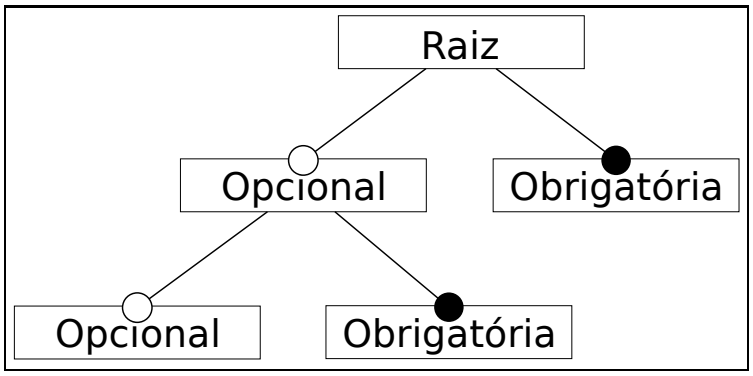

Figura 12: Exemplo Hipotético de um Modelo de Características

A árvore da Figura 12 possui cinco nós. No nível mais alto há uma raiz que agrupa todas as possíveis configurações. No nó intermediário, são exemplificadas uma característica opcional e uma obrigatória.

No nível inferior há outros exemplos de uma característica opcional e uma obrigatória. Como esses nós estão colocados como filhos de um nó opcional, eles só estarão presentes no software final caso todos os nós ancestrais estiverem selecionados. As características são mapeadas ao código fonte que representa toda a linha de produtos. Cada seleção de características é utilizada para gerar um membro da linha de forma otimizada para um uso específico, evitando que código não utilizado seja disponibilizado.

2.6 Frameworks Transversais

Um Framework Transversal (Crosscutting Framework) (FT) é um tipo de framework orientado a aspectos que encapsula comportamento genérico de um único interesse transversal, permitindo reúso desses interesses, como persistência, segurança e regras de negócio (CAMARGO; MASIERO, 2005). FTs possuem mecanismos de composição abstratos e podem ou não possuir variabilidades funcionais. Esse tipo de framework é bastante similar a um framelet em seu propósito. Framelets são frameworks orientados a objetos que têm o objetivo de facilitar o reúso de interesses transversais (PREE et al., 1999), porém, seu reúso não é totalmente modularizado, exigindo a inserção de chamadas de métodos transversais dentro dos métodos das classes base. FTs diferem na técnica empregada na implementação, modularizando o reúso sem causar espalhamento e entrelaçamento de código.

Assim como qualquer framework, um FT também possui “requisitos de reúso”. Esses requisitos de reúso são atividades que devem ser obrigatoriamente feitas pelo engenheiro de aplicação para conseguir reusar o FT. Por exemplo, os requisitos de reúso de um FT de Controle de Acesso podem ser:

1. Informar pontos da aplicação base em que o acesso deve ser controlado;
2. Informar o nome dos papéis que os usuários do sistema podem assumir;
3. Informar o número de vezes que um usuário pode errar sua identificação, etc. Muitas vezes esses requisitos de reúso são detalhados em documentos textuais ou manuais de instanciação de frameworks, conhecidos como cookbooks.

Diferentemente de um framework de aplicação que gera uma aplicação completa, um FT sempre deve ser acoplado a uma aplicação (ou código-base) que já existe para que possa operar adequadamente. Esse código-base deve ser compatível com os requisitos de reúso do FT.

O processo de reúso de um FT contempla duas etapas distintas: instanciação e composição. A instanciação é o processo convencional de reúso dos frameworks e consiste em especializar o código que foi especialmente projetado para isso. No processo pelo qual implementam-se os ganchos do framework, escolhe-se alguma funcionalidade alternativa ou implementa-se uma nova. A etapa de composição por sua vez, consiste em identificar pontos de junção e fornecimento de regras de composição. Também é necessário fornecer regras que unem as variabilidades escolhidas com o código-base.

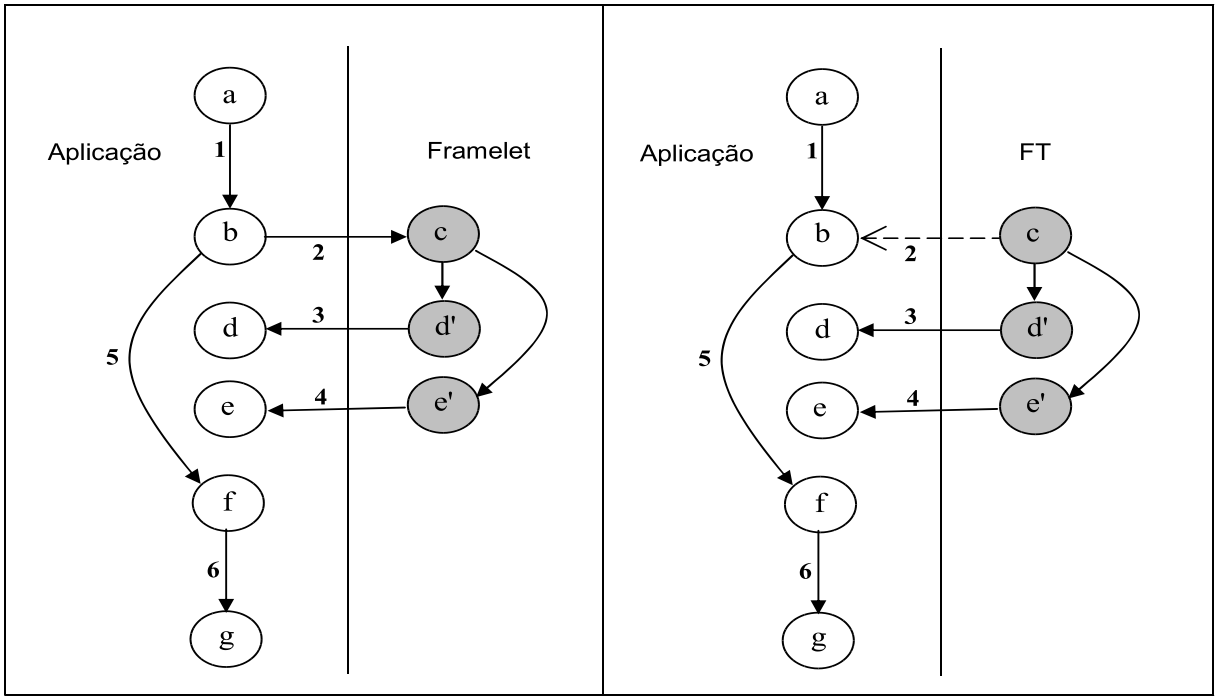

Figura 13: Inversão de Controle nos Framelets e nos Frameworks Transversais
Fonte: Camargo (2006)

A execução de um FT é similar à de um framelet, o fluxo de controle principal é da aplicação, porém, ao contrario de um framelet, a aplicação não necessita invocar o FT. Na Figura 13 há uma exemplificação gráfica de seu funcionamento. No caso do FT (grafo do lado direito), ocorre uma inversão de dependências, visto que as abstrações do paradigma permitem que o framework atue na aplicação base, aplicando o comportamento transversal sem invadir o conteúdo do código base, mantendo a modularização.

Por exemplo, se o interesse transversal a ser tratado é o de persistência, o nó 'b' representaria uma alteração de um valor que deve ser persistido e o nó 'c' a atualização no banco de dados. Nota-se que no caso do Framelet, a aplicação base deve invocar o tratamento de persistência, enquanto que no FT, o tratamento é inserido externamente. Na Subseção 2.6.1 há uma exemplificação de um FT para tratar o interesse de persistência, incluindo sua especificação, variabilidades e características.

2.6.1 Família de Frameworks Transversais de Persistência

Nesta subseção é apresentado uma Família de Frameworks Transversais (FFT) de Persistência, cujo objetivo é facilitar o desenvolvimento de uma aplicação orientada a objetos que utiliza banco de dados relacional. Os mecanismos necessários para tratar as incompatibilidades entre os dois paradigmas (relacional e orientado a objetos) são implementados pelos membros dessa família, que são frameworks transversais de

persistência, e podem ser reusados durante o desenvolvimento de novas aplicações.

A Família de Frameworks Transversais é uma aplicação do conceito de linhas de produtos de software ao contexto de Frameworks Transversais. Dessa forma, os membros da família são FTs que podem ser derivados de uma FFT.

Na Figura 14 é mostrado um diagrama de características que representa a FFT de Persistência seguindo a notação proposta por Gomma (2004). Algumas características foram implementadas como FTs, mas também há outras que foram implementadas como aspectos normais e com mecanismos tradicionais da orientação a objetos, como métodos e classes abstratas. Apenas a título de ilustração, as características implementadas como FTs/aspectos estão destacadas em cinza.

As características “Operações Persistentes” (*Persistent Operations*) e Conexão (*Connection*), ambas implementadas como FTs, são comuns a todos os frameworks de persistência membros dessa família e representam as operações que devem ser inseridas nas classes de aplicação persistentes e o interesse de conexão com o banco de dados, respectivamente. Assim, a etapa de instanciação do membro da família consiste em escolher as variabilidades da característica “Operações Persistentes” e também da Conexão. Essa é a configuração mínima do FT de Persistência, já que as demais características são opcionais. Embora os FTs de “Operações Persistentes” e Conexão estejam sendo mostrados como duas características, poderiam ser representados em apenas uma, já que ocorrem sempre juntos nos membros da família. Em virtude disso, no decorrer do texto o termo “FT de Persistência” é utilizado para representar essas duas características.

Todas as características implementadas em FFTs contêm camada de precomposição, o que é feito para facilitar a adoção da característica em um novo membro da família, sem a necessidade, ou minimizando, o trabalho de instanciação e composição. Por exemplo, a característica de Memória Auxiliar “Object Loader” está implementada como um FT. Embora essa característica esteja projetada como um framework, visando seu reúso em outros contextos, é dentro da família que provavelmente ela será mais utilizada. Assim, com esta camada, os FFTs são providos com um código para facilitar o acoplamento de seus membros.

Na Figura 15 é ilustrada a estrutura da família, assim, todas as características da FFT estão visíveis. Essa FFT consiste de dois módulos com objetivos distintos: O módulo “operações persistentes” é um conjunto de operações de persistência que devem ser herdadas por classes de aplicação persistentes (classes que possuem informações que devem ser persistidas) e que podem ser utilizadas para armazenar, remover, atualizar e realizar consultas no banco de dados. A estratégia de implementação adotada no projeto do FT foi introduzir todas as operações de persistência em

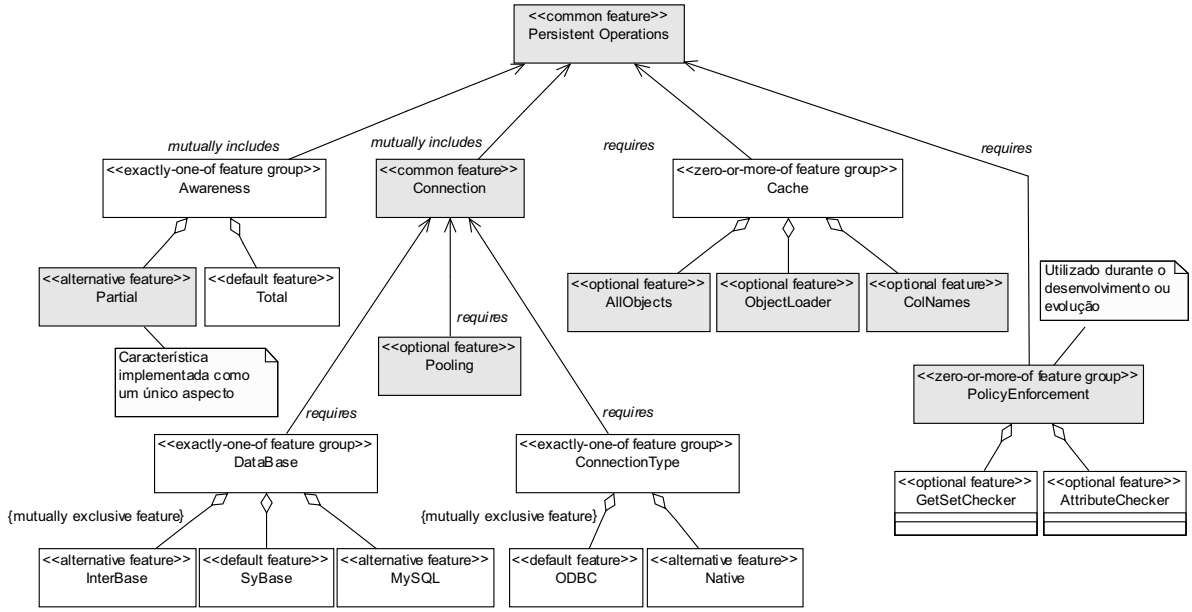

Figura 14: Características de Frameworks de Persistência
Fonte: Camargo (2006)

uma interface e fazer com que as classes de aplicação persistentes implementem essa interface.

O módulo “conexão” é referente ao interesse de conexão com o banco de dados, que tem como objetivo identificar locais do código-base em que a conexão deve ser aberta e fechada. A primeira parte depende da segunda para conseguir executar as operações de persistência.

Nessa figura, os aspectos estão sendo representados por retângulos destacados em cinza, e os relacionamentos de associação que partem dos aspectos para alguma entidade representam que o aspecto afeta a entidade entrecortando (*crosscutting*) a execução/chamada de seus métodos ou introduzindo (*intertype declaration*) operações, por meio de declarações intertipo. Por exemplo, a classe *ConnectionManager* é responsável por gerenciar a conexão com o banco de dados, o aspecto *PersistentEntities* é responsável por inserir atributos e operações relacionadas à persistência em classes marcadas com a interface *PersistentEntities*.

A aplicação base não é dependente de um FT pelo fato de empregar as abstrações presentes no paradigma para permitir inversão de dependência. Porém, um FT pode possuir restrições arquiteturais com relação às aplicações base, como por exemplo, pode ser necessário que a aplicação base possua certas classes, operações e atributos, para que seja possível inserir o comportamento transversal.

Por exemplo, os FTs membros da FFT de persistência apresentada possuem algumas restrições quando às aplicações base:

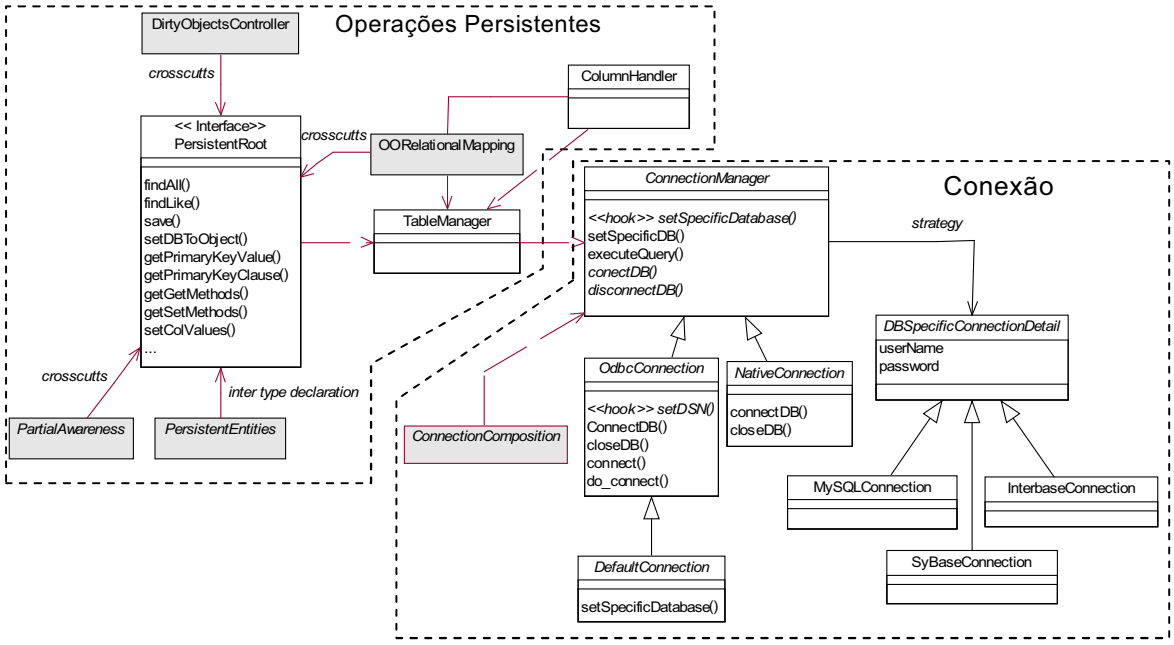

Figura 15: Estrutura do FT de Persistência
Fonte: Camargo (2006)

1. Toda classe de aplicação persistente deve possuir construtores com e sem parâmetros;
2. Cada classe de aplicação persistente deve ter uma tabela correspondente no banco de dados com o mesmo nome da classe;
3. Toda classe de aplicação persistente deve possuir métodos de acesso aos atributos (operações *setters* e *getters*);
4. Cada atributo de uma classe de aplicação persistente deve possuir uma coluna na tabela correspondente no banco de dados com o mesmo nome do atributo;
5. Todo método que altera um atributo do tipo int deve receber um parâmetro do tipo *Integer*.

Essas restrições arquiteturais, também chamadas de políticas de implementação, podem ser vistas como contratos que devem ser seguidos para o correto funcionamento do FT (GRISWOLD et al., 2006).

Para reusar esses FTs, inicialmente é necessário possuir uma aplicação base projetada conforme as restrições arquiteturais. Em seguida é necessário realizar a escolha de variabilidades, como o banco de dados a ser utilizado e a conexão a ser empregada. Por fim, definir classes persistentes da aplicação base. Para isso é preciso que seja criado um código de reúso, que é um conjunto de classes e aspectos que realiza a composição do framework à aplicação base.

Por exemplo, ao criar o código de reúso para um membro dessa família, é necessário escolher a variabilidade do tipo de conexão, para isso, deve-se estender uma classe filha de *ConnectionManager*: *ODBCConnection* ou *NativeConnection*. Para definir classes de aplicação como classes persistentes, é necessário criar um aspecto para inserir o relacionamento de implementação da interface *PersistentRoot* nessas classes.

É importante ressaltar que esse processo manual de criação do código de reúso possui um baixo nível de abstração e exige que o desenvolvedor da aplicação possua conhecimento de muitos detalhes do FT, além de estar mais propício a cometer erros. No contexto desse trabalho, o processo de desenvolvimento dirigido a modelos é utilizado com o objetivo de elevar o nível de abstração.

2.7 Considerações Finais

Neste capítulo foram apresentados conceitos necessários para a compreensão e elaboração da abordagem proposta. No Capítulo 3 são apresentados trabalhos relacionados que também utilizam de ferramentas para apoiar o reúso de frameworks. A abordagem proposta neste trabalho é apresentada e descrita no Capítulo 4.

Capítulo 3

TRABALHOS RELACIONADOS

3.1 Considerações Iniciais

Nesta seção são apresentados quatro trabalhos que propõem estratégias de reúso de frameworks com apoio de ferramentas. Estão inclusas abordagens que também fazem uso de desenvolvimento dirigido por modelos e orientação a aspectos.

3.2 Ferramenta ALFAMA

Santos, Koskimies e Lopes (2008) desenvolveram uma abordagem que também suporta um conjunto maior de frameworks e faz uso de linguagem de modelagem específica. Nessa abordagem, o engenheiro de domínio deve estabelecer uma "aplicação modelo" e um metamodelo para cada framework a ser reusado. Estes dois artefatos devem ser criados seguindo um conjunto de diretrizes para formar uma camada que adapta o gerador de código da aplicação final ao framework a ser reusado.

Dessa forma, essa abordagem permite que um único gerador de código seja empregado desde que essa camada seja criada corretamente. A "aplicação modelo" é criada utilizando a linguagem AspectJ agregada de anotações que ligam o código ao metamodelo do framework, ou seja, as metaclasses às classes da aplicação modelo e os atributos das metaclasses às propriedades das classes da aplicação modelo.

O engenheiro de aplicação deve então editar uma instância do metamodelo criado para o framework a ser reusado. Ao preencher os atributos dos objetos instanciados do metamodelo, o gerador de código pode ser executado para gerar uma aplicação com base na aplicação modelo que contém os nomes e propriedades do modelo criado pelo engenheiro de aplicação.

Na Figura 16 são mostradas telas do protótipo criado por estes autores para implementar sua abordagem. Esse protótipo foi nomeado de ALFAMA (Automatic DSLs

for using Frameworks by combining Aspect-oriented and Meta-modeling Approaches). No lado esquerdo da figura é possível visualizar a edição do metamodelo e da aplicação modelo, que são tarefas do engenheiro de domínio. No lado direito da figura é representado um editor de modelo, que é a tarefa do engenheiro de aplicação para criar uma aplicação que reúsa o framework.

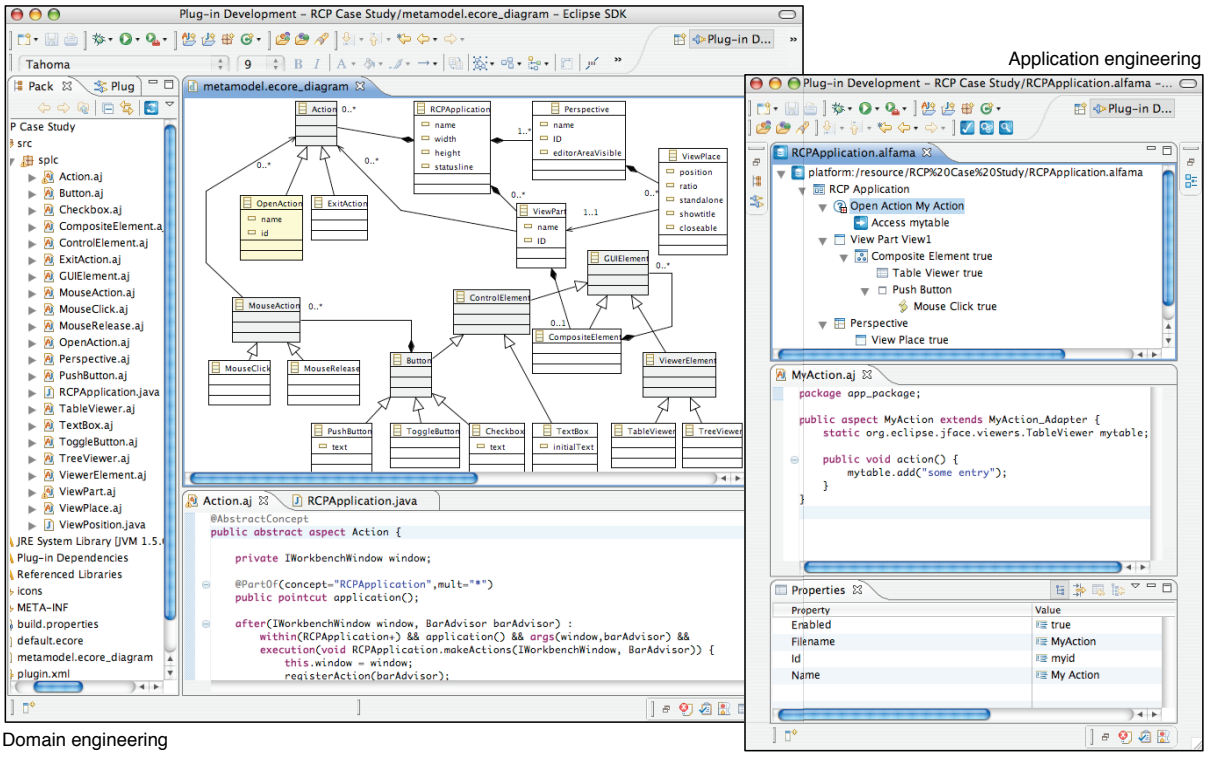

Figura 16: Ilustração da ferramenta ALFAMA
Fonte: Santos, Koskimies e Lopes (2008).

3.3 Reuse Definition Language

Oliveira et al. (2007) desenvolveram um processo dirigido a modelos para apoiar o reúso de frameworks orientados a objetos com o objetivo de permitir rápido reúso de frameworks. O processo permite selecionar as variabilidades e os recursos que serão usados do framework.

Na Figura 17 é ilustrado o processo proposto por Oliveira et al. (2007). A partir do modelo de características (features) à esquerda, são selecionadas as features do framework. Isso é possível, pois o modelo possui relacionamentos com as classes do framework, chamados de “Trace”.

O modelo de classes do framework possui elementos marcados por estereótipos presentes na Tabela 1. O objetivo desses estereótipos é definir regras de instanciação

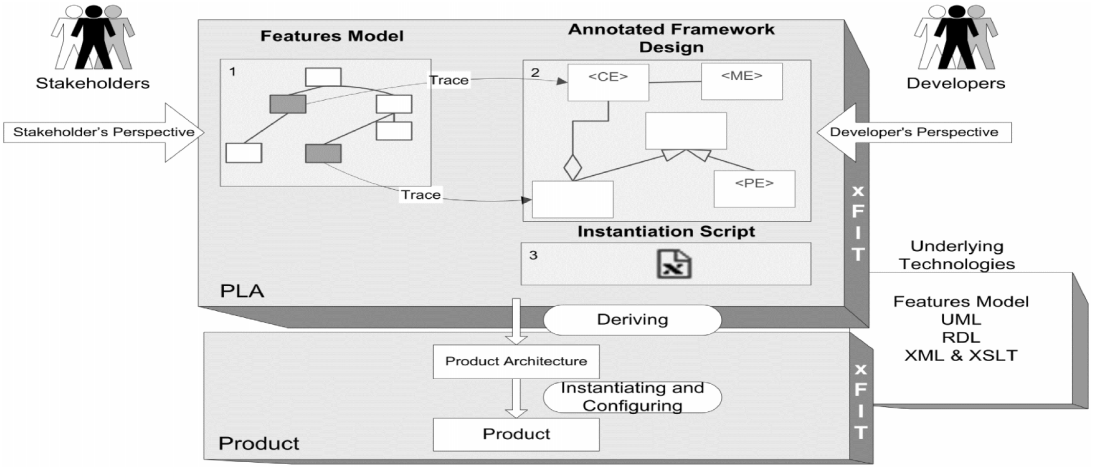

Figura 17: Processo de Reúso de Framework
Fonte: Oliveira et al. (2007)

do framework. Por exemplo, se uma classe do framework deve ser estendida para permitir o reúso, então ela pode receber o estereótipo <<CLASS_EXTENSION>>.

Tabela 1: Elementos do Perfil UML-FI

Estereótipo	Metaclasse	Descrição
<<CLASS_EXTENSION>>	*Class*	Indica que a classe precisa ser estendida.
<<PATTERN_CLASS_ EXTENSION>>	*Class*	Indica que a classe precisa ser estendida por meio de aplicação de um padrão.
<<SELECT_CLASS_ EXTENSION>>	*Class*	Indica que a classe precisa ser estendida, por meio de uma subclasse já existente.
<<METHOD_EXTENSION>>	*Operation*	Indica que o método deve realizar sobrescrita em uma subclasse.
<<PATTERN_METHOD_ EXTENSION>>	*Operation*	Indica que o método deve realizar sobrescrita por meio de aplicação de um padrão.
<<VALUE_ASSIGNMENT_ EXTENSION>>	*Property*	Indica que o atributo necessita de um valor.
<<VALUE_SELECTION_ EXTENSION>>	*Property*	Indica que o atributo necessita de um valor, que pode ser selecionado de uma lista de valores possíveis.

Fonte: Oliveira et al. (2007)

Com o modelo do framework marcado com os estereótipos, é possível aplicar uma linguagem especifica de domínio para representar o reúso do framework, chamada de Reuse Definition Language (RDL). Essa linguagem é utilizada para especificar como deve ser realizado o processo de reúso. Portanto, seguindo o exemplo de estender uma classe do framework, na Figura 18 é mostrado como utilizar a RDL para criar uma subclasse. Assim, “Classe2” é criada como filha de “Classe”, processo realizado inteiramente pela ferramenta de transformação.

Em seguida da especificação da RDL, um processo de transformação irá identificar quais características foram selecionadas para o processo de reúso e gerará o código da aplicação final.

Código RDL:

```
COOBOOK Exemplo
    RECIPE main
     // Estendendo uma classe
     CLASS_EXTENSION (Classe,Classe2);
     //
    END_RECIPE;
END_COOKBOOK
```

Diagrama:

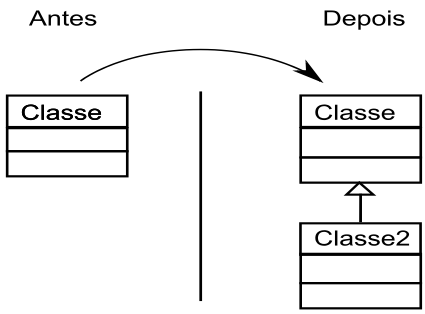

Figura 18: Uso da RDL para Estender Classes
Fonte: Oliveira et al. (2007)

3.4 Framework Specification Modeling Language

A abordagem de Antkiewicz e Czarnecki (2006) é embasada na criação de um tipo especifico de linguagens específicas de domínio denominadas FSMLs (Framework Specification Modeling Languages, ou seja, Linguagens de Modelagem para Especificação de Frameworks). Essas linguagens são destinadas a representar as informações que a API (Application Programming Interfaces) de um framework de aplicação necessita durante o reúso.

Antkiewicz e Czarnecki (2006) criaram um processo de desenvolvimento *round-trip* (ida e volta). Uma ilustração do processo pode ser visto na Figura 19, em que são criados modelos a partir de um código de exemplo de reúso do framework de aplicação e os códigos que podem ser gerados a partir desse novo modelo são comparados com o código original, em uma etapa chamada de conciliação.

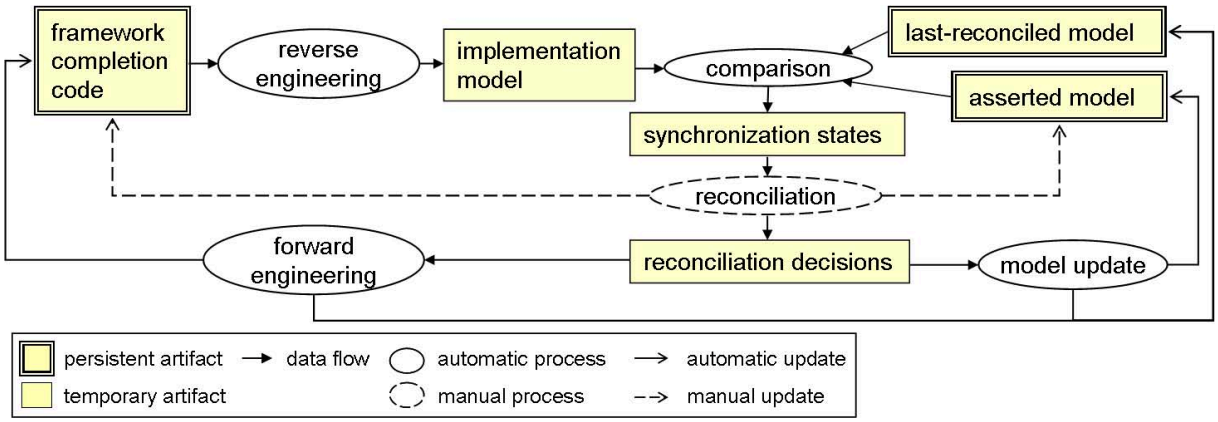

Figura 19: Processo de Antkiewicz e Czarnecki (2006)

Os modelos são mapeados ao código com definições em uma linguagem textual. Ao utilizar este mapeamento, um processo de transformação realiza a etapa de geração de modelo do código de exemplo de uma aplicação e outro processo de transfor-

mação para gerar o código da aplicação a partir do modelo.

A etapa de conciliação permite uma tomada de decisões de forma a melhorar tanto o código de reúso quanto o modelo. Esse processo pode ser realizado de forma iterativa e incremental até adquirir um código de aplicação final ideal e um modelo ideal que o represente.

Como estudo de caso, uma foi criada uma FSML para um framework de aplicação do domínio de plugins do Eclipse IDE (ECLIPSE CONSORTIUM, 2012) denominado WPI (Workbench Part Interactions). Essa abordagem foi implementada como um plugin de Eclipse IDE (ECLIPSE CONSORTIUM, 2012) e possui um metamodelo implementado em MOF, que é uma linguagem de metamodelagem utilizada para definir outras linguagens, como a UML (OMG, 2010c). Este metamodelo esta representado graficamente na Figura 20.

