Uma Abordagem de Reestruturação de Sistemas Baseada em Requisitos de Qualidade Pré-Estabelecidos

RELATÓRIO CIENTÍFICO PARCIAL - 01/05/2013 a 28/02/2015
Apresentado à Fundação de Amparo à Pesquisa do Estado de São Paulo - FAPESP

Número do Processo:	FAPESP 2012/05168-4
Período:	Maio/2013 a Fevereiro/2015

Bolsista:	Rafael Serapilha Durelli (rdurelli@icmc.usp.br)
Orientador:	Prof. Dr. Márcio Eduardo Delamaro (delamaro@icmc.usp.br)

USP - São Carlos
Janeiro de 2012

Resumo

Relatório Científico Parcial apresentado à Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) com o objetivo de elucidar as atividades realizadas pelo bolsista Rafael Serapilha Durelli durante o segundo período de vigência da bolsa concebida sob o Processo Número 2012/05168-4. O referido período teve inicio em Maio de 2013 e foi finalizado em Fevereiro de 2015. Além disso, este relatório também descreve as atividades que foram finalizadas, as atividades que estão em andamentos, bem como as atividades a serem realizadas no próximo período. Durante o período de vigência anterior, o bolsista conduziu atividades voltadas principalmente à obtenção de uma ampla visão da literatura científica sobre os principais temas de pesquisa relacionado ao projeto em questão. Para tal, uma revisão sistemática foi conduzida. Os resultados da revisão forneceram evidências que suportam e motivam a realização do projeto proposto. A condução da revisão sistemática bem como os principais resultados foram apresentados no relatório anterior. Vale ressaltar que no segundo período de vigência da bolsa, correspondente a este relatório, cinco principais atividades foram conduzidas. Tais atividades são: (*i*) redação de artigos, (*ii*) adaptação de um catalogo de refatoração para o metamodelo Knowledge Discovery Metamodel (KDM), (*iii*) implementação de uma ferramenta semi-automática que fornecer suporte ao catalogo de refatoração adaptado para o KDM, (*iv*) criação de um metamodelo de refatorações, padronizado que contenha características similares ao KDM, ou seja, independente de plataforma e linguagem, com o principal objetivo de auxiliar o engenheiro de modernização durante a elaboração de refactorações e (*v*) definição de uma Linguagem Especifica de Domínio (do inglês Domain-Specific Language - DSL) para auxiliar a instanciação do metamodelo de refatoração. Esse relatório apresenta as atividades desenvolvidas no período de Maio/2013 a Fevereiro/2015.

1 Introdução

Este relatório tem por objetivo apresentar as atividades realizadas pelo bolsista Rafael Serapilha Durelli durante o período de Maio/2013 a Fevereiro/2015, referente à bolsa de doutorado concebida pela Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) sob o Processo Número 2012/05168-4.

É importante salientar que o trabalho em questão tem sido desenvolvido no Departamento de Ciências da Computação e Estatística do Instituto de Ciência Matemáticas e de Computação (ICMC) da Universidade de São Paulo (campos São Carlos/SP). Este trabalho se insere no contexto do grupo de pesquisas em Engenheira de Software, sob a orientação do Prof. Dr. Márcio Eduardo Delamaro. Além disso, é importante salientar que este trabalho esta sendo executado em colaboração com o grupo de engenharia de software da Universidade Federal de São Carlos (UFSCAR)[1]. Mais especificamente em colaboração com o Prof. Dr. Valter Vieira de Camargo[2], o qual tem grande experiência na área de engenharia de software com ênfase no desenvolvimento de frameworks no contexto da programação orientada a aspectos e reuso de software. Ressalta-se que o bolsista criou um vínculo científico com o *Institut National de Recherche en Informatique et en Automatique* (INRIA), onde realizou um ano de doutorado sanduíche sobre orientação do Prof. Dr. Nicolas Anquetil[3] o qual tem grande experiência na área de manutenção e reengenharia de software. O doutorado sanduíche em questão foi realizado em Maio de 2013 até Março de 2014.

Durante o período concernente a este relatório, o bolsista dedicou-se às atividades técnicas requeridas para concretização do seu projeto de doutorado bem com às atividades exigidas pelo Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional do ICMC[4]. Considerando as atividades técnicas previstas no cronograma para o período vigente, todas foram devidamente realizadas ou estão progredindo de acordo com o estipulado, a saber: (*i*) redação da monografia e aprovação no exame de qualificação, (*ii*) implementação da abordagem proposta no exame de qualificação, (*iii*) redação de artigos. O exame de proficiência em língua inglesa, exigido pelo Programa de Pós-Graduação do ICMC, foi devidamente realizado e uma pontuação satisfatória foi obtida.

[1] `http://dc.ufscar.br`
[2] `http://buscatextual.cnpq.br/buscatextual/visualizacv.do?id=S819089`
[3] `http://rmod.lille.inria.fr/web/pier/team/Nicolas-Anquetil`
[4] Destaca-se que algumas atividades já foram reportadas no relatório anterior

1.1 Convenções adotadas neste relatório

Ao longo deste relatório, *Itálico* é utilizado para dar ênfases, introduzir novos termos e palavras em inglês. `Typewriter` é utilizado para operador Java, operador da DSL, palavras chaves, nome de métodos, variáveis e URL que aparecem no texto. Símbolos ❶, ❷, ❸, ❹ ou ⓐ, ⓑ, ⓒ, ⓓ, são utilizados para chamar a atenção do leitor para informações importantes em figuras e códigos.

1.2 Estrutura do relário

Neste relatório enfatiza-se adaptação de um catalogo de refatoração para o metamodelo *Knowledge Discovery Metamodel* (KDM). Além disso, neste relatório também é enfatizado a implementação de uma ferramenta semi-automática que fornecer suporte ao catalogo de refatoração adaptado para o KDM. Em seguida é apresentado a criação de um metamodelo de refatorações, padronizado que contenha características similares ao KDM, ou seja, independente de plataforma e linguagem. E a definição de uma Linguagem Especifica de Domínio (do inglês *Domain-Specific Language - DSL*) para auxiliar a instanciação do metamodelo de refatoração.

Organização do relatório

2 Plano de Trabalho

Hoje em dia diversas companhias estão enfrentando problemas de gerenciamento, manutenção e/ou troca (de parte) de um sistema existente. Esses sistemas geralmente são caracterizados como sistemas legados (SL). SL geralmente são aplicações de grande porte que possuem um papel crucial e importante no contexto de gerenciamento de informação de companhias. Melhorar a compreensão desses SLs (e.g., arquitetura, características, acoplamento, modularização, etc) é o fator principal durante a evolução/modernização desses sistemas. O processo de obtenção de uma representações de alto nível de um determinado SL é chamado de Engenharia Reversa (ER).

Diferentemente da Engenharia Avante (EA), a ER é comumente definida como o processo de examinar um SL para representa-lo formalmente em um modelo de alto nível de abstração (**?**). O principal objetivo para realizar esse processo é para facilitar o entendimento do estado atual do SL. Por exemplo, utilizando esse modelo de alto nível de abstração é possível corrigir possíveis erros, adicionar novas características, reutilizar partes do SL em outros sistemas, ou até mesmo moderniza-lo completamente (**?**). De acordo com **?**, isto está acontecendo com mais frequência nos dias atuais, em função da necessidade de não só satisfazer novas exigências e expectativas dos usuários, mas também para a adaptação dos SLs para modelos de negócios emergentes, aderindo à mudança da legislação, lidar com a inovação tecnológica (frameworks, *Application Programming Interface* (API), ambientes de desenvolvimento, etc) e ainda preservar a estrutura do sistema para que o mesmo não se deteriore.

Claramente, uma vez que a ER é um processo demorado e sujeito a erros, qualquer solução que auxilie (semi)automaticamente o processo de ER traria ajuda para os engenheiros de software/modernização e, assim, facilitaria sua utilização (**??**). No entanto, essa solução teria de enfrentar vários problemas, a saber: (*i*) heterogeneidade técnica dos sistemas legados; (*ii*) complexidade estrutural destes sistemas legados; (*iii*) escalabilidade da solução desenvolvida; e (*iv*) edaptabilidade.

Na década de 90 algumas pesquisas tinham como intuito desenvolver soluções (semi)automática para auxiliar a ER. No entanto, tais soluções eram focadas em tecnologias orientadas a objectos (OO) (**?**). Também surgiu-se o interesse em processos e ferramentas para auxiliar a compreensão de programas desenvolvidos em OO. Entre muitas propostas, algumas focaram na extração e análise de informações relevantes a partir do código fonte ou componentes de software (**?**), enquanto outras focaram em banco de dados relacionais (**?**), código compilado ou arquivos binários (**?**), etc. No entanto, estas pesquisas eram bastante específicas para uma tecnologia em particular ou um determinado cenário de ER (por exemplo, migração técnica, análise de software).

Com o surgimento de *Model-Driven Engineering* (MDE) (**?**), suas diretrizes e técnicas fundamentais tem sido utilizadas para auxiliar a construção de soluções eficazes de ER, ou seja, *Model-Driven Reverse Engineering* (MDRE). MDRE formaliza as representações (modelos) derivadas de SL para garantir um entendimento comum sobre o seu conteúdo. Estes modelos são então utilizados como ponto de partida para a ER. Dessa forma, MDRE beneficia diretamente da extensibilidade, da cobertura, reutilização, integração e automação das tecnologias MDE para fornecer um bom suporte para a ER. No entanto, ainda há uma falta de soluções completas destinadas a cobrir todo o processo de MDRE.

Neste contexto, em 2003 a *Object Management Group* (OMG) criou uma força tarefa para analisar e evoluir os tradicionais processos de ER, formalizando-os e fazendo com que eles fossem totalmente apoiados pelas diretrizes e princípios de MDE. Logo, o termo Modernização Dirigida à Arquitetura (*Architecture-Driven Modernization* - ADM) surgiu como uma solução para os problemas de padronização. A ADM é um processo de modernização de SL que utiliza um conjunto de metamodelos para representar complemente um sistema por meio de diferentes representações arquiteturais. Esses modelos são então submetidos à refatorações e otimizações e o código-fonte é então gerado novamente. Durante a modernização de um sistema são gerados vários modelos de acordo com os metamodelos da ADM, que representam diferentes partes do sistema, como: fluxos de dados, banco de dados, elementos de programação (métodos, classes, tipos de dados, etc.) e arquitetura (**??**).

O *Knowledge Discovery Metamodel* (KDM) é o principal metamodelo da ADM com uma ampla quantidade de metaclasses, cobrindo desde os níveis mais baixos de abstração de um sistema, como o código-fonte, até níveis mais altos, permitindo a representação de conceitos de qualquer domínio. A idéia principal da ADM é que a comunidade comece a desenvolver ferramentas que atuem somente sobre instâncias do KDM, ao invés de serem dependentes de plataformas e linguagens específicas. Por exemplo, um catálogo de refatorações para o KDM (**?**)[5] tem o poder de reestruturar um sistema independentemente da linguagem de

[5]No periodo de vigência pertinente a esse relatório, um artigo descrevendo um Catalogo de Refatoração Adaptado para o KDM foi publicado em um evento qualis B2 voltado para Engenharia de Software

programação que foi usada em seu desenvolvimento, uma vez que as refatorações ocorrem em nível do KDM, ou seja, um modelo independente de plataforma e/ou linguagem.

Sistemas Legados precisam ser refatorados durante toda a sua vida útil para se que adequem a novos requisitos. No entanto, geralmente a má aplicação de refatorações/-modernizações em SLs pode causar desvios arquiteturais. Embora refatoração (tanto de baixa granularidade, quanto alta granularidade) seja um técnica poderosa, e uma atividade recorrente durante a modernização em SLs, foi constatado durante a condução de um mapeamento sistemático (**?**)[6] que a versão original da ADM, e consequentemente do KDM, não fornecem apoio (por exemplo, a catálogos de refatoração, a metamodelos para definir refatorações, etc) para tal atividade. Além disso, dificilmente a arquitetura de um sistema legado permanece intacta depois de anos de manutenção, isso é, sua arquitetura atual possivelmente é diferente da arquitetura que foi previamente planejada. ADM também não fornece apoio à checagem de conformidade entre a arquitetura planejada e a atual.

Nesse sentido, este relatório enfatiza-se quatro principais atividades, a saber: (*i*) adaptação de um catalogo de refatoração para o metamodelo KDM, (*ii*) implementação de uma ferramenta semi-automática que fornecer suporte ao catalogo de refatoração adaptado para o KDM, (*iii*) criação de um metamodelo de refatorações, padronizado que contenha características similares ao KDM, ou seja, independente de plataforma e linguagem, com o principal objetivo de auxiliar o engenheiro de modernização durante a elaboração de refactorações e (*iv*) definição de uma Linguagem Especifica de Domínio (do inglês Domain-Specific Language - DSL) para auxiliar a instanciação do metamodelo de refatoração. A seção seguinte expõe os principais conceitos que dão embasamento para o entendimento das atividades realizadas pelo outorgado. Na Seção 3.1 é apresentado brevemente o conceito sobre Refatoração, na Seção 3.2 é descrito os conceitos sobre *Model-Driven Development* (MDD). A Seção 3.3 brevemente discorre sobre *Model Driven Reverse Engineering* (MDRE), bem como, *Architecture-Driven Modernization* (ADM) e *Knowledge Discovery*

[6]No periodo de vigência pertinente a esse relatório, um artigo descrevendo um Mapeamento Sistemático foi publicado em um evento qualis B2 voltado para Engenharia de Software

Metamodel (KDM) que são os temas guarda-chuva para este projeto. Finalmente, na Seção3.4 alguns dos apoios ferramentais utilizados nesse projeto são destacados.

Na Seção Y são descritas detalhadamente as atividades realizadas pelo outorgado durante o período de vigência da bolsa. Por sua vez, o plano de trabalho para as etapas seguintes são mencionados na Seção Z.

3 Fundamentação Teórica

3.1 Refatoração

A refatoração (*refactoring*), de acordo com **?**, surgiu na comunidade de programadores Smalltalk[7]. Refatoração, consiste no processo de alterar um software, melhorando a sua estrutura interna, de forma que o comportamento externo do código não seja alterado. Além disso, refatoração permiti a distribuição de classes, variáveis e métodos na hierarquia de classes, com o objetivo de facilitar futuras atividades de desenvolvimento ou de manutenção **????**.

No contexto da reengenharia, a refatoração é empregada para converter o código legado em um código mais modular e estruturado ou até com o objetivo de migrá-lo para uma nova linguagem de programação **?**. No entanto, a medida que o código é alterado o mesmo torna-se gradualmente difícil de entender.

De acordo com **?** a maioria das refatorações introduz indireção, ou seja, tendem a dividir objetos de maior granularidade em objetos menores e métodos longos são transformado em vários métodos menores. A seguir é apresentado algumas vantagens relacionadas à indireção:

- **Explicar intenção e implementação separadamente**: o nome de cada método, variável ou classe fornece a oportunidade de explicar sua intenção. A implementação de classes ou métodos explicam como a intenção é realizada;

[7]Linguagem de programação orientada a objeto fracamente tipada

- **Isolar a mudança**: facilita a introdução de funcionalidade.
- **Codificar a lógica condicional**: alterando a lógica condicional por mensagens polimórficas evita-se duplicações de código e aumenta-se a flexibilidade.

Hoje em dia várias *Integrated Development Environments* (IDE) conseguem automatizar algumas refatorações. Porém, nenhuma IDE consegue determinar qual trecho de código deve ser refatorado e nem quais tipos de refatoração devem ser aplicadas. Portanto, o desenvolvedor ainda é responsável por determinar qual trecho de código deve ser refatorado. De acordo com **?** essa etapa pode ser feita por meio de análise de "*bad smells*". Vale ressaltar que nenhum critério exato pode ser utilizado para determinar quando o código deve ser refatorado ou não.

3.2 *Model-Driven Development* (MDD)

Pesquisas apontam que com a utilização de MDD muitos benefícios podem ser obtidos ao mover de abordagens que são totalmente centradas a código-fonte habituais para outras baseadas em modelos. Este paradigma (MDD) é amplamente baseada na suposição de que "Tudo é um modelo" (**?**). Dessa forma, MDD basicamente se baseia em quatro principais conceitos: **meta-metamodel**, **metamodelo**, **modelo** e **transformações de modelos**. Um **meta-metamodelo** define linguagens de modelagem, como a UML, por exemplo. Um exemplo de meta-metamodel é o padrão *Meta-Object Facility* (MOF)[8]. Um **metamodelo** define os possíveis elementos e estrutura dos **modelos**, de forma semelhante à relação entre a gramática e programas correspondentes no campo de programação. **Transformações de modelos** são na verdade definido ao nível do **metamodelo**, e depois aplicado no nível do **modelo**, a partir dos **modelos** que conformam aos **metamodelos**. Por exemplo, as **transformações de modelos** são executadas entre um modelo fonte e um modelo alvo. Além disso, **transformações de modelos** podem ser ou do tipo *Model-To-Model* (e.g., Eclipse ATL[9]) ou do tipo *Model-To-Text* (e.g., Eclipse Acceleo[10]).

[8] `http://www.omg.org/mof/`
[9] `https://www.eclipse.org/atl/`
[10] `https://www.eclipse.org/acceleo/`

Note-se que o MDD, também conhecido como *Modelware* (**?**), não é tão diferente do grammarware (ou seja, onde as linguagens de programações são definidas em termos de gramáticas), em termos de definição de base e infra-estrutura. Tal afirmação pode ser visualizada na Figura 1.

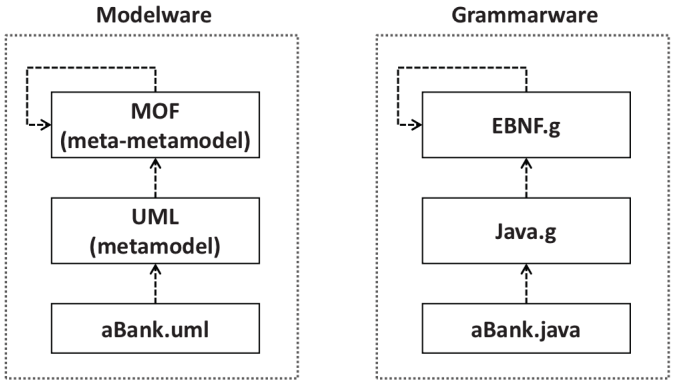


Figura 1: Modelwarevs.Grammarware.

MDD é ainda considerado um paradigma atual no contexto de Engenharia de Software. MDD foi popularizado pela *Object Management Group* (OMG) como *Model Driven Architecture* (MDA) (**?**). Hoje em dia o Eclipse, o qual é um *Integrated Development Environment*, é o ambiente padrão para MDD uma vez que o mesmo contêm uma implementação de referência do MOF, nomeado *Eclipse Modeling Framework* (EMF).