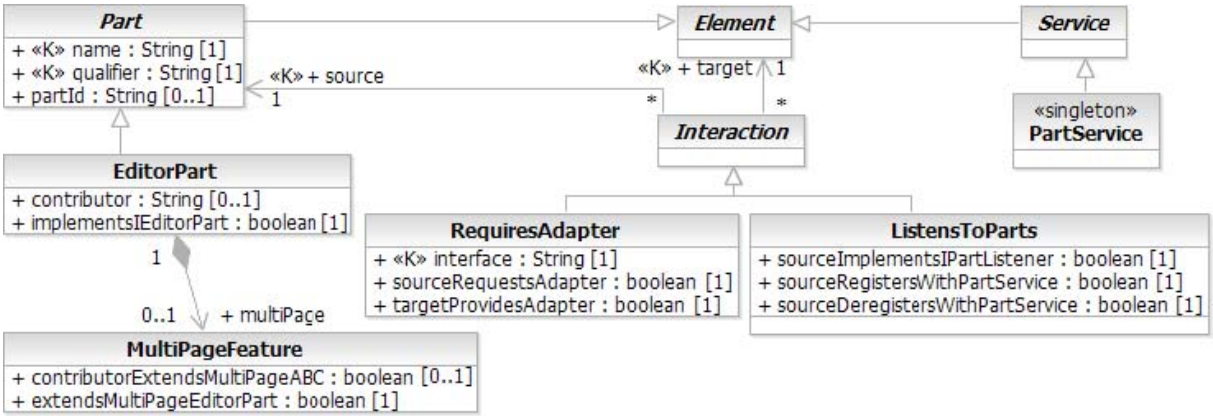

Figura 20: Metamodelo da FSML para WPI
Fonte: Antkiewicz e Czarnecki (2006)

Por exemplo, a metaclasse *EditorPart* é instanciada em nível de modelo com um objeto denominado “ep”. O código da aplicação possui uma classe denominada “editor” que é mapeada ao objeto “ep”. Isso é feito por meio do código na linguagem textual especifica de domínio para realizar o mapeamento com a seguinte linha: “*mapping EditorPart (EditorPart ep<->Class editor);*”.

Esse mapeamento é bilateral, ou seja, permite ao gerador de código gerar a classe “editor” a partir do objeto “ep” do modelo e também permite ao gerador de modelos gerar o objeto “ep” a partir da classe “editor”. Os autores argumentam que após estabelecer um modelo que gera um código correto e vice-versa, o modelo está pronto para ser utilizado para gerar outras aplicações com base no mesmo framework. Além disso, o processo inverso pode ser utilizado para gerar um modelo de forma a documentar aplicações criadas com o framework.

3.5 On-Board Software

Na Figura 21 é ilustrado o processo da abordagem de Cechticky et al. (2003), que foi utilizada ao criar a ferramenta On-Board Software. Essa ferramenta possui três entradas: configuração, o framework e algoritmo de controle. A configuração consiste de regras e definições apoiadas pela ferramenta. O framework é sempre um mesmo framework no contexto de sistemas de tempo real. O algoritmo de controle tem como objetivo impor a sequência de execução.

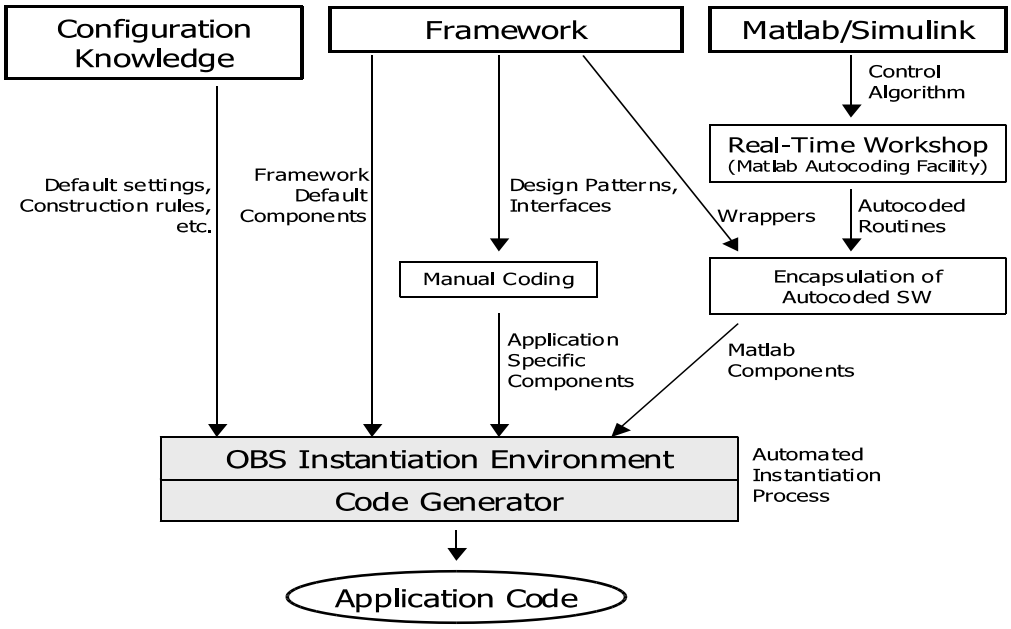

Figura 21: Abordagem de Cechticky et al. (2003)

Na Figura 22 é ilustrado o processo de transformação de modelos empregado na abordagem de Cechticky et al. (2003). A ferramenta de transformação processa arquivos XML que descrevem os componentes do framework, configurações de transformação e a aplicação a ser criada. O arquivo de descrição dos componentes do framework possui estruturas com as informações requeridas pelos módulos, que serão relacionadas à aplicação, por meio de Proxy Beans, que é um padrão arquitetural para mapear chamadas. A descrição da aplicação a ser criada é conhecida como modelo de aplicação. A ferramenta também provê um editor gráfico para facilitar a especificação desse modelo. Após a definição da composição, é gerado um arquivo XML especificando a descrição da aplicação final, conhecido como descrição formal de configuração da aplicação. Esse arquivo é utilizado de entrada para a geração do código da aplicação final. Além disso, esse arquivo pode ser alterado para melhor adequar a transformação do código da aplicação.

Essa abordagem se diferencia da proposta deste trabalho por três razões. A primeira é que ela é restrita para poucos frameworks; a segunda é que ela é específica

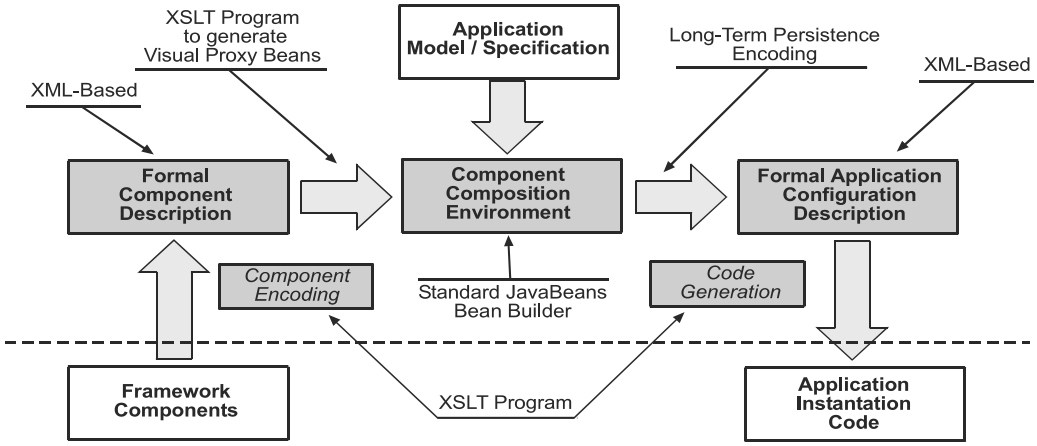

Figura 22: Processo de Transformação na Abordagem de Cechticky et al. (2003)

para Orientação a Objetos e a terceira é que o processo de reúso gera uma nova aplicação.

3.6 Processo de Desenvolvimento da GREN-WIZARD

Braga e Masiero (2003) propuseram um processo para criar ferramentas para apoiar o reúso de frameworks de aplicação. Para a aplicação do processo, é necessário que o domínio do framework seja representado por uma linguagem de padrões. Esta linguagem é criada com o vocabulário do domínio do problema cujas possíveis soluções são propostas. Pelo fato dessas soluções possíveis serem nomeadas de “padrões”, isso também nomeia o tipo da linguagem.

Com o framework a ser reusado mapeado a uma linguagem de padrões, o primeiro passo do processo consiste em criar um repositório de padrões, que são as soluções possíveis para o domínio do problema. O segundo passo consiste em armazenar, em outro repositório, a arquitetura prevista das aplicações criadas com base no framework de forma a solucionar um problema.

No terceiro passo, é criada uma aplicação que irá assistir o engenheiro de aplicação a reusar o framework. Nessa aplicação, também chamada de assistente, são providos campos com base na estrutura da linguagem de padrões que permite ao engenheiro de aplicação entrar com dados sobre a aplicação a ser instanciada.

Durante o último passo do processo, um gerador de código deve ser criado, o que deve ser capaz de gerar o código de uma aplicação reusando o framework de acordo com as informações inseridas no assistente.

Os autores aplicaram o processo com sucesso para criar uma ferramenta para reusar um framework que implementa a linguagem de padrões GRN (BRAGA; GERMANO;

MASIERO, 1999). Esta aplicação foi denominada “GREN-Wizard” e possui a interface gráfica ilustrada na Figura 23.

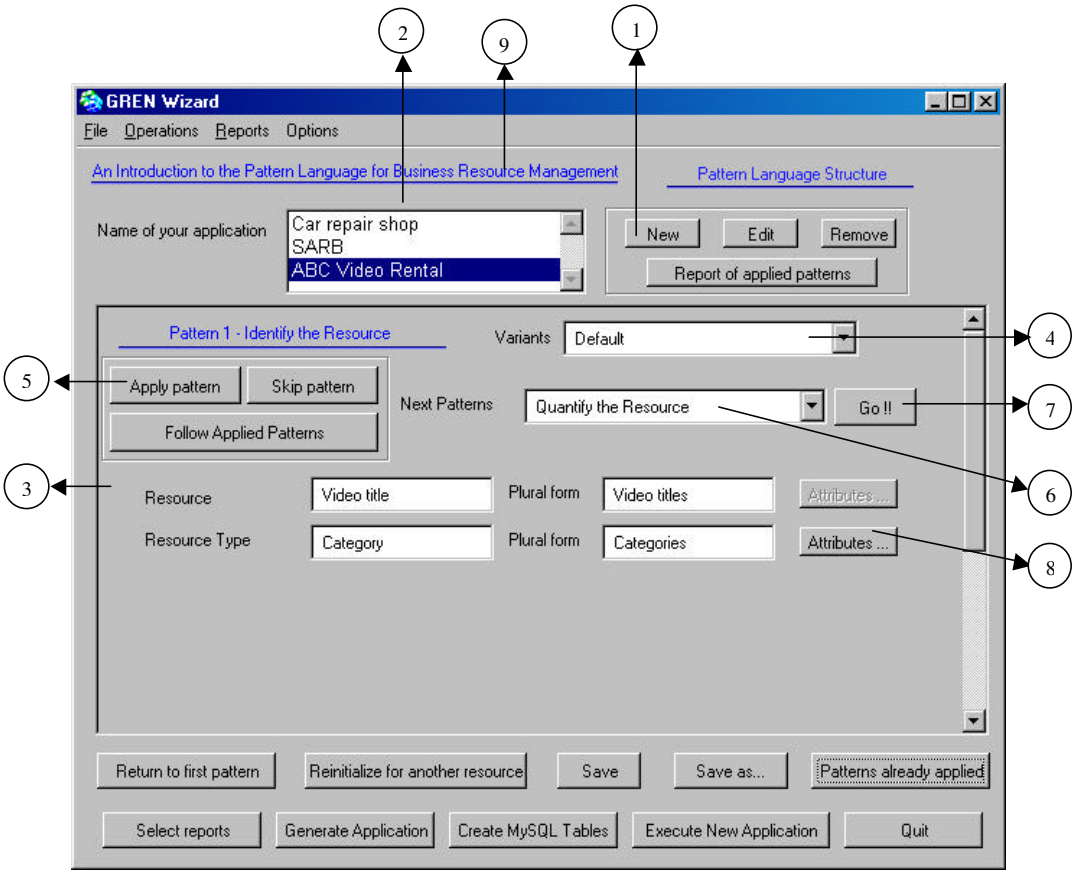

Figura 23: Abordagem de Braga e Masiero (2003)

Novas aplicações que reusam o framework podem ser criadas por um botão (indicado pelo número 1). A nova aplicação é então listada na caixa de aplicações (2) com um padrão da GRN (3). Conforme o padrão a ser aplicado, campos serão exibidos para serem preenchidos com informações da aplicação a ser instanciada. A variante do padrão (4) deve ser selecionada de acordo com o padrão a ser aplicado. O botão “Apply Pattern” (5) (aplicar padrão) é utilizado para salvar as definições da aplicação a ser instanciada. O botão “Attributes” (8) (atributos) permite alterar nomes de atributos para classes de aplicação ou adicionar novos atributos. O próximo padrão a ser aplicado pode ser selecionado em uma lista (6). O botão “Go” (7) (ir) é ativado para adaptar a interface gráfica às necessidades do padrão selecionado (6). Ao concluir o assistente, uma aplicação pode ser gerada.

3.7 Considerações Finais

Neste capítulo foram apresentados cinco trabalhos relacionados. O processo de Braga e Masiero (2003) permite a criação de ferramentas para apoiar o reúso de qualquer framework de aplicação orientado a objetos que implemente uma linguagem de padrões. Cechticky et al. (2003) criaram uma ferramenta para apoiar o reúso de um framework de aplicação específico.

As vantagens da abordagem de Oliveira, Alencar e Cowan (2011) com relação a abordagem de Cechticky et al. (2003) são: (*i*) Maior capacidade de configuração do processo de reúso; (*ii*) a RDL é uma linguagem poderosa que pode ser utilizada para inserir código no software final e o perfil proposto permite aplicar o processo em uma enorme gama de frameworks orientados a objetos, e não apenas um número restrito.

A proposta de Santos, Koskimies e Lopes (2008) faz uso de orientação a aspectos, mas, apenas foi exemplificado o reúso de frameworks de aplicação. Para adaptar a abordagem a cada framework, é necessário criar um novo metamodelo de acordo com diretrizes propostas para manter o mesmo gerador de código.

A proposta de Antkiewicz e Czarnecki (2006) envolve linguagens especificas de domínio para representar o que é necessário conhecer durante o reúso de frameworks de aplicação orientados a objetos. Estes autores também elaboraram um processo de desenvolvimento *round-trip* (ida e volta) em que modelos são transformados em código e códigos são transformados em modelos. Essa abordagem permite realizar comparações entre o item original com o resultado da transformação.

O reúso de frameworks transversais com abstrações orientadas a aspectos permite a completa modularização do código de reúso. A falta de modularização do código de reúso pode acarretar vários problemas, por exemplo: (*i*) Não é possível aplicar dois frameworks a uma mesma aplicação pois cada execução de processo gera uma nova e isolada aplicação final; (*ii*) A equipe de desenvolvimento da aplicação depende da completa elaboração do processo de reúso para iniciar o desenvolvimento; (*iii*) Caso o processo de reúso necessite de alterações, todo o código final será afetado e (*iv*) Não é possível estabelecer um desenvolvimento incremental com relação às características dos frameworks.

Por fim, a abordagem proposta neste trabalho é focada em frameworks transversais e permite o uso de mesmo metamodelo e gerador de código para vários frameworks transversais. Além disso o código gerado é modularizado por utilizar de conceitos orientados a aspectos.. Esta abordagem é apresentada e descrita no Capítulo 4.

Capítulo 4

REÚSO DE FTS COM APOIO DE MODELOS

4.1 Considerações Iniciais

Neste capítulo é apresentada a abordagem proposta de reúso de Frameworks Transversais com apoio de modelos. A abordagem é apoiada por dois modelos, o primeiro é denominado de Modelo de Requisitos de Reúso e o segundo de Modelo de Reúso. Ambos modelos são definidos por um único metamodelo. O metamodelo e os modelos propostos são descritos na Seção 4.2. Com o uso dos modelos propostos, é definido um processo de reúso de frameworks transversais. Este processo é apresentado na Seção 4.3. Uma ferramenta foi implementada para suportar o processo proposto, que é apresentada na Seção 4.4.

Para melhor compreensão deste capítulo, é importante o conhecimento de dois engenheiros de software com papéis diferentes. O primeiro engenheiro de software é responsável pelo desenvolvimento do Framework Transversal, sendo assim referenciado como “Engenheiro de Framework”; o segundo engenheiro de software é responsável pelo desenvolvimento de uma aplicação base que reusa o framework, sendo assim referenciado como “Engenheiro de Aplicação”.

4.2 Modelos Propostos

4.2.1 Modelo de Requisitos de Reúso

O primeiro modelo foi proposto com o objetivo de documentar todos os requisitos de reúso de um FT isolado ou uma FFT. Dessa forma, ele complementa documentos e manuais que descrevem as variabilidades do framework, assim como é feito em “Cookbooks” (JOHNSON, 1992).

Neste trabalho, Requisito de Reúso (RR) é o nome atribuído para toda informação

necessária para reusar um Framework Transversal e acoplá-lo a uma aplicação base, por exemplo, nome de métodos ou classes da aplicação base que devem receber tratamento transversal do framework.

Note-se que esses requisitos de reúso são informações que o framework depende para ser corretamente acoplado com uma aplicação base. A ideia é que estes modelos sejam criados pelo engenheiro de framework para documentar seu framework.

Esse modelo pode conter os requisitos de reúso agrupados por características da família de frameworks transversais. Esses grupos de caraterísticas podem ser utilizados para instanciar um outro modelo contendo apenas as características (*features*) selecionadas. Dessa forma, os elementos de grupo podem ser utilizados para definir a hierarquia de características.

Além disso, esse modelo pode ser analisado por um engenheiro de aplicação de forma a identificar quais informações de sua aplicação base serão necessárias para que o framework seja acoplado corretamente.

É importante notar que apesar desse artefato ser referenciado como modelo, ele não possui detalhes de uma aplicação base, o que deve ser preenchido por um engenheiro de aplicação.

Por exemplo, na Figura 24 é mostrado um formulário de um modelo hipotético de requisitos de reúso que foi criado para representar todas as informações necessárias para acoplar um framework transversal hipotético a uma aplicação base.

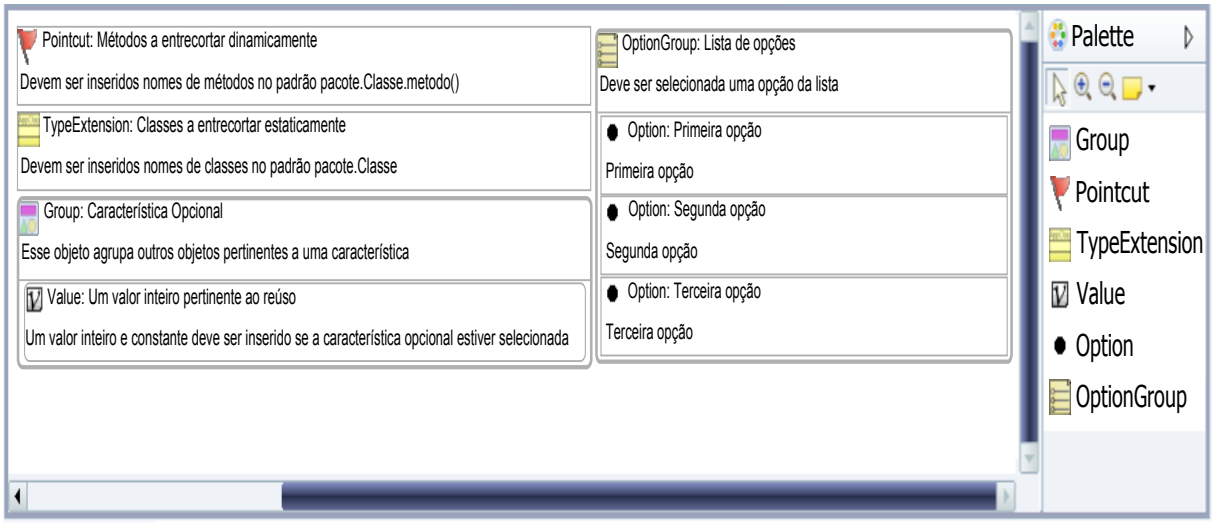

Figura 24: Exemplo de Modelo de Requisitos de Reúso

Os Modelos de Requisitos de Reúso (MRR) possuem seis tipos de elementos: “Conjuntos de Junção” (“Pointcut”), “Extensões de Tipo” (“TypeExtension”), “Valores” (“‘Value”),“Grupos de Opções” (“OptionGroup”), “Opções” (“Option”) e “Grupos” (“Group”). Com a paleta (“Pallete”), visível na direita da Figura 24, é possível criar objetos destes

tipos no modelo de requisitos de reúso.

As caixas à esquerda da paleta da Figura 24 representam objetos de um modelo de requisitos de reúso para um framework transversal hipotético. Nessas caixas, a primeira linha contém o nome e a segunda linha contém uma descrição para instruir o engenheiro de aplicação.

A primeira caixa da figura contém a o nome “Pointcut: Métodos a entrecortar dinamicamente” e denota que o framework entrecorta métodos da aplicação base e o nome desses métodos devem ser informados quando o engenheiro de aplicação estiver conduzindo o processo de reúso. Apesar do exemplo de “Pointcut” depender de nome de métodos, é importante notar que esta abordagem também suporta outros tipos de conjuntos de junção, que, por exemplo, capturam acesso a atributos, entre outras formas possíveis em AspectJ.

A segunda caixa do formulário contém o nome “TypeExtension: Classes a entrecortar estaticamente” e denota que o framework depende da definição de classes da aplicação base que devem receber atributos por meio de entrecorte estático. A abordagem também suporta o uso deste objeto para definir aspectos e interfaces.

A terceira caixa é um grupo e é identificado pelo nome “Group: Característica Opcional”. Nessa abordagem grupos são utilizados para agrupar qualquer tipo de elemento de modelo, portanto, todos os elementos, incluindo “Pointcut”, “TypeExtension” e o próprio “Group” podem ser inseridos de forma aninhada.

Grupos também são empregados para agrupar requisitos específicos de uma característica (*feature*). Dessa forma, se a característica com nome do grupo não for selecionada, os elementos aninhados não serão necessários durante o reúso. No exemplo, o quarto objeto está dentro do grupo e é identificada pelo nome “Value: Um valor inteiro pertinente ao reúso”. Esse tipo de objeto é empregado para fornecer uma constante de variados tipos. Neste caso, é requisitado um valor inteiro.

Os últimos elementos do modelo são as opções (“Option”). Essas opções são uma forma alternativa de fornecer um valor durante o reúso do framework, porém, apenas valores booleanos são aceitos. Essa forma de representação é útil para permitir ao engenheiro de aplicação ativar ou desativar funções do framework. Essas opções podem ser agrupadas em grupos de opções (“OptionGroup”) para agrupar os elementos e facilitar a escolha. Nota-se que o grupo de opções é empregado apenas para unir elementos de opção e não afeta o comportamento destes elementos.

O Modelo de Requisitos de Reúso também tem uma função importante durante a criação do Modelo de Reúso. Isso será explicado na Seção 4.3 e também exemplificado no Capítulo 5. Na Seção 4.3 também é descrito como proceder para criar um

modelo de requisitos de reúso, o que também é exemplificado no Capítulo 5.

4.2.2 Modelo de Reúso

O Modelo de Reúso compartilha o mesmo conjunto de tipos de elementos do modelo anterior e sua visão em formulário também é semelhante graficamente. Este modelo se diferencia do anterior em dois pontos:

1. No formulário do Modelo de Reúso há um campo visível que deve ser preenchido pelo engenheiro de aplicação;
2. Este modelo é utilizado para apoiar o processo de reúso;
3. Ele representa requisitos de um FT isolado ou apenas de um membro da família de FTs e não toda a família.

Assim como o formulário do Modelo de Requisitos de Reúso, o formulário do Modelo de Reúso possui caixas que representam requisitos de reúso. Cada requisito de reúso representado além de ter um nome e descrição, neste modelo, também tem um campo para ser preenchido com informações relativas à aplicação base. Dessa forma, a ideia é que essa informação relativa à aplicação base seja incluída pelo engenheiro de aplicação sem necessidade de conhecer detalhes da implementação do framework ou de orientação a aspectos.

Assim que o Modelo de Reúso estiver concluído, é possível executar um processo de transformação de modelo para texto com o objetivo de gerar o código de reúso. Por exemplo, na Figura 25 há um exemplo de Formulário de Modelo de Reúso. Neste modelo há caixas representando requisitos de reúso e com os campos relativos à aplicação base já preenchidos.

Nessa figura são especificados métodos para os conjuntos de junção requeridos pelo framework. É importante notar que esta é uma continuação do exemplo da Seção 4.2.1.

Na Caixa “Pointcut”, observa-se que a última linha foi preenchida com “*pacoteA.ClasseB.MetodoC()*”. Esse e os outros valores presentes nas outras caixas devem ser digitados pelo engenheiro de aplicação no momento que ele estiver fazendo o reúso do FT. Na caixa “TypeExtension”, possui a última linha preenchida com “pacoteA.ClasseB”, contida na aplicação base para ser afetada pelo entrecorte estático do framework.

O elemento de grupo que está presente no modelo de requisitos de reúso apresentado na Seção 4.2.1 representa uma característica que não foi selecionada ao derivar

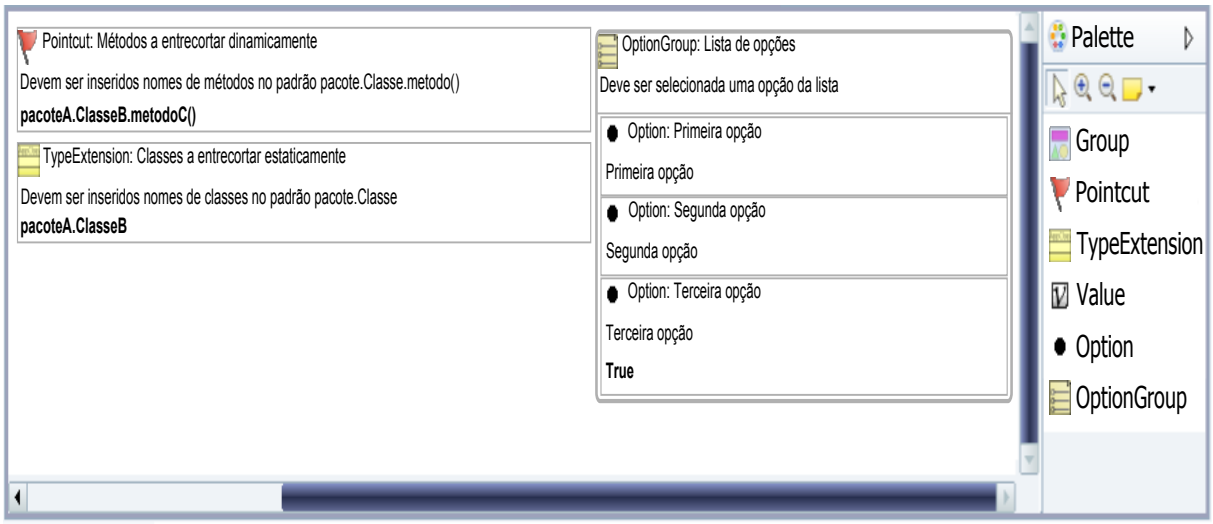

Figura 25: Exemplo de Modelo de Reúso

o modelo de reúso, o que também elimina os elementos aninhados. Caso o grupo estivesse selecionado, ele não seria eliminado do modelo de reúso e a Caixa “Value: Um valor inteiro pertinente ao reúso” estaria presente. Dessa forma, seria possível fornecer um valor inteiro na terceira linha desta caixa, assim como foi feito com as outras caixas descritas. Por fim, dentre os objetos de opção, a terceira opção foi escolhida ao ser definida como “true”.

O exemplo mostrado nesta seção contém apenas um tipo de cada elemento, mas é puramente sintético e não representa frameworks ou aplicações reais. Um exemplo real é mostrado no Capítulo 5.

4.2.3 Especificação dos modelos propostos

Os modelos propostos neste trabalho foram criados por meio de uma análise do domínio de reúso de frameworks transversais. Foi identificado um vocabulário com quatro elementos essenciais utilizados em manuais de instanciação (“Cookbooks”). Este vocabulário é importante para representar objetos, conceitos e outras entidades relacionadas ao domínio especificado.

Os elementos do vocabulário são as informações que o engenheiro de aplicação precisa para: (*i*) criar e sobrescrever conjuntos de junção; (*ii*) criar e sobrescrever métodos; (*iii*) selecionar variabilidades e (*iv*) estender classes, interfaces e aspectos.

Portanto, o metamodelo, que é a definição dos modelos propostos, foi elaborado com metaclasses concretas que representam esses conceitos do vocabulário. Um diagrama simplificado do metamodelo proposto pode ser visto na Figura 26. Este metamodelo possui enumerações e metaclasses abstratas e concretas. As metaclasses

concretas estão abaixo da linha tracejada enquanto que os outros elementos estão acima.

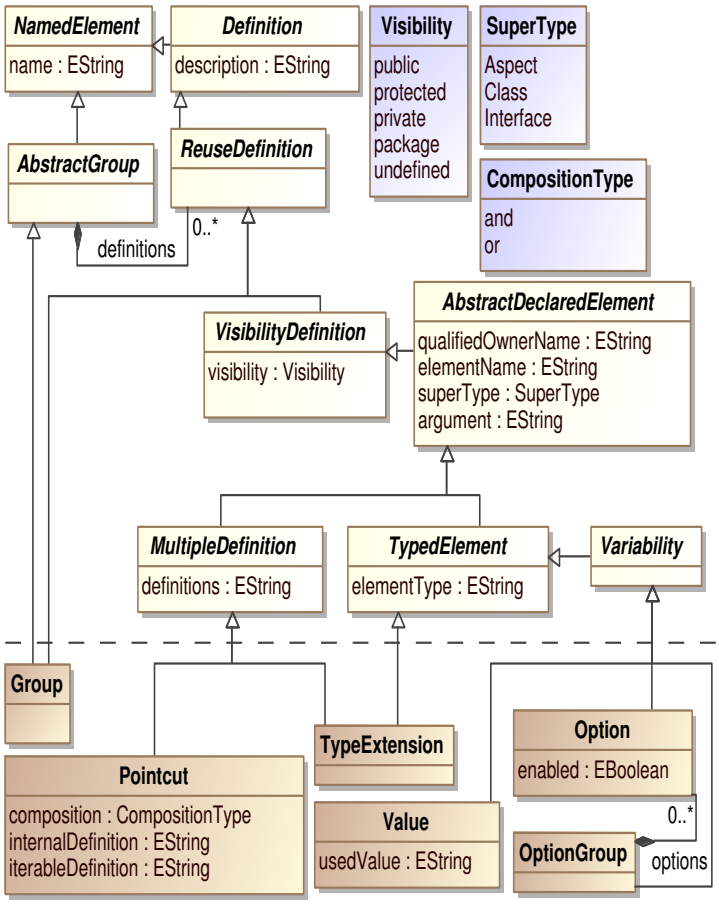

Figura 26: Diagrama Conceitual do Metamodelo Proposto

O metamodelo também possui um outro elemento concreto suplementar aos definidos pelo vocabulário. Esse elemento tem o nome de “Group” e agrupa qualquer elemento de modelo, incluindo outros grupos de forma recursiva para estabelecer uma hierarquia de elementos, o que é importante para agrupar funções ou características.

As metaclasses abstratas do metamodelo foram criadas ao generalizar atributos das classes concretas. Essa separação é importante para facilitar a elaboração do gerador de código evitando a replicação de código. As metaclasses abstratas estão acima da linha da Figura 26 ao lado das seguintes enumerações: “Visibility”, “SuperType” e “CompositionType”, que são conjuntos de literais utilizados em atributos das metaclasses.

As metaclasses concretas são aquelas que definem tipos de objetos válidos nos modelos propostos. O metamodelo possui seis metaclasses concretas:

1. “Group” (Grupo): um elemento para agrupar objetos válidos no modelos, incluindo grupos aninhados;

2. “Pointcut” (Conjunto de Junção): representa um conjunto de junção a ser sobrescrito pelo engenheiro de aplicação durante o reúso;

3. “TypeExtension” (Extensão de Tipo): representa um tipo do framework (aspecto, classe ou interface) que deve ser herdado ou implementado por elementos da aplicação base;

4. “Value” (Valor): representa a sobrescrita de um método do framework empregado para retornar um valor que seja requerido pelo framework durante o reúso;

5. “Option” (Opção): representa uma opção selecionável ao definir um valor booleano. Esse objeto é similar à “Value”, porém restrito a tipos booleanos;

6. “OptionGroup” (Grupo de Opções): agrupa objetos de opções para representar um conjunto de opções selecionáveis.

Este metamodelo permite a criação dos dois tipos de modelos, denominados neste trabalho de “Modelo de Requisitos de Reúso” (MRR) e “Modelo de Reúso” (MR). Um engenheiro de domínio pode construir um MRR que represente requisitos de reúso de FFTs ou de um FT individual. Este modelo representa os requisitos de reúso de todas as variabilidades da FFTs/FT e serve também como documentação.

O outro modelo, o MR, fornece apoio ao processo de reúso efetivamente, pois, o engenheiro de aplicação o utiliza para conduzir o processo de reúso propriamente dito, fornecendo valores para propriedades específicas. O objetivo de criar dois modelos é que o engenheiro possa utilizar o primeiro para escolher um subconjunto das características disponíveis da família de FTs e que, com o segundo modelo, ele possa usar apenas com o membro da família, contendo apenas elementos relativos às características que foram escolhidas.

Os modelos podem ser representados graficamente como formulários. O metamodelo foi desenvolvido utilizando a metalinguagem ECore, que é parte do EMF (Eclipse Modeling Framework). O editor para os formulários propostos foi desenvolvido com o GMF (Graphical Modeling Framework). Tanto o EMF quanto o GMF são frameworks integrantes do Eclipse Consortium (2011). Detalhes técnicos da definição do metamodelo em ECore estão disponíveis no Apêndice A.

4.3 Processo de Reúso Dirigido por Modelos

Com os novos formulários, é possível estabelecer um novo processo de reúso de frameworks transversais. Na Seção 4.3.1 é descrito o quanto a engenharia de domínio

é impactada pelo processo, enquanto que na Seção 4.3.2 é descrito o impacto na engenharia de aplicação.

4.3.1 Engenharia de Domínio

Neste trabalho, além de prover a FFT, o engenheiro de domínio deve também prover outros dois artefatos: um diagrama de características (*features*) e um modelo de requisitos de reúso.

O diagrama de características é utilizado para visualizar a hierarquia de características e também é entrada do processo de derivação do membro da família de frameworks transversais. Para maior compatibilidade, recomenda-se que no caso de prover um FT isolado sem características a selecionar, um diagrama de características contendo apenas o nó raiz deve ser provido.

Com o processo proposto neste trabalho, em seguida ao desenvolvimento do framework e da definição do modelo de características, o engenheiro de framework precisa também prover um modelo de requisitos de reúso. Para criação desse modelo, um editor desenvolvido neste trabalho pode ser empregado, o qual é descrito com maiores detalhes na Seção 4.5.

4.3.2 Engenharia de Aplicação

De forma a explicar o novo processo de reúso, há um diagrama de atividades na Figura 27. Este diagrama ilustra a perspectiva de engenheiros reusando um framework transversal.

O processo de reúso se inicia no lado esquerdo da figura. A aplicação em desenvolvimento é composta pelos módulos “Base” e “Reúso”.

Durante a fase de análise (‘a’), o engenheiro de aplicação identifica os interesses de software. Por exemplo, isso pode ser realizado com o auxílio de um diagrama de análise (‘b’).

Ao identificar esses interesses, o engenheiro de aplicação tem a oportunidade de selecionar Frameworks Transversais a partir desse momento e iniciar o processo de reúso desde as fases iniciais do desenvolvimento do software (‘c’).

Após realizar a seleção de um Framework Transversal adequado para a aplicação corrente, esse engenheiro deve então selecionar um conjunto de características (*features*) do framework (‘d’). Um processo de derivação cria um membro da FFT a partir das características selecionadas. Também é instanciado (‘e’) um modelo de reúso

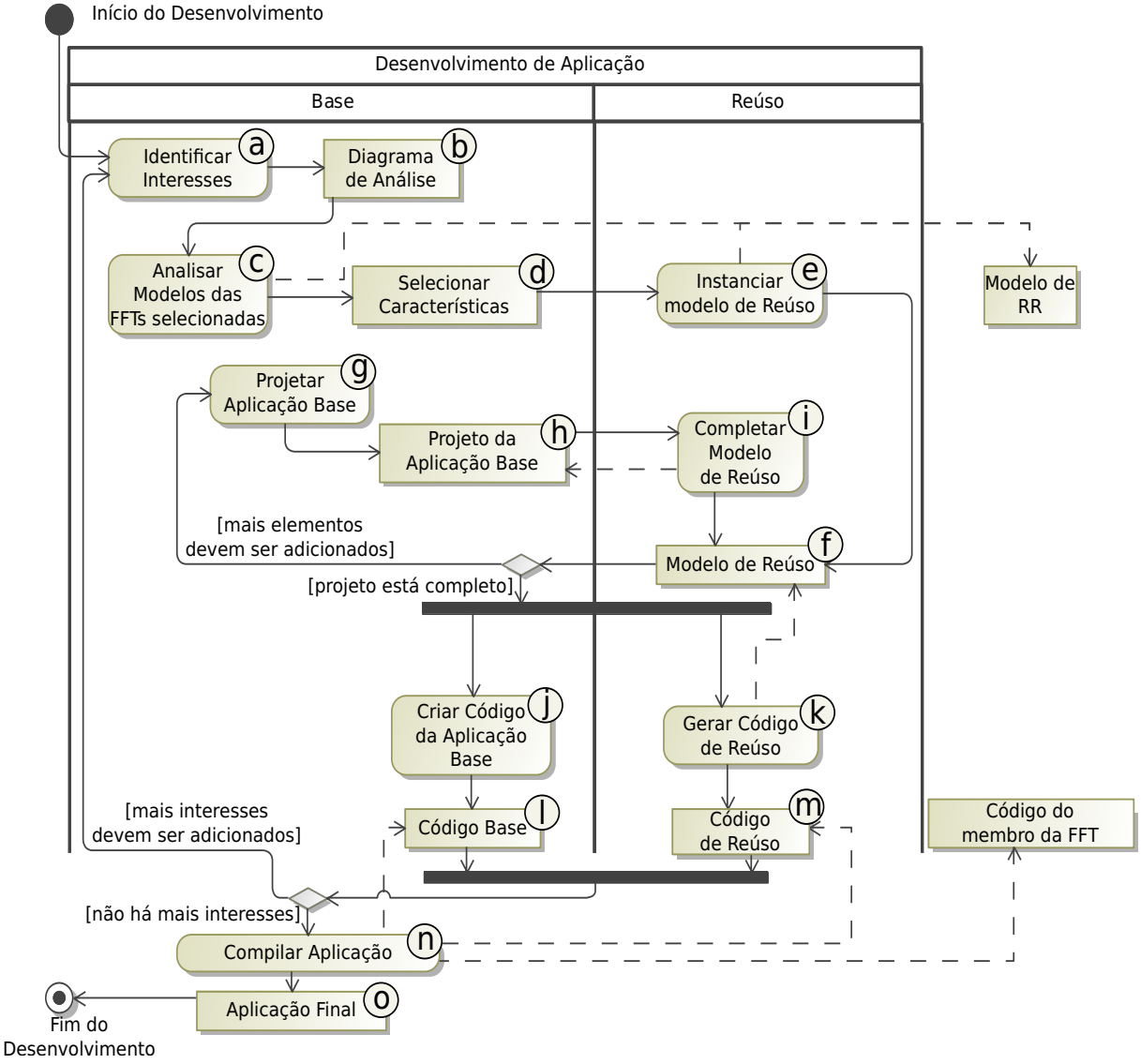

Figura 27: Diagrama de Atividades do Processo de Reúso

sem o preenchimento dos campos pertinentes à aplicação base (‘f’).

Esse engenheiro deve então projetar a aplicação base (‘g’). Com o projeto da aplicação base (‘h’), o código base já pode implementado (‘j’) (‘l’).

Além disso, a partir do momento em que o projeto da aplicação base estiver concluído, os nomes dos elementos da aplicação base referenciados pelo framework estarão disponíveis, portanto, neste momento já é possível especificar esses elementos no Modelo de Reúso , o que pode ser realizado graficamente ao utilizar o Formulário de Reúso .

Se o modelo de processo de desenvolvimento do software for iterativo, todos os elementos necessários pelo framework disponíveis durante a iteração devem ser especificados.

É importante apontar que os desenvolvimentos do código base e o código de reúso podem ser realizados em paralelo. Para concluir o Formulário de Reúso, os nomes de unidades, métodos e atributos da aplicação que são necessários para acoplar o framework à aplicação base são providos.

Assim que o Modelo de Reúso estiver completo, é possível executar um processo de geração de código (‘k’). Esse processo de geração de código é uma transformação de modelo para texto para gerar o “Código de Reúso” (‘m’) a partir de um “Modelo de Reúso” (‘f’).

Após completar o “Código da Aplicação Base” (‘l’) e o “Código de Reúso” (‘m’), o engenheiro de aplicação pode decidir entre adicionar um novo interesse à aplicação base, reusar um novo framework ou revisar o “Modelo de Reúso ” (‘f’) caso as alterações no projeto da aplicação base impactem as informações necessárias para acoplar os framework reusados. É importante lembrar que o código de reúso deverá ser gerado novamente se o “Modelo de Reúso” for alterado.

Após adicionar todos os interesses da aplicação base, o engenheiro de aplicação terá três tipos de códigos disponíveis:

1. Um “Código Base”;
2. *N* “Códigos de Reúso”;
3. *N* “Códigos de Frameworks” .

A letra *N* é definida como um número inteiro maior que zero. Dessa forma, é necessário um “Código de Reúso” para cada Framework Transversal reusado. Ao processar esses três tipos de códigos em um compilador, se torna possível a geração da “Aplicação Final” (‘h’), concluindo o processo e a aplicação.

O “Modelo de Reúso” contém informações que auxiliam o reúso do framework. Isso permite identificar informações necessárias para estabelecer o reúso antes de projetar a aplicação base. Isso facilita o projeto de uma aplicação base compatível com os frameworks selecionados para o reúso e cujas informações necessárias para o acoplamento sejam facilmente extraídas.

Consequentemente, a aplicação base não é totalmente inconsciente com relação aos frameworks reusados e seu comportamento. Porém, o código da aplicação base não depende do framework transversal.

O “Código de Reúso ” pode ser gerado antes de concluir o código da aplicação base. Porém, este código é dependente da aplicação base e do framework transversal em tempo de compilação.

Modularizar o código de reúso permite que o processo de geração seja repetido sem afetar o código da aplicação base ou do framework, o que não é possível em alguns trabalhos relacionados apresentados no Capítulo 3. Uma nova ferramenta foi implementada e é apresentada na Seção 4.4.

4.4 Ferramenta Proposta

Este trabalho envolve o desenvolvimento de um ambiente de desenvolvimento para apoiar o desenvolvimento e reúso de frameworks transversais. A parte da ferramenta destinada à engenharia de domínio é descrita na Subseção 4.4.1 e a parte específica da engenharia de aplicação é descrita na Subseção 4.5.

4.4.1 Engenharia de Domínio

Para auxiliar a aplicação do processo, foi desenvolvida uma ferramenta denominada CrossFIRE (DURELLI; GOTTARDI; CAMARGO, 2012a). Esta ferramenta inclui um repositório de Famílias de FTs que permite armazenar o código da família com seu modelo de características (*features*) da família e seu modelo de requisitos de reúso.

Vale ressaltar que os artefatos da Engenheira de Domínio devem estar desenvolvidos antes do engenheiro de software utilizar a CrossFIRE. Dessa forma, o engenheiro de domínio deve utilizar abordagens tradicionais (CAMARGO, 2006) para desenvolver os artefatos. Na fase de engenharia de domínio, dois principais artefatos devem ser desenvolvidos: o código-fonte da Família de FT e um modelo de característica da Família de FTs. O modelo de característica deve ser desenvolvido utilizando a FeatureIDE, que é a ferramenta que auxilia o desenvolvido de tais modelos. Além disso, o engenheiro de domínio deve também criar um Modelo de Requisitos de Reúso para a Família de FTs.

Com relação à engenharia de domínio, apenas a distribuição e armazenamento dos artefatos é apoiada. Na Figura 28 é apresentada uma visão geral da ferramenta CrossFIRE. Na Figura 28(A) é apresentado como o engenheiro de software pode armazenar uma determinada Família de FT. Dessa forma, o engenheiro de domínio pode fornecer algumas informações sobre a Família de FTs que almeja armazenar no repositório. No exemplo, o engenheiro está especificando o caminho do modelo de característica, um arquivo (*.xml*) criado pela ferramenta FeatureIDE, o nome da Família de FTs, o autor da Família de FTs e uma breve descrição sobre a Família de FTs. Após especificar todas as informações necessárias de uma determinada Família de FTs, o engenheiro de domínio deve clicar no botão “Upload”. Assim, o modelo de ca-

característica e todas as informações da Família de FT serão enviados e armazenados em um repositório remoto.

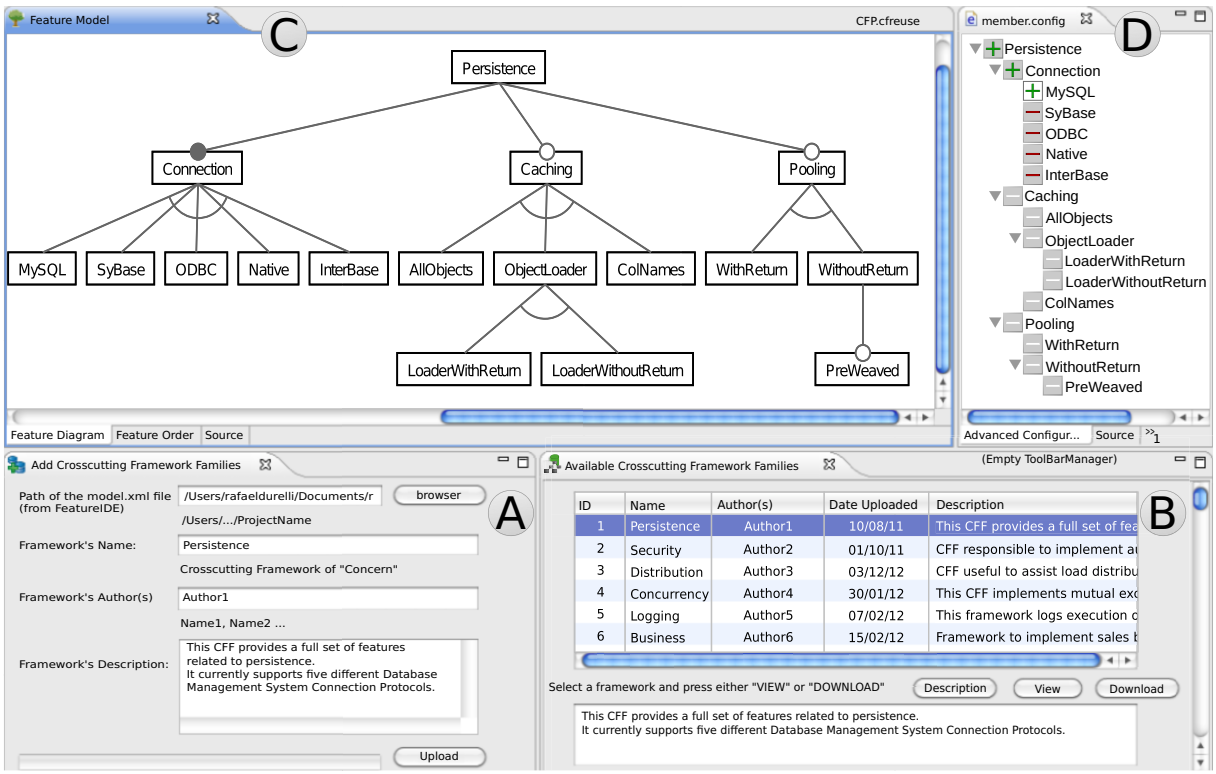

Figura 28: Visão geral da Ferramenta CrossFIRE

Conforme o processo descrito na Seção 4.3 (Figura 27), de forma a apoiar a atividade “Analisar Modelos das FFTs selecionadas” (‘c’), um engenheiro de aplicação pode utilizar a CrossFIRE para verificar se existe alguma Família de FTs que pode ser reutilizada em sua aplicação. Dessa forma, CrossFIRE fornece uma maneira de visualizar todos as Famílias de FTs disponíveis no repositório remoto, ou seja, todas as Famílias de FT que os engenheiros de domínio disponibilizaram.

Na Figura 28(B) é apresentada a visualização de todos as Famílias de FTs disponíveis para serem reutilizados. Nessa figura pode ser observado que existem seis Famílias de FTs: persistência, segurança, distribuição, concorrência, *logging* e regra de negócio, respectivamente. CrossFIRE permite que o engenheiro de aplicação visualize uma breve descrição de cada Família de FTs. No entanto, caso tal descrição não for suficiente para auxiliar o engenheiro de aplicação a decidir se a Família de FTs pode ser reutilizada, CrossFIRE fornece uma representação gráfica do modelo de característica de uma Família de FTs. Por exemplo, o modelo de característica ilustrado na Figura 28(C) representa a Família de FT selecionada na Figura 28(B), ou seja, Família de Persistência. Posteriormente, o botão “Download” deve ser clicado para realizar a transferência do modelo de características escolhido do repositório para o computador do engenheiro de aplicação.

Durante a atividade seguinte, denominada “Selecionar Características” (‘d’), o engenheiro de aplicação pode utilizar do apoio da ferramenta para visualizar uma árvore de características em forma gráfica, conforme mostrado na Figura 28(C). Com o uso de uma outra visualização desta árvore, presente na Figura 28(D), é possível selecionar um conjunto de características ao clicar nas caixas de seleção. Por exemplo, como mostrado na Figura 28(D), quando o engenheiro de aplicação seleciona uma determinada característica (representada por ‘+’), as variações e restrições são geradas automaticamente (representada por ‘-’). Com base nessa seleção, a ferramenta é capaz de instanciar código de um membro da Família de FTs e um modelo de reúso, ambos possuindo apenas elementos relacionados às características selecionadas.

Na Figura 29 é apresentada a arquitetura da ferramenta CrossFIRE. Essa ferramenta foi desenvolvida com base na plataforma Eclipse (ECLIPSE CONSORTIUM, 2012), mais especificamente, como um *plug-in* para o IDE Eclipse. Além disso, o código-fonte da FeatureIDE (KASTNER et al., 2009) foi estudado e estendido para auxiliar a criação e visualização de forma gráfica dos modelos de características pertencentes a Famílias de FT.

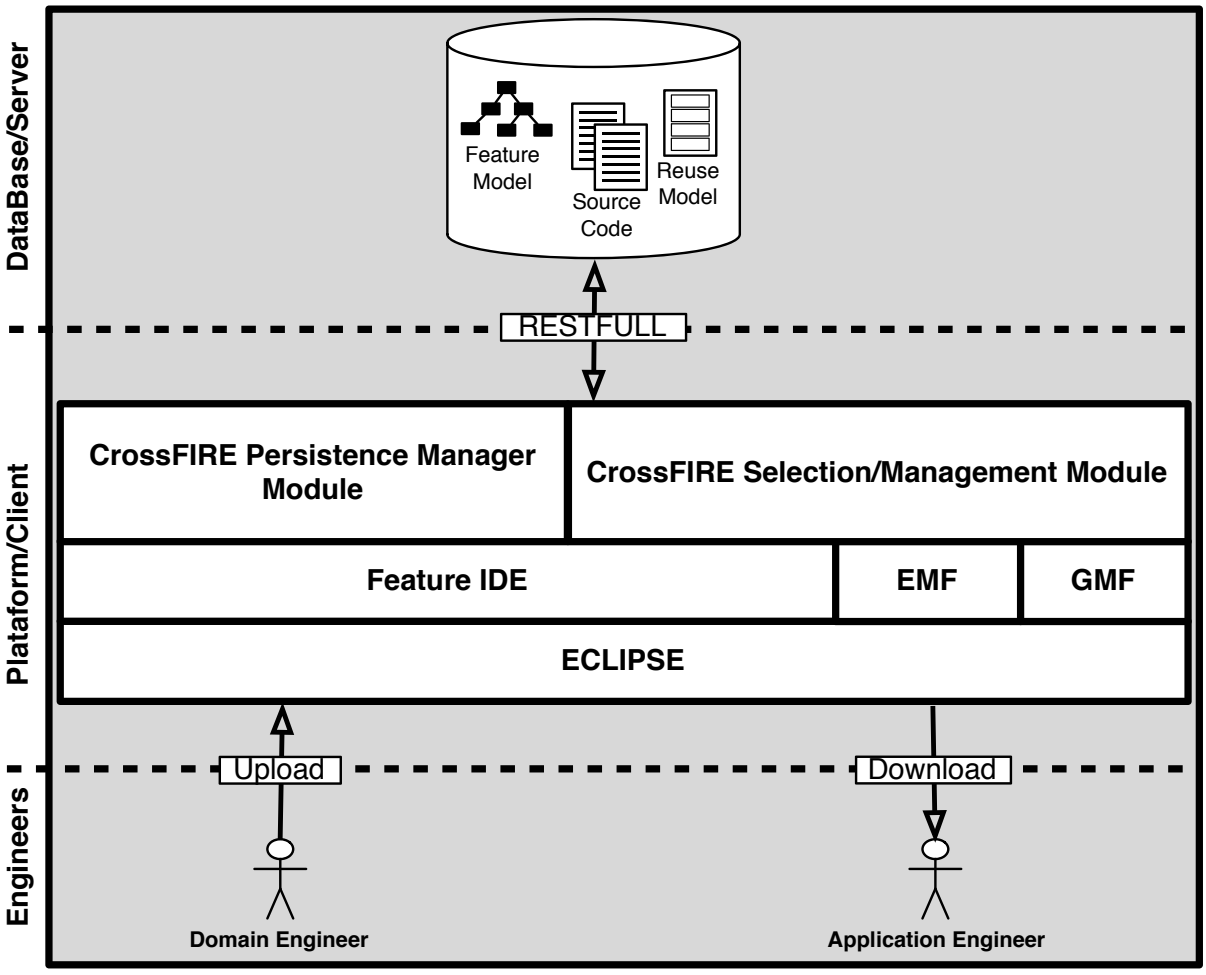

Figura 29: Arquitetura da Ferramenta CrossFIRE

Com pode ser observado na Figura 29, todos os artefatos (código-fonte, modelo de características, modelo de reúso e modelo de requisitos de reúso) que a CrossFIRE apoia são persistidos em um servidor remoto, os quais são disponíveis para serem reutilizados durante a fase de Engenharia de Aplicação. Este servidor é dedicado para executar solicitações RESTful (JSR 224 EXPERT GROUP, 2009). Portanto, todos os artefatos são enviados e recebidos por meio da API RESTful. Java Persistence API (JPA) 2.0 (JSR 317 EXPERT GROUP, 2009) e o banco de dados MySQL (ORACLE CORPORATION, 2012) foram utilizados para realizar as persistências dos artefatos.

Para garantir a correta geração do modelo de reúso pelo servidor de repositório de frameworks transversais, o engenheiro de domínio deve criar modelos de requisitos de reúso cujo nó raiz possua o mesmo nome do nó raiz do modelo de características do framework. Com o uso de elementos de grupo, a estrutura do modelo de características deve ser seguida. Cada grupo deve ter o mesmo nome que a característica que representa. Grupos vazios podem ser omitidos sem acarretar problemas na derivação do modelo de reúso.

Ao editar um modelo de requisitos de reúso, o engenheiro de domínio pode utilizar a paleta do editor, que é visível à direita da Figura 30, para adicionar objetos ao formulário. Os objetos possuem propriedades omitidas no formulário. Para editá-los é necessário utilizar a visão “Properties”, visível na parte inferior da Figura 30. Nesta figura é exemplificada a edição da propriedade “Name”.

Para cada conjunto de junção a ser sobrescrito durante o reúso, o engenheiro de framework deverá modelar um elemento do tipo *Pointcut* com as seguintes propriedades:

- *name*: Nome do requisito de reúso;
- *description*: Descrição do requisito de reúso;
- *visibility*: deve ser compatível com a visibilidade do conjunto de junção a redefinir de acordo com as regras de AspectJ;
- *qualifiedOwnerName*: Nome completo do aspecto ou classe em que o conjunto de junção está presente no padrão “*pacote.Classe*”. Deixar em branco caso o conjunto de junção abstrato pertencer a uma interface;
- *elementName*: Nome da classe a ser estendida;
- *superType*: tipo de unidade a ser estendida, valores válidos são: “aspecto”, “classe” ou “interface”;

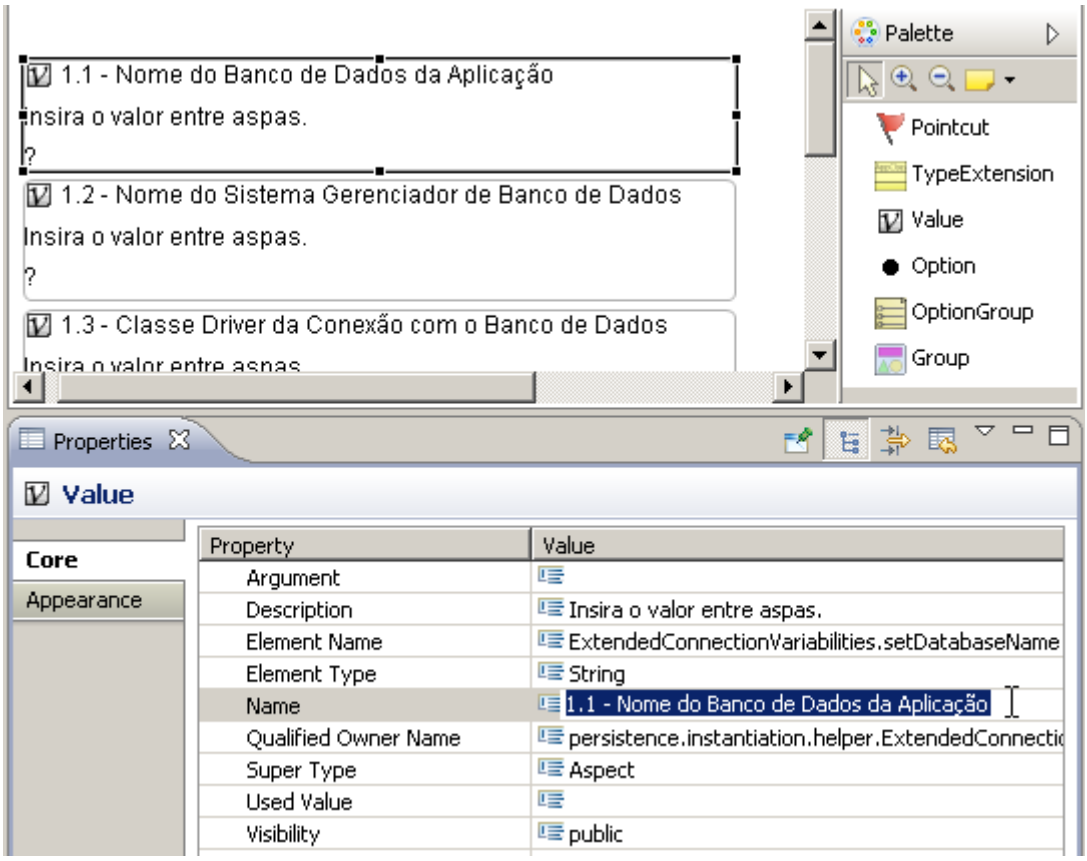

Figura 30: Edição de Modelos de Requisito de Reúso e Suas Propriedades

- *argument*: deve ser deixado para o engenheiro de aplicação preencher parâmetros do conjunto de junção (opcional);
- *internalDefinition*: definição de um conjunto de junção composto que agrupa conjuntos de junção iteráveis (*iterableDefinition*): exemplo: *!this(Classe) && (%s)*, em que "%s" é substituído pelos conjuntos de junção iteráveis;
- *iterableDefinition*: definição de conjuntos de junção iteráveis, que são colocados dentro da definição interna. Exemplo: o uso de *call(%s)* com a definição interna exemplificada no item anterior permite a criação de conjuntos de junção como: *!this(Classe) && (call(classe.Metodo1() || call(classe.Metodo2())))*;
- *composition*: utilizado para definir o operador de composição lógica entre os conjuntos de junção iteráveis. Portanto, no exemplo anterior, "or" foi empregado, porém, semanticamente "and" também seria permitido.
- *definitions*: deve ser deixado para ser preenchido pelo engenheiro de aplicação. Essa é a propriedade editável diretamente ao visualizar o modelo de reúso como um formulário;

Para toda unidade do framework a ser estendida durante o reúso, o engenheiro

de framework deverá modelar um elemento do tipo *TypeExtension* com as seguintes propriedades:

- *name*: Nome do requisito de reúso;
- *description*: Descrição do requisito de reúso;
- *visibility*: Deve ser compatível com a visibilidade do método estendido, similarmente como é feito ao estender conjuntos de junção;
- *qualifiedOwnerName*: Nome completo da unidade proprietária do método a ser estendido, como em "*pacote.Classe*";
- *elementName*: Nome do método a ser estendido;
- *superType*: utilizado para indicar se a unidade proprietária deve ser estendida por um "aspecto", "classe" ou "interface";
- *elementType*: Apenas empregado em herança múltipla. No caso de AspectJ, neste campo deve ser colocado o nome das interfaces a implementar. Isso só é valido para extensão com classes e aspectos;
- *definitions*: deve ser deixado para ser preenchido pelo engenheiro de aplicação. Essa é a propriedade editável diretamente ao visualizar o modelo de reúso como um formulário.

Para todo valor constante (em tempo de compilação) que deve ser especificado durante o reúso, o engenheiro de framework deverá modelar um elemento do tipo *Value ou Option* com as seguintes propriedades:

- *name*: Nome do requisito de reúso;
- *description*: Descrição do requisito de reúso;
- *visibility*: Compatível com o método a sobreescrever: "public", "protected" ou "package";
- *qualifiedOwnerName*: Nome completo do pacote ou classe proprietária da unidade a ser estendida, como em "*pacote.subPacote*";
- *elementName*: Nome da classe a ser estendida;
- *superType*: utilizado para indicar se a unidade proprietária deve ser estendida por um "aspecto", "classe" ou "interface";

- *elementType*: Nome textual do tipo de dado. Exemplo: “java.lang.String”. No caso de option, apenas tipos booleanos são válidos para o código final, como “boolean” e “java.lang.Boolean”;
- *usedValue* (apenas em Value): deve ser deixado para ser preenchido pelo engenheiro de aplicação. Essa é a propriedade editável diretamente ao visualizar o modelo de reúso como um formulário. Qualquer tipo de valor pode ser informado com esse tipo de objeto de modelo;
- *enabled* (apenas em Option): deve ser deixado para ser marcado ou não pelo engenheiro de aplicação. Dessa forma, apenas valores booleanos são usáveis.

Os outros atributos não são empregados no gerador de código de reúso em AspectJ. A definição completa das propriedades está descrita no Apêndice A.

Neste ponto conclui-se o suporte destinado à engenharia de domínio, o apoio ferramental é continuado na Subseção 4.5.

4.5 Engenharia de Aplicação

De forma a apoiar a tarefa do engenheiro de aplicação, após instanciar o membro da linha de produtos de frameworks e o modelo de reúso, foi implementado um editor gráfico para edição de modelos de reúso e requisitos de reúso.

Esse editor pode ser utilizado para completar o modelo de reúso instanciado pelo servidor de repositório. Conforme ilustrado na Figura 31, a última linha de texto das caixas visíveis nos diagramas podem ser clicadas pelo engenheiro de aplicação. Isto ativa um campo para que este engenheiro complete a definição necessária ao reúso.

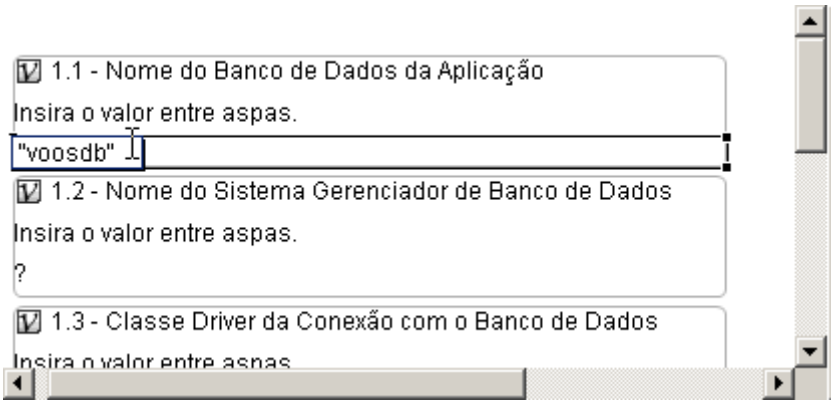

Figura 31: Edição de Modelo de Reúso

Foi implementado um gerador de código com validação do modelo de entrada e

capaz de validar e verificar os itens informados ao preencher o modelo. Conforme ilustrado na Figura 32, gerador de código é estruturado da seguinte forma:

1. Artefato de entrada: o artefato de entrada é uma instância do metamodelo CFReuse em arquivo que deve representar um modelo de reúso completo. Maiores detalhes sobre o metamodelo estão descritos no Apêndice A;

2. API XTend: a api XTend (ECLIPSE CONSORTIUM, 2010) é empregada para decodificar o arquivo em que está escrito o modelo e transformá-lo em uma árvore que representa o mesmo modelo em memória;

3. CFReuse em Memória: representação em memória do modelo de reúso. Cada nó é uma instância de uma classe que representa uma metaclasse do metamodelo CFReuse;

4. Gerador de Código de Reúso: O gerador de código de reúso transforma a árvore do modelo de reúso para uma outra árvore em memória. Essa nova árvore é uma árvore sintática que representa as classes, aspectos e métodos do código de reúso;

5. AJGenerator: Durante este trabalho foi implementada uma API que transforma uma árvore sintática que representa códigos AspectJ e Java em código fonte. Dessa forma, essa API transforma o código de reúso em memória para arquivos.