3.3 *Model-Driven Reverse Engineering* (MDRE)

A utilização de MDD no contexto da ER (ou seja, MDRE) é um campo relativamente recente (**?**). No início, os modelos eram apenas utilizados, principalmente, para especificar os sistemas antes da sua implementação (durante a Engenharia Avante). MDRE propõe que modelos não sejam apenas artefatos que "guiam" o engenheiro durante tarefas de desenvolvimento e manutenção de software, mas como parte integrante do software (**?**).

Devido ao grande interesse em MDRE, a OMG em 2003 inicio uma força tarefa (ADM *taskforce*) que tinha como intuito padronizar o processo de engenharia reversa. Da mesma forma que MDA, a OMG criou a *Architecture-Driven Mordernization* (ADM) que tem como objetivo criar especificações/padronizações para auxiliar processo de modernização

de sistemas legados por meio de um conjunto de metamodelos padronizados. O fluxo de um processo de MDRE apoiada pela ADM possui três fases e é semelhante ao contorno de uma ferradura, são elas: Engenharia Reversa (ER) (do inglês, *Reverse Engineering*), Reestruturação (do inglês, *Restructuring*) e Engenharia Avante (EA) (do inglês, *Forward Engineering*), como pode ser visto na Figura 2. Partindo do lado inferior esquerdo, na parte da ER, o conhecimento é extraído do sistema legado e um modelo *Plataform Specific Model* (PDM) é gerado. O modelo PSM serve como base para a geração de um modelo *Platform Independent Model*, ou seja, durante a fase de ER, transformações são feitas como o intuito de se obter uma representação de alto nível do software, independentemente da plataforma utilizada anteriormente.

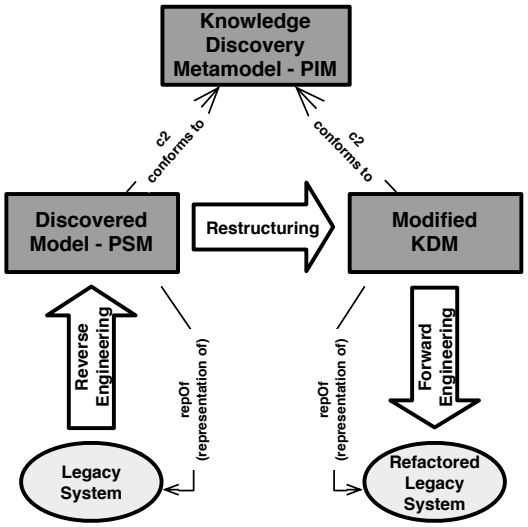


Figura 2: Fluxo do processo de modernização apoiada pela ADM (Adaptada (**?**))

O modelo PSM é um modelo especifico de uma plataforma, ou seja, nele existem metadados relacionados a uma plataforma ou linguagem de programação especifica. O modelo PIM é um modelo de abstração mais alto, pois não contém informações especificas de uma determinada plataforma ou linguagem.

Na fase de reestruturação, refatorações, melhorias e novas regras de negócios podem ser introduzidas no sistema, e, com uma representação independente de plataforma do software modernizado, segue-se para a fase de Engenharia Avante. Nessa ultima fase, os

modelos são novamente submetidos a uma série de transformações para chegar ao nível de artefatos executáveis, ou seja, código-fonte.

No contexto da ADM, para dar suporte ao processo de modernização independentemente de plataforma e linguagem (PIM), foi criado um metamodelo que possibilita a comunicação entre diferentes plataformas e linguagens, e foi denominado pela ADM *task-force* de *Knowledge Discovery Metamodel* (KDM) (**?**). O KDM provê representações para os sistemas de software existentes em diferentes camadas de abstrações. Cada modelo é representado por um conjunto de visões arquiteturais, ou seja, modelos KDM representando diferentes perspectivas de conhecimento sobre os artefatos dos sistemas de software existentes. Esses modelos são criados automaticamente, semi-automaticamente ou manualmente por meio da aplicação de várias técnicas de extrações de conhecimento, de análises e de transformações. (**?**).

O KDM representa artefatos físicos e lógicos de software dos sistemas legados em diferentes níveis de abstração e constitui dozes pacotes organizados em quatro camadas, são elas: infraestrutura (*infrastructure*), elementos de programa (*program elements*), recursos de tempo de execução (*runtime resources*) e abstração (*abstractions*) (**?**). Na Figura 3 está representada a arquitetura do KDM ilustrando a forma como as camadas se relacionam, quais são os pacotes pertencentes a cada camada e a separação dos seus interesses. Cada camada baseia-se na camada anterior, dessa forma, elas estão organizadas em pacotes que definem um conjunto de elementos do metamodelo, cujo propósito é representar um interesse específico e independente do conhecimento relacionado a sistemas legados.

Três pacotes importantes do KDM no contexto deste trabalho são: (*i*) pacote *Code*, (*ii*) *Action* e (*iii*) *Structure*. Os dois primeiros pacotes contêm metaclasses para representar elementos de programação, tais como, classes, métodos, atributos, etc. Dessa forma, utilizando esses dois pacotes foi possível criar/adaptar um catalogo de refatoração para o contexto do KDM. Assim, todos as refatorações que foram adaptadas para o KDM agora são independentes de linguagem e plataforma. O terceiro pacote por sua vez (*Structure*), define elementos de metamodelo que representam componentes de arquitetura de sistemas

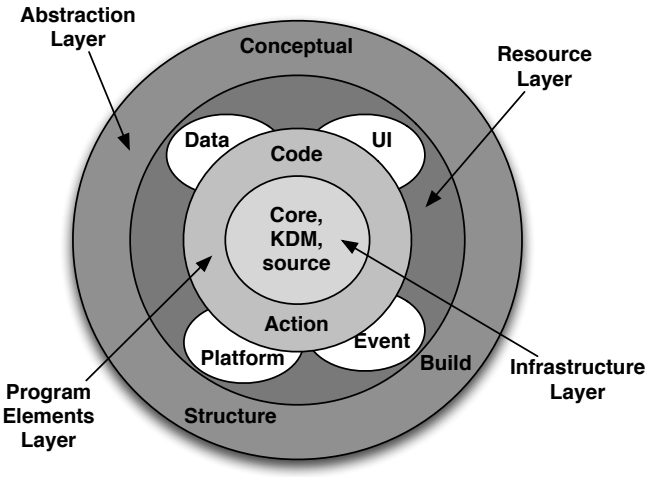


Figura 3: Arquitetura do KDM (adaptada (**?**))

de software existentes. Por exemplo, esse pacote define as seguintes metaclasses: *Subsystem*, *Component*, *Layer*, *SoftwareSystem*, *ArchitectureView*. Ressalta-se ainda que esse ultimo pacote, é de suma importância para as próximas atividades do outorgado, uma vez que verificação de conformidade nível arquitetural serão implementadas para o KDM. Além disso, também serão definidas refatorações em nível arquitetural.

3.4 Apoio ferramental

Diversas ferramentas de modelagem estão disponíveis para o desenvolvimento empresarial baseado em modelo, uma plataforma de ferramentas que se tornou proeminente no mundo da MDRE é o IDE Eclipse. Um conjunto de ferramentas interessantes para MDRE foram disponibilizados para essa IDE, permitindo, assim diversas iniciativas sobre esta plataforma. No contexto, do projeto em questão algumas iniciativas foram utilizadas: (*i*) *Eclipse Modeling Framework* (EMF), (*ii*) Xtext, (*iii*) Acceleo e (*iv*) *ATL Transformation Language* (ATL).

Eclipse Modeling Framework (EMF) é a principal iniciativa do IDE Eclipse no contexto de MDRE por várias razões. Primeiro, EMF permite a definição de metamodelos tendo como base uma linguagem de metamodelagem denominada Ecore. Em segundo lugar, EMF

fornece um gerador de metamodelos, ou seja, uma API baseada em Java para manipulação de modelos. Em terceiro lugar, EMF contêm uma poderoso API que abrangem diferentes aspectos, tais como a serialização e deserialização de modelos de/para *XML Metadata Interchange* (XMI). Xtext é um framework para desenvolvimento de Linguagens Específicas de Domínio (do inglês, *Domain-Specific Language*). Acceleo é uma implementação pragmática do OMG para fornecer suporte a transformações *Model2Text*. Por fim, ATL é uma linguagem de transformação de modelos, ou seja provê suporte a transformações *Model2Model*.

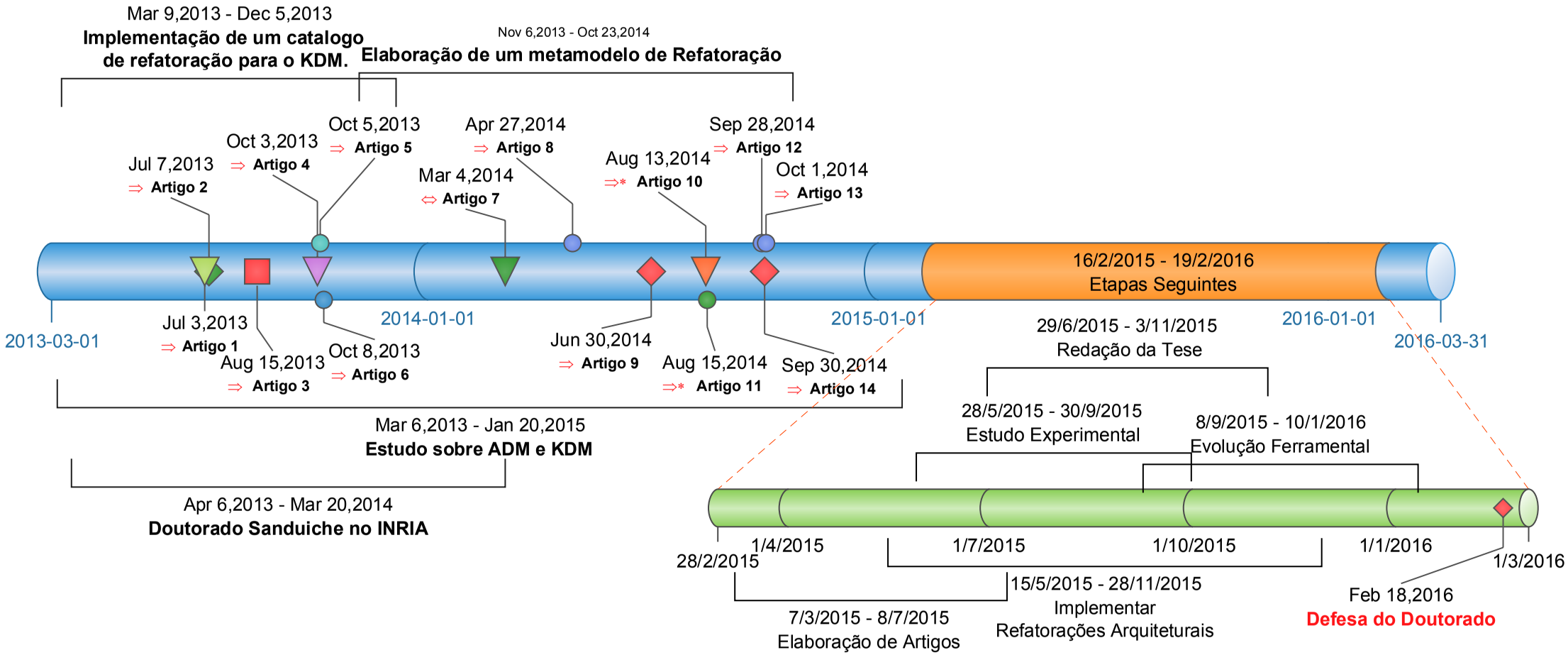
4 Cronograma e Resumo das Atividades Realizadas no Período

A seguir é apresentado uma *timeline* que descreve todas as atividades realizadas durante o período de vigência e que serão realizadas o período pelo bolsista para o desenvolvimento deste projeto. Em seguida, é apresentada uma descrição de cada evento presente na *timeline*. Ressalta-se que todos os artigos que foram publicados durante o período de vigência desse relatório estão destacados tanto na *timeline* quanto na descrição pelo símbolo $\Rightarrow$ ou $\Leftrightarrow$. $\Leftrightarrow$ ilustra capítulo de livro, enquanto o símbolo $\Rightarrow$ refere-se a conferências. Além disso, vale destacar que maiores informações sobre cada evento serão descritas nas seções seguintes.

- **Mar 6, 2013 - Jan 20, 2015** - Estudo detalhado sobre ADM e KDM;
- **Mar 9, 2013 - Dec 5, 2013** - Implementação de um catalogo de refatoração para o KDM;
- **Apr 6, 2013 - Mar 20, 2014** - Realização do Doutorado Sanduíche no INRIA, Lille, França;
- **Nov 6, 2013 - Oct 23, 2014** - Elaboração de um metamodelo de refatoração;

- ⇒ Artigo 1 - Concern-Based Refactorings Supported by Class Models to Reengineer Object-Oriented Software into Aspect-Oriented Ones;
- ⇒ Artigo 2 - F3: From features to frameworks;
- ⇒ Artigo 3 - An Approach to Develop Frameworks from Feature Models;
- ⇒ Artigo 4 - F3T: From Features to Frameworks Tool;
- ⇒ Artigo 5 - CCKDM - A Concern Mining Tool for Assisting in the Architecture-Driven Modernization Process;
- ⇒ Artigo 6 - A Combined Approach for Concern Identification in KDM models;
- ⇔ Artigo 7 - Developing Frameworks from Extended Feature Models;
- ⇒ Artigo 8 - Evaluating the Effort for Modularizing Multiple-Domain Frameworks towards Framework Product Lines with Aspect-Oriented Programming and Model-Driven Development;
- ⇒ Artigo 9 - Data Network in Development of 3D Collaborative Virtual Environments: A Systematic Review;
- ⇒ Artigo 10 - Towards a Refactoring Catalogue for Knowledge Discovery Metamodel;
- ⇒ Artigo 11 - A Mapping Study on Architecture-Driven Modernization;
- ⇒ Artigo 12 - KDM-AO: An Aspect-Oriented Extension of the Knowledge Discovery Metamodel;
- ⇒ Artigo 13 - Investigating Lightweight and Heavyweight KDM Extensions for Aspect-Oriented Modernization;
- ⇒ Artigo 14 - KDM-RE: A Model-Driven Refactoring Tool for KDM;
- **16/2/2015 - 19/2/2016** - Esse período representa as Etapas Seguintes;
  - **7/3/2015 - 8/7/2015** - Elaboração de Artigos;

- **15/5/2015 - 28/11/2015** - Implementar Refatorações Arquiteturais;
- **28/5/2015 - 30/09/2015** - Estudo Experimental;
- **29/06/2015 - 3/11/2015** - Redação da Tese;
- **08/09/2015 - 10/01/2016** - Evolução Ferramental;
- **Fevereiro** - Defesa do Doutorado;


Mar 9,2013 - Dec 5,2013
Implementação de um catalogo de refatoração para o KDM.
Nov 6,2013 - Oct 23,2014
Elaboração de um metamodelo de Refatoração
Jul 7,2013
⇒ Artigo 2
Oct 3,2013
⇒ Artigo 4
Oct 5,2013
⇒ Artigo 5
Mar 4,2014
⇔ Artigo 7
Apr 27,2014
⇒ Artigo 8
Aug 13,2014
⇒* Artigo 10
Sep 28,2014
⇒ Artigo 12
Oct 1,2014
⇒ Artigo 13
2013-03-01
Jul 3,2013
⇒ Artigo 1
Aug 15,2013
⇒ Artigo 3
Oct 8,2013
⇒ Artigo 6
2014-01-01
Jun 30,2014
⇒ Artigo 9
Aug 15,2014
⇒* Artigo 11
Sep 30,2014
⇒ Artigo 14
2015-01-01
16/2/2015 - 19/2/2016
Etapas Seguintes
2016-01-01
2016-03-31
29/6/2015 - 3/11/2015
Redação da Tese
Mar 6,2013 - Jan 20,2015
Estudo sobre ADM e KDM
Apr 6,2013 - Mar 20,2014
Doutorado Sanduiche no INRIA
28/5/2015 - 30/9/2015
Estudo Experimental
8/9/2015 - 10/1/2016
Evolução Ferramental
28/2/2015
1/4/2015
1/7/2015
1/10/2015
1/1/2016
1/3/2016
15/5/2015 - 28/11/2015
Implementar
Refatorações Arquiteturais
7/3/2015 - 8/7/2015
Elaboração de Artigos
Feb 18,2016
Defesa do Doutorado

4.1 Estágio no Exterior

O estágio contou com o apoio financeiro da CNPQ (Processo: 241028/2012-4) e foi realizado no *Institut national de recherche en informatique et en automatique* (INRIA), França. O aluno ficou na França por um ano (Abril de 2013 - Abril de 2014) sob a supervisão do professor Doutor Nicolas Anquetil. O grupo de pesquisa é especializado em remodularização e modernização de sistemas orientados a objetos com enfoque em *Model-Driven Development*.

Dois projetos foram conduzidos durante a estadia na França, descritos nas Seções X e Y. É importante enfatizar que ambos os projetos são diretamente relacionados à realização deste projeto de doutorado.

O vínculo com o CNPQ foi encerrado no mês de Abril e a bolsa da FAPESP foi reatividade. Uma cópia do parecer emitido pelo orientador no exterior pode ser encontrado no Apêndice A.

4.2 Mapeamento Sistemático

Quando se conduz uma revisão de literatura sem o pré-estabelecimento de um protocolo de revisão há um direcionamento por interesses pessoais, o que leva a resultados pouco confiáveis. Neste contexto, pesquisadores vem utilizando uma técnica denominada de Mapeamento Sistemático (MS) para auxiliar o pesquisador a conduzir um revisão bibliográfica de forma totalmente sistemática com o intuito de evitar que trabalhos importantes fiquem fora de suas pesquisas. Um MS é caracterizada por ser um meio de avaliar e interpretar todas as pesquisas disponíveis, referentes a um questão de pesquisa, tema, área ou fenômeno de interesse. O MS tem como objetivo apresentar uma avaliação justa de um tema de pesquisa, utilizando uma metodologia confiável, rigorosa e auditável **?**.

De acordo com **?** o MS implica na forma mais adequada para se identificar, avaliar e interpretar toda pesquisa importante para um tema em particular. Resume-se que um MS configura um alicerce para novas atividades de pesquisa acerca de determinado

tema. Dessa forma, foi realizado um MS sobre *Architecture-Driven Modernization* (ADM) e *Knowledge-Discovery Metamodel* (KDM). A nossa motivação para realizar esse MS é identificar os temas que têm sido mais investigados, bem como os temas que ainda não foram investigados. Embora, a ADM é uma abordagem relativamente nova, OMG afirma que ela é uma importante abordagem pois combina dois dos principais campos da Engenharia de Software, ou seja, *Model-Driven Development* e Engenharia Reversa. Desde a definição da ADM muitos esforços têm enfatizado a modernização de sistemas legados por meio desta abordagem. Neste contexto, é importante realizar uma investigação mais sistemática dos temas englobados por esta área de pesquisa.