6. Artefato de Saída: Código de Reúso: É o artefato final do processo de transformação, que é utilizado para acoplar o framework transversal à aplicação base.

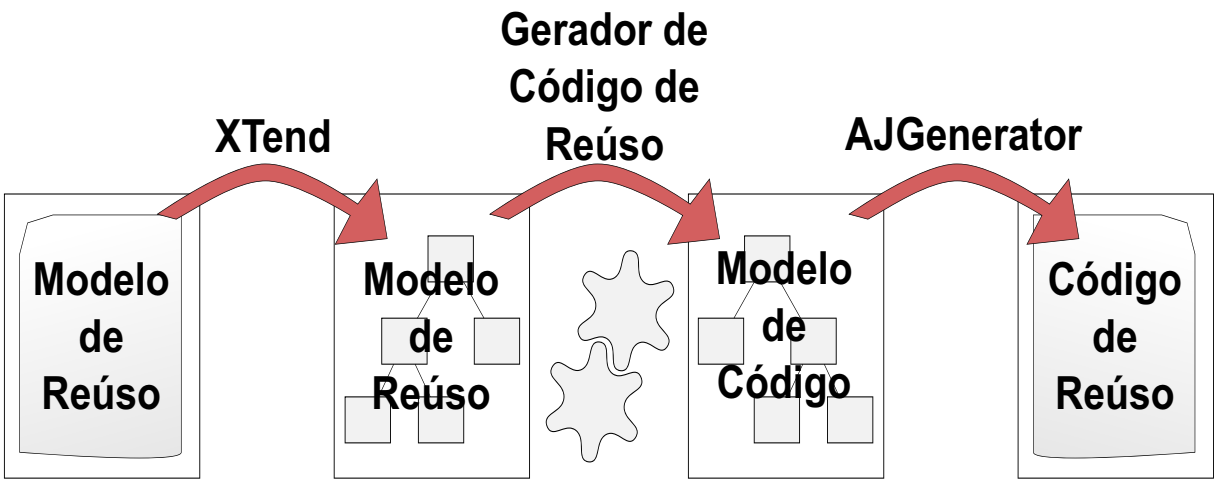

Figura 32: Camadas do Processo de Geração de Código

A separação do processo em camadas permite um nível maior de abstração. Dessa forma, é possível alterar a linguagem do código gerado para uma linguagem

que possua conceitos similares a AspectJ (exemplo: herança de pointcuts) ao substituir o AJGenerator.

O gerador de código realiza uma validação do modelo de reúso antes de gerar código. São verificados se os dados informados estão sintaticamente corretos. Esta validação detecta identificadores de métodos e classes inválidos, *Strings* sem aspas duplas, *Chars* não limitados por aspas simples, valores inteiros ou valores numéricos fracionários escritos de forma incorreta. O tipo da validação depende do tipo de informação especificada nos requisitos de reúso do modelo de entrada.

O gerador de código de reúso também é capaz de gerar outro código, chamado de Código de Verificação. O Código de Verificação também é gerado por meio do AJGenerator e é executado no ambiente da ferramenta proposta.

O Código de Verificação utiliza de reflexão e é executado no contexto da aplicação final para verificar se as classes e os métodos da aplicação base referenciados pelo modelo de reúso realmente estão presentes na aplicação base.

4.6 Considerações Finais

Com os modelos e a ferramenta proposta neste capítulo, é definido um novo processo de reúso. No Capítulo 5, este processo de reúso é exemplificado com as perspectivas de engenharia de domínio e aplicação.

Capítulo 5

EXEMPLO DE USO DA ABORDAGEM

5.1 Considerações Iniciais

Neste capítulo é apresentado um exemplo de uso da abordagem. Conforme descrito na Seção 5.2, um engenheiro de domínio deve criar um modelo de características e um modelo de requisitos de reúso para um framework transversal. Na Seção 5.2.1, um engenheiro de aplicação utiliza os modelos providos para criar um modelo de reúso, o que é utilizado para apoiar o reúso do framework e gerar o código de reúso, concluíndo o processo.

5.2 Engenharia de Domínio

Neste exemplo, ao realizar a atividade de um engenheiro de domínio, é necessário estabelecer um modelo de características e um modelo de requisitos de reúso que são utilizados por um engenheiro de aplicação interessado em reusar o framework transversal.

O modelo de características criado está visível na Figura 33. Com base nesse modelo, o engenheiro de domínio deve criar um modelo de requisitos de reúso que possua grupos em conformidade com a hierarquia de características.

O modelo de requisitos de reúso criado ao identificar as informações necessárias para reusar este FT está representado na Figura 34.

Esse modelo foi criado considerando que as seguintes informações devem ser fornecidas:

1. Valores relativos à conexão ao banco de dados:

   (a) Nome do Banco de Dados – Um objeto do tipo “Value” é criado com as seguintes propriedades:

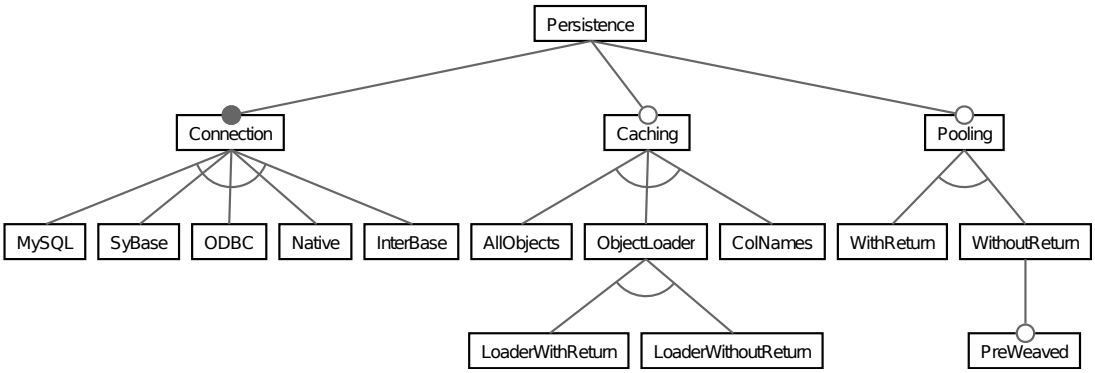

Figura 33: Árvore de Características do FT de Persistência

1.1 - Nome do Banco de Dados da Aplicação
Insira o valor entre aspas.

1.2 - Nome do Sistema Gerenciador de Banco de D...
Insira o valor entre aspas.

1.3 - Classe Driver da Conexão com o Banco de Da...
Insira o valor entre aspas.

1.4 - Protocolo da Conexão com o Banco de Dados
Insira o valor entre aspas.

2.1 - Pontos em que a Conexão Deve ser Aberta
Insira nome de método no estilo "pacote.Classe.metodo".

2.2 - Pontos em que a Conexão deve ser Fechada
Insira nome de método no estilo "pacote.Classe.metodo".

3.1 - Classes que Representam Objetos Persistentes
Insira nome de classes no padrão pacote.Classe

Figura 34: Modelo de requisitos Reúso para o FT

- Nome – (*name*) =
  "1.1 - Nome do Banco de Dados da Aplicação";
- Descrição – (*description*) =
  "Insira o valor entre aspas.";
- Nome da unidade a ser estendida – (*qualifiedOwnerName*) =

"persistence.instantiation.helper.ExtendedConnectionVariabilities";

- Nome do método a ser sobrescrito – (*elementName*) = "ExtendedConnectionVariabilities.setDatabaseName";
- Tipo de dados da constante – (*elementType*) = "String";
- Tipo de unidade a ser criada – (*superType*) = "Class";

(b) Nome do Sistema Gerenciador de Banco de Dados – Um objeto do tipo "Value" é criado com as seguintes propriedades:

- Nome – (*name*) = "1.2 - Nome do Sistema Gerenciador de Banco de Dados";
- Descrição – (*description*) = "Insira o valor entre aspas.";
- Nome da unidade a ser estendida – (*qualifiedOwnerName*) = "persistence.instantiation.helper.ExtendedConnectionVariabilities";
- Nome do método a ser sobrescrito – (*elementName*) = "ExtendedConnectionVariabilities.setSpecificDatabase";
- Tipo de dados da constante – (*elementType*) = "String";
- Tipo de unidade a ser criada – (*superType*) = "Class";

(c) Nome do Sistema Gerenciador de Banco de Dados – Um objeto do tipo "Value" é criado com as seguintes propriedades:

- Nome – (*name*) = "1.3 - Classe Driver da Conexão com o Banco de Dados";
- Descrição – (*description*) = "Insira o valor entre aspas.";
- Nome da unidade a ser estendida – (*qualifiedOwnerName*) = "persistence.instantiation.helper.ExtendedConnectionVariabilities";
- Nome do método a ser sobrescrito – (*elementName*) = "ExtendedConnectionVariabilities.getDriver";
- Tipo de dados da constante – (*elementType*) = "String";
- Tipo de unidade a ser criada – (*superType*) = "Class";

(d) Nome do Sistema Gerenciador de Banco de Dados – Um objeto do tipo "Value" é criado com as seguintes propriedades:

- Nome – (*name*) =
  "1.4 - Protocolo da Conexão com o Banco de Dados";
- Descrição – (*description*) =
  "Insira o valor entre aspas.";
- Nome da unidade a ser estendida – (*qualifiedOwnerName*) =
  "persistence.instantiation.helper.ExtendedConnectionVariabilities";
- Nome do método a ser sobrescrito – (*elementName*) =
  "ExtendedConnectionVariabilities.getJDBC";
- Tipo de dados da constante – (*elementType*) =
  "String";
- Tipo de unidade a ser criada – (*superType*) =
  "Class";

2. Definição de conjuntos de junção:

   (a) Pontos de abertura de conexão – Um objeto do tipo “Pointcut” é criado com as seguintes propriedades:

   - Nome – (*name*) =
     "2.1 - Pontos em que a Conexão Deve ser Aberta";
   - Descrição – (*description*) =
     "Insira nome de método no estilo "pacote.Classe.metodo";.";
   - Nome da unidade a ser estendida – (*qualifiedOwnerName*) =
     "persistence.instantiation.helper.ExtendedConnectionCompositionRules";
   - Nome do conjunto de junção a ser sobrescrito – (*elementName*) =
     "openConnection";
   - Operador de composição – (*composition*) =
     "or";
   - Definição interna – (*internalDefinition*) =
     ;
   - Definição iterável – (*iterableDefinition*) =
     "execution (* %s(..))";
   - Tipo de unidade a ser criada – (*superType*) =
     "Class";

   (b) Pontos de fechamento de conexão – Um objeto do tipo “Pointcut” é criado com as seguintes propriedades:

   - Nome – (*name*) =
     "2.2 - Pontos em que a Conexão Deve ser Fechada";

- Descrição – (*description*) =
  "Insira nome de método no estilo "pacote.Classe.metodo";.";
- Nome da unidade a ser estendida – (*qualifiedOwnerName*) =
  "persistence.instantiation.helper.ExtendedConnectionCompositionRules";
- Nome do conjunto de junção a ser sobrescrito – (*elementName*) =
  "closeConnection";
- Operador de composição – (*composition*) =
  "or";
- Definição interna – (*internalDefinition*) =
  ;
- Definição iterável – (*iterableDefinition*) =
  "execution (* %s(..))";
- Tipo de unidade a ser criada – (*superType*) =
  "Class";

3. Extensão de unidades:

   (a) Pontos de fechamento de conexão – Um objeto do tipo “Pointcut” é criado com as seguintes propriedades:

   - Nome – (*name*) =
     "3.1 - Classes que Representam Objetos Persistentes";
   - Descrição – (*description*) =
     "Insira nome de classes no padrão pacote.Classe".";
   - Nome da unidade a ser estendida – (*qualifiedOwnerName*) =
     "persistence.PersistentRoot";
   - Tipo de unidade a ser extendida – (*superType*) =
     "Interface";

Este modelo de requisitos de reúso é então provido em conjunto com o modelo de características para ser analisado durante a engenharia de aplicação, o que é descrito na Seção 5.2.1.

5.2.1 Engenharia de Aplicação

Nessa seção é exemplificada uma aplicação base denominada “Aplicação Sistema Gerenciamento de Passagens Aéreas” que deve ser acoplada ao framework transversal de persistência (CAMARGO; MASIERO, 2005).

Como atividade do engenheiro de aplicação, o diagrama de características do framework é analisado. Ao analisar as características disponíveis, as características adequadas para a aplicação base devem ser selecionadas. Isso é realizado ao editar o arquivo de configuração. Na Figura 35 é mostrada a edição de um arquivo de configuração em um editor gráfico. As caixas marcadas com "+" são aquelas que foram selecionadas e as caixas marcadas com "-" não foram selecionadas.

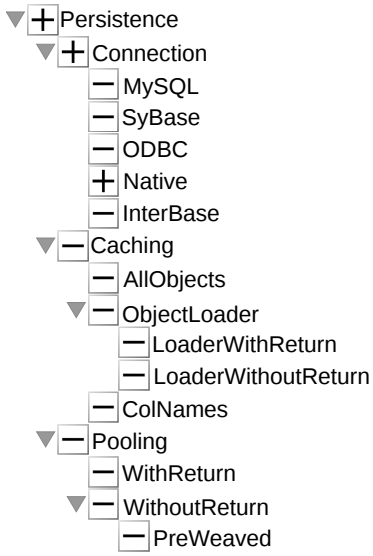

Figura 35: Arquivo de Configuração para Seleção de Características

Após executar a validação do arquivo de configuração e obter a instância da linha de produtos de frameworks em conjunto com o modelo de reúso, resta realizar a composição do código base com o framework instanciado com o apoio do modelo de reúso.

Com os requisitos selecionados por meio da instanciação de um modelo de reúso, o engenheiro de aplicação obtêm o conhecimento da necessidade de completar três campos de valores (*Value*), dois campos de conjuntos de junção (*Pointcut*) e uma extensão de tipos (*TypeExtension*).

Ao projetar a aplicação base e analisar os itens requeridos, foram identificadas as informações necessárias para completar o modelo de reúso:

1. Valores relativos à conexão ao banco de dados:
   - Nome do Banco de Dados:
     - "voosdb";
   - Nome do Sistema Gerenciador de Banco de Dados:
     - "derby";

- Nome do Sistema Gerenciador de Banco de Dados:
  - “org.apache.derby.jdbc.EmbeddedDriver”;
- Nome do Sistema Gerenciador de Banco de Dados:
  - “jdbc:derby:”;

2. Definição de conjuntos de junção:
   - Pontos de abertura de conexão:
     - Classe **Aeronave**
       - Método **pouso**;
     - Classe **Aeroporto**
       - Método **abertura**;
     - Classe **Bagagem**
       - Método **deposito**;
     - Classe **Checkin**
       - Método **confirma**;
     - Classe **Passageiro**
       - Método **embarca**;
     - Classe **Voo**
       - Método **partida**.
   - Pontos de fechamento de conexão:
     - Classe **Aeronave**
       - Método **decolagem**;
     - Classe **Aeroporto**
       - Método **fechamento**;
     - Classe **Bagagem**
       - Método **retirada**;
     - Classe **Checkin**
       - Método **cancela**;
     - Classe **Passageiro**
       - Método **desembarca**;
     - Classe **Voo**
       - Método **chegada**.
3. Extensão de unidades:
   - Pontos de fechamento de conexão:
     - Classe **Aeronave**;
     - Classe **Aeroporto**;
     - Classe **Bagagem**;
     - Classe **Checkin**;
     - Classe **Passageiro**;

– Classe **Voo**.

Com essas informações, é possível completar o modelo de reúso. O modelo de reúso resultante será similar ao da Figura 36. Por meio da geração de código, o código de reúso necessário para acoplar a aplicação ao framework é criado. Neste exemplo, o código de reúso é composta de três unidades.

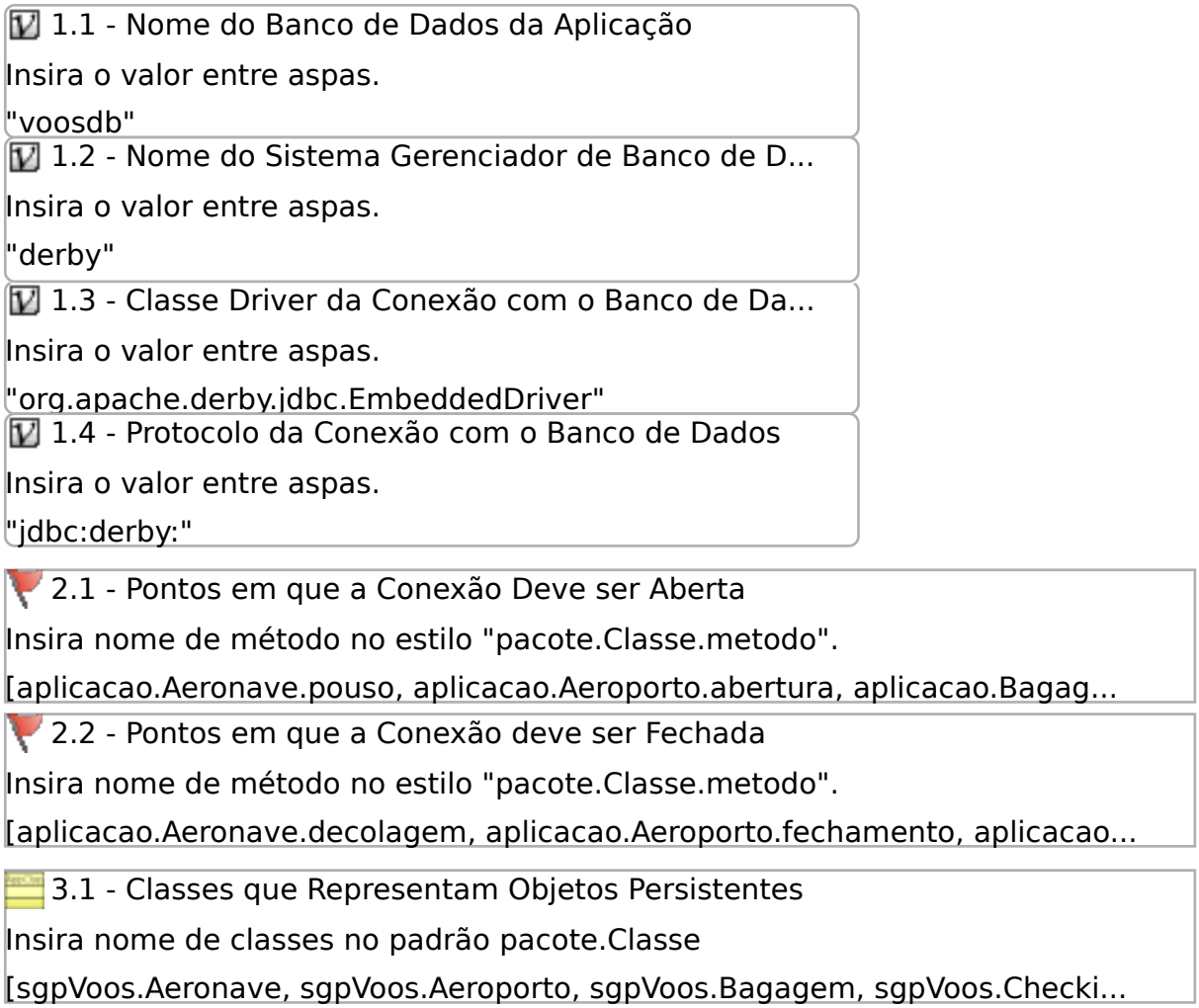

Figura 36: Modelo de Reúso para a Aplicação “Voos”

A primeira unidade criada é um aspecto que realiza sobrescrita de métodos para definir os valores constantes necessários durante o reúso. Dessa forma, na Figura 37, são implementados os valores passados nos objetos do tipo valor do modelo de reúso. É importante notar que os valores foram inseridos em uma mesma unidade porque foram definidos para estender a mesma superclasse.

Na Figura 38, é mostrado um aspecto que foi gerado para implementar os conjuntos de junção da aplicação. Dessa forma, os métodos de abertura de conexão foram mapeados no conjunto de junção *openConnection* e os métodos de fechamento de conexão são mapeados no conjunto de junção *closeConnection*. Estes conjuntos

```
                ConcreteExtendedConnectionVariabilities0.aj

package persistence.reuse;
import persistence.instantiation.helper.ExtendedConnectionVariabilities;
public aspect ConcreteExtendedConnectionVariabilities0 extends
 ExtendedConnectionVariabilities {
    public String ExtendedConnectionVariabilities.setDatabaseName (){
        return "voosdb";
    }
    public String ExtendedConnectionVariabilities.setSpecificDatabase (){
        return "derby";
    }
    public String ExtendedConnectionVariabilities.getDriver (){
        return "org.apache.derby.jdbc.EmbeddedDriver";
    }
    public String ExtendedConnectionVariabilities.getJDBC (){
        return "jdbc:derby:";
    }
}
```

Figura 37: Definição de Valores Constantes para a Aplicação “Voos”

de junção foram inseridos no mesmo aspecto porque foi definido no modelo que eles compartilham o mesmo aspecto abstrato.

```
              ConcreteExtendedConnectionCompositionRules1.aj

package persistence.reuse;
import persistence.instantiation.helper.ExtendedConnectionCompositionRules;
public aspect ConcreteExtendedConnectionCompositionRules1 extends
    ExtendedConnectionCompositionRules {
    public pointcut openConnection ():
        (
                execution (* aplicacao.Aeronave.pouso(..)) ||
                execution (* aplicacao.Aeroporto.abertura(..)) ||
                execution (* aplicacao.Bagagem.deposito(..)) ||
                execution (* aplicacao.Checkin.confirma(..)) ||
                execution (* aplicacao.Passageiro.embarca(..)) ||
                execution (* aplicacao.Voo.partida(..))
        );
    public pointcut closeConnection ():
        (
                execution (* aplicacao.Aeronave.decolagem(..)) ||
                execution (* aplicacao.Aeroporto.fechamento(..)) ||
                execution (* aplicacao.Bagagem.retirada(..)) ||
                execution (* aplicacao.Checkin.cancela(..)) ||
                execution (* aplicacao.Passageiro.desembarca(..)) ||
                execution (* aplicacao.Voo.chegada(..))
        );
}
```

Figura 38: Especificação de Conjuntos de Junção para a Aplicação “Voos”

Na Figura 39, é mostrado outro aspecto. Esse aspecto foi gerado para realizar a definição das extensões de tipo. As extensões de tipo são, por padrão, unificadas em um único aspecto que faz uso de “Declare Parents” para definir que as classes ou interfaces informadas pelo engenheiro de aplicação estendem as unidades do framework.

Concluída a geração do código de reúso, opcionalmente, o engenheiro de apli-

ParentsDeclaration.aj

```
package persistence.reuse;
public aspect ParentsDeclaration extends Object {
    declare parents: sgpVoos.Aeronave implements
        persistence.PersistentRoot;
    declare parents: sgpVoos.Aeroporto implements
        persistence.PersistentRoot;
    declare parents: sgpVoos.Bagagem implements
        persistence.PersistentRoot;
    declare parents: sgpVoos.Checkin implements
        persistence.PersistentRoot;
    declare parents: sgpVoos.Passageiro implements
        persistence.PersistentRoot;
    declare parents: sgpVoos.Voo implements
        persistence.PersistentRoot;
}
```

Figura 39: Definição de Valores Constantes para a Aplicação “Voos”

cação pode executar o código de verificação para verificar se os itens referenciados estão presentes na aplicação base.

Com o código de reúso definido corretamente, a aplicação base está corretamente acoplada ao framework transversal. Isso finaliza o processo de reúso.

5.3 Considerações Finais

Foi exemplificado um uso da abordagem neste capítulo. No Capítulo 6, este processo de reúso é avaliado, em termos de produtividade, ao reusar frameworks transversais e manter aplicações acopladas com esses frameworks.

Capítulo 6

AVALIAÇÃO DA ABORDAGEM

6.1 Considerações Iniciais

Neste capítulo são apresentados dois experimentos para avaliar a abordagem proposta. Em ambos os experimentos é considerada a produtividade, por meio da métrica de tempo utilizado, comparando a abordagem com o processo convencional de edição de código. O primeiro experimento considera a produtividade durante o reúso de um framework transversal, o que é apresentado na Seção 6.2. O segundo experimento considera a produtividade durante a manutenção de uma aplicação acoplada a um framework transversal, o que é apresentado na Seção 6.3. Por fim, na Seção 6.4 são apresentadas ameaças à validade dos experimentos.

6.2 Experimento de Reúso

Foram conduzidos estudos empíricos para comparar as diferenças entre o uso da ferramenta e o processo convencional quando utilizados para reusar um framework transversal em termos de esforço. O planejamento dos estudos foram realizados considerando as diretrizes propostas por Wohlin et al. (2000).

6.2.1 Definição do Estudo Empírico

6.2.1.1 Objetivo

O objetivo do estudo é comparar o esforço de reusar frameworks transversais em duas formas distintas de reúso: o processo convencional e o processo dirigido a modelos.

6.2.1.2 Sujeito do Estudo

Um framework transversal foi considerado para ser reusado nas duas técnicas de reúso.

6.2.1.3 Enfoque Quantitativo

O enfoque quantitativo envolve a identificação de esforço com base nos tempos tomados por cada participante para concluir cada processo de reúso.

6.2.1.4 Enfoque Qualitativo

O enfoque qualitativo é usado para determinar a técnica que exige menos esforço.

6.2.1.5 Perspectiva

Os experimentos foram conduzidos da perspectiva de engenheiros de aplicação que possuam interesse em reusar frameworks transversais.

6.2.1.6 Objeto de Estudo

O objeto dos estudos deste capítulo é o “esforço” para concluir um processo de reúso de framework transversal.

6.2.2 Planejamento do Estudo

Os experimentos foram planejados considerando a seguinte questão: “Qual técnica de reúso exige menos esforço para reusar um FT com sucesso?”; Essa questão é então analisada no término da seção para avaliar se ela foi respondida com sucesso.

Foi solicitado aos participantes para reusar o framework com distintas aplicações e com ambas as técnicas. Os seguintes dados pertinentes a este estudo foram coletados e analisados:

- Técnica de Reúso empregada;
- Aplicação Base;
- Data e Hora do Início de cada Execução;

- Data e Hora do Término de cada Execução;
- Tipo de Execução.

O item “Técnica de Reúso empregada” é utilizado para definir se o participante do estudo utilizou a técnica dirigida a modelos ou o processo convencional durante um processo de reúso. O item “Aplicação Base” é utilizado para identificar qual aplicação base está sendo acoplada ao framework. Os item “Data e Hora do Início de cada Execução” e “Data e Hora do Término de cada Execução” são usados para registrar o momento de início e fim de cada execução, valores considerados para calcular o tempo tomado em cada processo. “Tipo de Execução” é a finalidade de cada execução da aplicação final, que pode ser caso de teste, geração de código, validação ou registro de tempo inicial.

Um sistema de informação foi empregado para capturar os dados especificados. Foi adicionado código nas aplicações a serem acopladas ao framework para enviar os dados do experimento para um servidor que acumulou os dados de todos os experimentos em uma base de dados.

Todos os dados, com exceção das execuções de “registro de tempo inicial” foram coletadas de forma transparente sem exigir esforço ou preocupação dos participantes.

6.2.2.1 Seleção de Contexto

Este estudo foi conduzido com alunos de Ciência da Computação que são aqui chamados de participantes. Dezesseis alunos participaram do experimento. Os primeiros oito (números de um a oito) eram alunos de pós-graduação e os outros oito (números de nove a dezesseis) eram alunos de graduação.

Estes participantes foram selecionados por possuírem experiencia prévia em AspectJ e reúso convencional de frameworks transversais. Durante o experimento, os participantes tiveram de reusar um framework transversal de persistência (CAMARGO; MASIERO, 2008), compondo-o a uma aplicação base. Cada processo de reúso foi efetuado utilizando ou da técnica convencional ou da técnica dirigida por modelos.

6.2.2.2 Formulação de Hipóteses

Na Tabela 2 estão listadas as hipóteses para o experimento de reúso. As hipóteses foram criadas para comparar a produtividade da abordagem dirigida a modelos com a forma convencional durante o reúso de frameworks transversais.

Tabela 2: Hipóteses para o Estudo de Reúso

$H0_r$	Não há diferença entre usar o processo dirigido por modelos e utilizar um processo convencional "ad-hoc" para concluir um processo de reúso com sucesso em termos de produtividade (tempo). Portanto, as técnicas são equivalentes. Tc_r - $Tm_r \approx 0$
Hp_r	Há uma diferença positiva entre usar o processo dirigido por modelos e utilizar um processo convencional "ad-hoc" para concluir um processo de reúso com sucesso em termos de produtividade (tempo). Portanto, a técnica convencional leva mais tempo do que a técnica dirigida por modelos. Tc_r - $Tm_r > 0$
Hp_r	Há uma diferença negativa entre usar o processo dirigido por modelos e utilizar um processo convencional "ad-hoc" para concluir um processo de reúso com sucesso em termos de produtividade (tempo). Portanto, a técnica convencional leva menos tempo do que a técnica dirigida por modelos. Tc_r - $Tm_r < 0$

Na Tabela 2 são listadas duas variáveis: "Tc_r" e "Tm_r". "Tc_r" representa o tempo total necessário para reusar o framework utilizando a técnica convencional. "Tm_r" representa o tempo total necessário para reusar o framework utilizando a técnica dirigida a modelos.

Na Tabela 2 são listadas três hipóteses: "$H0_r$", "Hp_r" e "Hn_r". A hipótese "$H0_r$", também chamada de hipótese nula, é verdadeira quando ambas as técnicas são equivalentes; portanto, o tempo necessário para completar o processo utilizando a técnica convencional menos o tempo necessário para completar o processo utilizando a ferramenta da técnica dirigida a modelos é aproximadamente zero.

A hipótese "Hp_r" é verdadeira quando a técnica convencional leva mais tempo do que a técnica dirigida a modelos; portanto, o tempo necessário para completar o processo utilizando a técnica convencional menos o tempo necessário para completar o processo utilizando a ferramenta da técnica dirigida a modelos é positivo.

A hipótese "Hn_r" é verdadeira quando a técnica convencional leva menos tempo do que a técnica dirigida a modelos; portanto, o tempo necessário para completar o processo utilizando a técnica convencional menos o tempo necessário para completar o processo utilizando a ferramenta da técnica dirigida a modelos é negativo.

Considerando que as hipóteses consideram intervalos de um valor real, só há uma hipótese verdadeira para um experimento de sucesso. A hipótese nula possui precedência com relação às outras, visto que isso depende da margem de erro estipulada no teste estatístico.

6.2.2.3 Seleção de Variáveis

As variáveis são divididas em duas categorias: dependentes e independentes.

As variáveis dependentes são aquelas que são analisadas neste experimento. Dessa forma, é analisado o tempo necessário para completar o processo de reúso de um framework.

As variáveis independentes são controladas e manipuladas pelos pesquisadores, neste caso, são “Aplicação Base”, “Técnica de Reúso empregada” e “Tipo de Execução”.

6.2.2.4 Critério de Seleção dos Participantes

Foram selecionados participantes alunos de um curso de programação orientada a aspectos. Todos os participantes com disponibilidade foram selecionados. O número igual de participantes de graduação e pós-graduação foi mera coincidência, já que um nono aluno de graduação não pôde participar do experimento. Nenhum participante do experimento foi omitido neste trabalho.

6.2.2.5 Projeto do Estudo

Os participantes foram divididos em dois grupos. Cada um dos grupos foi composto por quatro alunos de graduação e quatro estudantes de pós-graduação. Cada grupo foi balanceado considerando um formulário de caracterização e resultados coletados durante o estudo piloto. Na Tabela 3 são listadas as fases executadas no estudo empírico.

6.2.2.6 Instrumentação

As aplicações foram fornecidas em conjunto com dois documentos. O primeiro documento fornecido é um manual relativo à técnica de reúso empregada e o segundo documento é uma lista de detalhes da aplicação a ser acoplada. Estes detalhes são importantes para efetuar a composição do framework à aplicação base. Os manuais e outros documentos utilizados durante os experimentos estão disponíveis no Apêndice C.

O processo para reusar cada umas das aplicações fornecidas possuía a mesma complexidade, portanto, para concluir o processo de reúso para cada aplicação exigia

Tabela 3: Projeto do Estudo de Reúso

Fase	Grupo 1	Grupo 2
Treinamento	Treinamento de Técnicas de Reúso	
	Oficina de Manutenção	
1ª Fase Piloto	Convencional	Modelos
	Aplicação Hotel	
2ª Fase Piloto	Modelos	Convencional
	Aplicação Biblioteca	
1ª Fase Reúso (Primária)	Convencional	Modelos
	Aplicação de Entregas	
2ª Fase Reúso (Primária)	Modelos	Convencional
	Aplicação Voos	
1ª Fase Reúso (Secundária)	Convencional	Modelos
	Aplicação Clínica Médica	
2ª Fase Reúso (Secundária)	Modelos	Convencional
	Aplicação Restaurante	

dos participantes a especificação de quatro valores, a assinatura de doze métodos e nomes de seis classes.

Cada fase representada na Tabela 3 é dividida em nomes da aplicação e a técnica empregada para reusar o framework. Por exemplo, durante a 1ª Fase de Reúso (Primária), os participantes do primeiro grupo compuseram o framework à "Aplicação Hotel" utilizando a técnica convencional de reúso ao mesmo instante que os participantes do segundo grupo utilizaram a técnica dirigida a modelos para executar a mesma tarefa com a mesma aplicação.

6.2.3 Operação

6.2.3.1 Preparação

Inicialmente, cada aluno foi apresentado à ferramenta que implementa a técnica de reúso dirigido a modelos e foi ensinado como reusar o framework transversal com esta ferramenta. Em seguida, os alunos foram treinados em como reusar o framework com a forma convencional, mesmo que eles já possuíam experiência com essa forma.

Deste modo, todo participante teve de reusar cada aplicação utilizando apenas uma das técnicas em números iguais. Além disso, durante todo o experimento, cada grupo utilizou uma técnica diferente do que o outro grupo.

6.2.3.2 Execução

Inicialmente, os participantes assinaram o formulário de consentimento e então responderam um formulário de caracterização. O formulário de caracterização possuía questões com relação ao conhecimento de sintaxe de AspectJ, Eclipse IDE e Frameworks Transversais.

Após a conclusão dos formulários de caracterização, os participantes foram treinados em como reusar o FT fornecido com as duas técnicas de reúso. Em seguida ao treinamento, foi executado o experimento piloto. Os participantes foram divididos em grupos considerando o formulário de caracterização, simulando os experimentos reais.

No piloto, as aplicações eram diferentes, porém, equivalentes e os participantes eram autorizados a fazer perguntas sobre o que não haviam compreendido durante o treinamento. Isso afetaria a validade dos valores, portanto, os dados dessa atividade foram utilizados somente para rebalancear os grupos.

Durante os experimentos reais, os participantes reusaram outras aplicações iniciando com técnicas diferentes para cada grupo. A execução secundária foi uma replicação da execução primária com aplicações diferentes. Esta execução foi realizada para evitar o risco de coletar resultados desbalanceados durante a primeira execução, visto que os dados coletados durante o experimento piloto foram invalidadas.

6.2.3.3 Validação de Dados

Os formulários preenchidos pelos participantes foram validados com dados coletados durante o experimento piloto.

Os pesquisadores também observaram notificações do sistema de informação dos dados de experimento para confirmar se os participantes concluíram os processos antes de ficarem ociosos.

6.2.3.4 Coleta de Dados

Os tempos registrados durante os processos de reúso em ambas as técnicas durante o experimento na execução primária estão listados na Tabela 4 e os tempos para a execução secundária estão na Tabela 5.

Cada tabela de tempos registrados possuem cinco colunas. Na coluna “G.” é indicado o grupo do participante, em “Ap.” é indicada a aplicação do exercício; em “T.” é indicada a técnica empregada no reúso, que pode ser “C” para convencional

ou “M” para dirigida por modelos; a coluna “P.” lista um código único para identificar cada participante, em que os oito primeiros números (de um a oito) indicam alunos de pós-graduação e os outros oito números (de nove a dezesseis) indicam alunos de graduação; A coluna “Tempo” lista o tempo utilizado pelo participante para completar o processo.

O sistema de informação empregado para coletar os dados do experimento, registrou os tempos com precisão de milissegundos considerando o relógio do servidor e das máquinas cliente. Porém, os valores apresentados nesse capítulo apenas consideram o relógio do servidor, portanto, o atraso de transmissão dos computadores foi desconsiderado. Esse atraso é considerado ser insignificante, pois o servidor estava na rede local e cálculos preliminares considerando os relógios das máquinas clientes não alteraram a ordem dos resultados.

Esse sistema coletou os dados de tempo e outros valores de forma transparente. Os participantes somente eram requeridos para disparar o tempo inicial, o que era supervisionado, e então deveriam trabalhar no processo de reúso sem ajuda exterior. Os outros valores foram coletados transparentemente. Assim que o caso de teste fosse executado com sucesso, o tempo de término era automaticamente enviado ao servidor antes mesmo de notificar o sucesso ao participante.

6.2.4 Análise de Dados e Interpretação

Os dados da primeira execução, que são encontrados na Tabela 4, estão ordenados pelo tempo utilizado para completar o processo. Ao analisar a tabela, nota-se que as linhas que representam o uso da técnica dirigida por modelos, as quais são identificadas pela letra ‘M’, são encontradas nos primeiros doze resultados. As linhas da técnica convencional, as quais são identificadas pela letra ‘C’, são os últimos quatro resultados.

Para permitir uma melhor visualização dos resultados, neste capítulo também é fornecida uma versão gráfica da Tabela 4 em forma de um gráfico de barras. O gráfico de barras para a primeira execução é encontrado na Figura 40. Os mesmos códigos de identificação são usados para diferenciar cada participante. Os valores de tempo para completar os processos são visíveis em formato de segundos.

A segunda informação importante encontrada na primeira execução é que nenhum participante foi capaz de reusar o framework de forma mais rápida ao utilizar a técnica convencional durante a mesma atividade.

Os dados da segunda execução são encontrados na Tabela 5. Esta execução resultou em valores similares. Os primeiros onze resultados foram registrados por

Tabela 4: Tempos Coletados na Execução Primária de Reúso

G.	Ap.	T.	P.	Tempo minutos : segundos
1	Voos	M	15	04:19.952015
1	Voos	M	13	04:58.604963
1	Voos	M	8	05:18.346829
2	Entregas	M	11	05:24.249952
2	Entregas	M	5	05:31.653952
2	Entregas	M	9	05:45.484577
2	Entregas	M	3	06:16.392424
2	Entregas	M	10	06:45.96879
2	Entregas	M	14	07:05.858718
2	Entregas	M	6	07:39.300214
2	Entregas	M	2	08:02.570996
1	Voos	M	1	08:38.69836
2	Voos	C	2	08:42.389884
1	Voos	M	16	10:18.809487
1	Entregas	C	13	10:25.359836
2	Voos	C	9	10:51.761493
1	Voos	M	7	10:52.183247
2	Voos	C	10	10:52.495216
1	Entregas	C	8	11:39.151434
1	Entregas	C	15	12:03.519008
1	Voos	M	4	12:17.693128
2	Voos	C	3	12:26.993837
2	Voos	C	14	12:49.585392
2	Voos	C	11	13:04.272941
1	Entregas	C	4	13:16.470523
1	Entregas	C	1	15:47.376327
1	Entregas	C	16	18:02.259692
1	Voos	M	12	20:03.920754
2	Voos	C	5	21:32.272442
2	Voos	C	6	23:10.72776
1	Entregas	C	7	23:20.991158
1	Entregas	C	12	41:29.414342

participantes utilizando a ferramenta proposta enquanto os últimos quatro foram registrados por participantes utilizando a técnica convencional. Também é mostrado um gráfico de barras para essa execução na Figura 41.

Apenas o participante de número dezesseis foi mais rápido ao utilizar a técnica convencional, o que contradiz os resultados coletados pelo mesmo participante durante a execução anterior. Este participante argumentou ter se confundido com a ordem de preencher os elementos do modelo, acarretando essa diferença de resultados.

Tabela 5: Tempos Coletados na Execução Secundária de Reúso

G.	Ap.	T.	P.	Tempo minutos : segundos
2	Clinica	M	10	02:59.467569
1	Restaurante	M	13	03:56.785359
1	Restaurante	M	15	04:23.629206
2	Clinica	M	11	04:25.196135
1	Restaurante	M	8	04:33.954349
2	Clinica	M	9	04:41.25492
1	Restaurante	M	12	05:05.524264
2	Clinica	M	3	05:45.333167
2	Clinica	M	14	05:57.00931
2	Clinica	M	5	06:31.365498
2	Clinica	M	2	06:59.96749
2	Restaurante	C	2	07:18.927029
2	Clinica	M	6	07:45.403075
2	Restaurante	C	10	08:56.765163
1	Clinica	C	16	09:20.284593
1	Restaurante	M	7	09:23.574403
1	Restaurante	M	4	09:25.089084
2	Restaurante	C	14	09:27.112225
2	Restaurante	C	3	09:55.736324
1	Clinica	C	15	10:25.475603
2	Restaurante	C	5	10:37.460834
2	Restaurante	C	9	10:49.014842
1	Restaurante	M	16	10:56.743477
1	Clinica	C	13	11:04.48539
1	Clinica	C	4	12:06.690347
1	Clinica	C	8	13:38.014602
1	Clinica	C	12	14:37.19726
1	Restaurante	M	1	17:09.073104
2	Restaurante	C	11	17:11.980052
1	Clinica	C	7	19:35.816561
2	Restaurante	C	6	28:02.391335
1	Clinica	C	1	28:18.301114

Na Tabela 6 estão listados a média de tempo e as suas proporções. A coluna “Execução” lista qual das atividades é considerada no cálculo da média, que pode ser “Primária” ou “Secundária”. A coluna “Execução” lista se a técnica é “Convencional” ou “Modelos”. Na coluna “Média” há os resultados do cálculo considerando as técnicas e execuções especificadas em cada linha. Na coluna “Soma das Médias” é mostrado o valor das médias para cada técnica somando ambas as execuções. Na coluna “Porcentagem” é mostrado a relação entre o valor das somas parciais com a soma total do tempo utilizado durante todo o experimento.

Por considerar o tempo médio dos participantes de ambos os grupos para comple-

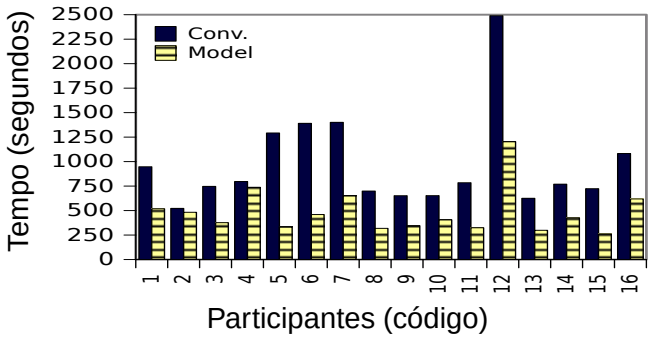

Figura 40: Gráfico de Barras para a Execução Primária de Reúso

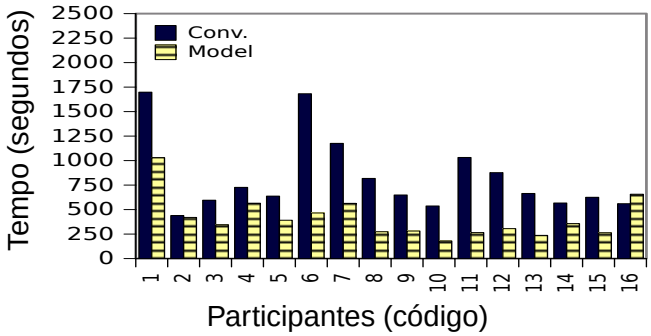

Figura 41: Gráfico de Barras para a Execução Secundária de Reúso

tar os processos de forma convencional dividido pelo tempo médio utilizado para completar os processos com a ferramenta proposta, a técnica convencional levou aproximadamente 100,01% mais tempo para ser concluída do que a técnica dirigida a modelos, ou seja, aproximadamente o dobro.

Tabela 6: Médias de Tempo do Estudo de Reúso

Execução	Técnica	Média	Soma das Médias	Porcentagem
Primária	Convencional	16:13,44008	30:03,79341	66,7766%
Secundária		13:50,35333		
Primária	Modelos	08:04,980525	14:57,441176	33,2234%
Secundária		06:52,460651		
Total		45:01,234586		100,0000%

6.2.5 Teste de Hipóteses

Foram aplicados Teste-T Pareados para cada execução apresentada sem remoção de *outliers* e outro Teste-T unindo todos os dados com a remoção de oito *outliers*. Os valores de tempo registrados foram processados no formato de segundos utilizando o ambiente computacional “R” Free Software Foundation, Inc. (2011). Os resultados dos Testes-T estão listados na Tabela 7. A primeira coluna indica se o teste-t é *Paired*

(pareado) ou *Two-Sided*.

A coluna “Means” indica uma média das diferenças entre os pares para um teste-t pareado ou médias para cada conjunto em um teste-t não pareado. No segundo caso, a média para a técnica convencional é mostrado primeiro e a média para a técnica dirigida a modelos em seguida. A coluna “d.t.” indica o grau de liberdade; “t” e “p” são variáveis consideradas no teste das hipóteses.

Um Teste-T Pareado é utilizado para comparar diferenças entre duas amostras. Em ambas amostras, é conhecido o par de valores para cada participante, neste caso, é calculado a diferença individual de tempo que cada participante levou para concluir o processo de forma convencional menos o tempo necessário para concluir de forma dirigida a modelos.

O outro Teste-T não é pareado, as médias de todo o grupo são calculadas independentemente, porque não foram mantidos os pares de execução de cada participante, pois, um participante pode ser *outlier* em uma execução e não na outra. As diferenças são calculadas depois do cálculo da média de cada amostra. Nesse caso, o teste é chamado de *Two-Sided* porque ambas as amostras também possuem o mesmo número de elementos, já que foram removidos *outliers* em números iguais.

Tabela 7: Resultados dos Testes-T do Experimento de Reúso

T-Test	Data	Means	d.f.	t	p
Paired	Primária	488,4596	15	5,841634	$3{,}243855 \cdot 10^{-05}$
Paired	Secundária	417,8927	15	5,285366	$9{,}156136 \cdot 10^{-05}$
Two-Sided	Ambos	771,4236 409,4295	43,70626	6,977408	$1{,}276575 \cdot 10^{-08}$

A remoção de *outliers* para o cálculo do último teste-t foi realizada com base no calculo de um Teste “Chi Quadrado”. Os resultados do Teste “Chi Quadrado” estão dispostos na Tabela 8. A letra ‘M’ na coluna ‘T.’ indica o uso da técnica dirigida a modelos enquanto que a letra ‘C’ indica o uso da técnica convencional. A coluna ‘G.’ indica o número dos grupos. O X^2 indica o resultado de um calculo de diferença do elemento detectado como *outlier* com a variância da amostra em que está inserido. A coluna “posição” indica a posição do elemento na amostra, que pode ser ‘maior’ ou ‘menor’ valor. A coluna de *outliers* indica o valor em segundos do elemento que foi considerado anormal para a amostra.

Para melhor visualização dos *outliers*, dois gráficos de caixas são mostrados. Na Figura 42 há um gráfico de caixas para a execução primária, enquanto que na Figura 43 há um gráfico de caixas para a execução secundária. As caixas representam a concentração dos valores. Os quadrados preenchidos dentro das caixas representam a média da amostra. As distancias entre os símbolos “+” indicam o desvio padrão

Tabela 8: Teste Chi-Quadrado para detecção de *outliers* no Experimento de Reúso

Execução	T.	G.	X^2	p	posição	outlier
Primária	C	1	5,104305	0,02386654	maior	2489,414342
		2	2,930583	0,08691612	maior	1390,72776
	M	1	4,091151	0,04310829	maior	1203,920754
		2	2,228028	0,1355267	maior	482,570996
Secundária	C	1	4,552248	0,03287556	maior	1698,301114
		2	5,013908	0,02514448	maior	1682,391335
	M	1	3,917559	0,04778423	maior	1029,073104
		2	2,943313	0,08623369	menor	179,467569

e os símbolos “o” indicam os *outliers*.

Nas Figuras 44 e 45 estão gráficos de linhas que são também utilizados para visualizar a dispersão dos valores registrados. Nesses gráficos o números dos participantes não condizem com os códigos únicos, pois cada amostra é ordenada de forma independente.

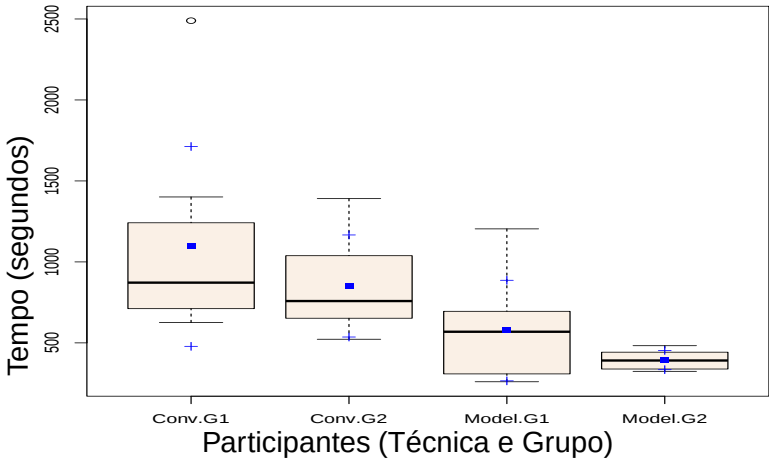

Figura 42: Gráfico de Caixas da Execução Primária

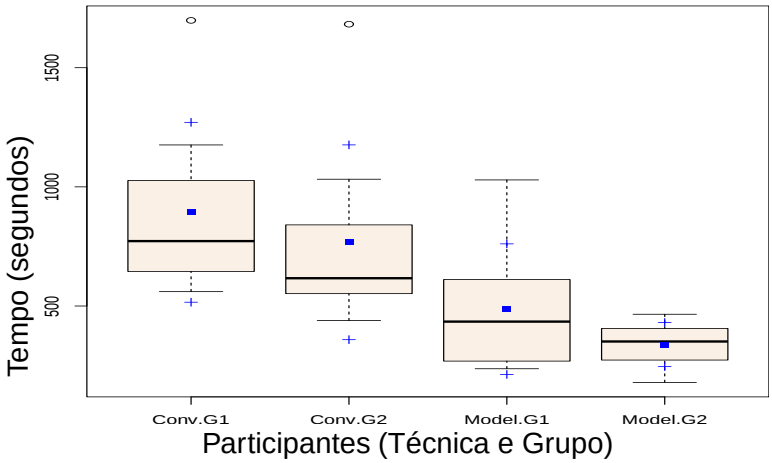

Figura 43: Gráfico de Caixas da Execução Secundária

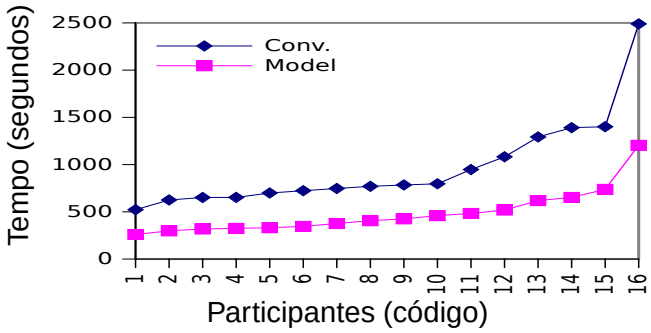

Figura 44: Gráfico de Linhas da Execução Primária

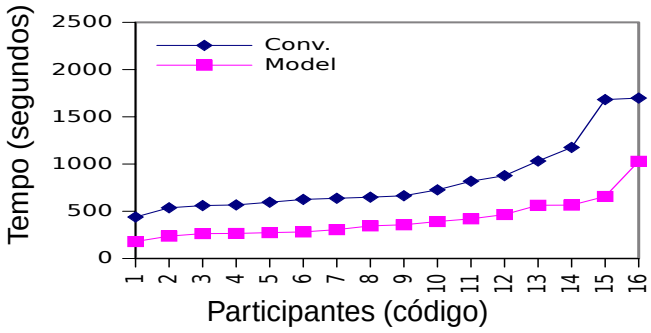

Figura 45: Gráfico de Linhas da Execução Secundária

De acordo com uma análise da Tabela 7, como o valor absoluto de todos os valores 'p' são menores que a margem de erro (0,01%), o que corresponde ao complemento do nível de significância estabelecido de 99,99%, portanto, estatisticamente, pode-se rejeitar a hipótese "H0$_r$" em que as técnicas deveriam ser equivalentes.Como todo valor 't' é positivo, a hipótese positiva é verdadeira, ou seja, "Hp$_r$", em que o uso da técnica dirigida a modelos leva menos tempo que a técnica convencional.

6.3 Experimento de Manutenção

Neste trabalho também foram realizados experimentos para comparar as diferenças entre o uso da ferramenta e o processo convencional no contexto de manutenção. Dessa forma, o objetivo foi comparar o esforço para realizar alterações em um modelo de reúso (técnica dirigida a modelos) com realizar alterações no código de reúso (técnica convencional). Assim como o experimento de reúso, o planejamento dos estudos apresentados nesta seção realizados considerando as diretrizes propostas por Wohlin et al. (2000).

6.3.1 Definição do Estudo Empírico

6.3.1.1 Objetivo

O objetivo do estudo é comparar o esforço para manter aplicações acopladas a frameworks transversais em duas formas distintas de reúso: o processo convencional e o processo dirigido a modelos.

6.3.1.2 Sujeito do Estudo

Um framework transversal é acoplado a duas aplicações que carecem de manutenção.

6.3.1.3 Enfoque Quantitativo

O enfoque quantitativo envolve a identificação de esforço com base nos tempos tomados por cada participante para concluir cada processo de manutenção.

6.3.1.4 Enfoque Qualitativo

O enfoque qualitativo é usado para determinar a técnica que exige menos esforço.

6.3.1.5 Perspectiva

Os experimentos foram conduzidos da perspectiva de engenheiros de aplicação que possuam interesse em alterar aplicações acopladas a frameworks transversais.

6.3.1.6 Objeto de Estudo

O objeto dos estudos deste capítulo é o “esforço” para concluir um processo de manutenção de uma aplicação acoplada a um framework transversal.

6.3.2 Planejamento do Estudo

Os experimentos foram planejados considerando a seguinte questão: “Qual artefato de reúso exige menos esforço durante a manutenção de uma aplicação acoplada a um FT?”; Essa questão é então analisada no término da seção para avaliar se ela foi respondida com sucesso.

Foi solicitado aos participantes para corrigir erros em um artefato de reúso de uma aplicação já fornecida de modo a fazer com que a aplicação fosse corretamente acoplada ao framework e com ambas as técnicas. Assim como o experimento de reúso, os seguintes dados pertinentes a este estudo foram coletados e analisados:

- Técnica de Reúso empregada;
- Aplicação Base;
- Data e Hora do Início de cada Execução;
- Data e Hora do Término de cada Execução;
- Tipo de Execução.

O item “Técnica de Reúso empregada” é utilizado para definir se o participante do estudo utilizou a técnica dirigida a modelos ou o processo convencional durante um processo de reúso. O item “Aplicação Base” é utilizado para identificar qual aplicação base está sendo acoplada ao framework. Os item “Data e Hora do Início de cada Execução” e “Data e Hora do Término de cada Execução” são usados para registrar o momento de início e fim de cada execução, valores considerados para calcular o tempo tomado em cada processo. “Tipo de Execução” é a finalidade de cada execução da aplicação final, que pode ser caso de teste, geração de código, validação ou registro de tempo inicial.

O mesmo sistema de informação para coleta de dados experimentais foi empregado para capturar os dados especificados.

6.3.2.1 Seleção de Contexto

Este estudo foi conduzido com os mesmos participantes do experimento de reúso. Dezesseis alunos participaram do experimento. Os primeiros oito eram alunos de pós-graduação e os outros oito eram alunos de graduação.

Todos os participantes possuíam experiencia prévia em AspectJ e corrigiram erros no reúso de aplicações acopladas a um framework transversal de persistência (CAMARGO; MASIERO, 2008). Cada processo de manutenção foi efetuado utilizando ou da técnica convencional ou da técnica dirigida por modelos.

6.3.2.2 Formulação de Hipóteses

Na Tabela 9 estão listadas as hipóteses para o experimento de manutenção. As hipóteses foram criadas para comparar a produtividade de editar o artefato “código

de reúso" com o artefato "modelo de reúso" durante a manutenção de uma aplicação acoplada a um framework transversal.

Tabela 9: Hipóteses para o Estudo de Manutenção

$H0_m$	Não há diferença entre editar o código de reúso e o modelo de reúso durante um processo de manutenção de uma aplicação que reusa o framework transversal em termos de produtividade (tempo). Portanto, os artefatos são equivalentes. Tc_m - $Tm_m \approx 0$
Hp_m	Há uma diferença positiva entre editar o código de reúso e o modelo de reúso durante um processo de manutenção de uma aplicação que reusa o framework transversal em termos de produtividade (tempo). Portanto, a edição do código de reúso leva menos tempo do que editar um modelo de reúso durante a manutenção. Tc_m - $Tm_m > 0$
Hp_m	Há uma diferença negativa entre editar o código de reúso e o modelo de reúso durante um processo de manutenção de uma aplicação que reusa o framework transversal em termos de produtividade (tempo). Portanto, a edição do código de reúso leva mais tempo do que editar um modelo de reúso durante a manutenção. Tc_m - $Tm_m < 0$

Na Tabela 9 são listadas duas variáveis: "Tc_m" e "Tm_m". "Tc_m" representa o tempo total necessário para concluir o processo de manutenção editando o código de reúso. "Tm_m" representa o tempo total necessário para concluir o processo de manutenção editando o modelo de reúso.

Na Tabela 9 são listadas três hipóteses: "$H0_m$", "Hp_m" e "Hn_m". A hipótese "$H0_m$", também chamada de hipótese nula, é verdadeira quando ambos os artefatos são equivalentes; portanto, o tempo necessário para completar o processo editando o código de reúso menos o tempo necessário para completar o processo editando o modelo de reúso é aproximadamente zero.

A hipótese "Hp_m" é verdadeira quando a edição do código de reúso leva mais tempo do que a edição do modelo de reúso; portanto, o tempo necessário para completar o processo editando o código de reúso menos o tempo necessário para completar o processo editando o modelo de reúso é positivo.

A hipótese "Hn_m" é verdadeira quando a edição do código de reúso leva menos tempo do que a edição do modelo de reúso; portanto, o tempo necessário para completar o processo editando o código de reúso menos o tempo necessário para completar o processo editando o modelo de reúso é negativo.

Considerando que as hipóteses consideram intervalos de um valor real, só há uma hipótese verdadeira para um experimento de sucesso. A hipótese nula possui prece-

dência com relação às outras, visto que isso depende da margem de erro estipulada no teste estatístico.

6.3.2.3 Seleção de Variáveis

As variáveis são divididas em duas categorias: dependentes e independentes.

As variáveis dependentes são aquelas que são analisadas neste experimento. Dessa forma, é analisado o tempo necessário para completar o processo de manutenção de uma aplicação base acoplada a um framework.

As variáveis independentes são controladas e manipuladas pelos pesquisadores, neste caso, são “Aplicação Base”, “Técnica de Reúso empregada” e “Tipo de Execução”.

6.3.2.4 Critério de Seleção dos Participantes

Os participantes foram selecionados em uma abordagem não probabilística por conveniência, portanto, a probabilidade de todos os elementos da população pertencerem à amostra é desconhecida.

6.3.2.5 Projeto do Estudo

Os participantes foram divididos em dois grupos. Cada um dos grupos foi composto por quatro alunos de graduação e quatro estudantes de pós-graduação. Cada grupo foi balanceado considerando um formulário de caracterização e resultados coletados durante o experimento anterior. Na Tabela 10 são listadas as fases executadas no estudo empírico.

Tabela 10: Projeto do Estudo de Manutenção

Fase	Grupo 1	Grupo 2
Treinamento	Treinamento de Técnicas de Manutenção	
	Oficina de Manutenção	
1^a Fase Manutenção (Primária)	Convencional	Modelos
	Aplicação de Entregas	
2^a Fase Manutenção (Primária)	Modelos	Convencional
	Aplicação Voos	
1^a Fase Manutenção (Secundária)	Convencional	Modelos
	Aplicação Clínica Médica	
2^a Fase Manutenção (Secundária)	Modelos	Convencional
	Aplicação Restaurante	

6.3.2.6 Instrumentação

As aplicações base fornecidas para este experimento eram versões modificadas das mesmas aplicações fornecidas no experimento de reúso. Estas aplicações foram fornecidas com códigos de reúso incorretos (para a técnica convencional) ou com modelos de reúso incorretos (para a técnica dirigida a modelos). Para que o processo de manutenção fosse concluído com sucesso, os participantes deveriam corrigir estes artefatos.

Os participantes receberam um manual com possíveis erros que podem ser encontrados e suas possíveis soluções. Neste experimento, não foram fornecidos documentos com detalhes da aplicação, os participantes deveriam procurar manualmente no código fonte da aplicação base e corrigir os erros nos artefatos de reúso. Os manuais e outros documentos utilizados durante os experimentos estão disponíveis no Apêndice C.

O processo para corrigir cada um dos artefatos de reúso possuía a mesma complexidade, portanto, para concluir o processo de manutenção de cada aplicação exigia dos participantes a atualização de três nomes de classes, a atualização de três nomes de métodos e a correção de três caracteres inválidos.

Também é importante salientar que erros específicos à edição manual de código de reúso não foram inseridos nesse estudo. Exemplos de erros específicos à edição manual de código incluem: erros de sintaxe de AspectJ. Também não foram realizadas omissões no código de reúso.

Cada fase representada na Tabela 10 é dividida em nomes da aplicação e a técnica empregada para reusar o framework. Por exemplo, durante a 1^a Fase de Manutenção (Primária), os participantes do primeiro grupo executaram um processo de manutenção da “Aplicação Entregas” editando o código de reúso ao mesmo instante que os participantes do segundo grupo utilizaram editaram o modelo de reúso para executar a mesma tarefa com a mesma aplicação.

6.3.3 Operação

6.3.3.1 Preparação

Este experimento foi aplicado em sequência ao experimento de reúso, que foi considerado para a caracterização dos participantes e para a divisão dos grupos. Em seguida, cada aluno foi apresentado a possíveis erros que poderiam ser encontrados e suas possíveis soluções.

Deste modo, assim como no experimento de reúso, todo participante teve de editar os artefatos de reúso de cada aplicação utilizando apenas um dos tipos de artefatos em números iguais. Além disso, durante todo o experimento, cada grupo editou um tipo de artefato diferente do que o outro grupo.

6.3.3.2 Execução

Após a execução do experimento de reúso, foi realizado um treinamento específico para mostrar possíveis erros que podem ser encontrados quando um artefato de reúso está incorretamente definido.

Os participantes corrigiram artefatos de reúso em duas aplicações do experimento. O uso de duas execuções foi realizada para evitar o risco de coletar resultados desbalanceados durante a primeira execução.

6.3.3.3 Validação de Dados

Os pesquisadores observaram notificações do sistema de informação dos dados de experimento para confirmar se os participantes concluíram os processos antes de ficarem ociosos.

6.3.3.4 Coleta de Dados

Os tempos registrados durante os processos de manutenção em ambas as técnicas durante o experimento na execução primária estão listados na Tabela 11 e os tempos para a execução secundária estão na Tabela 12.

Cada tabela de tempos registrados possuem cinco colunas. Na coluna “G.” é indicado o grupo do participante, em “Ap.” é indicada a aplicação do exercício; em “T.” é indicada a técnica utilizada na manutenção, que pode ser “C” para edição de código de reúso ou “M” para edição de modelo de reúso; a coluna “P.” lista um código único para identificar cada participante, em que os oito menores valores indicam alunos de pós-graduação e os oito maiores valores indicam alunos de graduação; A coluna “Tempo” lista o tempo utilizado pelo participante para completar o processo.

O sistema de informação empregado para coletar os dados do experimento registrou os tempos com precisão de milissegundos considerando o relógio do servidor e das máquinas cliente. Porém, os valores apresentados nesse capítulo apenas consideram o relógio do servidor, portanto, o atraso de transmissão dos computadores foi desconsiderado. Esse atraso é considerado ser insignificante, pois o servidor estava

na rede local e cálculos preliminares considerando os relógios das máquinas clientes não alteraram a ordem dos resultados.