Para atingir este objetivo, foi realizado um MS. Os resultados, bem como o protocolo de pesquisa elaborado, foram publicados no *IEEE Information Reuse and Integration* (IRI 2014) , o qual tem qualis B2. O artigo foi apresentado pelo bolsista em Agosto de 2014 em São Francisco, California. Maiores detalhes sobre esse artigo pode ser obtido no Apêndice A deste relatório. O mesmo anexo contem uma cópia da obra submetida.

4.3 Estabelecimento do Catalogo de Refatoração para o metamodelo KDM

Foi elaborado um catalogo dedicado para ser aplicado no metamodelo do KDM. Mais especificamente esse catalogo foi adaptado com base em catálogos já disponíveis na literatura. Para a definição desse catalogo de refatoração, foi utilizado um formato inspirado pelo **?**. Em outras palavras, o catalogo foi descrito da seguinte forma: (*i*) o nome do refatoração, (*ii*) descrição da típica situação onde a refatoração deve ser aplicada, (*iii*) descrição da solução para solucionar uma determinada situação problemática, (*iv*) pre-condições que devem ser satisfeitas para aplicar a refatoração, (*v*) os parâmetros necessários para executar a refatoração e (*vi*) descrição dos passos para realizar a refatoração.

O catalogo de refatoração é estruturado em quatro grupos como pode ser observado na Tabela **??**, a qual contêm 17 refatorações. O primeiro grupo chamada *Rename Feature* consiste de refatorações para renomear *ClassUnit*, *StorableUnits* e *MethodUnits*. O segundo

grupo, *Moving Features Between Objects* consiste de refatorações simples, tais como mover ou criar características, ou seja, criar ou mover atributos, métodos ou classes. O terceiro grupo, *Organizing Data*, é responsável por definir um conjunto de refatorações para ser organizar a estrutura do código-fonte. Por fim, o quatro grupo, *Dealing With Generalization*, representa refatorações para mover métodos e/ou atributos sobre uma especifica hierarquia de classes.

Tabela 1: Refactorings Adapted to KDM

Rename Feature	Moving Features Between Objects	Organing Data	Dealing with Generalization
Rename ClassUnit	Move MethodUnit	Replace data value with Object	Push Down MethodUnit
Rename StorableUnit	Move StorableUnit	Encapsulate StorableUnit	Push Down StorableUnit
Rename MethodUnit	Extract ClassUnit	Replace Type Code with ClassUnit	Pull Up StorableUnit
	Inline ClassUnit	Replace Type Code with SubClass	Pull Up MethodUnit
		Replace Type Code with State/Strategy	Extract SubClass
			Extract SuperClass
			Collapse Hierarchy

Vale ressaltar que um artigo descrevendo o catalogo de refatoração adaptado para o KDM também foi no *IEEE Information Reuse and Integration* (IRI 2014) , o qual tem qualis B2. O artigo foi apresentado pelo bolsista em Agosto de 2014 em São Francisco, California. Além disso, o bolsista participou das sessões técnicas e palestras. Maiores detalhes sobre esse artigo pode ser obtido no Apêndice B deste relatório. O mesmo contem uma cópia da obra submetida.

4.4 Um ambiente integrado para desenvolvimento e apoio para o catalogo de refatorações do metamodelo KDM

Durante a revisão sistemática pode-se averiguar que tanto o processo ADM e o seu metamodelo KDM têm sido vastamente utilizado na literatura para auxiliar a modernização de sistemas legados. No entanto, também pode-se verificar qua até o momento não existe nenhum ambiente integrado de desenvolvimento para guiar o engenheiro para automaticamente aplicar as refatorações e modernizações como existe em outros paradigmas, tais como o paradigma orientado a objetos. Para mitigar tal limitação, durante o período de vigência da bolsa foi desenvolvido um *plug-in* utilizando a plataforma do Eclipse.

Esse *plug-in* fornece um ambiente para realizar as refatorações apresentadas na Seção 4.3 de forma totalmente automatizada. Na Figura 4 é ilustrado todo o processo no qual o *plug-in* é baseado. Como pode ser observado nessa figura a utilização do *plug-in* pode ser ilustrada em três passos: (*i*) engenharia reversa (*reverse engineering*), (*ii*) refatorações (*refactorings*) e (*iii*) engenharia avante (*forward engineering*). Maiores detalhes sobre tais passos são descritos nas próximas seções.

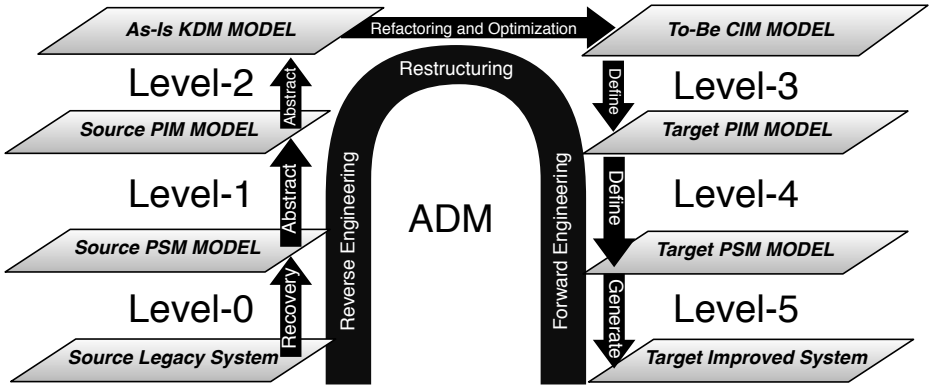


Figura 4: Passos para utilização do *plug-in*

4.4.1 Engenharia Reversa

Para iniciar esse passo o engenheiro de software deve entrar com um arquivo que representa uma instância do metamodelo KDM ou um determinado código-fonte para realizar a refatoração. No caso do código-fonte ser a entrada escolhida, o engenheiro de software começa o processo no **Level-0** escolhendo um projeto no Eclipse[11] que contenha o código-fonte para realizar as refatorações. Posteriormente, no **Level-1** o código-fonte precisa ser transformado para um modelo especifico de plataforma (no inglês - Platform-Specific Model (PSM)). Esse PSM representa uma instância do código-fonte em um nível mais abstrato do código-fonte. Para realizar essa transformação (código-fonte para PSM) foi implementado um extrator de modelo em Java.

Após criar o PSM o próximo level (**Level-2**) consiste em transformar o PSM para um Modelo Independente de plataforma (no inglês - Platform-Indented Model (PIM)) o qual

[11] https://www.eclipse.org/

é baseado no metamodelo do KDM. Nesse level o *plug-in* utiliza o *framework* MoDisco[12] para realizar a transformação de PSM para PIM.

Na Figura 5 é apresentado uma visão geral do *plug-in* desenvolvido pelo bolsista durante a período de vigência da bolsa. Apenas para o propósito de explicação, foi identificado quatro principais regiões do *plug-in*, veja Figura5 ⓐ, ⓑ, ⓒ and ⓓ.

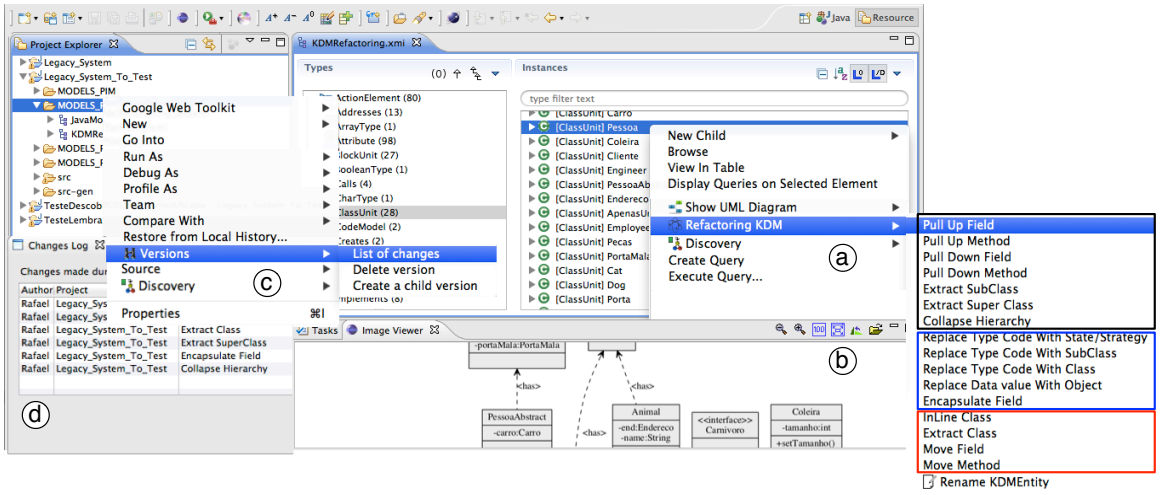


Figura 5: Visão geral do *plug-in* desenvolvido

Como já salientado anteriormente todas as refatorações fornecidas pelo *plug-in* são feitas com base no metamodelo KDM. Dessa forma, para auxiliar o engenheiro de software a realizar as refatorações um menu chamado *Refactoring KDM* foi adicionado, veja Figura 5ⓐ. Utilizando esse menu o engenheiro de software pode interagir com o metamodelo do KDM e escolher qual refatoração deve ser executada. Note que na região ⓐ da Figura 5 é possível ver todas as 17 refatorações implementadas no *plug-in*.

Na região ⓑ da Figura 5 é apresentado um diagrama de classe. Esse diagrama pode ser utilizado pelo engenheiro de software antes ou depois de aplicar as refatorações no modelo KDM. Usualmente o engenheiro pode utilizar esse diagrama antes de aplicar refatorações para decidir onde deve-se realmente aplicar as refatorações. Além disso, esse diagrama é útil por que geralmente sistemas legados não contêm nenhum tipo de documentação, sendo o código-fonte o único artefato disponível do mesmo. Portanto, criar um diagrama de classe

[12]http://www.eclipse.org/MoDisco/

durante a execução de refatorações no sistema legado pode ser uma boa alternativa para melhorar a documentação de um determinado sistema.

O *plug-in* também fornece múltiplas versões de um sistema em nível de modelos, ou seja, em nível de modelos KDM. O objetivo é permitir que o engenheiro de software trabalhe interativamente em vários modelos, permitindo assim que o engenheiro de software escolha e explore diferentes caminhos de refatoração. Como pode ser observado na região ⓒ da Figura 5, o engenheiro deve selecionar o arquivo KDM e escolher a opção "*Versions*". Três opções são disponíveis nesse menu (*i*) *List of Changes*, (*ii*) *Delete version* and (*iii*) *Create a child version.* A primeira opção mostra todas refatorações que já foram feitas pelo engenheiro (ver região ⓓ) - a segunda opção é responsável por deletar uma versão - e a ultima opção criar uma cópia do arquivo KDM, permitindo assim que o engenheiro aplique outras refatorações e explore diferentes caminhos de refatoração sem afetar o modelo KDM principal do projeto.

4.4.2 Executando Refatorações no *Plug-in*

Após o engenheiro clicar no menu da região ⓐ (ver Figura 5) e escolher qual refatoração aplicar um *Wizard* será mostrado. Para explicação, considere que o engenheiro de software escolheu aplicar a refatoração denominada *Extract Class.* Então o engenheiro deve selecionar qual metaclass o mesmo deseja extrair, esse passo é ilustrado na Figura 6 (a). Posteriormente o *plug-in* executa o *Wizard* como ilustrado na Figura 6 (b). Como pode ser observado, aqui o engenheiro de software pode atribuir um nome para a nova metaclasse. Além disso, uma prévia de todos os detectados *StorableUnits* e *MethodUnis* que podem ser extraídos e adicionados na outra classe também são mostrados. O engenheiro pode também selecionar se a nova classe será uma classe interna ou uma classe normal. Também é possível especificar se métodos assessores (*getters* e *setters*) devem ser criados ou não.

Após o engenheiro preencher todos os campos necessários, ele pode clicar no botão *Finish* e então a refatoração *Extract Class* é executada. Como pode ser observado na

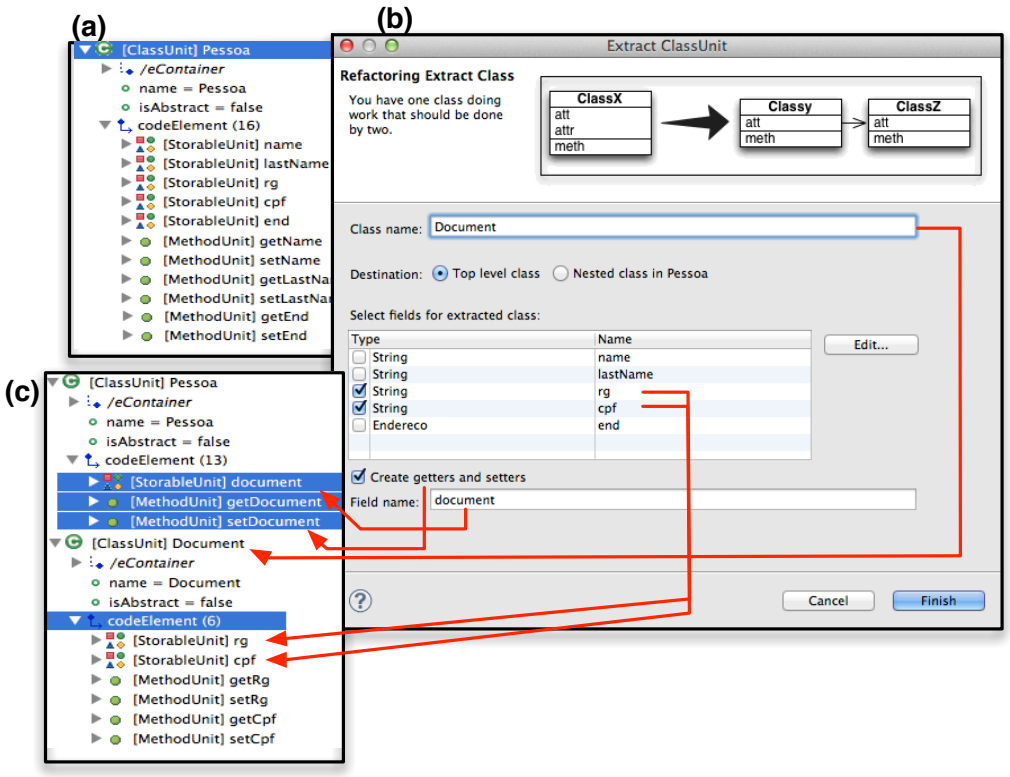


Figura 6: Extract Class Wizard

Figura 6 (c) uma nova instância de *ClassUnit* denominada *Document* foi criada - dois *StorableUnits* da metaclasse *Pessoa*, "rg" e "CPF" foram movidas para *Document*.

4.4.3 Engenharia Avante

Após o engenheiro realizar todas as refatorações no KDM os próximos passos são: (*i*) transformar o KDM para um PSM e (*ii*) transformar o PSM para artefatos físicos (código-fonte). O primeiro passo é executado baseado em um conjunto de transformações utilizando a linguagem ATL Transformation Languag (ATL)[13]. O último passo consiste em utilizar *templates* para gerar o código-fonte refatorado. Na Figura 7 é apresentado como é feita a geração de código-fonte.

[13]https://www.eclipse.org/atl/

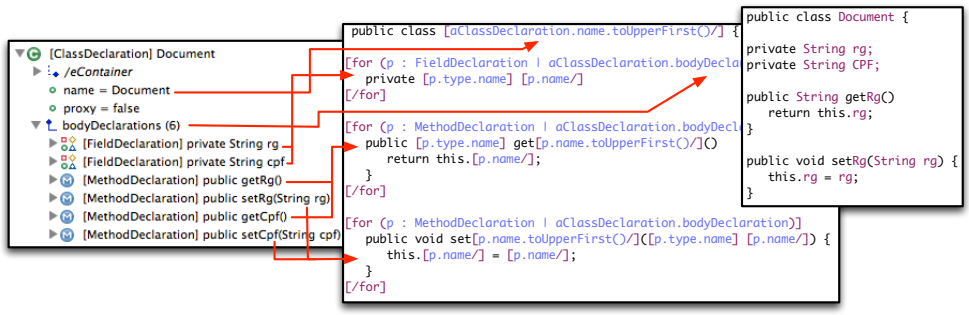


Figura 7: Passos da Engenharia Avante

4.4.4 Arquitetura do *plugin*

Na Figura 8 é ilustrado a arquitetura do *plug-in* desenvolvido durante o período de vigência da bolsa. Como pode ser observado nessa figura, a primeira camada (*layer*) é o *Core Framework*. Essa camada representa que o *plug-in* foi desenvolvida utilizando como base a plataforma de desenvolvimento Eclipse. Além disso, nessa camada pode-se observar que também foi utilizado Java e Groovy como linguagem de programação. Também é possível identificar que nessa camada alguns *plug-ins* da plataforma de desenvolvimento Eclipse foram utilizados, tais como MoDisco[14] e EMF[15]. Modisco e EMF ambos foram utilizados pois fornecem uma *Application Programming Interface* (API) para facilitar o acesso ao metamodelo KDM.

A segunda camada, *Tool Core*, é onde todas as refatorações fornecidas pelo *plug-in* foram implementadas. A ultima camada é onde a interface gráfica do *plug-in* foi desenvolvida. Vale ressaltar que um artigo sobre esse *plugin* foi publicado no 2nd *Workshop on Software Visualization, Evolution and Maintenance* (VEM). O bolsista esteve presente nesse evento, apresentado o artigo e participando das sessões técnicas e palestras. Mais detalhes sobre este *plugin* podem ser encontrados na cópia desse artigo no Apêndice C.

[14]http://www.eclipse.org/MoDisco/
[15]https://www.eclipse.org/modeling/emf/

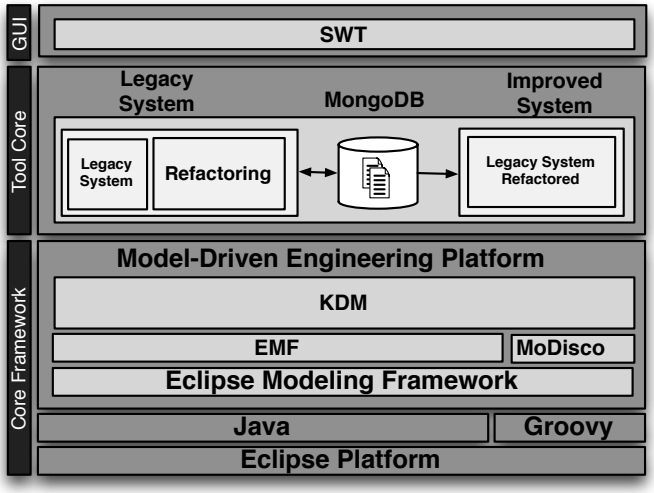


Figura 8: Arquitetura do *plug-in*

4.5 Um Metamodelo para Especificar Refatorações para o KDM - (*Refac-KDM*)

Durante o desenvolvimento e evolução de software novas funcionalidades geralmente são adicionadas ou são ajustadas a novas exigências. Devido a tais alterações, a flexibilidade da arquitetura desse sistema pode ser um fator desafiador. Em outros casos, após aplicar um conjunto de alterações a arquitetura continua perfeitamente adequada. Porém, usualmente é necessário aplicar mudanças na arquitetura dessa sistema para confortar tais alterações.

Para melhorar o projeto de um determinado software, geralmente os engenheiros de software gastam uma quantidade significativa de tempo reestruturando o software manualmente. No entanto, reestruturar código manualmente além de ser uma tarefa que demanda tempo é totalmente propicia a erros. Não importa a atenção dispendida pelo engenheiro de software durante a atividade de reestruturação, se o sistema é relativamente grande, há uma boa chance de que o mesmo irá ter o seu comportamento alterado após tal atividade. Como ressaltado anteriormente é de suma importância que o comportamento do sistema seja preservado após a reestruturação, assim, o conceito de refatoração deve ser aplicado (ver Seção 3.1).

Hoje em dia refatoração é utilizada tanto no âmbito acadêmico quanto na industria. Refatoração é uma área muito madura e muito difundida. Além disso, várias IDEs executam facilmente e de forma segura um conjunto refatorações para um vasto número de linguagens de programação. Com o surgimento de MDD é importante que os conceitos de refatorações sejam adaptados tanto para MDRE (ver Seção 3.3) e ADM. Uma iniciativa têm sido conduzida pelo outorgado. Mais especificadamente o bolsista definiu um ambiente integrado para desenvolvimento e apoio para o catalogo de refatorações do metamodelo KDM (**?**). Maiores informações podem ser obtidas no artigo publicado, o mesmo pode ser visualizado na integra no Apêndice X.

Um olhar mais atento à reutilização de refatorações levanta a questão sobre o que pode ser reutilizado e o que não pode. Trabalhos anteriores nesta área tem mostrado que há potencial para reutilização, mas é limitado, de uma forma ou de outra. No entanto, pesquisas também apontam que algumas características não podem ser capturadas na parte reutilizável de uma refatoração. Por exemplo, uma refatoração genérica não pode fazer suposições sobre a semântica de uma linguagem de programação.

Embora a ADM e, principalmente o KDM, tenham sido propostos para apoiar a modernização de sistemas legados, até o momento não existem propostas de um metamodelo para especificar refatorações para o KDM. Portanto, os engenheiros de modernização precisam desenvolver suas próprias soluções para transformar instâncias KDM origem em alvo. Durante o mapeamento sistemático conduzido e publicado pelo outorgado (**?**) (ver Apêndice X) pôde-se observar na literatura a carência de estudos que definem um metamodelo para especificar refatorações para KDM. Sem a adequada representação de refatorações no KDM, a realização de uma refatoração pode se tornar propensa a erros. Assim, é de suma importância definir uma extensão para o KDM em que os engenheiros de modernização possam especificar refatorações independentes de plataforma.

Como já ressaltado existe uma forte necessidade de reutilizar refatorações no contexto da ADM e, principalmente para o metamodelo KDM. Para permitir tal reutilização, as partes de refatorações que podem ser generalizadas devem ser separadas das que são es-

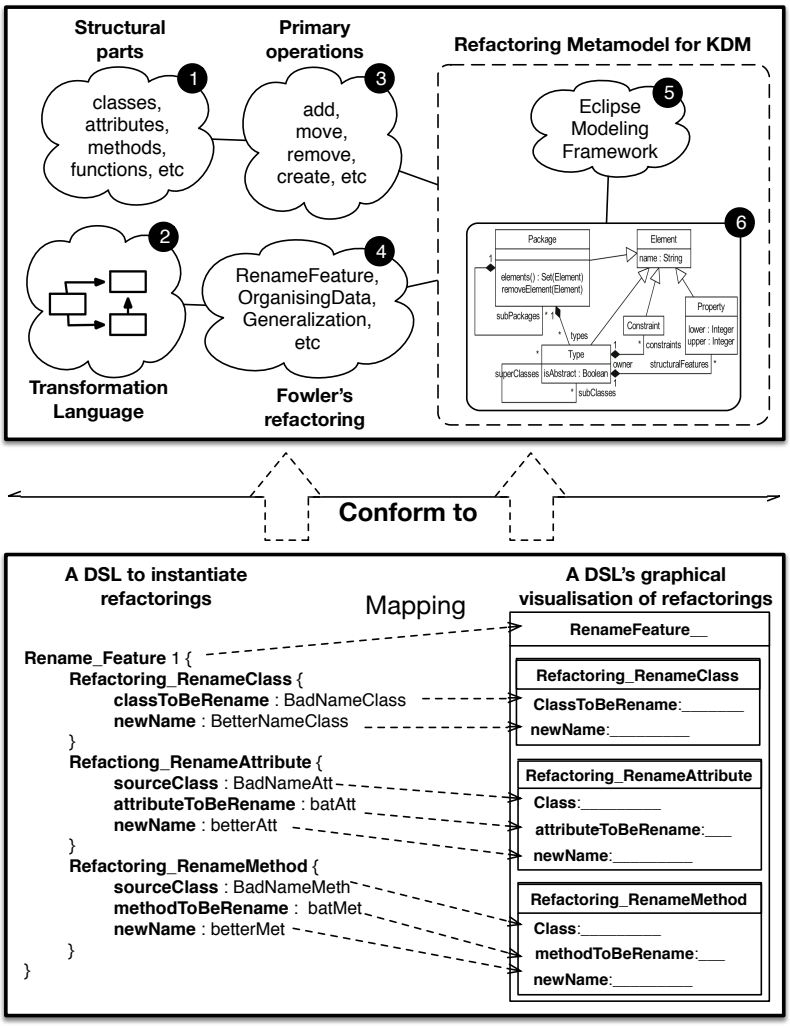


Figura 9: Princípios e critérios utilizados para criar o metamodelo de refatoração para o KDM

pecíficas. Por exemplo, considere a refatoração *RenameElement*. As etapas necessárias para executar tal refatoração são iguais, não importa que tipo de elemento precisa ser renomeado. Por exemplo, depois de alterar o valor de um atributo do tipo *String*, todas as referências daquele elemento precisa também ser atualizado. O atributo concreto pode variar dependendo da linguagem de programação, mas o procedimento é o mesmo.

Com base no exemplo anterior é possível identificar algumas idéias iniciais. Na parte superior da Figura 9. Na parte superior são ilustrados os principais conceitos que foram utilizados para criar o metamodelo de refatoração para o KDM. Em primeiro lugar, as par-

tes estruturais de uma refatoração (classes, métodos, atributos, etc), ou seja, os elementos que são transformados, foram considerados bons candidatos para o reuso (ver Figura 9 ❶) e assim utilizados como base para a criação do metamodelo de refatoração. Em segundo lugar, foi possível identificar na literatura pesquisas que executam uma refatoração utilizando um conjunto de composições de transformações por meio de linguagens de transformações, tais como OCL e ATL (ver Figura 9 ❷). Além disso, também foi constatado que a grande maioria das refatorações são realizadas utilizando operações primárias, tais como: *add*, *remove*, *move*, *create*, entre outras. Tais operações primárias também foram incluídas no metamodelo de refatoração para o KDM (ver Figura 9 ❸). Assim, os engenheiros de modernização podem realizar um *chain of primary operations* permitindo a especificar/-criar novas refatorações. O catalogo de refatoração proposto por Fowler (**?**) também foi considerado durante a criação do metamodelo de refatoração para o KDM como pode ser observado na Figura 9 ❹.

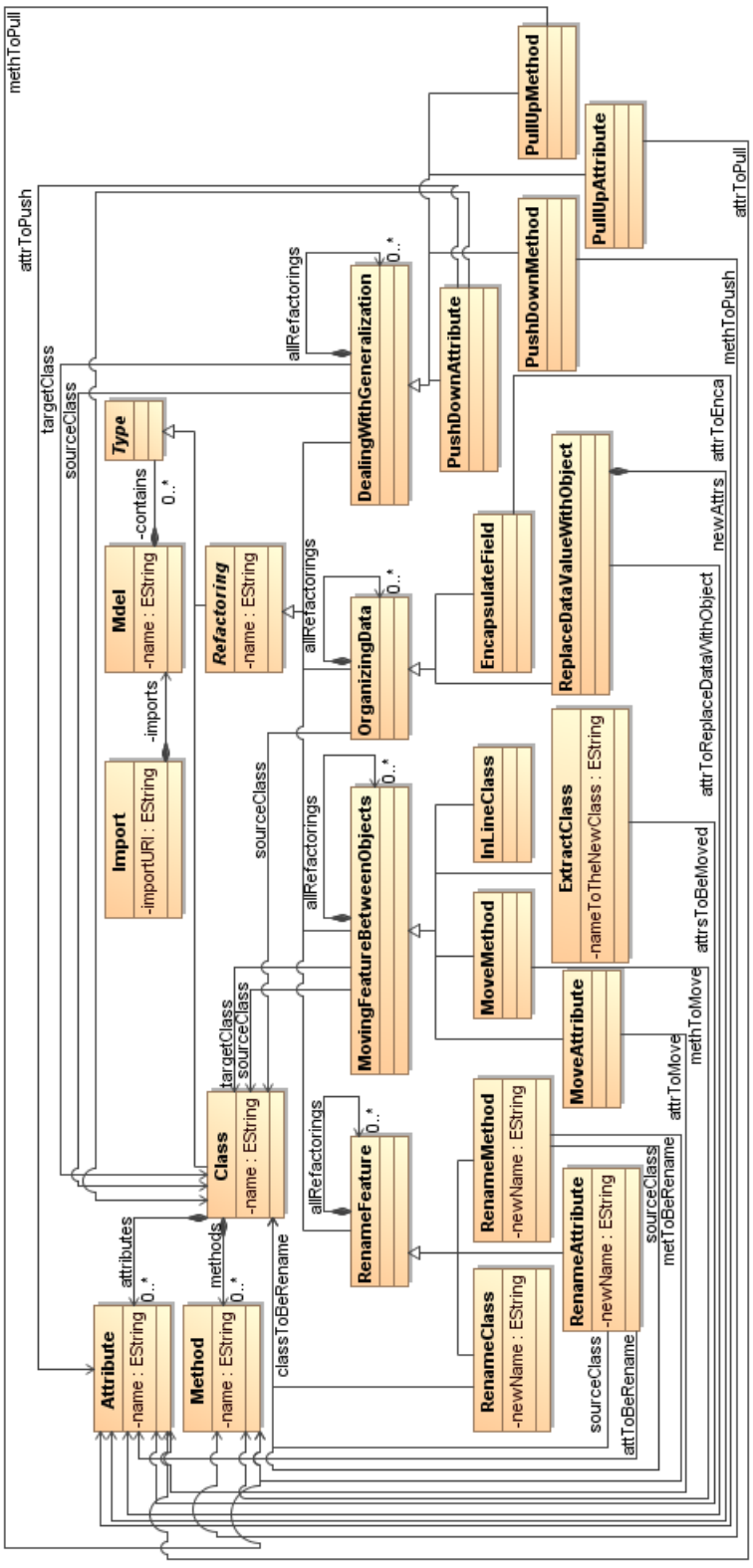


Figura 10: Metamodel de Refatoração para o KDM

Para reutilizar a parte estrutural de uma refatoração, um modelo que represente essa estrutura é necessário. Para o exemplo *RenameElement*, a estrutura basicamente consiste de uma *metaclass* em um *metamodel*. Outras refatorações, tais como *ExtractClass* possuem restrições estruturais mais complexas, como a exigência de que a classe que será extraída esteja contida em um *container object*. O metamodelo de refatorações para o KDM foi definido utilizado EMF (ver Figura 9 ❺). O metamodelo propriamente dito, Refac-KDM, pode ser visualizado na Figura 10. As *metaclasses* que compõem o metamodelo de refatoração, ou seja, Refac-KDM são definidas da seguinte forma:

- *Refactoring* é o elemento raiz utilizado para armazenar todas as informações sobre as refatorações.

- *RenameFeature* corresponde às categorias de refatorações relacionadas a ação de renomear.*RenameFeature* também armazena todas as refatorações relacionadas com a ação de renomear utilizando o *metaattribute allRefactorings*.

- *RenameClass* representa uma instância da refatoração *rename Class*. A semântica para a utilização da *RenameClass* consiste em: (i) especificar uma *ClassUnit* (*classToBeRenamed*), (ii) bem como uma *string* que representa o novo nome (*newName*) da *ClassUnit* após a aplicação da refatoração *renameClassUnit*.

- *RenameAttribute* representa uma instância da refatoração *rename Attribute*. A semântica para a utilização da *metaclass RenameAttribute* consiste em: (i) especificar a *ClassUnit* (*sourceClass*) que armazena todos os atributos e todos os métodos, (ii) o atributo que será renomeado (*attToBeRename*) e (iii) uma *string* que representa o novo nome (*newName*) do atributo após a aplicação da refatoração *renameAttribute*.

- *RenameMethod* representa uma instância da refatoração *rename Method*. A semântica para a utilização da *metaclass RenameMethod* consiste em: (i) especificar a *ClassUnit* (*sourceClass*) que armazena todos os atributos e todos os métodos, (ii) o método que será renomeado (*methToBeRename*) e (iii) uma *string* que representa o novo nome (*newName*) do método após a aplicação da refatoração *renameMethod*.

- *MovingFeatureBetweenObjects* corresponde às categorias de refatorações relacionadas a ação de movimentar características entre objetos.

- *MoveAttribute* representa uma instância da refatoração *move field*. A semântica para a utilização da *metaclass MoveAttribute* consiste em: (*i*) especificar a *ClassUnit* (*sourceClass*) que armazena todos os atributos e todos os métodos, (*ii*) a *ClassUnit* (*targetClass*) que irá receber o atributo a ser movido, (*iii*) e o atributo que será movimentado (*attrToMove*).

- *MoveMethod* representa uma instância da refatoração *move method*. A semântica para a utilização da *metaclass MoveMethod* consiste em: (*i*) especificar a *ClassUnit* (*sourceClass*) que armazena todos os atributos e todos os métodos, (*ii*) a *ClassUnit* (*targetClass*) que irá receber o método a ser movido, (*iii*) e o método que será movimentado (*methToMove*).

- *ExtractClass* representa uma instância da refatoração *extract class*. A semântica para a utilização da *metaclass ExtractClass* consiste em especificar em: (*i*) a *ClassUnit* (*sourceClass*) que armazena todos os atributos e todos os métodos, (*ii*) uma lista de atributos (*attrsToBeExtracted*) e uma lista de métodos (*methsToBeExtracted*) para serem extraídos, (*iii*) uma *string* que representa o nome (*newName*) da classe que foi extraída, (*iv*) bem como um *Package* (*targetPackage*) que irá receber a classe criada após a execução da refatoração.

- **TERMINAR AQUI TODAS AS METACLASSES**

4.5.1 Uma DSL para auxiliar a instânciação de refatorações com base no Refac-KDM

A fim de utilizar plenamente as vantagens das refatorações, os desenvolvedores precisam ter um bom conhecimento de linguagem de programação avançada. Na verdade os desenvolvedores devem estar familiarizados como as semânticas das refatorações (por exemplo, qual(is) é (são) o(s) pré-requisito(s) para a execução de uma refatoração) e como/onde

utilizar programar tais refatorações. A instanciação de uma refatoração utilizando o Refac-KDM é bastante verbosa, complexa e propensa a erros, uma vez que exige conhecimento avançadas de refatoração e habilidades avançadas de programação em relação a API Ecore. Com o objetivo de diminuir a quantidade de código-fonte, esforço e competência necessários para instanciar refatorações utilizando o Refac-KDM, foi desenvolvido uma linguagem especifica de domínio (do inglês, *Domain-Specific Language* - DSL) que auxilia a instanciação de refatorações sistematicamente. Na parte inferior a esquerda da Figura 9 é possível visualizar um exemplo da sintaxe da DSL criada para auxiliar a instanciação do metamodelo Refac-KDM.

A DSL para auxiliar a instanciação do Refac-KDM foi desenvolvida utilizando Xtext 3.4[16]. Xtext é um *framework* do Eclipse[17] que facilita a definição de gramática[18] com a utilização de um metamodelo que foi definido utilizando EMF. Xtext tem como principal objetivo automatizar e agilizar o processo de desenvolvimento de DSLs.

Em Xtext a gramática para especificar DSLs segue uma notação similar ao BNF chamada de regras do *parser*. Tais regras representam a sintaxe concreta da DSL. Note que para facilitar o entendimento da DSL, trechos da mesma são mostradas em listagens de códigos separados, bem como símbolos para explanar o propósito de uma terminada linha da gramática. Na Listagem de código 1 é ilustrado o primeiro trecho da gramática da DSL.

Listing 1: Gramática da DSL - parte 1

```
❶ grammar com.br.refactoring.xtext.RefacKdm with org.eclipse.xtext.common.Terminals
❷ import ''platform:/resource/com.br.refactoring.RefacKdm/model/RefacKdm.ecore'
❸ import ''http://www.eclipse.org/emf/2002/Ecore'' as ecore
Model:
   ❹ 'model' name=ID
   ❺ (imports+=Import)*
   ❻ (contains+=Type)*;
```

[16] https://www.eclipse.org/Xtext/

[17] https://www.eclipse.org

[18] Gramáticas representam a definição formal de um sintaxe textual concreta. Consistem em um conjunto de regras de produção para definir como o *textual input* (, i.e., sentenças) são representadas. Basicamente, as regras de produção podem ser representadas utilizando *Backus–Naur Form* (BNF), por exemplo, *S ::= P1 ... Pn*, essa gramática define um símbolo *S* por um conjunto de expressões *P1 ... Pn*.

A gramática começa com a definição do nome da DSL (RefacKdm) (ver Listagem de código 1 ❶). Em sequência é definido os metamodelos que devem ser importados para serem utilizados durante a criação da DSL, ou seja, o metamodelo RefacKdm❷ e o Ecore❸.