Esse sistema coletou os dados de tempo e outros valores de forma transparente. Os participantes somente eram requeridos para disparar o tempo inicial, o que era supervisionado, e então deveriam trabalhar no processo de reúso sem ajuda exterior. Os outros valores foram coletados transparentemente. Assim que o caso de teste fosse executado com sucesso, o tempo de término era automaticamente enviado ao servidor antes mesmo de notificar o sucesso ao participante.

Tabela 11: Tempos Coletados na Execução Primária de Manutenção

G.	Ap.	T.	P.	Tempo minutos : segundos
2	Voos	C	10	02:30.944685
2	Voos	C	9	02:54.232578
1	Entregas	C	8	03:02.751342
2	Voos	C	2	03:11.695431
1	Entregas	C	15	03:31.801582
2	Voos	C	12	03:45.692316
2	Voos	C	3	05:09.817914
2	Voos	C	5	05:44.462030
1	Voos	M	8	05:53.407296
2	Voos	C	11	07:08.687074
2	Voos	C	6	07:38.576312
1	Voos	M	4	07:53.595699
1	Voos	M	14	08:14.148937
2	Entregas	M	3	08:27.092566
1	Entregas	C	1	08:37.138931
1	Voos	M	13	08:50.185469
1	Voos	M	1	09:15.253791
2	Entregas	M	5	09:15.934211
1	Entregas	C	14	09:32.031612
1	Entregas	C	7	10:04.694800
1	Voos	M	15	11:07.617639
2	Entregas	M	6	11:32.482992
2	Entregas	M	2	11:49.247460
1	Entregas	C	16	12:12.576158
1	Voos	M	7	12:27.297563
1	Entregas	C	13	12:49.443610
2	Entregas	M	11	13:00.604583
1	Entregas	C	4	13:25.433748
2	Entregas	M	9	15:51.117061
2	Entregas	M	12	15:56.048486
2	Entregas	M	10	21:23.533192
1	Voos	M	16	32:32.875079

Tabela 12: Tempos Coletados na Execução Secundária de Manutenção

G.	Ap.	T.	P.	Tempo minutos : segundos
2	Clinica	M	5	01:43.801965
2	Clinica	M	3	02:17.158954
1	Clinica	C	8	02:34.248260
1	Clinica	C	14	02:57.405545
2	Restaurante	C	2	03:01.547524
2	Restaurante	C	10	03:09.169865
2	Clinica	M	2	03:25.640129
2	Restaurante	C	3	03:39.443080
1	Clinica	C	7	04:28.998071
2	Restaurante	C	6	04:35.517498
2	Restaurante	C	12	04:41.052812
2	Restaurante	C	11	04:46.028085
1	Restaurante	M	8	04:51.290971
2	Clinica	M	6	04:53.800449
1	Restaurante	M	15	04:58.094389
1	Clinica	C	15	05:21.846560
2	Restaurante	C	5	05:42.389865
2	Clinica	M	10	07:18.533351
1	Restaurante	M	14	07:24.342788
1	Clinica	C	16	07:37.332151
1	Restaurante	M	1	07:44.516376
2	Clinica	M	11	08:08.144168
2	Restaurante	C	9	08:13.115942
1	Restaurante	M	13	08:32.056119
1	Restaurante	M	16	11:28.592180
1	Restaurante	M	7	11:45.459699
2	Clinica	M	9	12:42.958789
1	Restaurante	M	4	13:57.879299
1	Clinica	C	1	14:46.465482
1	Clinica	C	4	17:55.176353
1	Clinica	C	13	18:02.486509
2	Clinica	M	12	25:54.176697

6.3.4 Análise de Dados e Interpretação

Os dados da primeira execução, que são encontrados na Tabela 11, e os dados da segunda execução são encontrados na Tabela 12. Ambas as tabelas estão ordenados pelo tempo utilizado para completar o processo. Ao analisar a tabela, nota-se que as técnicas estão dispostas de forma muito mais aleatória do que os dados do experimento de reúso.

Para fornecer uma melhor visualização dos resultados, neste capítulo também é mostrada uma versão gráfica da Tabela 11 em forma de um gráfico de barras. O

gráfico de barras para a primeira execução é encontrado na Figura 46. Também é mostrado um gráfico de barras para a segunda execução na Figura 47. Os mesmos códigos de identificação são usados para diferenciar cada participante. Os valores de tempo para completar os processos são visíveis em formato de segundos.

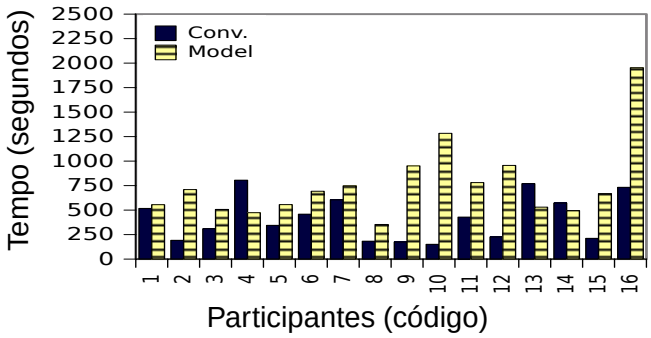

Figura 46: Gráfico de Barras para a Execução Primária de Manutenção

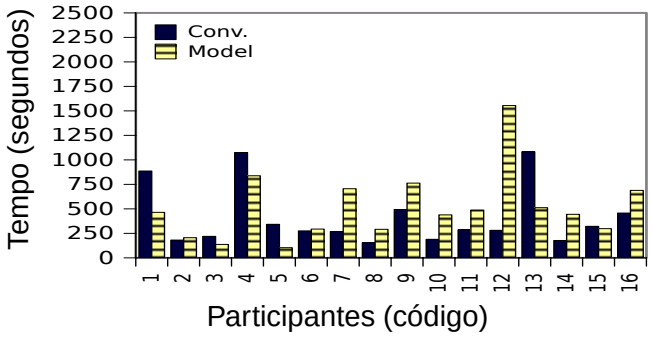

Figura 47: Gráfico de Barras para a Execução Secundária de Manutenção

Na Tabela 13 estão listados a média de tempo e as suas proporções. A coluna “Execução” lista qual das atividades é considerada no cálculo da média, que pode ser “Primária” ou “Secundária”. A coluna “Execução” lista se a técnica é “Convencional” ou “Modelos”. Na coluna “Média” há os resultados do cálculo considerando as técnicas e execuções especificadas em cada linha. Na coluna “Soma das Médias” é mostrado o valor das médias para cada técnica somando ambas as execuções. Na coluna “Porcentagem” é mostrado a relação entre o valor das somas parciais com a soma total do tempo utilizado durante todo o experimento.

Por considerar o tempo médio dos participantes de ambos os grupos para completar os processos de forma convencional dividido pelo tempo médio utilizado para completar os processos com a ferramenta proposta, a técnica dirigida por modelos levou aproximadamente 52,83% mais tempo para ser concluída do que a técnica convencional.

Tabela 13: Médias de Tempo do Estudo de Manutenção

Execução	Técnica	Média	Soma das Médias	Porcentagem
Primária	Convencional	06:57,498758	13:55,762733	39,5521%
Secundária		06:58,263975		
Primária	Modelos	12:43,152626	21:17,305521	60,4479%
Secundária		08:34,152895		
Total		35:13,068254		100,0000%

6.3.5 Teste de Hipóteses

Foram aplicados Teste-T Pareados para cada execução apresentada sem remoção de *outliers* e outro Teste-T após a remoção de oito *outliers*. Os valores de tempo registrados foram processados no formato de segundos utilizando o ambiente computacional "R" Free Software Foundation, Inc. (2011). Os resultados dos Testes-T estão listados na Tabela 14. A primeira coluna indica se o teste-t é *Paired* (pareado) ou *Two-Sided*.

A coluna "Means" indica uma média das diferenças entre os pares para um teste-t pareado ou médias para cada conjunto em um teste-t não pareado. No segundo caso, a média para a técnica convencional é mostrado primeiro e a média para a técnica dirigida a modelos em seguida. A coluna "d.t." indica o grau de liberdade; "t" e "p" são variáveis consideradas no teste das hipóteses.

Um Teste-T Pareado é utilizado para comparar diferenças entre duas amostras. Em ambas as amostras, é conhecido o par de valores para cada participante, neste caso, é calculado a diferença individual de tempo que cada participante levou para concluir o processo de forma convencional menos o tempo necessário para concluir de forma dirigida a modelos.

O outro Teste-T não é pareado, as médias de todo o grupo são calculadas independentemente, porque não foram mantidos os pares de execução de cada participante, pois, um participante pode ser *outlier* em uma execução e não na outra. As diferenças são calculadas depois do cálculo da média de cada amostra. Nesse caso, o teste é chamado de *Two-Sided* porque ambas as amostras também possuem o mesmo número de elementos, já que foram removidos *outliers* em números iguais.

Tabela 14: Resultados dos Testes-T do Estudo de Manutenção

T-Test	Data	Means	d.f.	t	p
Paired	Primária	-345,6539	15	-3,971923	0,001227479
Paired	Secundária	-95,88892	15	-1,191781	0,2518624
Two-Sided	Ambos	431,3323	24,22097	-2,662684	0,0135614
		641,0024			

A remoção de *outliers* para o cálculo do último teste-t foi realizada com base no calculo de um Teste “Chi Quadrado”. Os resultados do Teste “Chi Quadrado” estão dispostos na Tabela 15. A letra ‘M’ na coluna ‘T.’ indica o uso da técnica dirigida a modelos enquanto que a letra ‘C’ indica o uso da técnica convencional. A coluna ‘G.’ indica o número dos grupos. O X^2 indica o resultado de um calculo de diferença do elemento detectado como *outlier* com a variância da amostra em que está inserido. A coluna “posição” indica a posição do elemento na amostra, que pode ser ‘maior’ ou ‘menor’ valor. A coluna de *outliers* indica o valor em segundos do elemento que foi considerado anormal para a amostra.

Tabela 15: Teste Chi-Quadrado para detecção de *outliers* no Estudo de Manutenção

Execução	T.	G.	X^2	p	posição	outlier
Primária	C	1	2,350449	0,1252469	menor	182,751342
		2	2,152789	0,1423112	maior	458,576312
	M	1	5,788559	0,0161308	maior	1952,875079
		2	3,598538	0,05783041	maior	1283,533192
Secundária	C	1	1,771974	0,183138	maior	1082,486509
		2	4,338041	0,03726978	maior	493,115942
	M	1	2,422232	0,1196244	maior	837,879299
		2	4,87366	0,02726961	menor	1554,176697

para melhor visualização dos *outliers*, dois gráficos de caixas são mostrados. Na Figura 48 há um gráfico de caixas para a execução primária, enquanto que na Figura 49 há um gráfico de caixas para a execução secundária. As caixas representam a concentração dos valores. Os quadrados preenchidos dentro das caixas representam a média da amostra. As distancias entre os símbolos “+” indicam o desvio padrão e os símbolos “o” indicam os *outliers*.

Nas Figuras 50 e 51 estão gráficos de linhas que são também utilizados para visualizar a dispersão dos valores registrados. Nesses gráficos o números dos participantes não condizem com os códigos únicos, pois cada amostra é ordenada de forma independente.

De acordo com uma análise da Tabela 14, como o valor absoluto de todos os valores ‘p’ são maiores que a margem de erro (0,01%), o que corresponde ao complemento do nível de significância estabelecido de 99,99%, portanto, estatisticamente, pode-se aceitar a hipótese “$H0_m$” em que as técnicas são equivalentes. Dessa forma, de acordo com este experimento, não há vantagens em utilizar a ferramenta durante o esforço de manutenção e que, estatisticamente, as técnicas são equivalentes de acordo com a margem de erro estabelecida.

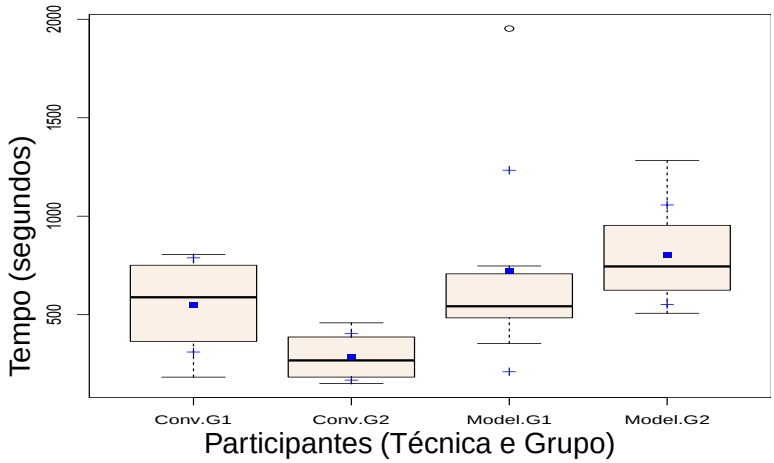

Figura 48: Gráfico de Caixas da Execução Primária do Estudo de Manutenção

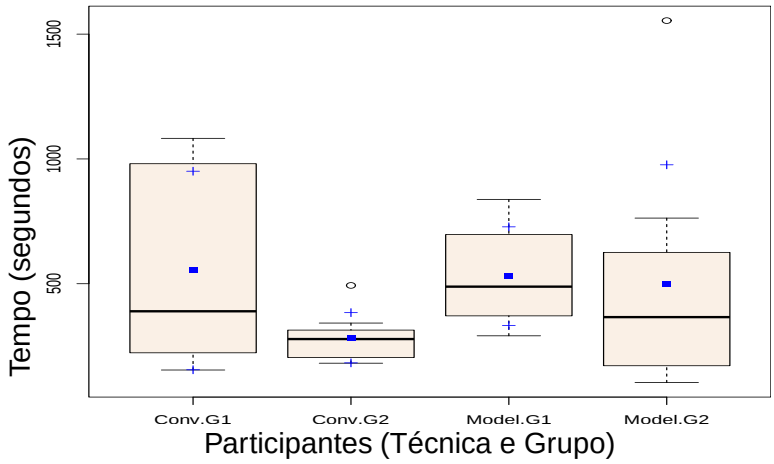

Figura 49: Gráfico de Caixas da Execução Secundária do Estudo de Manutenção

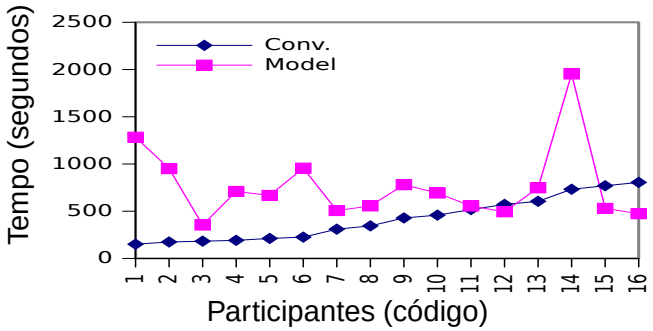

Figura 50: Gráfico de Linhas da Execução Primária do Estudo de Manutenção

6.4 Ameaças à Validade

Nesta seção são apresentados itens que podem afetar os valores e a conclusão dos experimentos apresentados neste capítulo. Ambos os experimentos são afetados pelas mesmas ameaças à validade listadas abaixo.

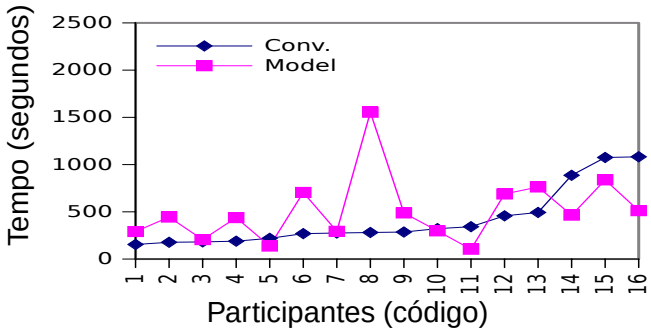

Figura 51: Gráfico de Linhas da Execução Secundária do Estudo de Manutenção

Validade Interna:

- Nível de experiência dos participantes: os variados níveis de experiência e conhecimento dos participante que podem afetar os dados coletados. Para mitigar essa ameaça, os participantes foram divididos em dois grupos equilibrados, considerando o nível de experiência. Os grupos foram rebalanceados após cada atividade do experimento de acordo com os resultados obtidos. Além disso, os participantes tinham experiência prévia sobre como reutilizar o framework de forma convencional. Durante o treinamento, os participantes foram treinados em como reutilizar o framework com a ferramenta dirigida por modelos e, em seguida, novamente em como reutilizá-lo convencionalmente, o que pode fazer com que os participantes tenham mais experiência com a técnica convencional do que com a ferramenta proposta.

- Produtividade em avaliação: existe a possibilidade de que isso poderia influenciar os resultados do experimento, porque os alunos muitas vezes tendem a pensar que estão sendo avaliados pelos resultados dos experimentos. A fim de mitigar isso, foi explicado aos alunos que ninguém estava sendo avaliado e suas participações eram consideradas anônimas.

- Instalações utilizadas durante o estudo: computadores e instalações diferentes poderiam afetar os tempos registrados. No entanto, os diferentes grupos utilizaram a mesma configuração, modelo, sistema operacional e em números iguais. Os participantes não foram autorizados a mudar de máquinas durante a mesma atividade, o que significa que um participante não pode realizar um execício de forma convencional usando um computador diferente que foi usado para reutilizá-lo com a técnica proposta.

Validade por Construção:

- Expectativa nas Hipóteses: os participantes já conheciam os pesquisadores e sabiam que a ferramenta de reúso dirigido por modelos foi feita com o intuito de agilizar o processo de reúso, o que reflete uma hipótese de um experimento. Estes dois itens podem afetar os dados do experimento e torná-lo menos imparcial. Para evitar imparcialidade, foi requerido que os participantes mantivessem um ritmo constante durante todos os exercícios.

Validade Externa:

- Interação entre configuração e tratamento: Existe a possibilidade que os exercícios providos não são suficientemente precisos para representar todo reúso e manutenção de frameworks transversais para aplicações reais. Apenas um único framework foi empregado e as aplicações base tinham mesma complexidade. Para mitigar essa ameaça, os exercícios foram criados considerando aplicações com base no mundo real.

Validade de Conclusão:

- Confiabilidade das Métricas: este item é relacionado com a precisão das métricas empregadas durante as atividades. Para mitigar essa ameaça, apenas foi considerado o tempo capturado por um sistema de informação com precisão de milissegundos;
- Baixo poder estatístico: O número de participantes pode ser pequeno para gerar dados estatísticos confiáveis. Para mitigar essa ameaça, os dados foram analisados com a aplicação de três Testes-T para cada experimento.

6.5 Considerações Finais

Dois experimentos foram apresentados neste capítulo. O experimento de reúso terminou com bons resultados para a abordagem proposta, porém, no experimento de manutenção foi constatado que não há benefícios em usar a abordagem proposta para o esforço de manutenção.

Capítulo 7

CONCLUSÃO

7.1 Considerações Iniciais

Neste trabalho foi apresentada uma abordagem que dá apoio ao engenheiro de domínio no processo de criação de modelos que representam a Família de Frameworks Transversais, bem como ao engenheiro de aplicação no processo de reúso baseado em modelos de famílias de FTs. Para isso, dois modelos foram propostos, o “Modelo de Requisitos de Reúso” e o “Modelo de Reúso”, que são utilizados para documentar e apoiar o reúso de frameworks transversais.

O uso dos modelos é definido em um novo processo de reúso, que também é proposto neste trabalho. Uma ferramenta foi implementada para apoiar a visualização e edição dos modelos. O uso da abordagem com a ferramenta desenvolvida foi avaliado em dois experimentos.

As contribuições obtidas pelas atividades realizadas neste trabalho estão disponíveis na Seção 7.2. Também foram identificadas limitações neste trabalho, que estão disponíveis na Seção 7.3. Algumas sugestões para continuidade deste trabalho estão descritas na Seção 7.5.

7.2 Contribuições deste Trabalho

Os dois novos modelos que documentam e apoiam o processo de reúso de Frameworks Transversais são utilizados para aumentar o nível de abstração durante o reúso desses frameworks.

Os modelos também podem ser visualizados como formulários com o uso do editor desenvolvido neste trabalho. O primeiro formulário proposto permite uma visualização dos requisitos de reúso do framework enquanto o segundo é utilizado para apoiar o processo de reúso. Com isso, é apresentado uma abordagem de reúso diferente do

processo convencional de edição de código.

Com o processo de reúso apresentado, o engenheiro de aplicação é permitido a iniciar o reúso desde as fases de análise do ciclo de vida da aplicação a ser desenvolvida, ou seja, não é necessário esperar até a fase de implementação para iniciar o reúso. Além disso, não é necessário conhecer o paradigma orientado a aspectos e nem a linguagem. Também não é necessário conhecer detalhes da implementação do framework transversal. Somente é necessário ter conhecimento da finalidade do framework e o projeto da aplicação base para completar os modelos de reúso e, consequentemente, gerar o código de reúso. Um gerador de código foi implementado neste trabalho para tornar esta geração possível.

Também é importante notar que este trabalho foi realizado dentro de um projeto para desenvolver uma infraestrutura para Frameworks Tranversais que envolve toda a ferramenta implementada neste trabalho e também abrange um conjunto maior de funções. Por exemplo, o repositório de frameworks citado na Subseção 4.4.1 está fora do escopo deste trabalho mas também faz parte desta infraestrutura. Esta infraestrutura foi denominada CrossFIRE (Crosscutting Framework Integrated Reuse Environment – Ambiente Integrado de Reúso de Frameworks Transversais) (DURELLI; GOTTARDI; CAMARGO, 2012a). Um tutorial de uso da funcionalidade implementada está disponível com imagens da ferramenta e vídeo de exemplo de uso (DURELLI; GOTTARDI; CAMARGO, 2012b).

A abordagem foi avaliada em dois experimentos. Os resultados dos experimentos foram utilizados para responder as questões descritas nos planejamentos dos estudos, o que indica um sucesso conclusivo. Os dados de tempo em formato de planilha podem ser acessados em *http://www2.dc.ufscar.br/~valter/crossfire/*.

Os resultados do estudo de reúso foram positivos em termos de produtividade. Porém, os resultados do estudo de manutenção não mostraram benefícios durante o esforço de manutenção. Durante o planejamento dos experimentos, a preocupação foi evitar qualquer tipo de situação que favorecesse a ferramenta proposta neste trabalho. Acredita-se que isso aumente a confiabilidade dos resultados obtidos.

7.3 Limitações deste Trabalho

Foram identificadas algumas limitações neste trabalho. A primeira é a dependência de tecnologia acarretada ao usar Eclipse Modeling Framework e outras bibliotecas relacionadas que executam sobre a plataforma Java (ECLIPSE CONSORTIUM, 2011). Essas tecnologias foram empregadas para elaborar a especificação dos modelos e

desenvolver o editor e gerador de código, tornando essa ferramenta dependente das tecnologias.

O metamodelo foi especificado ao considerar frameworks implementados em AspectJ. Outras linguagens podem permitir abstrações diferentes que não são apoiadas pelo metamodelo e o gerador de código desenvolvido apenas gera códigos nesta linguagem.

Acredita-se que qualquer framework, não apenas transversal, que possa ser reusado com AspectJ utilizando-se de sobrescrita de métodos, conjuntos de junção e herança de unidades possa ser reusado com a abordagem proposta. Porém, não foi avaliado se todo framework transversal pode ser reusado com a abordagem e não foi identificado se a abordagem necessita de melhorias ou extensões.

Uma aplicação pode ser acoplada a vários frameworks transversais, porém, não foi tratado como reusar múltiplos frameworks com uma única aplicação base utilizando a abordagem proposta. Apesar desta funcionalidade já ser suportada, alguns frameworks podem conflitar entre si e acarretar resultados indesejados.

Também não foi analisada a representatividade gráfica dos formulários e o quanto isso auxilia os engenheiros de aplicação durante o reúso. Acredita-se que a grande similaridade dos formulários possa trazer confusões durante a aplicação da abordagem. Esses pontos incentivam esforços para melhorias de representatividade.

O foco deste trabalho concentra-se na engenharia de aplicação. Não foi identificado se engenheiros de domínio podem ser dificuldades ao criar modelos de requisitos de reúso de forma correta.

As limitações relacionadas aos experimentos conduzidos neste trabalho são descritos na Seção 6.4.

7.4 Publicações Realizadas neste Trabalho

Até o momento de escrita desta seção, foram confirmadas quatro publicações com conteúdo produzido durante o desenvolvimento deste trabalho. A proposta original deste trabalho foi publicada no “Workshop de Teses e Dissertações do Congresso Brasileiro de Software 2011” (GOTTARDI; PENTEADO; CAMARGO, 2011b). O processo de reúso e, naquele momento, a ferramenta ainda como um protótipo foi publicada no “V Latin American Workshop on Aspect-Oriented Software” (GOTTARDI; PENTEADO; CAMARGO, 2011a). Uma versão estável da ferramenta de reúso e o experimento de reúso foram publicados no “International Conference on Enterprise Information Systems 2012” (GOTTARDI; PASTOR; CAMARGO, 2012). Por fim, na última publicação confir-

mada, é descrito o metamodelo em conjunto com a ferramenta e os dois experimentos realizados neste trabalho, publicada no “Simpósio Brasileiro de Engenharia de Software” (GOTTARDI et al., 2012). Além disso, a ferramenta CrossFIRE foi publicada no CBSoft Tools 2012 (DURELLI; GOTTARDI; CAMARGO, 2012a).

7.5 Sugestões de Trabalhos Futuros

No momento da escrita desta dissertação, algumas atividades são consideradas para trabalhos futuros. A primeira delas é analisar o uso da abordagem em conjunto com outros frameworks transversais e não transversais considerando o esforço de reúso e manutenção.

Também é planejada a avaliação do nível de abstração dos elementos de modelo, para apurar se estes são suficientes e ideais para representar os conceitos durante o reúso. Outro objeto de estudo é o esforço exigido de engenheiros de domínio ao empregar a abordagem, pelo fato que a abordagem proposta neste trabalho teve um foco maior em facilitar a engenharia de aplicação.

Com base na análise da representatividade dos elementos de modelo, pode ser necessário efetuar melhorias gráficas nos formulários propostos.

Por fim, considerando a divisão das metaclasses do metamodelo em concretas e abstratas, resta estudar a reusabilidade das metaclasses abstratas para outros contextos de reúso de software e representação gráfica de documentação.

REFERÊNCIAS

ANDRADE, C.; SANTOS, A.; BORBA, P. AspectH: Uma extensão orientada a aspectos de Haskell. In: *WASP' 04 - workshop do 18º SBES . Brasília, Brasil.* [S.l.: s.n.], 2004.

ANTKIEWICZ, M.; CZARNECKI, K. Framework-specific modeling languages with round-trip engineering. In: SPRINGER-VERLAG. *ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS)*. Genova, Italy: Springer-Verlag, 2006. p. 692–706. Disponível em: <http://www.springerlink.com/content/y08152212701l160/fulltext.pdf>.

ARACIC, I. et al. An overview of CaesarJ. In: *Transactions on Aspect-Oriented Software Development I*. Springer, 2006. (Lecture Notes in Computer Science, v. 3880), p. 135–173. Disponível em: <http://dx.doi.org/10.1007/11687061_5>.

ASPECTJ TEAM. *The AspectJ(TM) Programming Guide*. 2003. Acessado em 20 de fevereiro de 2012. Disponível em: <http://www.eclipse.org/aspectj/doc/released/progguide/index.html>.

BERG, K. van den; CONEJERO, J. M.; CHITCHYAN, R. *AOSD Ontology 1.0 - Public Ontology of Aspect-Orientation*. maio 2005. Disponível em: <http://doc.utwente.nl/64104/1/BergConChi2005.pdf>.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *UML: guia do usuário*. Elsevier, 2006. ISBN 9788535217841. Disponível em: <http://books.google.com.br/books?id=ddWqxcDKGF8C>.

BRAGA, R.; MASIERO, P. Building a wizard for framework instantiation based on a pattern language. In: KONSTANTAS, D. et al. (Ed.). *Object-Oriented Information Systems*. Springer Berlin / Heidelberg, 2003, (Lecture Notes in Computer Science, v. 2817). p. 95–106. ISBN 978-3-540-40860-4. 10.1007/978-3-540-45242-3_10. Disponível em: <http://dx.doi.org/10.1007/978-3-540-45242-3_10>.

BRAGA, R. T. V.; GERMANO, F. S. R.; MASIERO, P. C. A pattern language for business resource management. In: *In Proceedings of the 6th Pattern Languages of Programs Conference (PLoP'99*. [S.l.: s.n.], 1999. p. 1–33.

BYNENS, M. et al. Towards reusable aspects: The mismatch problem. In: *Workshop on Aspect, Components and Patterns for Infrastructure Software (ACP4IS'10)*. [S.l.: s.n.], 2010. p. 17–20.

CAMARGO, V.; MASIERO, P. Frameworks orientados a aspectos. In: *Anais Do 19º Simpósio Brasileiro De Engenharia De Software (SBES'2005), Uberlândia-MG, Brasil, Outubro.* [S.l.: s.n.], 2005.

CAMARGO, V.; MASIERO, P. A pattern to design crosscutting framework families. In: *ACM Annual Symposium On Applied Computing (ACM-SAC), Fortaleza, Brasil*. [S.l.: s.n.], 2008.

CAMARGO, V. V. *Frameworks Transversais: Definições, Classificações e Utilização em um Processo de Desenvolvimento.* Tese (Doutorado) — Tese de Doutorado. Instituto de Ciências Matemáticas e de Computação, USP, São Carlos, 2006.

CECHTICKY, V. et al. A generative approach to framework instantiation. In: *Proceedings of the 2nd international conference on Generative programming and component engineering*. New York, NY, USA: Springer-Verlag New York, Inc., 2003. (GPCE '03), p. 267–286. ISBN 3-540-20102-5. Disponível em: <http://portal.acm.org/citation.cfm?id=954186.954203>.

CLEMENTS, P.; NORTHROP, L. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002. (SEI Series in Software Engineering). ISBN 9780201703320. Disponível em: <http://books.google.com.br/books?id=tHGFQgAACAAJ>.

CONSTANTINIDES, C. A.; ELRAD, T. Towards a two-dimensional separation of concerns (poster session). In: *OOPSLA '00: Addendum to the 2000 proceedings of the conference on Object-oriented programming, systems, languages, and applications (Addendum)*. New York, NY, USA: ACM, 2000. p. 63–64. ISBN 1-58113-307-3.

CUNHA, C.; SOBRAL, J.; MONTEIRO, M. Reusable aspect-oriented implementations of concurrency patterns and mechanisms. In: *Aspect-Oriented Software Development Conference (AOSD'06)*. Bonn, Germany: [s.n.], 2006.

DURELLI, R. S.; GOTTARDI, T.; CAMARGO, V. V. Crossfire: An infrastructure for storing crosscutting framework families and supporting their model-based reuse. In: *CBSoft2012 - Tools 2012*. [S.l.: s.n.], 2012.

DURELLI, R. S.; GOTTARDI, T.; CAMARGO, V. V. *Tutorial Cross-FIRE*. 2012. Acessado em 30 de Maio de 2012. Disponível em: <http://www2.dc.ufscar.br/ valter/crossfire.html>.

ECLIPSE CONSORTIUM. *Xtend Documentation*. May 2010. Disponível em: <http://www.eclipse.org/Xtext/documentation/2_0_0/01-Xtend_Introduction.php>.

ECLIPSE CONSORTIUM. *Graphical Modeling Framework, version 1.5.0*. 2011. Disponível em: <http://www.eclipse.org/modeling/gmp/>.

ECLIPSE CONSORTIUM. *Eclipse Integrated Development Environment, version 4.2*. 2012. Disponível em: <http://www.eclipse.org/>.

EVERMANN, J. A meta-level specification and profile for aspectj in UML. In: *AOM '07: Proceedings of the 10th international workshop on Aspect-oriented modeling*. New York, NY, USA: ACM, 2007. p. 21–27. ISBN 978-1-59593-658-5.

FRANCE, R.; RUMPE, B. Model-driven development of complex software: A research roadmap. In: *2007 Future of Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007. (FOSE '07), p. 37–54. ISBN 0-7695-2829-5. Disponível em: <http://dx.doi.org/10.1109/FOSE.2007.14>.

FREE SOFTWARE FOUNDATION, INC. *R*. December 2011. Http://www.r-project.org/.

GAMMA, E. et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1. ed. [S.l.]: Addison-Wesley Professional, 1994. Hardcover. ISBN 0201633612.

GOMMA, H. *Designing Software Product Lines With UML – From Use Case to Pattern-Based Software Architectures*. 1ª. ed. [S.l.]: Addison Wesley, 2004.

GONG, W. M.; JACOBSEN., H. A. *AspeCt-oriented C Specification (v0.8). Working Technical Draft*. January 2008.

GOTTARDI, T. et al. Model-based reuse for crosscutting frameworks: Assessing reuse and maintainability effort. In: *Simpósio Brasileiro de Engenharia de Software – SBES*. Natal,RN, Brasil: SBC, 2012. (CBSoft 2012).

GOTTARDI, T.; PASTOR, O. L.; CAMARGO, V. V. A model-based approach for reusing crosscutting frameworks. In: *International Conference on Enterprise Information Systems – ICEIS*. Wroclaw, Poland: INSTICC, 2012. (ICEIS 2012).

GOTTARDI, T.; PENTEADO, R.; CAMARGO, V. V. A model based process to support the reuse of aspect-oriented frameworks. In: *Proc. of the V Latin American Workshop on Aspect-Oriented Software Development, LA-WASP'2011*. São Paulo, SP, Brazil: SBC, 2011. (LA-WASP 2012).

GOTTARDI, T.; PENTEADO, R.; CAMARGO, V. V. Reuso de frameworks transversais com apoio de modelos. In: *I Workshop de Teses e Dissertações do CBSoft*. São Paulo, SP, Brazil: SBC, 2011. (WTDSoft 2011).

GRISWOLD, W. G. et al. Modular software design with crosscutting interfaces. *IEEE Softw.*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 23, n. 1, p. 51–60, jan. 2006. ISSN 0740-7459. Disponível em: <http://dx.doi.org/10.1109/MS.2006.24>.

HAN, Y.; KNIESEL, G.; CREMERS, A. Towards visual AspectJ by a meta model and modeling notation. In: *6^{th} Aspect-Oriented Modeling (AOM), AOSD*. [S.l.: s.n.], 2005.

HUANG, M.; WANG, C.; ZHANG, L. Towards a reusable and generic aspect library. In: *Workshop of the Aspect Oriented Software Development Conference at AOSDSEC'04*. Lancaster, UK: [s.n.], 2004.

JOHNSON, R. E. Documenting frameworks using patterns. In: *conference proceedings on Object-oriented programming systems, languages, and applications*. New York, NY, USA: ACM, 1992. (OOPSLA '92), p. 63–76. ISBN 0-201-53372-3. Disponível em: <http://doi.acm.org/10.1145/141936.141943>.

JSR 224 EXPERT GROUP. *JSR 224: Java(TM) API for XML-Based Web Services (JAX-WS) 2.0*. May 2009. Disponível em: <http://www.jcp.org/en/jsr/detail?id=224>.

JSR 317 EXPERT GROUP. *JSR 317: Java(TM) Persistence 2.0*. May 2009. Disponível em: <http://www.jcp.org/en/jsr/detail?id=317>.

KASTNER, C. et al. Featureide: A tool framework for feature-oriented software development. In: *Proceedings of the 31st International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2009. (ICSE '09), p. 611–614. ISBN 978-1-4244-3453-4. Disponível em: <http://dx.doi.org/10.1109/ICSE.2009.5070568>.

KICZALES, G. et al. An overview of AspectJ. In: . [S.l.]: Springer-Verlag, 2001. p. 327–353.

KICZALES, G. et al. Aspect-oriented programming. In: *ECOOP*. [S.l.]: Springer-Verlag, 1997.

KISELEV, I. *Aspect-Oriented Programming with AspectJ*. 1. ed. [S.l.]: Sams, 2002. 288 p.

KULESZA, U. et al. Improving extensibility of object-oriented frameworks with aspect-oriented programming. In: *Proc. of the 9th Intl Conf. on Software Reuse (ICSR'06)*. [S.l.: s.n.], 2006. p. 231–245.

LADDAD, R. *AspectJ in Action: Practical Aspect-Oriented Programming*. [S.l.]: Manning Publications, 2003. 512 p.

LAZANHA, R. et al. Uma arquitetura de referência baseada em papéis para frameworks transversais de persistência: Uma análise quantitativa. In: *XXXVI Clei – Conferência Latino-Americana de Informática*. Assunção, Paraguay: [s.n.], 2010.

MORTENSEN, M.; GHOSH, S. Creating pluggable and reusable non-functional aspects in AspectC++. In: *Proceedings of the Fifth AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software*. [S.l.: s.n.], 2006.

MORTENSEN, M.; GHOSH, S. Using aspects with object-oriented frameworks. In: *Proceedings of the Aspect-Oriented Software Development Conference – industry track*. Bonn, Alemanha: [s.n.], 2006.

OLIVEIRA, T. C.; ALENCAR, P.; COWAN, D. Reusetool-an extensible tool support for object-oriented framework reuse. *J. Syst. Softw.*, Elsevier Science Inc., New York, NY, USA, v. 84, n. 12, p. 2234–2252, dez. 2011. ISSN 0164-1212. Disponível em: <http://dx.doi.org/10.1016/j.jss.2011.06.030>.

OLIVEIRA, T. C. et al. Rdl: A language for framework instantiation representation. *J. Syst. Softw.*, Elsevier Science Inc., New York, NY, USA, v. 80, p. 1902–1929, November 2007. ISSN 0164-1212. Disponível em: <http://portal.acm.org/citation.cfm?id=1290192.1290220>.

OMG. *Overview and guide to* OMG*'s architecture*. May 2010. Disponível em: <http://www.omg.org/cgi-bin/doc?omg/03-06-01>.

OMG. *UML OCL 2.0 Adopted Specification.* May 2010. Disponível em: <http://www.omg.org/docs/ptc/03-10-14.pdf>.

OMG. *Unified Modeling Language Infrastructure Specification*. May 2010. Disponível em: <http://www.omg.org/spec/UML/2.3/Infrastructure/PDF/>.

ORACLE CORPORATION. *MySQL Community Server 5.5.27*. May 2012. Disponível em: <http://www.mysql.com/>.

PASTOR, O.; MOLINA, J. C. *Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling*. Secaucus, NJ, USA: Springer-Verlag New York, 2007. ISBN 3540718672.

PREE, W. et al. Building application frameworks: Object-oriented foundations of framework design. In: ____. [S.l.]: John Willey and Sons, 1999. cap. Hot-Spot-Driven Development, p. p. 379–393.

RAJAN, H.; LEAVENS, G. T. Ptolemy: A language with quantified, typed events. In: *Proceedings of the 22nd European conference on Object-Oriented Programming*. Berlin, Heidelberg: Springer-Verlag, 2008. (ECOOP '08), p. 155–179. ISBN 978-3-540-70591-8.

SAKENOU, D. et al. Patterns for re-usable aspects in object teams. In: *Net Object Days*. Erfurt: [s.n.], 2006.

SANTOS, A. L.; KOSKIMIES, K.; LOPES, A. Automated domain-specific modeling languages for generating framework-based applications. *Software Product Line Conference, International*, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 149–158, 2008.

SCHMIDT, D. C. Model-driven engineering. *IEEE Computer*, v. 39, n. 2, February 2006. Disponível em: <http://www.truststc.org/pubs/30.html>.

SHAH, V.; HILL, V. An aspect-oriented security framework: Lessons learned. In: *Proceedings of AOSDSEC'04 (AOSD Technology for Application-Level Security). Workshop of the Aspect Oriented Software Development Conference*. Lancaster, UK: [s.n.], 2004.

SOARES, S.; LAUREANO, E.; BORBA, P. Distribution and persistence as aspects. *Software: Practice and Experience*, v. 33, n. 7, p. 711–759, 2006.

SOUDARAJAN, N.; KHATCHADOURIAN, R. Specifying reusable aspects. In: *Asian Workshop on Aspect-Oriented and Modular Software Development (AOAsia'09)*. [S.l.: s.n.], 2009.

SPINCZYK, O.; GAL, A.; SCHRöDER-PREIKSCHAT, W. AspectC++: an aspect-oriented extension to the C++ programming language. In: *Proceedings of the Fortieth International Conference on Tools P. Sydney, Australia*. [S.l.: s.n.], 2002.

TARR, P.; OSSHER, H.; SUTTON, S. Hyper/J (tm) : Multi-dimensional separation of concerns for Java. In: *Proc. of the 24th International Conference on Software Engineering. Orlando, Florida*. [S.l.: s.n.], 2002.

WICHMAN, J. C. *ComposeJ: The Development of a Preprocessor to Facilitate Composition Filters in the Java Language*. Dissertação (Mestrado) — Department of Computer Science, University of Twente, 1999.

WOHLIN, C. et al. *Experimentation in software engineering: an introduction*. Norwell, MA, USA: Kluwer Academic Publishers, 2000. ISBN 0-7923-8682-5.

ZANON, I.; CAMARGO, V. V.; PENTEADO, R. A. D. Reestructuring an application framework with a persistence crosscutting framework. *INFOCOMP*, v. 1, p. 9–16, 2010.

Apêndice A

DETALHAMENTO DO METAMODELO PROPOSTO

Este apêndice foi elaborado para detalhar o metamodelo proposto nesse trabalho, o qual é usado para definir os modelos propostos.

Esse metamodelo foi desenvolvido utilizando-se a metalinguagem ECore, uma linguagem própria para criação de metamodelos. A implementação do metamodelo foi realizada utilizando ferramentas de desenvolvimento Eclipse Modeling Framework (ECLIPSE CONSORTIUM, 2011).

Similarmente ao MOF (OMG, 2010c), o ECore permite a definição de uma linguagem de modelagem por meio de modelos. No metamodelo criado neste projeto, são empregadas metaclasses e enumerações ECore. As metaclasses são caixas com quatro compartimentos, no primeiro é contido o nome da metaclasse, no segundo são contidas propriedades, no terceiro são contidas meta-operações e no último são contidas anotações. Neste projeto, foram empregadas apenas os dois primeiros compartimentos.

As enumerações são caixas com dois compartimentos. No primeiro compartimento é contido o nome da enumeração e no segundo são contidos valores literais válidos para a enumeração. Também são empregadas generalizações, que são setas que indicam que uma metaclasse herda atributos da metaclasse apontada pela seta.

Na Figura 52 está representado o diagrama do metamodelo proposto. Este diagrama possui todos os dezenove elementos do metamodelo que são representados graficamente como caixas.

O metamodelo proposto possui três enumerações:

1. *SuperType*: utilizado para definir o tipo do do elemento a ser estendido, que pode ser aspecto, classe ou interface;
2. *CompositionType*: utilizado para definir tipos de composição lógica entre dois ou mais elementos, que pode ser “e”, “ou” ou “ou exclusivo”;

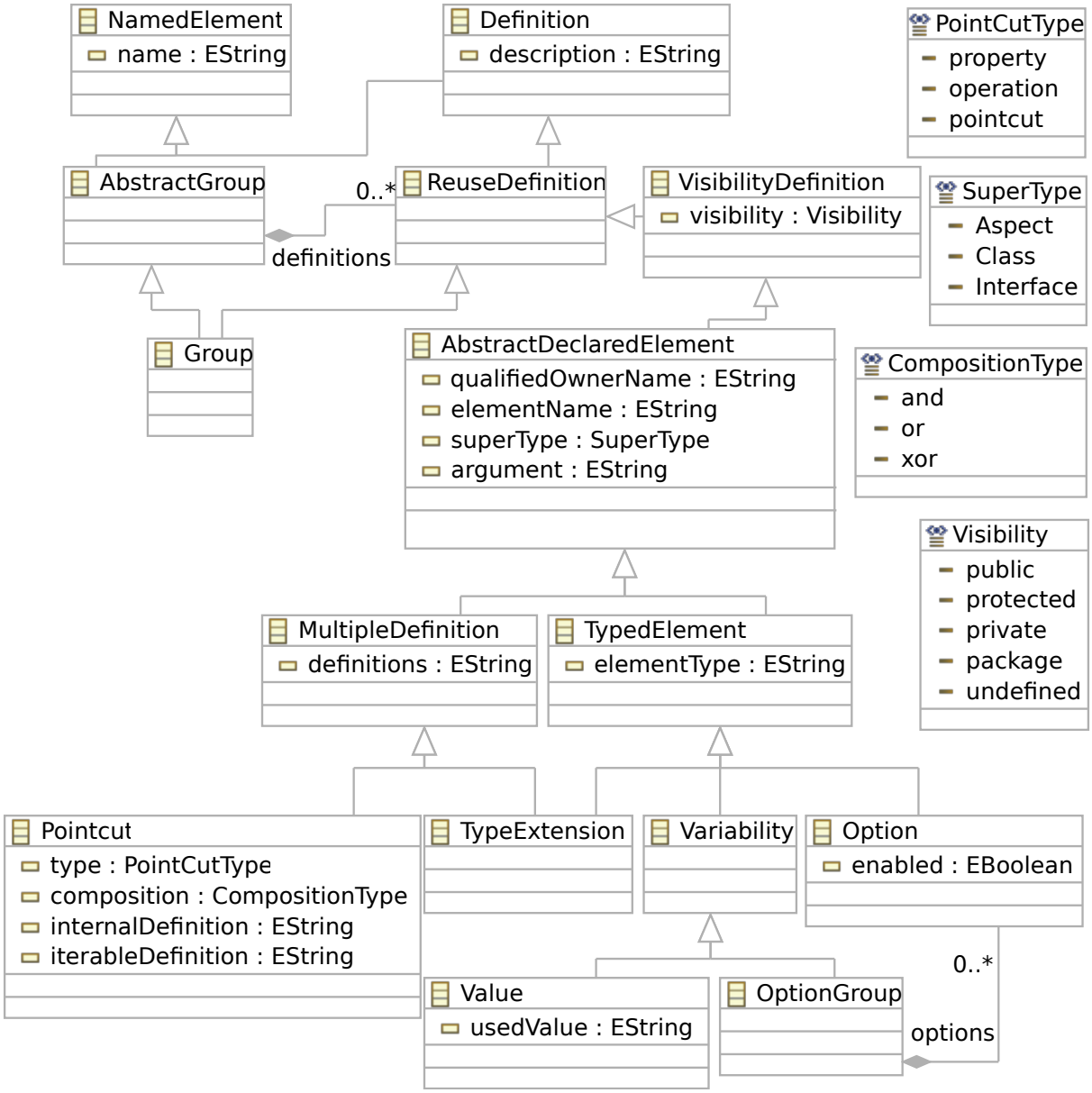

Figura 52: Metamodelo Proposto

3. *Visibility*: utilizado para indicar o nível de visibilidade de um elemento a ser estendido, são possíveis, público, protegido, privado, por pacote e indefinido.

No metamodelo há quinze metaclasses. Dentre os elementos, existem elementos concretos e abstratos. Os elementos abstratos são:

1. *NamedElement*: metaclasse que define o atributo nome para todos os elementos que possuem nome;

2. *Definition*: metaclasse que define o atributo de descrição utilizado por todos elementos que definem informação pertinente ao reúso;

3. *AbstractGroup*: metaclasse que define elementos que agrupam definições pre-

sentes em um modelo;

4. *ReuseDefinition*: metaclasse que define todos os tipos de definições de serem inseridas dentro de um modelo;

5. *VisibilityDefinition*: metaclasse que define o nível de visibilidade de definições de reúso;

6. *AbstractDeclaredElement*: metaclasse que define os seguintes atributos:

   (a) o nome do proprietário do elemento do código do Framework Transversal a ser estendido ou referenciado durante o reúso, por exemplo, o nome completo de uma classe;

   (b) o nome do elemento do Framework Transversal a ser estendido ou referenciado durante o reúso, por exemplo, um método ou conjunto de junção;

   (c) atributo que define se o tipo a ser estendido é uma classe, aspecto ou interface;

   (d) definição de um argumento a ser passado durante o reúso.

7. *MultipleDefinition*: metaclasse que define tipos de definição que permitem definições compostas, por exemplo, um conjunto de junção composto ou um conjunto de classes que estendem uma única superclasse.

8. *TypedElement*: metaclasse herdada por metaclasses utilizadas em definições de reúso que possuem tipos, como definição de valores constantes, definição de classes e opções;

9. *Variability*: metaclasse que define variabilidades do framework, estabelecidas pela definição de valores constantes ou seleção de opções.

No modelo final, apenas o elementos concretos são representados. Portanto, nos modelos propostos é possível representar elementos do tipo “PointCut”, “TypeExtension”, “Value”, “OptionGroup” e “Group”:

1. *PointCut*: elemento utilizado para representar definição de conjuntos de junção, como por exemplo, nome de métodos da aplicação base que serão entrecortados;

2. *TypeExtension*: elemento utilizado para definir quais classes, aspectos e interfaces da aplicação base herdarão de classes do framework. Por exemplo, isso pode ser utilizado para estabelecer quais classes representam objetos persistentes ao acoplar com um framework transversal de persistência;

3. *Value*: elemento utilizado para definir valores constantes necessárias para melhor adequar o framework às necessidades da aplicação final. Por exemplo, valores podem incluir nomes de úsuário e senha número de *pools*, valores úteis ao realizar a conexão a um banco de dados;

4. *OptionGroup*: elemento utilizado para definir seleção de uma opção (*Option*) dentro de uma lista de elementos possíveis. Isso pode ser utilizado por exemplo, para selecionar a estratégia de atualização dos objetos persistentes no banco de dados.

Apêndice B

Sistema de Captura dos Dados

Um sistema de informação foi implementado para capturar os dados dos experimentos. Esse sistema permitiu uma coleta de grande quantidade de dados e minimizou interrupções dos participantes.

O sistema opera em uma arquitetura cliente-servidor, em que o servidor acumula todas as informações enviadas pelas máquinas cliente, as quais são utilizadas pelos participantes do experimento.

Para que as máquinas cliente enviassem os dados, foram inseridos aspectos nos códigos em que os participantes trabalharam. Esses aspectos inseriam comportamento para transmitir os dados de forma transparente, sem interferir com o andamento do processo dos participantes. Dessa forma, os dados foram submetidos ao servidor antes que a notificação de sucesso, conforme a visível na Figura 53, fosse visível pelos participantes.

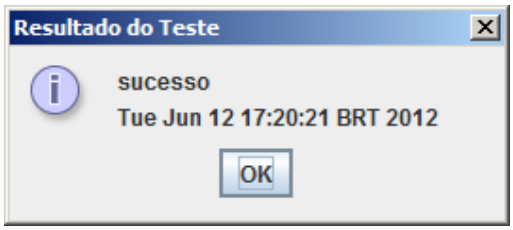

Figura 53: Notificação de Sucesso

Os participantes apenas foram requisitados a realizar um cadastro preliminar para que fosse possível a identificação. Na Figura 54 é visível o formulário que os participantes precisaram preencher no momento do início de suas participações.

Os arquivos de código editados pelos participantes não possuíam qualquer informação sobre esse sistema de captura. Eles apenas sabiam que dados estavam sendo coletados, mas não tinham contato maior com esse interesse.

Esse sistema foi inicialmente planejado com o objetivo de coletar os dados de tempo para completar os processos de reúso e manutenção desde a fase piloto e

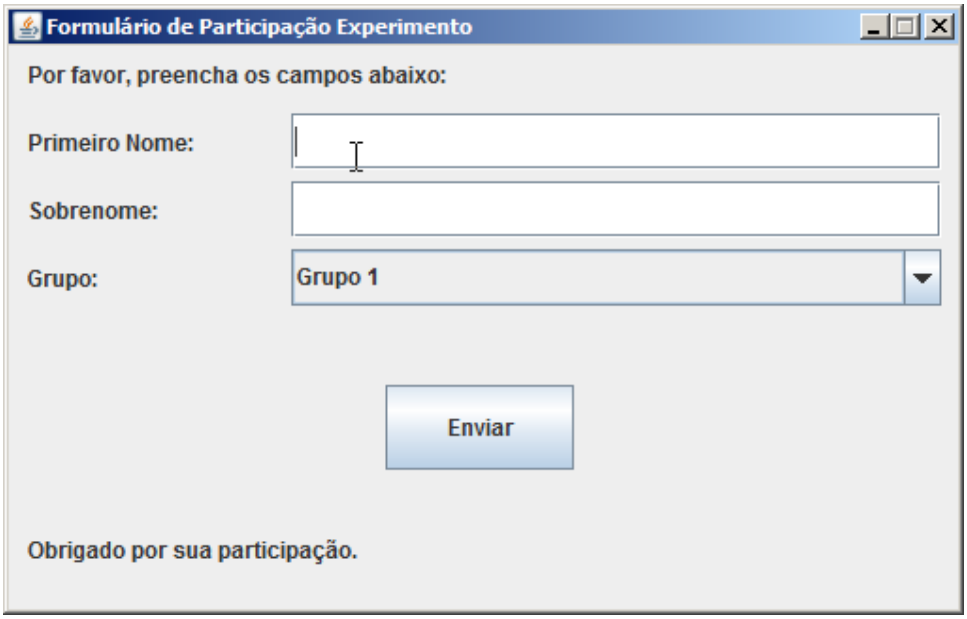

Figura 54: Formulário de Participação

prover informações pertinentes ao balanceamento de grupos em tempo real.

Além disso, o sistema capturou os dados de tempo com precisão de milissegundos, com base nos relógios dos clientes, e com base no relógio do servidor. Os dados do relógio do servidor foram utilizados na maior parte dos cálculos, o que permite uma melhor organização dos dados, porém, este não desconsidera os tempos de atraso de transmissão desses valores para o servidor. Em questão precisão dos dados, acredita-se que esse atraso é irrelevante, pois, todas as máquinas estavam em rede local de baixa latência, o que torna o atraso pequeno e proporcional a todas. Por exemplo, na Figura 55, há uma página de visualização de dados do sistema.

Visões

Ultimas 10 Execuções

Participantes

Nome e Grupo de Participantes

Processos (Geral)

Estatísticas dos Processos Tempo Bruto

Estatísticas dos Processos Removendo Pausa

Query: Select * from estatisticafinalreal Enviar

Inicio com Select é obrigatório.As seguintes sequências são proibidas: --,/*,*/, ;

N°	Atividade	Grupo	Aplicação	Técnica	Menor Tempo	Maior Tempo	Média de Tempo
1	Real	Grupo 1	Voos	Modelos	00:04:19.952015	00:20:03.920754	00:09:36.026098
2	Real	Grupo 2	Entregas	Modelos	00:05:24.249952	00:08:02.570996	00:06:33.934953
3	Real	Grupo 2	Voos	Convencional	00:08:42.389884	00:23:10.72776	00:14:11.312371
4	Real	Grupo 1	Entregas	Convencional	00:10:25.359836	00:41:29.414342	00:18:15.56779

4 registros

Figura 55: Exemplo de Visualização de Dados Estatísticos

No exemplo da Figura 55 é exibida uma visão dos dados com cálculos estatísticos do primeiro experimento de reúso, no caso, referenciada como “Real”. A tabela dentro da figura inclui valores referentes ao maior e menor tempo para concluir o processo, média de tempo, soma de tempo, variância e desvio padrão do tempo.

Dessa forma, ao analisar os dados apresentados na Figura 55, foi estudada a necessidade de rebalancear os grupos para a fase seguinte.

Apêndice C

DOCUMENTOS DO EXPERIMENTO

Durante os experimentos descritos no Capítulo 6, foram utilizados diversos documentos que são fornecidos dentro deste capítulo de apêndice.

C.1 Preparação

Nesta seção são providos os documentos utilizados durante a preparação dos experimentos. Na Figura 56 é mostrado o formulário de consentimento e na Figura 57 é mostrado o formulário de caracterização.

C.2 Treinamento e Manuais

Na Figura 58 é mostrado o sumário do framework transversal empregado no experimento, documento utilizado durante o treinamento.

Da Figura 59 à 61 está presente o manual de reúso do framework na forma convencional, documento utilizado em todas as fases do experimento de reúso quando a forma convencional foi empregada.

Da Figura 62 à 67 está presente o manual de reúso do framework na forma dirigida por modelos, documento utilizado em todas as fases do experimento de reúso quando a forma dirigida por modelos foi empregada.

Formulário de Consentimento

– Experimento
Este experimento visa realizar um estudo que tem como objetivo avaliar a efetividade, em termos de custo e eficácia do emprego de desenvolvimento dirigido por modelos no contexto de reúso de frameworks orientados a aspectos.

– Idade
Eu declaro ser maior de 18 (dezoito) anos de idade e concordar em participar do experimento conduzido por Thiago Gottardi na Universidade Federal de São Carlos (UFSCar).

– Procedimento
Este experimento ocorrerá em duas semanas, que incluirá o entendimento e execução dos processos reuso convencional e dirigido a modelos para frameworks transversais, a fim de comparar a efetividade, em termos de custo e eficácia entre os dois processos apresentados. Eu entendo que, uma vez que o experimento tenha sido concluído, os trabalhos que desenvolvi, bem como os dados coletados, serão estudados.

– Confidencialidade
Estou ciente de que toda informação coletada neste experimento é confidencial, e meu nome ou quaisquer outros meios de identificação não serão divulgados. Da mesma forma, me comprometo a não comunicar meus resultados aos demais participantes e/ou a outros grupos enquanto não terminar o experimento, bem como manter sigilo das técnicas e documentos apresentados que fazem parte do experimento.

– Benefícios e liberdade de desistência
Eu entendo que os benefícios que receberei deste experimento se limitam ao aprendizado do material que é distribuído e apresentado. Eu compreendo que sou livre pare realizar perguntas a qualquer momento, solicitar que qualquer informação relacionada à minha pessoa seja incluída no experimento, ou comunicar minha desistência de participação. Eu entendo que participo livremente com o único intuito de contribuir para o avanço e desenvolvimento de técnicas e processos para a Engenharia de Software.

– Pesquisador responsável
Thiago Gottardi - Programa de Pós-Graduação em Ciência da Computação

– Professor Responsável
Prof. Dr. Valter Vieira de Camargo - Programa de Pós-Graduação em Ciência da Computação.

Ao preencher e assinar este formulário, dou plena ciência e consentimento com os termos acima expostos.

Nome (em letra de forma): _____

Assinatura:_____

Data:__/__/____

Figura 56: Formulário de Consentimento

Nome:_____

Respostas:__,__,__,__,__,__,__,__.**Soma:**_____

Formulário de Caracterização

Responda todas as questões com a maior sinceridade possível. Os dados são considerados confidenciais e não serão divulgados de forma alguma.
Esses dados também não serão utilizados como avaliação na disciplina.

1) Qual é a sua experiência com a construção de pointcuts **call** e **execution**
(**SEM** necessidade de compor com **args**, **target**, **this**, **cflow**, **set**, **get**, etc.)?
1.1 – Utilizei apenas em aula;
1.2 – Já havia usado antes da disciplina, poucas vezes;
1.3 – Uso (ou utilizei recentemente) várias vezes;
1.4 – Uso com frequência.

2) Qual é a sua experiência com a construção de **declare parents**?
2.1 – Utilizei apenas em aula;
2.2 – Já havia usado antes da disciplina, poucas vezes;
2.3 – Uso (ou utilizei recentemente) várias vezes;
2.4 – Uso com frequência.

3) Qual é a sua experiência com a construção de **entrecorte estático com métodos e atributos** (exemplo: definir métodos em um aspecto para serem inseridos em classes)?
3.1 – Utilizei apenas em aula;
3.2 – Já havia usado antes da disciplina, poucas vezes;
3.3 – Uso (ou utilizei recentemente) várias vezes;
3.4 – Uso com frequência.

4) Qual é a sua experiência com a construção de **sobrecarga de métodos** (como em Java)?
4.1 – Utilizei apenas em aula;
4.2 – Já havia usado antes da disciplina, poucas vezes;
4.3 – Uso (ou utilizei recentemente) várias vezes;
4.4 – Uso com frequência.

5) Já utilizou um Framework Transversal de forma convencional?
5.1 – Utilizei apenas em aula;
5.2 – Já havia usado antes da disciplina, poucas vezes;
5.3 – Uso (ou utilizei recentemente) várias vezes;
5.4 – Uso com frequência.

6) Já utilizou um Framework Transversal com apoio de modelos?
6.1 – Nunca utilizei;
6.2 – Utilizei uma vez;
6.3 – Já havia usado antes da disciplina, poucas vezes;
6.4 – Utilizei várias vezes.

7) Qual a sua experiência em ferramentas de modelagem do Eclipse (Ecore, EMF, GMF, GEF, Papyrus, etc)
7.1 – Nunca utilizei;
7.2 – Utilizei uma delas uma vez;
7.3 – Já havia uma delas, poucas vezes;
7.4 – Uso (ou utilizei recentemente) várias vezes;

8) Qual a sua experiência com o IDE Eclipse em geral
8.1 – Nunca utilizei;
8.2 – Utilizei apenas em aula.
8.3 – Já havia usado antes da disciplina, poucas vezes.
8.4 – Uso (ou utilizei recentemente) várias vezes;

Figura 57: Formulário de Caracterização

Sumário – Framework Transversal de Persistência

O objetivo do Framework Transversal (FT) de Persistência é persistir objetos em um banco de dados relacional.

Para instanciar o framework provido nos exercícios, é necessário especificar as seguintes informações:

1 – Detalhes de Conexão com o Banco de Dados

1.1 – Nome do Banco de Dados da Aplicação:

O nome do esquema de banco de dados criado especificamente para agrupar as tabelas do banco de dados da aplicação base.

1.2 – Nome do Sistema Gerenciador de Banco de Dados:

O nome do software gerenciador do banco de dados que o framework irá realizar a conexão.

1.3 – Classe Driver da Conexão com o Banco de Dados:

O nome da classe JDBC que provê suporte à conexão entre o aplicativo Java e o sistema gerenciador de Banco de Dados.

1.4 – Protocolo da Conexão com o Banco de Dados:

Informação provida ao JDBC utilizada durante a abertura da conexão ao sistema gerenciador do Banco de Dados.

2 – Pontos de abertura e fechamento de Conexão

2.1 – Pontos em que a Conexão Deve ser Aberta:

Devem ser definidos os pontos do código base que necessitam da conexão aberta.

2.2 – Pontos em que a Conexão Deve ser Fechada:

A conexão ao banco de dados é finalizada após os pontos informados nesse item, portanto, o ideal é informar os últimos pontos que necessitam da conexão aberta.

3 – Definir Classes de Objetos Persistentes

Os objetos persistentes são aqueles que são persistidos em tabelas no banco de dados. Cada classe que representa esses objetos deve ser informada.

Além disso, é obrigatório que essas classes possuam:

- Métodos get e set para todos os atributos salvos;
- Um atributo inteiro para a chave primária;
- Prover um método get “int getID()” para retornar a chave primária;
- Prover um método set “void setID(Integer)” para definir a chave primária.

Figura 58: Sumário do Framework Transversal Empregado no Experimento

Processo de Reúso Convencional do Framework Transversal de Persistência

Etapas do Processo

Início – **Registre o Tempo Inicial**
Execução – **Escreva o código necessário conforme as instruções abaixo;**
Término – **Execute o caso de teste, volte para execução até o resultado ser positivo.**

1 – Especificar Detalhes de Conexão com Banco

1.1 – Criar **um aspecto estendendo** o aspecto abstrato de nome:
`persistence.instantiation.helper.ExtendedConnectionVariabilities`

1.1) Informar o **Nome do Banco de Dados da Aplicação:**:
Para isso é necessário implementar o método
`public String ExtendedConnectionVariabilities.setDatabaseName ()`
e retornar uma String com a informação necessária.

Exemplo:
Caso o **Nome do Banco de Dados da Aplicação** seja "bancodedados", o método seria:

```
public String ExtendedConnectionVariabilities.setDatabaseName (){
        return "bancodedados";
}
```

1.2) Informar o **Nome do Sistema Gerenciador de Banco de Dados:**
Para isso é necessário implementar o método
`public String ExtendedConnectionVariabilities.setSpecificDatabase ()`
e retornar uma String com a informação necessária.

Exemplo:
Caso o **Nome do Sistema Gerenciador de Banco de Dados** seja "mysql", o método seria:

```
public String ExtendedConnectionVariabilities.setSpecificDatabase (){
        return "mysql";
}
```

1.3) Informar a **Classe Driver da Conexão com o Banco de Dados:**
Para isso é necessário implementar o método
`public String ExtendedConnectionVariabilities.getDriver ()`
e retornar uma String com a informação necessária.

Exemplo:
Caso a **Classe Driver da Conexão com o Banco de Dados** seja "com.mysql.jdbc.Driver", o método seria:

Figura 59: Manual - Convencional – Página 1

```
public String ExtendedConnectionVariabilities.getDriver (){
        return "com.mysql.jdbc.Driver";
}
```

1.4) Informar o **Protocolo da Conexão com o Banco de Dados:**
Para isso é necessário implementar o método
`public String ExtendedConnectionVariabilities.getJDBC ()`
e retornar uma String com a informação necessária.