Em seguida é criado a primeira regra. Essa regra começa com a definição da *metaclass* `Model`, o corpo da regra começa após os `:`. Primeiramente para o entendimento da regra, é importante destacar que literais de *string* (que em Xtext podem ser expressas com aspas simples ou duplas) definem palavras-chave da DSL. Como pode ser observado na Listagem de código 1 é esperado a palavra-chave `model`❹ seguido por um `ID`. A gramática que rege o objeto `ID` é definida como uma sequência ilimitada de maiúsculas e minúsculas, números e o carácter de sublinhado, embora possa não começa por um dígito. A gramática que representa o nó `ID`❶ pode ser visualizada na Listagem de código 2.

Listing 2: Gramática da DSL - parte 2

```
❶ terminal ID: ('a'..'z' | 'A'..'Z'|'_')('a'..'z' | 'A'..'Z'|'_'|'0'..'9')*;
```

Ainda na Listagem de código 1 a expressão `(imports+=Import)*`❺ especifica que pode-se instanciar várias instancias da *metaclass* `Import`. O operador estrela, `*`, ilustra que o número de elementos (nesse caso `Import`) é arbitrário; em particular, ele pode ser qualquer número `>=` 0. Operador `+=` por sua vez representa que a propriedade `imports` será uma lista do tipo `Import`. A expressão `(contains+=Type)*`❻ descrita na Listagem de código 1 especifica que pode-se instanciar várias instâncias da *metaclass* `Type`.

Listing 3: Gramática da DSL - parte 3

```
Type:
  ❶ Refactoring | FormalizedDefinition;
Refactoring:
  ❷ (RenameFeature | MovingFeaturesBetweenObjects
    ➥ | OrganizingData | DealingWithGeneralization | PrimaryOperation);
```

A definição da *metaclass* `Type` é interessante, uma vez que `Type` representa uma *metaclass* abstrata com dois subtipos, como pode ser observado na Listagem de código 3. Ainda nessa listagem, é possível observar que o operador, `|`, é utilizado para expressar

alternativas ❶, o que é traduzido para o conceito de herança no metamodelo do Refac-KDM. A definição da *metaclass* `Refactoring` é ilustrada por um conjunto de *submetaclass* ❷ que são expressas na gramática da DSL pelo operador, `|`, a saber: `RenameFeature`, `MovingFeaturesBetweenObjects`, `OrganizingData`, `DealingWithGeneralization` and `Create`.

Listing 4: Gramática da DSL - parte 4

```
RenameFeature:
  ❶ 'Rename_Feature' name = ID '{'
    ➥ ❷ (allRefactorings+=RenameClass)*
    ➥ ❸ (allRefactorings+=RenameAttribute)*
    ➥ ❹ (allRefactorings+=RenameMethod)*
   '}';
```

A definição da *metaclass* `RenameFeature` é ilustrada na Listagem de código 4. Como pode ser observado essa regra começa com a definição da palavra-chave `Rename_Feature` seguida por um `ID` ❶. As expressões descritas em ❷, ❸ e ❹ representam que pode haver qualquer número de instancias das *metaclasses* `RenameClass`, `RenameAttribute` e `RenameMethod`, respectivamente.

Listing 5: Gramática da DSL - parte 5

```
RenameClass:
  ❶ 'Refactoring_RenameClass' name = ID '{'
    ➥ ❷ 'classToBeRenamed' ':' classToBeRenamed = [ClassUnit]
    ➥ ❸ 'newName' ':' newName = ID
  '}';
RenameAttribute:
  'Refactiong_RenameAttribute' name = ID '{'
    ➥ 'sourceClass' ':' sourceClass = [ClassUnit]
    ➥ ❹ 'attributeToBeRenamed' ':' attributeToBeRenamed = [StorableUnit]
    ➥ 'newName' ':' newName = ID
  '}';
RenameMethod:
  'Refactoring_RenameMethod' name = ID '{'
    ➥ 'sourceClass' ':' sourceClass = [ClassUnit]
    ➥ ❺ 'methodToBeRenamed' ':' methodToBeRenamed = [MethodUnit]
    ➥ 'newName' ':' newName = ID
  '}';
```

A definição da *metaclass* `RenameClass` é ilustrada na Listagem de código 5. A regra começa com a definição da palavra-chave `Refactoring_RenameClass` seguida por um `ID` e a chave, `{` ❶. Essa chave representa o inicio do escopo relacionado a refatoração *Rename Class*, ou seja, é um contexto delimitante aos quais valores e expressões estão associados a refatoração. Em seguida a palavra-chave `classToBeRenamed` é esperada, seguido por `:` ❷. Posteriormente uma instância da `ClassUnit` que será renomeada deve ser atribuída. Em Xtext, a notação `classToBeRenamed = [ClassUnit]` indica que é espero uma instância da *metaclass* `ClassUnit`, e não apenas um nome. Em seguida a palavra-chave `newName` é esperado, seguido por `:`, finalmente o novo nome da classe deve ser especificado ❸.

A gramática que define a semântica da refatoração *rename attribute* é similar a anterior. Como visualizado na Listagem de código 5 ❹, a única diferença é que deve-se atribuir uma instância da *metaclass* `StorableUnit`, ou seja, o atributo que almeja-se renomear. A semântica da refatoração *rename method* é praticamente a mesma que a refatoração *rename attribute*. A única diferença é que deve-se atribuir uma instância da *metaclass* `MethodUnit` ❺.

Listing 6: Gramática da DSL - parte 6

```
MovingFeaturesBetweenObjects:
  ❶ 'MovingFeaturesBetweenObjects' name = ID '{'
    ➥ ❷ (allRefactorings+=MoveAttribute)*
    ➥ ❸ (allRefactorings+=MoveMethod)*
    ➥ ❹ (allRefactorings+=ExtractClass)*
    ➥ ❺ (allRefactorings+=InlineClass)*
  '}';
```

A definição da *metaclass* `MovingFeaturesBetweenObjects` é ilustrada na Listagem de código 6. Perceba que tal definição é similar a da *metaclass* `RenameFeature` (ver Listagem de código 4). Como pode ser observado a regra começa com a definição da palavra-chave `MovingFeaturesBetweenObjects` seguida por um `ID` ❶. As expressões descritas em ❷, ❸, ❹ e ❺ representam que pode haver qualquer número de instancias das *metaclasses* `MoveAttribute`, `MoveMethod`, `ExtractClass` e `InlineClass`, respectivamente.

Listing 7: Gramática da DSL - parte 7

```
MoveAttribute:
  ❶ 'Refactoring_MoveAttribute' name = ID '{'
    ➥ 'sourceClass' ':' sourceClass = [ClassUnit]
    ➥ 'targetClass' ':' targetClass = [ClassUnit]
    ➥ 'attributeToBeMoved' ':' attributeToBeMoved = [StorableUnit]
  '}';
MoveMethod:
  'Refactoring_MoveMethod' name = ID '{'
    ➥ 'sourceClass' ':' sourceClass = [ClassUnit]
    ➥ 'targetClass' ':' targetClass = [ClassUnit]
    ➥ ❷ 'methodToBeMoved' ':' methodToBeMoved = [MethodUnit]
  '}';
ExtractClass:
  'Refactoring_ExtractClass' name = ID '{'
    ➥'sourceClass' ':' sourceClass = [ClassUnit]
    ➥ ❸ 'attribute(s)ToBeMoved' ':'
      ➥ '{'attributesToBeMoved += [StorableUnit]
        ➥ (','(attributesToBeMoved += [StorableUnit]))*'}'
    ➥ ❹ 'nameToTheNewClass' ':' newName = ID
  '}';
InlineClass:
  'Refactoring_InlineClass' name = ID '{'
    ➥ ❺ 'classToGetAllFeatures' ':' classToGetAllFeatures = [ClassUnit]
    ➥ ❻ 'classToRemove' ':' classToRemove = [ClassUnit]
  '}';
```

A semântica da refatoração *move attribute* é definido na gramática da Listagem de código 7 ❶. Como pode ser observado é esperado a palavra-chave `Refactoring_MoveAttribute` seguida por um `ID` e `{`. Posteriormente deve-se especificar a palavra-chave `sourceClass`, seguido por `:`. Uma instância da `ClassUnit` que contém o atributo que será movimentado deve ser atribuída. Em seguida, deve-se indicar a `targetClass`. Novamente, uma instância de *ClassUnit* deve ser atribuída, ou seja, a classe que receberá o atributo. Por fim, deve-se especificar qual atributo realmente será movimentado. Assim, a palavra-chave `attributeToBeMoved` deve ser especificada, seguido por `:`. Uma instância de `StorableUnit` que representa o atributo que será movimentado deve ser atribuída.

A regra que descreve a refatoração `move method` é similar a refatoração *move attribute* como pode ser observado na Listagem de código 7, duas principais diferenças podem ser notadas. A primeira diferença é a primeira palavra-chave da regra `Refactoring_MoveMethod`. A segunda diferença pode ser ser observado na expressão `methodToBeMoved = [MethodUnit]` ❷. Essa expressão mostra que deve-se especificar o método que será movimentado.

A refatoração *extract class* é definida na sequência. A regra sempre começa com a palavra-chave `Refactoring_ExtractClass` seguida por um `ID` e `{`. Similarmente, também deve-se especificar a palavra-chave `sourceClass` seguido por `:`. Uma instância da `ClassUnit` que contém os atributos que serão extraídos deve ser especificada. A diferença pode ser observado na expressão ilustrada pelo símbolo ❸. Essa expressão inicia-se com a palavra-chave `attribute(s)ToBeMoved` seguido por `:` e `{`, nesse contexto as chaves representam o(s) atributo(s) que será(ão) extraído(s). Em seguida a expressão descreve que ao menos uma instância de `StorableUnit` deve ser atribuída, porém, pode-se especificar mais do que uma instância de `StorableUnit` para ser extraído, simplesmente separando cada `StorableUnit` por virgula (`,`). Em seguida, a palavra-chave `nameToTheNewClass` deve ser especifica seguido por `:` e `ID`.

Para satisfazer a semântica da gramática relacionada a refatoração *inline class* deve-se primeiramente especificar a palavra-chave `Refactoring_InlineClass` seguida por um `ID` e `{`. Em seguida, deve-se especificar a palavra-chave `classToGetAllFeatures` e `:` (ver ❺). Uma instância de uma `ClassUnit` deve ser atribuída. Finalmente, deve-se especificar a palavra-chave `classToRemove`, `:` e uma instância de `ClassUnit` que representa a classe que será removida (ver ❺).

Listing 8: Gramática da DSL - parte 8

```
OrganizingData:
  ❶ 'OrganizingData' name = ID '{'
    ➥ ❷ (allRefactorings+=ReplaceDataValueWithObject)*
    ➥ ❸ (allRefactorings+=EncapsulateField)*
  '}';
```

A definição da *metaclass* `OrganizingData` é ilustrada na Listagem de código 8. Note que a regra começa com a definição da palavra-chave `OrganizingData` seguida por um `ID` ❶. As expressões descritas em ❷ e ❸ representam que pode haver qualquer número de instancias das *metaclasses* `ReplaceDataValueWithObject` e `EncapsulateField`, respectivamente.

Listing 9: Gramática da DSL - parte 9

```
ReplaceDataValueWithObject:
  'Refactoring_ReplaceDataValueWithObject' name = ID '{'
    ➥ ❶ 'sourceClass' ':' sourceClass = [ClassUnit]
    ➥ ❷ 'attributeToReplaceDataWithObject' ':'
      ➥ attributeToReplaceDataWithObject = [StorableUnit]
    ➥ ❸ 'newAttributes' ':' '{'
      ➥ (newAttributes+=StorableUnit)*
    '}'
  '}';
EncapsulateField:
  'Refactoring_EncapsulateField' name = ID '{'
    ➥ ❹' sourceClass' ':' sourceClass = [ClassUnit]
    ➥ ❺ 'attributeToEncapsulate' ':' attributeToEncapsulate = [StorableUnit]
  '}';
```

A semântica da refatoração *replace data value with object* é definido na gramática da Listagem de código 9. Note que é esperado a palavra-chave `Refactoring_ReplaceDataValueWithObject` seguida por um `ID` e `{`. Posteriormente deve-se especificar a palavra-chave `sourceClass`, `:` e uma instância da `ClassUnit` deve ser atribuída (ver ❶). Em seguida, deve-se especificar a palavra-chave `attributeToReplaceDataWithObject`, `:` e uma instância de *StorableUnit* também deve ser atribuído (ver ❷). Por fim, deve-se especificar a palavra-chave `newAttributes`, `:` seguida por um conjunto `StorableUnit` (ver ❸).

A refatoração *encapsulate field* é definida na sequência. A regra sempre começa com a palavra-chave `Refactoring_EncapsulateField` seguida por um `ID` e `{`. Similarmente, também deve-se especificar a palavra-chave `sourceClass` seguido por `:` e uma instância de `ClassUnit` (ver ❹). Em seguida, deve-se especificar a palavra-chave `attributeToEncapsulate`, `:` e uma instância de *StorableUnit*, o qual será encapsulado (ver ❺).

Listing 10: Gramática da DSL - parte 10

```
DealingWithGeneralization:
  ❶ 'DealingWithGeneralization' name = ID '{'
    ➥ ❷ (allRefactorings+=PushDownAttribute)*
    ➥ ❸ (allRefactorings+=PushDownMethod)*
    ➥ ❹ (allRefactorings+=PullUpAttribute)*
    ➥ ❺ (allRefactorings+=PullUpMethod)*
  '}';
```

A definição da *metaclass* `DealingWithGeneralization` é ilustrada na Listagem de código 10. Note que a regra começa com a definição da palavra-chave `DealingWithGeneralization` seguida por um `ID` e a chave, `{` ❶. As expressões descritas em ❷, ❸, ❹ e ❺ representam que pode haver qualquer número de instancias das *metaclasses* `PushDownAttribute`, `PushDownMethod`, `PullUpAttribute` e `PullUpMethod`, respectivamente.

Listing 11: Gramática da DSL - parte 11

```
PushDownAttribute:
  'Refactoring_PushDownAttribute' name = ID '{'
    ➥ 'sourceClass' ':' sourceClass = [ClassUnit]
    ➥ 'attributeToPushDown' ':' attributeToBePushed = [StorableUnit]
    ➥ 'targetClass' ':' targetClass = [ClassUnit] '}';
PushDownMethod:
  'Refactoring_PushDownMethod' name = ID '{'
    ➥ 'sourceClass' ':' sourceClass = [ClassUnit]
    ➥ 'methodToPushDown' ':' methodToBePushed = [MethodUnit]
    ➥ 'targetClass' ':' targetClass = [ClassUnit] '}';
PullUpAttribute:
  'Refactoring_PullUpAttribute' name = ID '{'
    ➥ 'sourceClass' ':' sourceClass = [ClassUnit]
    ➥ 'attributeToPullUp' ':' attributeToBePulled = [StorableUnit]
    ➥ 'targetClass' ':' targetClass = [ClassUnit] '}';
PullUpMethod:
  'Refactoring_PullUpMethod' name = ID '{'
    ➥ 'sourceClass' ':' sourceClass = [ClassUnit]
    ➥ 'methodToPullUp' ':' methodToBePulled = [MethodUnit]
    ➥ 'targetClass' ':' targetClass = [ClassUnit] '}';
```

As regras que definem a semântica das *metaclasses* `PushDownAttribute`, `PushDownMethod`, `PullUpAttribute` e `PullUpMethod` estão definidas na Listagem de código 11.

Listing 12: Gramática da DSL - parte 12

```
❶ PrimaryOperation:
  (CreateClass | CreateAttribute | CreateMethod |
    ➥ RemoveClass | RemoveAttribute | RemoveMethod |
      ➥ MoveClass | CreateInheritance)
;
CreateClass:
  'Refactoring_CreateClass' '{'
    ➥ ❷ 'package' ':' package = [Package]
    ➥ ❸ 'class' ':' name = ID
  '}';
CreateAttribute:
  'Refactoring_CreateAttribute' '{'
    ➥ 'class' ':'className = ID
    ➥ '@' attribute = ID
  '}';
```

A definição da *metaclass* `PrimaryOperation` é ilustrada na Listagem de código 12 ❶. Essa *metaclass* é responsável por representar algumas das operações básicas que podem constituir uma refatoração (*add*, *remove*, *move*, *create*). Na Listagem de código 12 apenas algumas dessas operações são destacadas. A operação *create class* é destacado na Listagem de código 12 ❷, perceba que a regra começa com a definição da palavra-chave `Refactoring_CreateClass` seguida pela outra palavra-chave `package` e `:`. Em sequência a palavra-chave `class` e `:` devem ser especificadas (ver ❸). A operação *create attribute* é similar a anterior.

Listing 13: Gramática da DSL - parte 13

```
❶ FormalizedDefinition:
  'Refactoring_FormalizedDefinition' '{'
    ➥ (allDefinition+=BooleanExpression)*
  '}';
BooleanExpression:
  ❷ AndExpression | OrExpression | NotExpression;
AndExpression:
    ➥ 'AndExpr' ':' ref1 = Refactoring
```

```
    ➥ '&&' ref2 = Refactoring;
OrExpression:
    ➥ 'OrExpr' ':' ref1 = Refactoring
    ➥ '| |' ref2 = Refactoring ;
NotExpression:
    ➥ 'NotExpr' ':' ref = Refactoring;
```

A definição da *metaclass* `FormalizedDefinition` é ilustrada na Listagem de código 13 ❶. Utilizando as regras ilustradas na Listagem de código 13 ❷ os engenheiros de modernização podem realizar um *chain of primary operations* permitindo a especificar/criar novas refatorações. Por exemplo, é possível criar expressões da seguinte forma, `createClass && (moveAttribute && createMethod)`.

4.5.2 Uma notação gráfica (concreta) da DSL para auxiliar a instânciação de refatorações com base no Refac-KDM

Usualmente os modelos gráficos são mais intuitivos de se utilizar. Assim, com o intuito de prover maior facilidade na instanciação do metamodelo Refac-KDM criou-se uma notação gráfica da DSL. Essa notação gráfica também auxilia o engenheiro de modernização, uma vez que o mesmo não precisa escrever as refatorações utilizando uma linguagem textual, ou seja, apenas elementos gráficos são utilizados para criar e definir as refatorações. Posteriormente, a IDE automaticamente fornece uma sincronização entre a DSL concreta e a DSL textual, provendo assim uma multi-visualização da DSL, gráfica e textual.

A notação gráfica é composta por elementos que permitem a visualização e a edição das informações definidas na DSL de forma gráfica. A construção da sintaxe concreta de uma DSL é dependente da ferramenta adotada para esse fim. Por exemplo, o *Graphical Modeling Framework* (GMF)[19] utiliza três tipos de modelos para definir a notação gráfica da DSL: (*i*) *Gmfgraph*, (*ii*) *Gmftool* e (*iii*) Gmfmap. Na parte inferior a direita da Figura 9 é possível visualizar um exemplo da notação gráfica que foi criada para dar suporte a DSL apresentada na Seção 4.5.1.

[19] `https://www.eclipse.org/modeling/gmp/`

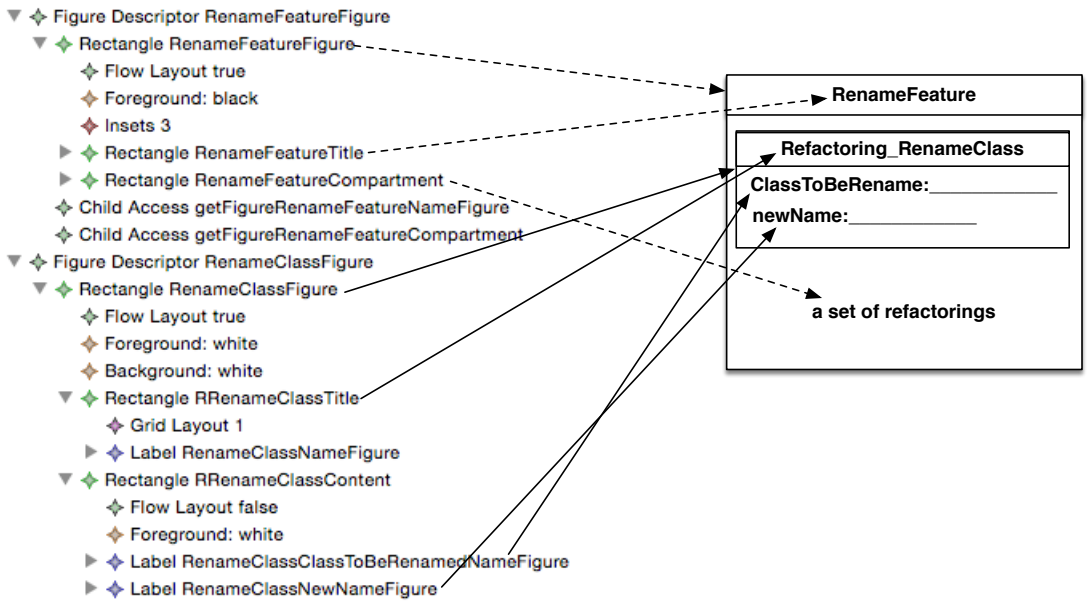


Figura 11: Definição da notação gráfica da DSL

No modelo *Gmfgraph* são definidos os componentes da notação gráfica da DSL que correspondem às refatorações e seus relacionamentos. Cada componente gráfico pode ser formado pela junção de vários componentes de forma aninhada. Por exemplo, na Figura 11 é mostrado que o conjunto de refatoração relacionado a *rename feature* é representada graficamente por um retângulo (*Rectangle RenameFeatureFigure*) composto de outros retângulos que contêm um rótulo (*RenameFeatureTitle*) e um compartimento (*RenameFeatureCompartment*). Similarmente, a refatoração *rename class* também é representada graficamente por um retângulo (*Rectangle RenameClassFigure*) composto de outros retângulos que contêm um rótulo (*RRenameClassTitle*) e um compartimento (*RRenameClassContent*) que também contêm dois outros rótulos (*RenameClassClassToBeRenamedNameFigure* e (*RenameClassNewNameFigure*)).

No modelo *Gmftool* são definidos os itens de menu dos elementos e dos relacionamentos que podem ser utilizados. Na Figure 12 é mostrado o modelo *Gmftool* da DSL, no qual são definidos as refatorações. Note que foi criado um item para cada uma das refatorações: *extract class*, *move attribute*, *move method*, etc.

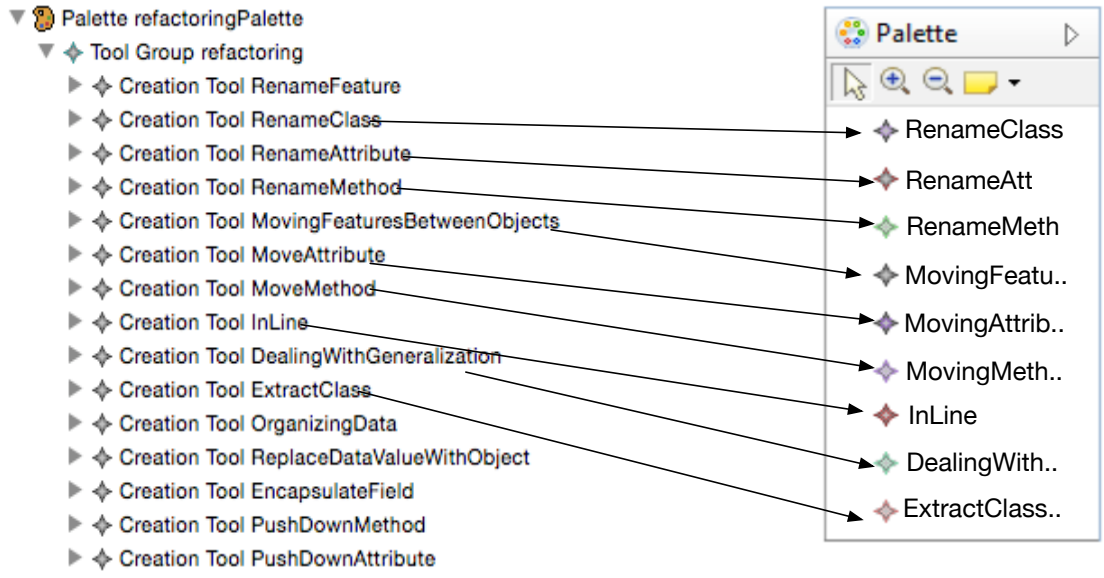


Figura 12: Definição dos itens do menu da DSL

No modelo *Gmfmap* os elementos do metamodelo são combinados com os seus respectivos componentes da notação gráfica e itens de menu. Na Figura 13 é ilustrado um exemplo de modelo *Gmfmap* que combina as *metaclasses* e os relacionamentos do metamodelo (Figura 10) com a notação gráfica definida no modelo *Gmfgraph* (Figura 11) e com os itens de menu definidos no modelo *Gmftool* (Figura 12). Nesse modelo cada refatoração é definida com a inclusão de um *Top Node Reference* contendo um *Node Mapping*. *Feature Labels* são acrescentados para permitir a visualização e a edição dos nomes e dos tipos das refatorações.

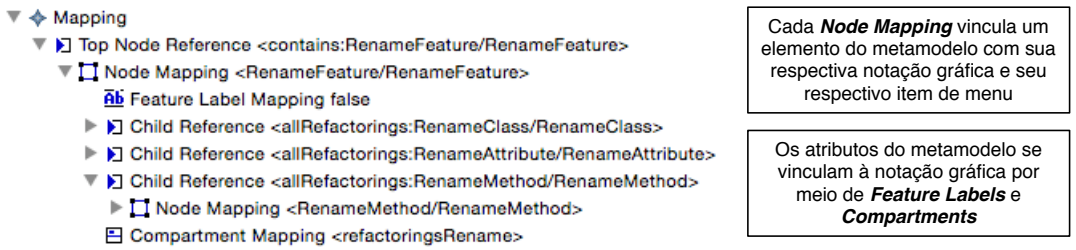


Figura 13: Modelo *Gmfmap da DSL*

4.6 Publicações, Participação em Eventos Científicos e Apresentações de Trabalhos

Nesta seção são apresentadas as publicações, participações em eventos científicos e apresentações de trabalho realizadas pelo bolsista durante o primeiro período de vigência da bolsa de doutorado.

Por enquanto, o resultado mais relevente do projeto é proveniente de uma revisão sistemática conduzida pelo bolsista com o intuito de identificar e obter conhecimento da área de pesquisa. Assim, um artigo sumarizando as observações dessa revisão sistemática foi submetido e aceito para o evento ACM SAC 2013, o qual possui Qualis A1. Tal evento será realizado em Coimbra, Portugal e será apresentado pelo bolsista em Março/2013. O artigo é denominado *A Systematic Review on Mining Techniques for Crosscutting Concerns* e encontra-se em anexo a este relatório. A seguir são descritos outras publicações que foram conduzidas como atividades complementares e não estão diretamente relacionados ao projeto de pesquisa do bolsista. Vale resssaltar que tais artigos foram desenvolvidos em parceria com outros membros do grupo de pesquisa. Tais artigos foram aceitos e apresentados por um dos autores. Além disso é importante salientar que não foi usado nenhum recurso disponível em Reserva Técnica para tais artigos, tais como: inscrição, viagem ou hospedagem.

- **Trabalhos completos publicados em anais de congressos**
  - DURELLI, R. S. ; DURELLI, V. H. S. . A Systematic Mapping Study on Formal Methods Applied to Crosscutting Concerns Mining. In: IX Experimental Software Engineering Latin American Workshop (ESELAW), 2012, Buenos Aires. IX Experimental Software Engineering Latin American Workshop (ESELAW), 2012 [20].

[20] Vale ressaltar que esse artigo é uma atividade complementar e não relacionada ao projeto de pesquisa. Esse artigo foi desenvolvido em parceria com outros membros do grupo de pesquisa. Tal artigo foi aceito e apresentado por um dos autores. Além disso é importante salientar que não foi usado nenhum recurso disponível em Reserva Técnica para inscrição, viagem ou hospedagem do bolsista.

- DURELLI, R. S. ; DURELLI, V. H. S. . F2MoC: A Preliminary Product Line DSL for Mobile Robots. In: Simpósio Brasileiro de Sistemas de Informação (SBSI), 2012, São Paulo. Simpósio Brasileiro de Sistemas de Informação (SBSI), 2012 [20].
- Gottardi ; DURELLI, R. S. ; PASTOR, O. L. ; CAMARGO, V. V. . Model-Based Reuse for Crosscutting Frameworks: Assessing Reuse and Maintainability Effort. In: Simpósio Brasileiro de Engenharia de Software, 2012, Natal. Simpósio Brasileiro de Engenharia de Software, 2012 [20].
- DURELLI, R. S. ; Gottardi ; CAMARGO, V. V. . CrossFIRE: An Infrastructure for Storing Crosscutting Framework Families and Supporting their Model-Based Reuse. In: XXVI Simpósio Brasileiro de Engenharia de Software - XXVI Sessão de Ferramenta, 2012, Natal. Simpósio Brasileiro de Engenharia de Software, 2012. v. 6. p. 1-6 [21].
- DURELLI, R. S. ; SANTIBANEZ, D. S. M. ; ANQUETIL, N. ; DELAMARO, M. E. ; CAMARGO, V. V. . A Systematic Review on Mining Techniques for Crosscutting Concerns (to appear). In: ACM SAC 2013, 2012, Coimbra. ACM SAC Software Engineering (SE) Track, 2013. v. 28th [22].

- **Resumos expandidos publicados em anais de congressos**

  - DURELLI, R. S. ; SANTIBANEZ, D. S. M. ; DELAMARO, M. E. ; CAMARGO, V. V. . A Systematic Review on Mining Techniques for Crosscutting Concerns. In: 6th, Latin American Workshop on Aspect-Oriented Software Development Advanced Modularization Techniques, 2012, Natal. Latin American Workshop on Aspect-Oriented Software Development Advanced Modularization Techniques, 2012

[21] Vale ressaltar que esse artigo é uma atividade complementar e não relacionada ao projeto de pesquisa. Esse artigo foi desenvolvido em parceria com outros membros do grupo de pesquisa. Tal artigo foi aceito e apresentado por um dos autores

[22] Foi usado recurso disponível em Reserva Técnica para pagar a inscrição desse evento. Outras despesas como passagem áerea e diários foram solicitadas e adquiridas pela Programa de Apoio à Pós-Graduação (PROAP).

- **Apresentação de Trabalho**

  – DURELLI, R. S. ; Gottardi ; CAMARGO, V. V. . CrossFIRE: An Infrastructure for Storing Crosscutting Framework Families and Supporting their Model-Based Reuse. Trabalho apresentado no XXIII Simpósio Brasileiro de Engenharia de Software - XVIII Sessão de Ferramentas, 2012, Natal - RN - Brasil (Apresentação de Trabalho/Simpósio).

4.7 Elaboração do Texto para o Exame de Qualificação

Durante o período ao qual se refere esse relatório, o bolsista realizou a elaboração do texto para o exame de qualificação exigido pelo Programa de Pós-Gradução em Ciências de Computação e Matemática Computacional. A realização da Revisão Sistemática, a escolha das disciplinas oferecidas pelo Programa, publicação e participações em eventos científicos foram fundamentais para o bom aproveitamento deste atividade.

O texto da qualificação possui um total de 4 Capítulos. O primeiro capítulo da qualificação contêm as seguintes seções: (*i*) **Contextualização**, que apresenta um resumo do trabalho a ser desenvolvido - (*ii*) **Motivação**, onde são descritas as motivações para a realização do trabalho a ser desenvolvido - (*iii*) **Objetivos**, a qual descreve os principais objetivos do projeto de doutorado - e, (*iv*) **Organização**, que descreve a organização do texto da qualificação.

O segundo capítulo consiste das seguintes seções: (*i*) **Considerações Iniciais**, que descreve o objetivo do capitulo - (*ii*) **Modernização Dirigida à Arquitetura**, onde são descritos os principais conceitos sobre essa abordagem - (*iii*) **Reengenharia de Software**, a qual descreve todos os conceitos sobre reengenharia de software, bem como os conceitos de refatoração - (*iv*) **Desenvolvimento Dirigido a Modelos**, descreve os conceitos relacionados a tal abordagem - (*v*) **Considerações Finais**, o qual conclui e descreve o conteúdo que foi apresentado nesse capítulo.

O terceiro capítulo descreve e apresenta vários trabalhos relacionados com o projeto de pesquisa. E o quarto capítulo contêm as seguintes seções: (*i*) **Considerações Iniciais**, que descreve o objetivo do capitulo - (*ii*) **Proposta de Projeto**, onde descreve os objetivos do trabalho a ser desenvolvido pelo bolsista - (*iii*) **Desafios de Pesquisa com Relação ao Projeto**, apresenta os principais desafios que o bolsita possivelmente irá enfrentar durante o desenvolvimento dos objetivos do trabalho - (*iv*) **Metodologia**, descreve os passos que o bolsista irá realizar para realizar os objetivos do trabalho - (*v*) **Avaliação**, apresenta como o bolsista irá avaliar o trabalho realizado - (*vi*) **Resultados Esperados**, descreve os principais resultados esperados ao término do trabalho - (*vii*) **Colaborações**, descreve as principais colaborações que o bolsista tem com outros pesquisadores - (*viii*) **Considerações Finais**, conclui e descreve o conteúdo que foi apresentado nesse capítulo.

É importante destacar que a apresentação da qualificação foi realizada no dia 12 de Dezembro de 2012. Tal apresentação resultou na aprovação do projeto proposto.

4.8 Atividades Complementares não Relacionadas ao Projeto

Nesta seção são descritas as atividades complementares não relacionadas ao projeto que foram realizadas até o momento. Vale ressaltar que embora tais atividades não estejam totalmente relacionadas ao projeto do bolsista, são de suma importância para o desenvolvimento e aperfeiçoamento da formação do bolsista.

4.8.1 Exame de proficiência em lingua estrangeira

No contexto do programa de Pós-Graduação do ICMC, um dos requisitos para obtenção do título de doutor é que o candidato deve submeter-se ao exame de proficiência em língua ingles em até 18 meses após o ingresso no curso. Dessa forma, o bolsista realizou o Test of English as a Foreign Language (TOEFL) em sua modalidade Internet-based test (iBT), obtendo uma pontuação de 102. O exame foi realizado dia 27 de Outubro de 2012, 8 meses antes do prazo estipulado pelo programa, e o resultado foi entregue na secretaria dia 13 de novembro de 2012.

4.8.2 Estágio no Programa de Aperfeicoamento do Ensino (PAE)

No período de Junho até Novembro de 2012 o bolsista foi estagiário do programa PAE, sob a supervisão do Prof. Dr. Paulo Cesar Masiero. Durante esse período, o bolsista colaborou com o supervisor na elaboração de material e na correção das atividades atribuídas aos alunos da disciplina Engenharia de Software para Sistemas Embarcados. O estágio teve carga horária de 6 horas semanais.

4.9 Atividades Previstas para o Próximo Período

A seguir são descritas as atividades que o bolsista pretende conduzir durante o próximo período de vigência da bolsa.

1. **Implementação da abordagem de modernização:** no próximo período de vigência do bolsista, o mesmo irá estudar qual a melhor forma para implementar a abordagem de modernização de sistemas legados. A princípio a implementação constituirá de um *plug-in* do Eclipse[23]. Nesse *plug-in* o engenheiro de software entrará com um sistema legado que almeja-se ser modernizado. Assim, a abordagem implementada irá modernizar tal sistema utilizando técnicas de MDD, ADM e KDM (ver Seções **??**, **??** and **??**).

2. **Redação de artigos:** pretende-se continuar produzindo artigos, com o intuito de documentar o progresso do projeto em questão. O próximo passo consiste em publicar um experimento avaliando a implementação que será realizada. Uma possível abordagem para condução do experimento seria constratar a abordagem implementada pelo bolsista com uma outra abordagem já disponível na literatura.

3. **Avaliação:** após a implementação da abordagem proposta pelo bolsista, pretende-se avaliar o ambiente de modernização resultante. Pretende-se realizar dois tipos de experimentos. O primeiro consiste em realizar um conjunto de estudos de casos para investigar a viabilidade da abordagem proposta, bem como avaliar o uso das

[23] www.eclipse.org

funcionalidades do apoio computacional para fornecer suporte à modernização de sistemas legados. O segundo experimento consistem em realizar avaliações controladas utilizando a metodologia experimental **?**, a fim de avaliar o impacto da abordagem proposta, bem como do apoio computacional relacionado a eficiência e impacto das equipes e também a qualidade em termos de modularidade, reuso e manutenibilidade dos sistema resultantes durante a atividade de modernização.

Referências

Bézivin, J. On the unification power of models. *Software and Systems Modeling*, vol. 4, no. 2, pp. 171–188, 2005.

Canfora, G.; Cimitile, A.; Munro, M. Re2: Reverse-engineering and reuse re-engineering. *Journal of Software Maintenance: Research and Practice*, vol. 6, pp. 53–72, 1994.

Canfora, G.; Di Penta, M.; Cerulo, L. Achievements and challenges in software reverse engineering. *Commun. ACM*, vol. 54, no. 4, pp. 142–151, 2011.