Exemplo:
Caso o **Protocolo da Conexão com o Banco de Dados** seja "jdbc:mysql:", o método seria:

```
public String ExtendedConnectionVariabilities.getJDBC (){
        return "jdbc:mysql:";
}
```

---

2 - Especificar Pontos de Abertura e Fechamento de Conexão

2.1 - Criar um aspecto estendendo o aspecto abstrato de nome:
`persistence.instantiation.helper.ExtendedConnectionCompositionRules`

2.1 Informar os **Pontos em que a Conexão Deve ser Aberta :**
É necessário sobrescrever o pointcut
`public pointcut openConnection ()`
e definir os pontos em que a conexão deve ser aberta.
Exemplo:
Para definir que os métodos **ClasseBase.metodo1** e **ClasseBase.metodo2** necessitam de conexão aberta, utilize:

```
public pointcut openConnection ():
                execution (* ClasseBase.metodo1(..)) ||
                execution (* ClasseBase.metodo2(..));
```

2.2 Informar os **Pontos em que a Conexão Deve ser Fechada:**
É necessário sobrescrever o pointcut
`public pointcut closeConnection ()`
e definir os pontos em que a conexão deve ser fechada.

Exemplo:
Para definir que após os métodos **ClasseBase.metodo1** e **ClasseBase.metodo2** a conexão deve ser fechada, utilize:

```
public pointcut closeConnection ():
                execution (* ClasseBase.metodo1(..)) ||
                execution (* ClasseBase.metodo2(..));
```

---

3 - Definir Classes que representam Objetos Persistentes

Figura 60: Manual – Convencional – Página 2

3.1 - Para **Definir Classes de Objetos Persistentes**, é necessário **criar um aspecto** sem super classe e utilizar **declare parents para cada classe de objeto persistente** de forma a **implementar: persistence.PersistentRoot**

Exemplo:
Caso seja necessário definir a classe pacote.ClasseObjetoPersistente como Classe de Objeto Persistente, uma implementação válida seria a seguinte:

```
declare parents: pacote.ClasseObjetoPersistente  implements persistence.PersistentRoot;
```

---

4 - No Término do processo

4.1 – Execute o caso de teste

O caso de teste é importante para verificar se as etapas foram realizadas corretamente.

Se o caso de teste falhar, verifique os dados informados e repita a operação.

Se o caso de teste obtiver sucesso, o processo está finalizado.

Figura 61: Manual – Convencional – Página 3

Processo de Reuso do Framework Transversal de Persistência Apoiado por Modelos

Todos os itens descritos devem ser realizados de forma completa.

Atenção: Antes de começar execute o registro de tempo correspondente.

Etapas do Processo

Início – Registre o Tempo Inicial
Execução –
1– Preencha o diagrama de acordo com as instruções;
2– Execute a geração de código;
3– Execute a validação de código gerado;
Término – Execute o caso de teste, volte para execução até obter sucesso.

Somente prossiga para a próxima etapa se a corrente obtiver sucesso.
Ao identificar erro, corrija o modelo e realize sequencialmente etapas a partir da geração de código até o caso de teste até obter total sucesso.

---

1 – Especificar Detalhes de Conexão com Banco

1 - Preencha as seguintes caixas para definir a **Informação Necessária para a Conexão com o Banco de Dados.**

1.1) Na caixa **Nome do Banco de Dados da Aplicação,**
Preencha a informação necessária.

Exemplo:
Caso o **Nome do Banco de Dados da Aplicação** seja "bancodedados", então o elemento de diagrama terá configuração igual à Figura 1.

Figura 1 – Nome do Banco de Dados

1.2) Na caixa **Nome do Sistema Gerenciador de Banco de Dados,**
Preencha a informação necessária.

Exemplo:
Caso o **Nome do Sistema Gerenciador de Banco de Dados** seja "mysql", então

Figura 62: Manual – Modelos – Página 1

o elemento de diagrama terá configuração igual à Figura 2.

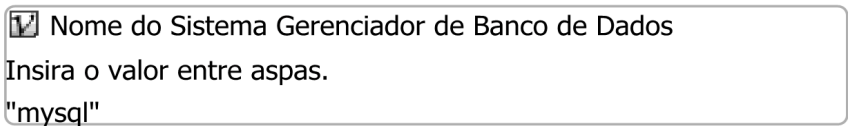

Figura 2 – Nome do Sistema Gerenciador de Banco de Dados

1.3) Na caixa **Classe Driver da Conexão com o Banco de Dados:**
Preencha a informação necessária.

Exemplo:
Caso a string do Classe Driver da Conexão com o Banco de Dados com o banco de dados seja **"com.mysql.jdbc.Driver"**, o elemento de modelo será igual à Figura 3.

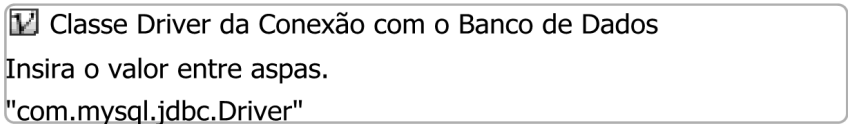

Figura 3 – Classe Driver da Conexão com o Banco de Dados

1.4) Na caixa **Protocolo da Conexão com o Banco de Dados:**
Preencha a informação necessária.

Exemplo:
Caso a string do Protocolo da Conexão com o Banco de Dados seja **"jdbc:mysql:"**, o elemento de modelo será igual à Figura 4.

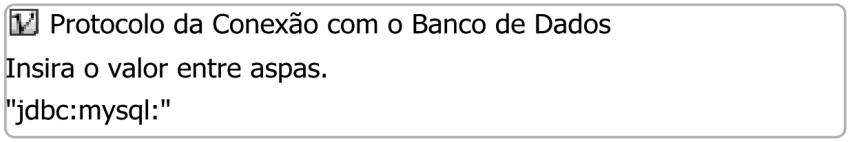

Figura 4 – Protocolo da Conexão com o Banco de Dados

---

2 - Especificar Pontos de Abertura e Fechamento de Conexão

2 - Preencha as seguintes caixas para definir os **Pontos de abertura e fechamento de Conexão** preencha as caixas a seguir:

Importante: Por causa de um bug na biblioteca de diagrama, recomenda-se utilizar o botão "..." de *Definitions*, conforme figura a seguir, Figura 5.

Figura 63: Manual – Modelos – Página 2

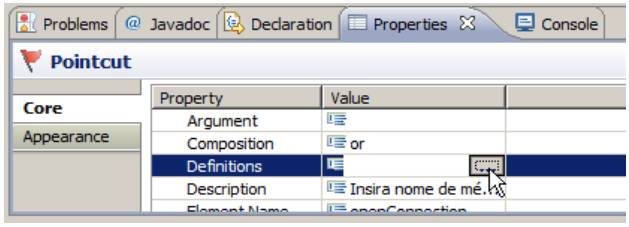

Figura 5 – Botão de *Definitions*

Utilize a janela a seguir, preencha o valor no campo Value, como visto na Figura 6.

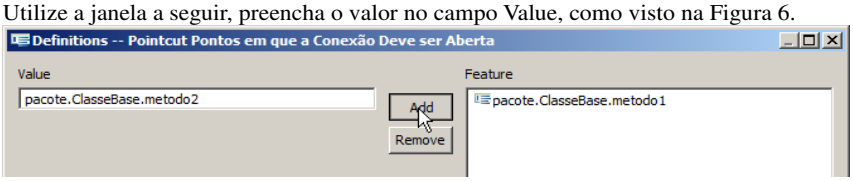

Figura 6 – Preenchimento e Add

E utilize Add (ou tecle Enter) até que todos os valores sejam passados para a direita. Conclua a definição com OK, conforme Figura 7.

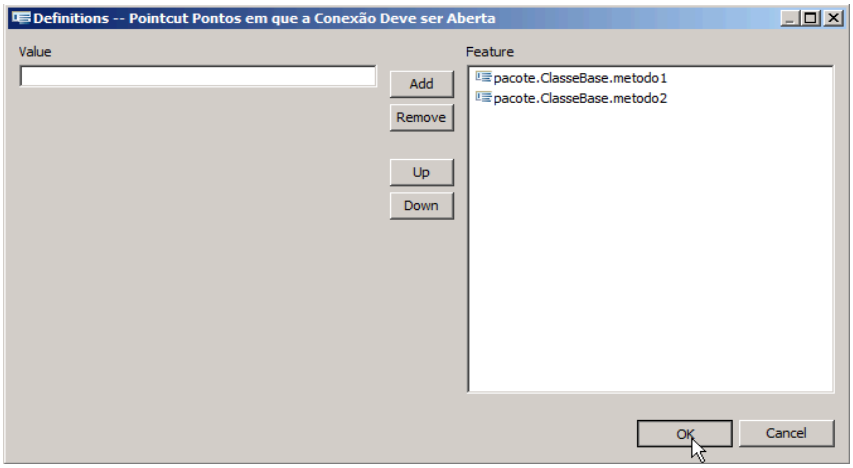

Figura 7 – Finalização de Definição

2.1 Na caixa **Pontos em que a Conexão Deve ser Aberta:**
Preencha a informação necessária.

> **Exemplo:**
> Caso o valores de **Pontos em que a Conexão Deve ser Aberta sejam os métodos pacote.ClasseBase.metodo1 e pacote.ClasseBase.metodo2**, o

Figura 64: Manual – Modelos – Página 3

elemento de modelo terá configuração igual à Figura 8:

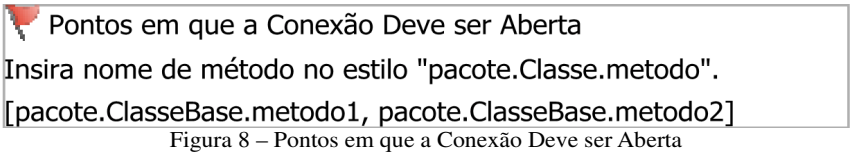

Figura 8 – Pontos em que a Conexão Deve ser Aberta

2.2 Na caixa **Pontos em que a Conexão Deve ser Fechada:**
Preencha a informação necessária.

Exemplo:
Caso o valores de **Pontos em que a Conexão Deve ser Fechada sejam os métodos pacote.ClasseBase.metodo1 e pacote.ClasseBase.metodo2**, o elemento de modelo terá configuração igual à Figura 9:

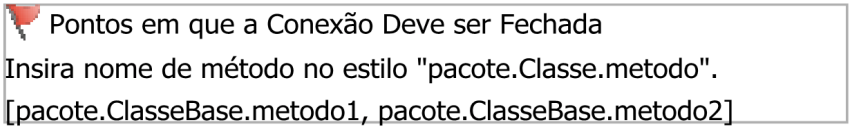

Figura 9 – Pontos em que a Conexão Deve ser Aberta

---

3 - Definir Classes que representam Objetos Persistentes

Importante: Por causa de um bug na biblioteca de diagrama, recomenda-se utilizar o botão "..." de *Definitions*, como visível na Figura 10.

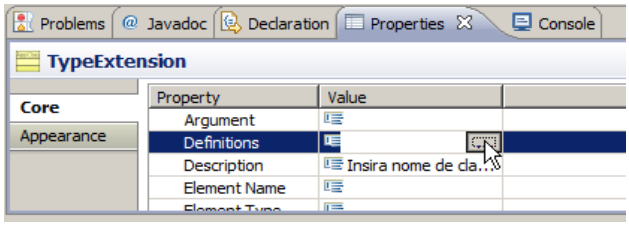

Figura 10 – Botão de *Definitions*

Utilize a janela a seguir, preencha o valor no campo Value, conforme Figura 11:

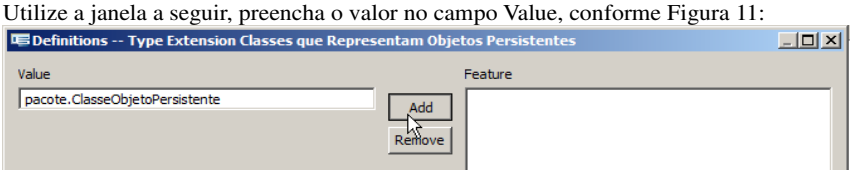

Figura 11 – Preenchimento e Add

Figura 65: Manual – Modelos – Página 4

E utilize Add (ou tecle Enter) até que todos os valores sejam passados para a direita. Conclua a definição com OK, conforme Figura 12.

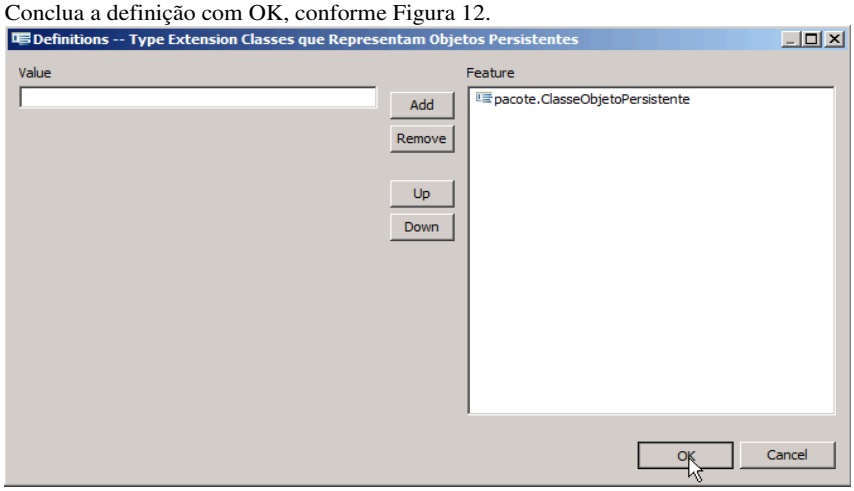

Figura 12 – Finalização de Definição

3 - Para **Definir Classes de Objetos Persistentes**, preencha a caixa de **Definição de Objetos Persistentes**.

Exemplo:
Caso o nome da classe de Objetos Persistentes seja pacote.ClasseObjetoPersistente, o modelo seria igual à Figura 13.

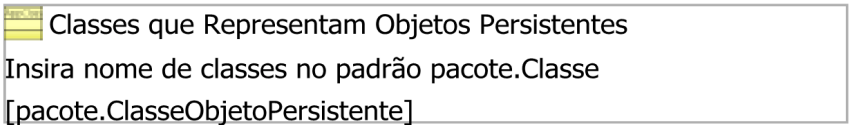

Figura 13 – Finalização de Definição

4 – Execute a Geração de código

A geração de código analisa a sintaxe dos valores inseridos e gera o código de reuso. Caso apareça uma mensagem de erro, verifique o console para a identificação.

Possíveis erros:

- **Valor inserido é invalido:**
  - **Se o valor requerido for uma String, o gerador irá exigir que a String esteja entre aspas;**
  - **Se o valor requerido for um Inteiro, o gerador irá exigir que o texto seja um valor inteiro valido;**
  - **Se o valor requerido for um número real, o gerador irá exigir que o texto seja um valor inteiro válido ou racional válido;**
  - **Se o valor requerido for booleano, o gerador irá exigir que o texto seja ou *true* ou *false*;**

Figura 66: Manual – Modelos – Página 5

- **Identificador inválido:**
  - **Para as definições de pontos ou classes, será validado se o valor informado é um identificador ou uma sequência de identificadores separados por pontos.**

5 – Execute a Validação

Embora não seja obrigatória, esta etapa verifica se as classes e métodos especificados existem na aplicação.
Caso apareça uma mensagem de erro, verifique o console para a identificação.
Possíveis erros:

- **Método não pode ser encontrado:**
  - **O método não pode ser encontrado no escopo da aplicação base.**
    - **Verifique se o método foi escrito corretamente com o nome correto de pacotes e classes.**
- **Classe não pode ser encontrada:**
  - **A classe não pode ser encontrada no escopo da aplicação base.**
    - **Verifique se a classe foi escrito corretamente com o nome correto de pacotes e classes.**

6 – Execute o Caso de Teste

O caso de teste é importante para verificar se as etapas foram realizadas corretamente.

Se o caso de teste não obtiver resultado positivo, verifique os seguintes itens:

- Valores definidos não estão corretos:
  - O valor pode passar na validação sintática e de tipos, mas o conteúdo em si está incorreto.
- Falta de pontos da aplicação base:
  - Não foram inseridos todos os pontos (métodos) necessários para a execução correta;
- Falta de classes da aplicação base:
  - Não foram inseridas todas as classes necessárias para a execução correta;

Se o caso de teste falhar, verifique os dados informados e repita a operação.

Se o caso de teste obtiver sucesso, o processo está finalizado.

Figura 67: Manual – Modelos – Página 6

C.3 Detalhes das Aplicações Base

Nesta seção são mostrados os documentos que foram utilizados para especificar detalhes da aplicação base aos participantes do experimento. É importante notar que esses manuais foram utilizados sempre que a aplicação em questão foi acoplada ao framework e que esses manuais valem para ambas as técnicas de reúso.

A especificação da Aplicação “Gerenciamento de Pedidos” está na Figura 68. Nas tabelas providas no Capítulo 6, esta aplicação é referida como “Pedidos” e apenas empregada durante o treinamento.

A especificação da Aplicação “Gerenciamento de Manutenção” está na Figura 69. Nas tabelas providas no Capítulo 6, esta aplicação é referida como “Manutenção” e apenas empregada durante o experimento piloto.

A especificação da Aplicação “Gerenciamento de Hotel” está na Figura 70. Nas tabelas providas no Capítulo 6, esta aplicação é referida como “Hotel”.

A especificação da Aplicação “Gerenciamento de Biblioteca” está na Figura 71. Nas tabelas providas no Capítulo 6, esta aplicação é referida como “Biblioteca”.

A especificação da Aplicação “Gerenciamento de Consultas Médicas” está nas Figuras 72 e 73. Nas tabelas providas no Capítulo 6, esta aplicação é referida como “Clinica”.

A especificação da Aplicação “Gerenciamento de Entregas” está nas Figuras 74 e 75. Nas tabelas providas no Capítulo 6, esta aplicação é referida como “Entregas”.

A especificação da Aplicação “Gerenciamento de Restaurante” está nas Figuras 76 e 77. Nas tabelas providas no Capítulo 6, esta aplicação é referida como “Restaurante”.

A especificação da Aplicação “Gerenciamento de Passagens Aéreas” está nas Figuras 78 e 79. Nas tabelas providas no Capítulo 6, esta aplicação é referida como “Voos”.

Aplicação Gerenciamento de Pedidos

Requisitos de Reúso – Framework Transversal de Persistência

1 – Detalhes de Conexão com o Banco de Dados

1.1 – Nome do Banco de Dados da Aplicação:
"bancopedido"

1.2 – Nome do Sistema Gerenciador de Banco de Dados:
"derby"

1.3 – Classe Driver da Conexão com o Banco de Dados:
"org.apache.derby.jdbc.EmbeddedDriver"

1.4 – Protocolo da Conexão com o Banco de Dados:
"jdbc:derby:"

2 – Pontos de abertura e fechamento de Conexão

2.1 – Pontos em que a Conexão Deve ser Aberta:
Métodos que necessitam da Conexão Aberta

Pacote **programas**
- Classe **CasoTeste**
  - Método **validaCliente;**
  - Método **validaEndereco;**
  - Método **validaItem;**
  - Método **validaPedido;**
  - Método **validaProduto.**

2.2 – Pontos em que a Conexão Deve ser Fechada:
Métodos que executam antes do fechamento da conexão

Pacote **programas**
- Classe **CasoTeste**
  - Método **validaCliente;**
  - Método **validaEndereco;**
  - Método **validaItem;**
  - Método **validaPedido;**
  - Método **validaProduto.**

3 – Definir Classes de Objetos Persistentes
Classes de Objetos Persistentes no Banco de Dados

Pacote **application**
- Classe **Pedido**;
- Classe **Endereco**;
- Classe **Cliente**;
- Classe **Produto**;
- Classe **Item**.

Figura 68: Aplicação "Gerenciamento de Pedidos"

Aplicação Gerenciamento de Manutenção

Requisitos de Reúso – Framework Transversal de Persistência

1 – Detalhes de Conexão com o Banco de Dados

1.1 - Nome do Banco de Dados da Aplicação:
"bancomanutencao"

1.2 - Nome do Sistema Gerenciador de Banco de Dados:
"derby"

1.3 - Classe Driver da Conexão com o Banco de Dados:
"org.apache.derby.jdbc.EmbeddedDriver"

1.4 - Protocolo da Conexão com o Banco de Dados:
"jdbc:derby:"

2 - Pontos de abertura e fechamento de Conexão

2.1 - Pontos em que a Conexão Deve ser Aberta:
Métodos que necessitam da Conexão Aberta

Pacote **teste**

- Classe **CasoTeste**
  - Método **validaComponente;**
  - Método **validaEndereco;**
  - Método **validaEquipamento;**
  - Método **validaFornecedor;**
  - Método **validaManutencao;**
  - Método **validaRequerente.**

2.2 - Pontos em que a Conexão Deve ser Fechada:
Métodos que executam antes do fechamento da conexão

Pacote **teste**

- Classe **CasoTeste**
  - Método **validaComponente;**
  - Método **validaEndereco;**
  - Método **validaEquipamento;**
  - Método **validaFornecedor;**
  - Método **validaManutencao;**
  - Método **validaRequerente.**

3 - Definir Classes de Objetos Persistentes
Classes de Objetos Persistentes no Banco de Dados

Pacote **oficinaEquipamentos**

- Classe **Componente**;
- Classe **Endereco**;
- Classe **Equipamento**;
- Classe **Fornecedor**;
- Classe **Manutencao;**
- Classe **Requerente**.

Figura 69: Aplicação "Gerenciamento de Manutenção"

Não se esqueça de executar o "Tempo Inicial" correspondente antes de começar. Hotel (2) Página 1

Aplicação Sistema Gerenciamento de Hotel

Requisitos de Reúso – Framework Transversal de Persistência.

1 Detalhes de Conexão com o Banco de Dados

1.1 Nome do Banco de Dados da Aplicação:

"bancohotel"

1.2 Nome do Sistema Gerenciador de Banco de Dados:

"derby"

1.3 Classe Driver da Conexão com o Banco de Dados:

"org.apache.derby.jdbc.EmbeddedDriver"

1.4 Protocolo da Conexão com o Banco de Dados:

"jdbc:derby:"

2 Pontos de Abertura e Fechamento de Conexão

2.1 Pontos em que a Conexão Deve ser Aberta:

Métodos que necessitam da conexão aberta.
Pacote **execucao**
- Classe **CasoTeste**
  - Método **entradaAluguel**;
  - Método **criaCategoria**;
  - Método **admiteFuncionario**;
  - Método **entradaHospede**;
  - Método **abreQuarto**;
  - Método **criaReserva**.

2.2 Pontos em que a Conexão Deve ser Fechada:

Métodos que executam antes do fechamento da conexão.
Pacote **execucao**
- Classe **CasoTeste**
  - Método **fechaAluguel**;
  - Método **salvaCategoria**;
  - Método **demiteFuncionario**;
  - Método **saidaHospede**;
  - Método **fechaQuarto**;
  - Método **cancelaReserva**.

3 Definir Classes de Objetos Persistentes

Classes de Objetos Persistentes no Banco de Dados.

Pacote **sistemaHotel**
- Classe **Aluguel**;
- Classe **Categoria**;
- Classe **Funcionario**;
- Classe **Hospede**;
- Classe **Quarto**;
- Classe **Reserva**.

Hotel (2) Página 1

Figura 70: Aplicação "Gerenciamento de Hotel"

Não se esqueça de executar o "Tempo Inicial" correspondente antes de começar. Biblioteca (4) Página 1

Aplicação Sistema Gerenciamento de Biblioteca

Requisitos de Reúso – Framework Transversal de Persistência.

1 Detalhes de Conexão com o Banco de Dados

1.1 Nome do Banco de Dados da Aplicação:

"bancobiblioteca"

1.2 Nome do Sistema Gerenciador de Banco de Dados:

"derby"

1.3 Classe Driver da Conexão com o Banco de Dados:

"org.apache.derby.jdbc.EmbeddedDriver"

1.4 Protocolo da Conexão com o Banco de Dados:

"jdbc:derby:"

2 Pontos de Abertura e Fechamento de Conexão

2.1 Pontos em que a Conexão Deve ser Aberta:

Métodos que necessitam da conexão aberta.
Pacote **biblioteca**

- Classe **CasoTeste**
  - Método **retiraLivro**;
  - Método **registraBibliotecario**;
  - Método **adicionaEscritor**;
  - Método **cadastraLeitor**;
  - Método **adicionaCategoria**;
  - Método **requereReserva**.

2.2 Pontos em que a Conexão Deve ser Fechada:

Métodos que executam antes do fechamento da conexão.
Pacote **biblioteca**

- Classe **CasoTeste**
  - Método **devolveLivro**;
  - Método **despedeBibliotecario**;
  - Método **removeEscritor**;
  - Método **suspendeLeitor**;
  - Método **removeCategoria**;
  - Método **concedeReserva**.

3 Definir Classes de Objetos Persistentes

Classes de Objetos Persistentes no Banco de Dados.

Pacote **sistemaDeBiblioteca**

- Classe **Livro**;
- Classe **Bibliotecario**;
- Classe **Escritor**;
- Classe **Leitor**;
- Classe **Categoria**;
- Classe **Reserva**.

Biblioteca (4) Página 1

Figura 71: Aplicação "Gerenciamento de Biblioteca"

Aplicação Sistema Gerenciamento de Consultas Médicas

Requisitos de Reúso – Framework Transversal de Persistência.

1 Detalhes de Conexão com o Banco de Dados

1.1 Nome do Banco de Dados da Aplicação:

"dataclinica"

1.2 Nome do Sistema Gerenciador de Banco de Dados:

"derby"

1.3 Classe Driver da Conexão com o Banco de Dados:

"org.apache.derby.jdbc.EmbeddedDriver"

1.4 Protocolo da Conexão com o Banco de Dados:

"jdbc:derby:"

2 Pontos de Abertura e Fechamento de Conexão

2.1 Pontos em que a Conexão Deve ser Aberta:

Métodos que necessitam da conexão aberta.
Pacote **appTest**

- Classe **Consulta**
  - Método **inicia**;
- Classe **Exame**
  - Método **requere**;
- Classe **Laboratorio**
  - Método **adiciona**;
- Classe **Medico**
  - Método **preescreve**;
  - Método **atende**;
- Classe **Paciente**
  - Método **comparece.**

2.2 Pontos em que a Conexão Deve ser Fechada:

Métodos que executam antes do fechamento da conexão.
Pacote **appTest**

- Classe **Consulta**
  - Método **finaliza**;
- Classe **Exame**
  - Método **expede**;
- Classe **Laboratorio**
  - Método **remove**;
- Classe **Medico**
  - Método **cancela**;
  - Método **atesta**;
- Classe **Paciente**
  - Método **falta.**

Figura 72: Aplicação "Gerenciamento de Consultas Médicas" – Página 1

3 Definir Classes de Objetos Persistentes

Classes de Objetos Persistentes no Banco de Dados.

Pacote **appClinicaSys**

- Classe **Consulta**;
- Classe **Exame**;
- Classe **Laboratorio**;
- Classe **Medicacao**;
- Classe **Medico**;
- Classe **Paciente**.

Figura 73: Aplicação "Gerenciamento de Consultas Médicas" – Página 2

Aplicação Sistema Gerenciamento de Entregas

Requisitos de Reúso – Framework Transversal de Persistência.

1 Detalhes de Conexão com o Banco de Dados

1.1 Nome do Banco de Dados da Aplicação:

"entregasbase"

1.2 Nome do Sistema Gerenciador de Banco de Dados:

"derby"

1.3 Classe Driver da Conexão com o Banco de Dados:

"org.apache.derby.jdbc.EmbeddedDriver"

1.4 Protocolo da Conexão com o Banco de Dados:

"jdbc:derby:"

2 Pontos de Abertura e Fechamento de Conexão

2.1 Pontos em que a Conexão Deve ser Aberta:

Métodos que necessitam da conexão aberta.
Pacote **aplicacao**

- Classe **Caminhao**
  - Método **carrega**;
- Classe **Destino**
  - Método **localiza**;
- Classe **Pacote**
  - Método **entrega**;
- Classe **Motorista**
  - Método **dirige**;
- Classe **Remetente**
  - Método **recebe**;
- Classe **Veiculo**
  - Método **partida**.

2.2 Pontos em que a Conexão Deve ser Fechada:

Métodos que executam antes do fechamento da conexão.
Pacote **aplicacao**

- Classe **Caminhao**
  - Método **descarrega**;
- Classe **Destino**
  - Método **atinge**;
- Classe **Pacote**
  - Método **retirada**;
- Classe **Motorista**
  - Método **retira**;
- Classe **Remetente**
  - Método **recusa**;
- Classe **Veiculo**
  - Método **estaciona**.

Figura 74: Aplicação "Gerenciamento de Entregas" – Página 1

3 Definir Classes de Objetos Persistentes

Classes de Objetos Persistentes no Banco de Dados.

Pacote **entregasGerenciamento**

- Classe **Carga**;
- Classe **Destino**;
- Classe **Pacote**;
- Classe **Motorista**;
- Classe **Remetente**;
- Classe **Veiculo**.

Figura 75: Aplicação "Gerenciamento de Entregas" – Página 2

Não se esqueça de executar o "Tempo Inicial" correspondente antes de começar. Restaurante (8) Página 1

Aplicação Sistema Gerenciamento de Restaurante

Requisitos de Reúso – Framework Transversal de Persistência.

1 Detalhes de Conexão com o Banco de Dados

1.1 Nome do Banco de Dados da Aplicação:

"restaurante"

1.2 Nome do Sistema Gerenciador de Banco de Dados:

"derby"

1.3 Classe Driver da Conexão com o Banco de Dados:

"org.apache.derby.jdbc.EmbeddedDriver"

1.4 Protocolo da Conexão com o Banco de Dados:

"jdbc:derby:"

2 Pontos de Abertura e Fechamento de Conexão

2.1 Pontos em que a Conexão Deve ser Aberta:

Métodos que necessitam da conexão aberta.
Pacote **baseApp**

- Classe **Bebida**
  - Método **separa**;
- Classe **Cliente**
  - Método **entra**;
- Classe **Comanda**
  - Método **abre**;
- Classe **Garcon**
  - Método **entrega**;
- Classe **Prato**
  - Método **prepara**;
- Classe **Mesa**
  - Método **concedeACliente**.

2.2 Pontos em que a Conexão Deve ser Fechada:

Métodos que executam antes do fechamento da conexão.
Pacote **baseApp**

- Classe **Bebida**
  - Método **abre**;
- Classe **Cliente**
  - Método **sai**;
- Classe **Comanda**
  - Método **fecha**;
- Classe **Garcon**
  - Método **retira**;
- Classe **Prato**
  - Método **entrega**;
- Classe **Mesa**
  - Método **limpa**.

Restaurante (8) Página 1

Figura 76: Aplicação "Gerenciamento de Restaurante" – Página 1

3 Definir Classes de Objetos Persistentes

Classes de Objetos Persistentes no Banco de Dados.

Pacote **gastronomicManagement**

- Classe **Bebida**;
- Classe **Cliente**;
- Classe **Comanda**;
- Classe **Garcon**;
- Classe **Prato**;
- Classe **Mesa**.

Figura 77: Aplicação "Gerenciamento de Restaurante" – Página 2

Aplicação Sistema Gerenciamento de Passagens Aéreas

Requisitos de Reúso – Framework Transversal de Persistência.

1 Detalhes de Conexão com o Banco de Dados

1.1 Nome do Banco de Dados da Aplicação:

"voosdb"

1.2 Nome do Sistema Gerenciador de Banco de Dados:

"derby"

1.3 Classe Driver da Conexão com o Banco de Dados:

"org.apache.derby.jdbc.EmbeddedDriver"

1.4 Protocolo da Conexão com o Banco de Dados:

"jdbc:derby:"

2 Pontos de Abertura e Fechamento de Conexão

2.1 Pontos em que a Conexão Deve ser Aberta:

Métodos que necessitam da conexão aberta.
Pacote **aplicacao**

- Classe **Aeronave**
  - Método **pouso**;
- Classe **Aeroporto**
  - Método **abertura**;
- Classe **Bagagem**
  - Método **deposito**;
- Classe **Checkin**
  - Método **confirma**;
- Classe **Passageiro**
  - Método **embarca**;
- Classe **Voo**
  - Método **partida**.

2.2 Pontos em que a Conexão Deve ser Fechada:

Métodos que executam antes do fechamento da conexão.
Pacote **aplicacao**

- Classe **Aeronave**
  - Método **decolagem**;
- Classe **Aeroporto**
  - Método **fechamento**;
- Classe **Bagagem**
  - Método **retirada**;
- Classe **Checkin**
  - Método **cancela**;
- Classe **Passageiro**
  - Método **desembarca**;
- Classe **Voo**
  - Método **chegada**.

Figura 78: Aplicação "Gerenciamento de Passagens Aéreas" – Página 1

3 Definir Classes de Objetos Persistentes

Classes de Objetos Persistentes no Banco de Dados.

Pacote **sgpVoos**

- Classe **Aeronave**;
- Classe **Aeroporto**;
- Classe **Bagagem**;
- Classe **Checkin**;
- Classe **Passageiro**;
- Classe **Voo**.

Figura 79: Aplicação "Gerenciamento de Passagens Aéreas" – Página 2