Chikofsky, E.; Cross, J.H., I. Reverse engineering and design recovery: a taxonomy. *Software, IEEE*, vol. 7, no. 1, pp. 13–17, 1990.

Demeyer, S.; Du Bois, B.; Verelst, J. Refactoring - Improving Coupling and Cohesion of Existing Code. *Reverse Engineering, 2004. Proceedings. 11th Working Conference on*, 2004.

Durelli, R.; Santos M, B.; Honda, R.; Delamaro, M.; Camargo, V. Kdm-re: A model-driven refactoring tool for kdm. *Workshop on Software Visualization, Evolution and Maintenance*, pp. 1–8, 2014a.

Durelli, R. S.; Santibáñez, D. S. M.; Delamaro, M. E.; Camargo, V. V. A mapping study on architecture-driven modernization. In: *IEEE 15th International Conference on Information Reuse and Integration (IRI)*, 2014b, pp. 168–178.

Durelli, R. S.; Santibáñez, D. S. M.; Delamaro, M. E.; Camargo, V. V. Towards a refactoring catalogue for knowledge discovery metamodel. In: *IEEE 15th International Conference on Information Reuse and Integration (IRI)*, 2014c, pp. 158–168.

Eilam, E. *Reversing: Secrets of Reverse Engineering.* Wiley, 624 pp., 2005.

Fowler, M.; Beck, K.; Brant, J.; Opdyke, W.; Roberts, D. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley, 1999.

Griffith, I.; Wahl, S.; Izurieta, C. Evolution of legacy system comprehensibility through automated refactoring. In: *Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering*, New York, NY, USA: ACM, 2011, pp. 35–42.

ISO/IEC DIS 19506 Knowledge Discovery Meta-model (KDM), v1.1. Online, `http://www.iso.org/iso/catalogue_detail.htm?csnumber=32625` - Acessado em 03/07/2012, 2011.

Kent, S. Model driven engineering. In: Butler, M.; Petre, L.; Sere, K., eds. *Integrated Formal Methods*, Springer Berlin Heidelberg, pp. 286–298, 2002.

Kitchenham, B.; Pearl Brereton, O.; Budgen, D.; Turner, M.; Bailey, J.; Linkman, S. Systematic literature reviews in software engineering - a systematic literature review. *Information and Software Technology*, vol. 51, pp. 7–15, 2009.

Kleppe, A. G.; Warmer, J.; Bast, W. *Mda explained: The model driven architecture: Practice and promise.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

Mens, T.; Tourwe, T. A Survey of Software Refactoring. *Transactions on Software Engineering, IEEE*, vol. 30, 2004.

OMG Object Management Group (OMG) Architecture-Driven Modernisation. Disponível em: *http://www.omgwiki.org/admtf/doku.php?id=start*, (Acessado 2 de Agosto de 2012), 2012a.

OMG Object Management Group (OMG) Model-Driven Development — OMG Available Specification without Change Bars. Disponível em: *http://www.omg.org*, (Acessado 2 de Agosto de 2012), 2012b.

Opdyke, W. F. *Refactoring Object-Oriented Frameworks.* Ph.D. Thesis, University of Illinois, 1992.

Perez-Castillo, R.; Garcia-Rodriguez de Guzman, I.; Piattini, M. Knowledge Discovery Metamodel-ISO/IEC 19506: A standard to modernize legacy systems. *Computer Standards & Interfaces*, vol. 33, no. 6, pp. 519–532, 2011a.

Pérez-Castillo, R.; de Guzmán, I. G. R.; Caballero, I.; Piattini, M. Software modernization by recovering web services from legacy databases. *Journal of Software: Evolution and Process*, vol. 25, no. 5, pp. 507–533, 2013.

Perez-Castillo, R.; de Guzmán, I. G.-R.; Piattini, M. *Architecture driven modernization.* San Francisco, CA, USA: Information Science Reference, 2011b.

Premerlani, W.; Blaha, M. An approach for reverse engineering of relational databases. In: *Proceedings of Working Conference on Reverse Engineering*, 1993, pp. 151–160.

Rugaber, S.; Stirewalt, K. Model-driven reverse engineering. *Software, IEEE*, vol. 21, no. 4, pp. 45–53, 2004.

Smith, J.; Nair, R. *Software Reuse and Reverse Engineering in Practice.* Chapman and Hall Ltd, 608 pp., 1992.

Thomas, D. Mda: revenge of the modelers or uml utopia? *Software, IEEE*, pp. 15 – 17, 2004.

Ulrich, W. M.; Newcomb, P. *Information systems transformation: Architecture-driven modernization case studies.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010a.

Ulrich, W. M.; Newcomb, P. *Information systems transformation: Architecture-driven modernization case studies.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010b.

Wohlin, C.; Runeson, P.; Höst, M.; Ohlsson, M. C.; Regnell, B.; Wesslén, A. *Experimentation in software engineering: an introduction.* Norwell, MA, USA: Kluwer Academic Publishers, 2000.

5 Anexos

Anexo 1: Comprovante da publicação no *Experimental Software Engineering Latin American Workshop* (ESELAW'12)

Cópia do email de aceitação (Somente a primeira página)

ESELAW 2012 <francine@icmc.usp.br>
To: rafael_durelli@dc.ufscar.br ,Cc: Vinícius Humberto Durelli <durelli@icmc.usp.br>
Your ESELAW 2012 paper 96336

Dear Ms. Rafael Durelli:

Congratulations - your paper "A Systematic Mapping Study on Formal Methods Applied to Crosscutting Concerns Mining" for ESELAW 2012 has been accepted. We would like to inform you that we had a highly competitive selection process as we received a large number of qualified paper s .

The reviews are below or can be found them at https://submissoes.sbc.org.br/PaperShow.cgi?m=96336. We expect to receive the final version no later than March 1st, 2012, 23:55PM (BR Time). The final version should be uploaded using the JEMS system (https://submissoes.sbc.org.br/eselaw2012) using the "Camera Ready" tracking.

Only PDF files are accepted. The maximum length is 14 pages and the Springer-Verlags LNCS format should be strictly followed. At least one author must be registered in ESELAW to present the paper!

Remember: ESELAW is being held together with CIbSE and WER. Please, double check the full program at http://cibse2012.unlam.edu.ar/home/. Besides the technical sessions, there are additional activities that can
be of your interest.

Thank you for your participation! Sincerely,

Ellen Francine Barbosa PC Chair of ESELAW 2012

- Uma cópia do artigo é apresentado a seguir (a partir da próxima página).

A Systematic Mapping Study on Formal Methods Applied to Crosscutting Concerns Mining

Rafael S. Durelli and Vinícius H. S. Durelli

Computer Systems Department
University of São Paulo
São Carlos – Brazil
{rdurelli, durelli}@icmc.usp.br

Abstract. Background: Crosscutting concerns consist in software system features having the implementation spread across modules as tangled and scattered code. Developers need of up-to-date knowledge about crosscutting concerns currently implemented in their systems and about the location of these concerns throughout the underlying code. Hence, within the academic community there are a bunch of techniques to mine such scattered concerns. To the best of our knowledge, there is no survey reporting on what methods have been employed in this area.
Objectives: To conduct a systematic mapping study to ascertain what formal methods have been used to assist the mining of crosscutting concerns.
Research Method: We have carried out a systematic mapping study of the literature based upon searching of major electronic databases.
Results: As a result, 10 primary studies have been selected and classified by their categories and data of publication. From analyzing the results of our mapping study, we found out that formal methods have scarcely been used to assist the identification of crosscutting concerns.
Conclusions: According to our results, there is still much research to be done on applying formal methods to support crosscutting mining. Therefore, this mapping study may be seen as an initial step towards identifying research gaps in the area and thus perspectives for future research.

Keywords: Systematic Mapping, Formal Method, Crosscutting Concerns

1 Introduction

A concern is commonly defined as anything that stakeholders regard as a conceptual unit [1]. Concerns range from development oriented tracing, and more general purpose caching, to domain-specific business rules. Developers and architects continuously need of up-to-date knowledge about concerns currently implemented in their systems as well as their location throughout the code. For

example, during maintenance and reengineering, developers need to locate specific concerns in the source code. Bug fixes must be propagated to the whole implementation of a concern [2], and possibly to other concerns with which the concern interacts. Architects need to map the currently implemented concerns to the reference architecture to verify architecture conformance. As such, crosscutting concern mining is indispensable for software maintenance, reverse engineering, reengineering and even for re-documentation. Despite their importance, as far as we know the literature, there are no surveys describing what techniques have been used to identify crosscutting concerns. Therefore, as a first step towards filling this gap, the main objective of this paper is to present a systematic mapping that attempts to identify whether formal methods have been used to aid the identification of crosscutting concerns.

2 The Mapping Process

The process applied to conduct the mapping study herein described is detailed by [3]. According to them, its essential steps are: *(i)* definition of research questions, *(ii)* conducting the search for relevant primary studies, *(iii)* screening of papers, *(iv)* keywording of abstracts, and *(v)* data extraction and mapping. Each step produces an intermediate outcome, the concluding result being the mapping study as shown in Figure 1. Following sections present details on how each step was carried out.

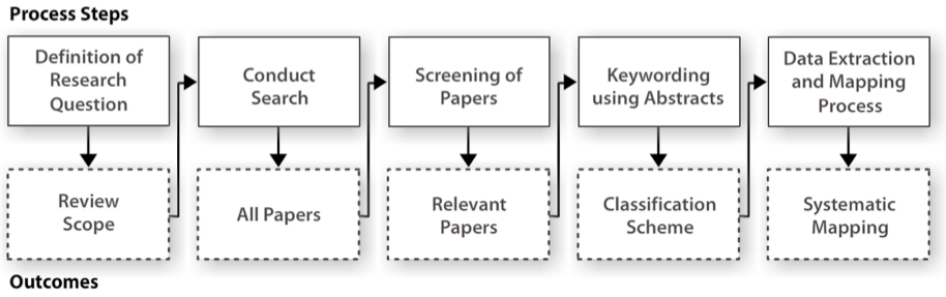


Fig. 1. Systematic mapping process. This Figure was adapted from [3]

2.1 Definition of Research Questions

Research questions must embody the mapping study purpose. Moreover, research questions reflect the general scope of the mapping study. The scope is comprised of population (i.e., population group observed by the intervention), intervention (i.e., what is going to be observed in the context of the planned mapping study), and outcomes of relevance (i.e., the results of the intervention). Accordingly,

during the conduction of this step, it was also necessary to establish the scope of the mapping study:

- **Population**: published scientific literature reporting on formal methods applied to aspects mining or/and crosscutting mining.
- **Intervention**: published scientific literature concerned with introducing formal methods related to aspects mining or/and crosscutting concerns mining. Furthermore, we also aim at determining whether there is a formal method that is applied in this domain and if it is used within academic circles.
- **Outcome of relevance**: an overview of the studies that have been conducted in the domain of formal methods, emphasizing primary studies that report on how these methods can be used to aid the identification of crosscutting concerns. From observing such an aggregated data set we also intend to provide insight into the frequencies of publication over time to inspect trends.

Hence, given that we set out to determine if formal methods are being applied to identify crosscutting concerns, our research question (RQ) reflects this purpose as follows:

- **RQ**$_1$: Have formal methods being used to support the mining of crosscutting concerns within the academic community?

2.2 Search for Primary Studies

Herein the search for primary studies basically involves defining both the search string and electronic databases to be used. The string we used for searching is composed of a combination of the following keywords and acronyms: ***concern mining, aspect mining, mining technique, crosscutting concerns and formal method***. Figure 2 depicts the search string devised from combining such keywords.

*(("aspect mining") **OR** ("concern mining") **OR** ("CC mining") **OR** ("mining technique") **OR** ("crosscutting concerns") **OR** ("crosscutting concerns")) **AND** (("formal method") **OR** ("formal-method") **OR** ("formal specification") **OR** ("formal verification"))*

Fig. 2. Search string.

Afterwards, we used such search string on the following electronic databases: ACM Digital Library[1], IEEE Xplore[2], ScienceDirect[3] and Springer Lecture Notes

[1] `http://portal.acm.org`
[2] `http://ieeexplore.ieee.org`
[3] `http://www.sciencedirect.com`

in Computer Science (LNCS)[4]. These databases have search engines that enable identifying for occurrences of the terms defined both in the title and abstract. Furthermore, no limits were placed on date of publication in order not to restrict the mapping study scope. Aimed at keeping track of the selected papers, we used JabRef[5], an open source system for bibliography reference management.

2.3 Screening of Papers for Inclusion and Exclusion Criteria

In order to determine which primary studies are relevant to answer our research question, we applied a set of inclusion and exclusion criteria to each retrieved study.

Inclusion criteria (IC) devised and applied are:

1. if several paper reported identical studies, only the most recent was selected;
2. papers describing more than one study had each study individually evaluated;
3. the primary study has to describe at least one technique that uses formal method or model checking or formal verification in the domain of crosscutting mining or aspect mining.

The set of exclusion criteria (EX) are:

1. papers that do not present studies related to formal methods in the domain of crosscutting or aspect mining techniques;
2. papers that do not describe studies related to either formal methods or crosscutting/aspect mining techniques;
3. the primary study is not available in an electronic format;
4. technical reports, documents that are available in the form of either abstracts or presentations (i.e., elements of "grey" literature), and secondary literature reviews (i.e., systematic literature reviews and mapping studies).

It is worth highlighting that we avoided imposing many restrictions on primary study selection since we wanted a broad overview of the research area as a whole. Taking into consideration only certain types of studies would lead to a biased overview and result in an inaccurate mapping study.

Although mapping studies are often carried out based solely on the abstracts, throughout primary study selection these criteria were applied to the following sections of each candidate study: *(i)* title, *(ii)* abstract, *(iii)* introduction, and *(iv)* conclusion. Overall, an initial figure of 11 candidate papers was obtained after applying the inclusion and exclusion criteria based only upon title and abstract. After going over introductions and conclusions, we ended up with a final set of 10 primary studies as shown in Table 1. Nevertheless, it is important to mention that other parts, besides introductions and conclusions, quite often had to be read in order to ascertain whether formal methods were used in the

[4] `http://www.springer.com/lncs`
[5] `http://jabref.sourceforge.net/`

domain of crosscutting/aspect mining. The titles, references, and the inclusion criteria (IC) that was applied for these primary studies are listed in Table 2.

Despite the reduced number of primary studies, we believe that our search retrieved all primary studies that deal with formal methods, formal verification and model-checking within the domain of crosscutting/aspect mining.

Table 1. Papers retrieved from each electronic database, total of candidate studies and the final set.

Electronic Database	Number
ACM Digital Library	93
IEEE Xplore	253
ScienceDirect	121
Springer LNCS	47
Total	514
Candidates	11
Final set	10

2.4 Keywording

As previously mentioned, as far as we know the literature, there is no similar study providing a taxonomy and an overview of research into formal methods and crosscutting/aspect mining. Hence, we applied a *keywording* strategy aimed at devising our own classification scheme and categories for the selected primary studies. Keywording reduces the time spent developing classification schemes and categories. By applying such strategy, initially, abstracts are read for the purpose of finding keywords and concepts that reflect their contribution. Subsequently, these keywords and concepts are combined together to produce a general understanding regarding the nature and contribution of the research. Eventually, the final set of keywords is used to define representative categories. The classification scheme gradually evolves toward its final version as new categories are added, merged, or split up. At the end of the depicted strategy, 3 categories have been obtained: *"**Model-Checking**"*, *"**Formal Verification**"* and *"**Formal Method**"*.

2.5 Data Extraction and Mapping

Each included primary study was assigned to one or more categories that we have devised in the foregoing step. Given these categories, in this section we outline each of the resulting categories as well as typical primary studies of each of them.

– **Model-Checking**: In this category are included primary studies that report on how to use model-checker in the domain of crosscutting/aspect mining.

As evidenced by the included studies there is a lack of formal methods in the domain of aspect oriented (AO) due to the fact that it is not straightforward to implement such methods in this paradigm. Only 3 studies explore how to use model-checker in such domain. In [4] the authors propose a model-checker to verify the correctness of AO programs. Similarly, in [5, **?**] the authors describe theoretical underpinnings for applying model checking to programs written using AO programming languages. Nevertheless, there were no studies reporting on utilizing model-checkers to support crosscutting concerns mining.

- **Formal Verification**: This category comprises studies focusing on introducing functionality for formal verification into crosscutting/aspect mining. For example, in [6] an approach to formally verifying properties of systems composed of multiple crosscutting concerns is presented. This approach models concerns as set of concurrent process and provides a method of composition that mimics the composition operators of existing multiple concern implementation language. In a similar way, in [7] propose an approach that uses the method B to support formal verification in the AO paradigm. Besides formal verification, this category also groups together primary studies concerned with formal specification of crosscutting concerns [8].
- **Formal Method**: This category contains primary studies focusing on providing support for new formal method in the context of crosscutting/aspect, as the study described in [9]. They have implemented a new formal method named AOZCL, an aspect-oriented extension to a formal framework (ZCL) with a built-in software architecture description language.

3 Analysis and Classification

The focus of this section is to present a broad overview of research within formal method considering the context of crosscutting and aspect mining we have acquired after classifying and categorizing primary studies. For this, we analyzed each paper, through the reading of title, keywords and abstract.

During the analysis step, we have identified three categories: *Model-Cheking*, *Formal Verification*, and *Formal Method*. We also used information drawn from this overview to answer our mapping study's research question. The fan plot in Figure 3 (b) depicts the number of primary studies according to their categories. It is worth highlighting that certain primary studies were grouped in more than one category, which affected the frequency count: the sum of the frequencies, i.e., 12, is greater than the total of selected studies presented in Table 1 (10).

As can be seen, the majority of the selected primary studies are published by ACM as shown in Figure 3 (a). The other electronic databases, IEEE, ScienceDirect, and Springer had 2, 2, and 1 selected studies, respectively.

It is fairly evident from observing Figure 3 (c) that the categories formal verification and model checking have not drawn much attention in the last decade. In the same way, formal methods have not been an active research as can be observed from 2001 up to 2006. Nevertheless, our results show that from 2006 up

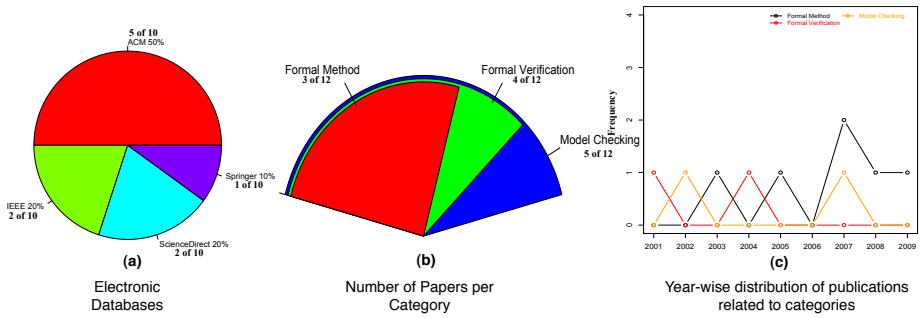


Fig. 3. Distribution of primary studies by electronic database.

to 2009 there has been an increase in the number of publications addressing this subject. In spite of reduced number of primary studies identified herein we argue that the answer to **RQ1** is that *formal methods*, *model checking* and *formal verification* have yet to enter the mainstream. As of now, they have been scarcely used to help identify crosscutting concerns within the academic community.

4 Threats to Validity

The threats to validity in this systematic mapping are the following:

Primary studies selection: Aimed at ensuring an unbiased selection process we defined research questions in advance and devised inclusion and exclusion criteria we believe are detailed enough to provide an assessment of how the final set of primary studies was obtained. However, we cannot rule out threats from a quality assessment perspective, for we simply selected studies without assigning any scores. In addition, we wanted to be as inclusive as possible, thus no limits were placed on date of publication and we avoided imposing many restrictions on primary study selection since we wanted a broad overview of the research area.

Missing important primary studies: The search for primary studies was conducted in several search engines, even though it is rather possible we have missed some primary studies. Nevertheless, this threat was mitigated by selecting search engines which have been regarded as the most relevant scientific sources [10] and therefore prone to contain the majority of the important studies.

Keywording reliability: The authors are software engineering researchers, and one of them is familiar with crosscutting mining, aspect mining, and software product line. However, none of them has theoretical or practical knowledge about formal methods, model-checking, and formal verification, thereby it is possible that we end up introducing some bias during the keywording step. Nevertheless, we believe that our search retrieved all primary studies that deal with formal methods, formal verification, and model-checking within the domain of crosscutting and aspect mining.

Table 2. Selected primary studies

#	Title and Reference	IC
1	**Locating crosscutting concerns in the formal specification of distributed reactive systems**. [11]	IC3
2	**Verifying aspect advice modularly**. [5]	IC3
3	**Foundations of incremental aspect model-checking.** [12]	IC3
4	**Revisiting a Formal Framework for Modeling Aspects in the Design Phase.** [9]	IC3
5	**Aspect-oriented programming with model checking.** [4]	IC3
6	**Using B to Verify the Weaving of Aspects.** [7]	IC3
7	**An Approach for Modeling and Analyzing Crosscutting Concerns.** [8]	IC3
8	**Supporting Formal Verification of Crosscutting Concerns.** [6]	IC3
9	**Coordinating Aspects and Objects.** [13]	IC3
10	**Using Aspects for Enforcing Formal Architectural Invariants.** [14]	IC3

5 Concluding Remarks

The main contribution of this paper is an overview of the formal methods that have been used to help the mining of crosscutting concerns. Toward this end, we have conducted a systematic mapping. As main result, we contend that formal methods have been barely used to mine crosscutting concerns. Another important contribution is the identification of new research lines. For instance, trying to add formal methods to techniques already stablished for mining crosscutting concerns such as, History-based Aspect Mining [15], Clustering-Based Fan-in Analysis [16] and Concern Mining using Mutual Information over Time [17]. Therefore, there are still different perspectives that could be investigated, aiming at improving the identification of crosscutting concerns.

References

1. Robillard, M., Murphy, G.: Concern graphs: finding and describing concerns using structural program dependencies. In: Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on. (may 2002) 406 –416
2. Eaddy, M., Zimmermann, T., Sherwood, K.D., Garg, V., Murphy, G.C., Nagappan, N., Aho, A.V.: Do crosscutting concerns cause defects? IEEE Trans. Softw. Eng. **34** (July 2008) 497–515
3. Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M.: Systematic Mapping Studies in Software engineering. In: EASE '08: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering. (2008)
4. Ubayashi, N., Tamai, T.: Aspect-oriented programming with model checking. In: Proceedings of the 1st international conference on Aspect-oriented software development. AOSD '02, ACM (2002) 148–154
5. Krishnamurthi, S., Fisler, K., Greenberg, M.: Verifying aspect advice modularly. SIGSOFT Softw. Eng. Notes **29** (October 2004) 137–146

6. Nelson, T., Cowan, D., Alencar, P.: Supporting formal verification of crosscutting concerns. In Yonezawa, A., Matsuoka, S., eds.: Metalevel Architectures and Separation of Crosscutting Concerns. Volume 2192 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2001) 153–169
7. Thuan, T.N., Ha, N.V.: Using b to verify the weaving of aspects. In: Software Engineering Conference, 2007. APSEC 2007. 14th Asia-Pacific. (dec. 2007) 199 –205
8. Fu, Y., Ding, J., Bording, P.: An approach for modeling and analyzing crosscutting concerns. In: Service Operations, Logistics and Informatics, 2009. SOLI '09. IEEE/INFORMS International Conference on. (july 2009) 91 –97
9. Paula, V.d., Batista, T.: Revisiting a formal framework for modeling aspects in the design phase. In: Proceedings of the Early Aspects at ICSE: Workshops in Aspect-Oriented Requirements Engineering and Architecture Design. EARLYASPECTS '07, Washington, DC, USA, IEEE Computer Society (2007) 6–
10. Dyba, T., Dingsoyr, T., Hanssen, G.K.: Applying systematic reviews to diverse study types: An experience report. In: First International Symposium on Empirical Software Engineering and Measurement. ESEM '07, Washington, DC, USA, IEEE Computer Society (2007) 225–234
11. Pazos-Arias, J.J., García-Duque, J., López-Nores, M.: Locating crosscutting concerns in the formal specification of distributed reactive systems. SIGSOFT Softw. Eng. Notes **30** (May 2005) 1–5
12. Krishnamurthi, S., Fisler, K.: Foundations of incremental aspect model-checking. ACM Trans. Softw. Eng. Methodol. **16** (April 2007)
13. Aaltonen, T., Helin, J., Katara, M., Kellomki, P., Mikkonen, T.: Coordinating aspects and objects. Electronic Notes in Theoretical Computer Science (2003) 248 – 267
14. Kallel, S., Charfi, A., Jmaiel, M.: Using aspects for enforcing formal architectural invariants. Electronic Notes in Theoretical Computer Science (2008) 5–21
15. Breu, S., Zimmermann, T.: Mining aspects from version history. In: Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering, Washington, DC, USA (2006) 221–230
16. Zhang, D., Guo, Y., Chen, X.: Automated aspect recommendation through clustering-based fan-in analysis. In: Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering, Washington, DC, USA (2008) 278–287
17. Adams, B., Jiang, Z.M., Hassan, A.E.: Identifying crosscutting concerns using historical code changes. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, New York, NY, USA (2010) 305–314

Anexo 2: Comprovante da publicação no *Simpósio Brasileiro de Sistemas de Informação* (SBSI'12)

Cópia do email de aceitação (Somente a primeira página)

Dear Ms. Rafael Durelli:

Congratulations! Your paper "F2MoC: A Preliminary Product Line DSL for Mobile Robots" for SBSI 2012 - Trilhas Técnicas (Technical Tracks) has been accepted for the "1.Trilha Regular - Artigos Completos (Regular Track - Full papers)" Track. The reviews are below or can be found at https://submissoes.sbc.org.br/PaperShow.cgi?m=96565.

There are a number of things we´d like to remind you:
1) the camera ready version should follow the SBC papers publishing model and be submitted in pdf format up to 18th march 2012 through the JEMS system (https://submissoes.sbc.org.br/home.cgi?c=1496);
2) full papers must have seven to 12 pages and short papers should be limited to six pages ("Regular Short Papers" and "Information Systems and Smart Cities" Tracks). Papers longer than these limits will not be included in the proceedings;
3) the copyright form must include all paper authors, but be signed by at least one author. The copyright form can be electronically signed and must be sent in pdf format with the camera ready version of the paper. The copyright form is available:
http://www.each.usp.br/sbsi2012/files/contratodecessodedireitosautorais_sbc_sbsi.doc
4) please incorporate all reviewers comments and feedback received from the conference, check for language and follow the SBC references format. Papers that do not follow these recommendations will not be included in the proceedings;
5) for each accepted paper at least one of the authors must be registered in the conference. The undergraduated students fee cannot be applied to accepted paper authors; in this case they must be registered as graduated students. In order to include the paper in the proceedings the authors registration must be done up to 31st march 2012.

Please, let us know if you need help.

Best Regards,
Célia Ghedini Ralha and Fátima L. S. Nunes
SBSI´2012 Technical Program Chairs

- Uma cópia do artigo é apresentado a seguir (a partir da próxima página).

F2MoC: A Preliminary Product Line DSL for Mobile Robots

Rafael S. Durelli[1], **Vinicius H. S. Durelli**[2]

[1]Computing Departament
Federal University of São Carlos (UFSCAR)
São Carlos, SP, Brazil.

[2]Computer Systems Departament
University of São Paulo
São Carlos, SP, Brazil.

rafael_durelli@dc.ufscar.br[1], durelli@icmc.usp.br[2]

Abstract. Background: *Software product line (SPL) and model-driven development (MDD) are two trends that have been drawing increased attention from the software development community. In the literature it is possible to find a set of articles that apply MDD techniques to assist the development of a SPL.*
Objectives: *To show how to create a Domain-Specific Language (DSL) based on a feature model.*
Methods: *A SPL and MDD techniques were used to create a DSL.*
Results: *A DSL named FeatureToModeOrCode (F2MoC) was developed in order to assist the development of the mobile robots.*
Conclusions: *Advantages can be gained from using the F2MoC: (i) an easier instantiation of SPL members; (ii) it also makes it possible for the application engineers to focus on an high level model (i.e., features model), obviating the need of dealing with platform-specific issues; (iii) source code is generated automatically from this high level model, e.i., the underlying application source-code.*

1. Introduction

Software product line (SPL) and model-driven development (MDD) are two trends that have been drawing increased attention from the software development community[Czarnecki et al. 2005].

A software product line (SPL) is a set of software systems that share a common and managed set of characteristics that satisfy the needs of a particular market segment or mission, and that are developed from a common set of core assets following a planned process[P. Clements 2001]. Traditional SPL development processes identify two main technical stages: Domain Engineering, where reusable assets are developed and maintained, and the scope and production plan are defined, and Application Engineering, where particular member requirements are gathered, and the product is built by arranging the reusable assets according to the production plan.

Model-driven development (MDD) aims at capturing every important aspect of a software system through appropriate models. Compared to implementation code, models capture the intentions of the stakeholders more directly, are freer from accidental implementation details, and are more amenable to analysis. In MDD, models are not just

auxiliary documentation artifacts; rather, they are source artifacts and can be used for automated code generation (model-to-code) and model transformation (model-to-model). Modeling languages play a central role in MDD. They range from the more generic modeling languages like UML (Unified Modeling Language) to the so called Domain-Specific Languages (DSLs) that is, formal languages whose constructs represent concepts from a specific problem domain.

The relationship between SPL and MDD is not new in the literature. In this way, it is possible to find articles which use MDD in the context of SPL in order to complement and assist the different aspects of a given SPL [Trujillo et al. 2007] [Avila-García et al. 2007] [Freeman et al. 2008] [Iris et al. 2007] [Polzer et al. 2009]. For instance, MDD can be used in the context of SPL to assist the application engineer in the activity of instantiation SPL's members.

This paper presents in detail the creation of a DSL, called FeatureToModelOrCode (F2MoC) to support instantiation of members of a given SPL. Our DSL comprises a metamodel[1] that represents all features of SPL feature models[2].

We have argued that several advantages can be gained from using the F2MoC: *(i)* an easier instantiation of SPL members, given that the application engineer has at his disposal all possible features belonging to the SPL; *(ii)* it also makes it possible for the application engineers to focus on an high level model (i.e., features model), obviating the need of dealing with platform-specific issues; *(iii)* source code is generated from this high level model, e.i., the underlying application source-code.

The remainder of this paper is organized as follows: Section 2 presents the foundation related to MDD and DSL, Section 3 describes the guidelines that we have developed to assist the development of a DSL based on a feature model and shows how to create a DSL named F2MoC, Section 4 gives an overview of the F2MoC, and finally in Section 5 we conclude the paper with some remarks and future directions.

2. Background

In what follows we present the main concepts about MDD and DSL.

2.1. Model-Driven Development

In Model-Driven Development (MDD) software engineer does not need to manually interact with the source code, focusing into models of higher level of abstraction. Mechanism that perform transformations are employed to generate other artifacts (source code, and/or more specific models) from the input models. These models guide the development tasks, maintenance and are also considered fundamental parts of the software as well as source code, serving as input to tools that perform automatic transformations of code, thereby reducing the efforts of developers [Stahl et al. 2006].

[1]Metamodel is a model containing elements that describe other models. Metamodels play a supportive role in MDD: they *(i)* define the abstract syntax of model languages, *(ii)* facilitate the understanding of concepts involved in the transformations, and *(iii)* are employed by the mechanism responsible for performing the transformations that map an entry-model to another model or code.

[2]A feature model is a tree based structure in which each node represents a variability point or unit of functionality in the product. The root of the tree represents the most generalized concept in the product, and successively deeper levels indicate software refinement. The parent-child relationships indicate configuration constraints that you must satisfy when choosing values for variability points.

Domain specific Languages (DSLs) play a cornerstone role in MDD for representing models and metamodels. DSLs are small languages that present limited expressiveness focused on a particular domain[van Deursen et al. 2000]. Usually, DSLs are not turing complete and have no imperative control structures, e.g., conditions and loops. Rather, most of them are declarative, consequently, they can be regarded as specification languages. These small, declarative, special-purpose languages have a simplified suite of notations that is tailored toward their domain abstractions, features, semantics, and jargon. Hence, by using DSLs, developers perceive themselves as dealing directly with domain concepts.

DSLs are differentiated into their appearance (textual vs. graphical) and their origin (internal vs. external)[Fowler 2009]. External DSLs have their own custom-built syntax. As a consequence, developing an external DSL implies in writing a full-fledged parser in order to process it. Internal DSLs use existing general-purpose languages structures and, in most cases, the underlying execution environment, as a hosting base. An advantage of this approach is that the compiler or interpreter of the base language is reused. The main limitation is related to the limited expressiveness that can be achieved by using the base language syntactic mechanisms[van Deursen et al. 2000].

3. Methodology for the Development of F2MoC

This section briefly describes the guidelines to assist the development of a DSL called FeatureToModelorCode (F2MoC) in which aims to increase the level of reuse and accelerate the instantiation of members belonging to a given SPL. In Figure 1 depicts these guidelines. It is worth highlighting that these guidelines are based in the following frame-

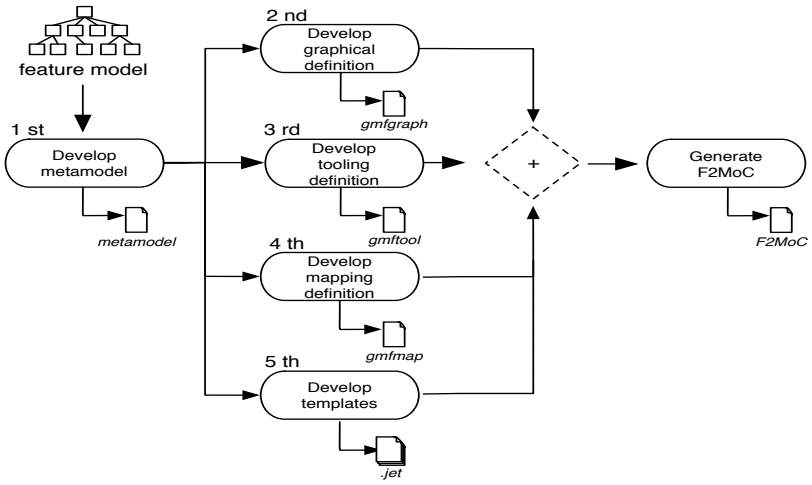


Figure 1. Guidelines to create the F2MoC

works: *(i)* Eclipse Modeling Framework[3] (EMF), *(ii)* Graphical Modeling Framework[4] (GMF) and *(iv)* Java Emitter Template[5] (JET).

As can be seen in Figure 1 these guidelines assumes that a SPL has already developed, so a SPL and consequently a feature model must be available at this point. Thus,

[3]http://www.eclipse.org/modeling/emf/
[4]http://www.eclipse.org/modeling/gmp/
[5]http://www.eclipse.org/modeling/m2t/?project=jet#jet

after develop or get a feature model we need develop five different input documents, they are:

- The metamodel (*genmodel*) wich it is created based on a feature model together with some guidelines. These guidelines are the following:
  - Insert in the F2MoC's metamodel, an abstract metaclass named Diagram;
  - Insert in the F2MoC's metamodel, an abstract metaclass named DiagramElements which must have an aggregation relationship with the metaclass diagram;
  - Insert in the F2MoC's metamodel, an abstract metaclass named Features and another called RelationShip. These metaclasses must extends the metaclass DiagramElements
  - Insert in the F2MoC's metamodel, for each features that belong the feature model a metaclasse with the same name of the features. These metaclasses must extends the metaclasse Features;
  - Insert in the F2MoC's metamodel, an enumerate that represents the kind of features: mandatory, optional or alternative.
- The graphical definition model wich specifies the geometric elements that make up the graphical representation of the metamodel, such as rectangles or text labels (*gmfgraph*). This file is created using the following guidelines:
  - The submetaclasses of Features must be represented as rectangular blocks;
  - The RelationShip must be represented by line.
- The tooling definition model comprises things related to editor palettes, menus, etc. (*gmftool*). This file is created based on the following guidelines:
  - All subclass of Feature must be created a menu;
  - For each RelationShip must be create a menu.
- The mapping model links the first three models together (*gmfmap*);
- The templates are used to realized transformation. The templates must be created base on the following guidelines:
  - For each "alternative" feature the engineer must develops templates using the design pattern named Abstract Factory;
  - For each "optional" feature the engineer must develops templates using the design pattern called Builder;
  - For each "or" feature the engineer must develops templates using the design pattern called Chain of Responsibility.

In the following subsection is described how we have developed the F2MoC applied these guidelines.

3.1. Developing the F2MoC

The SPL and the feature model that we have devised and used in this paper is describe by [Durelli et al. 2010]. The domain of this SPL is mobile robots application[6]. In Figure 2 presents the feature model of which is used how the basis for the development of F2MoC. This feature model has 37 features and it can derivates approximately 17 members. Having the feature model the domain engineer must create the metamodel using the guidelines previously defined in Section 3.

[6]Mobile robots are usually equipped with contact, distance and visual sensors that enable them to perceive the environments and avoid collisions, allowing for safe autonomous navigation.

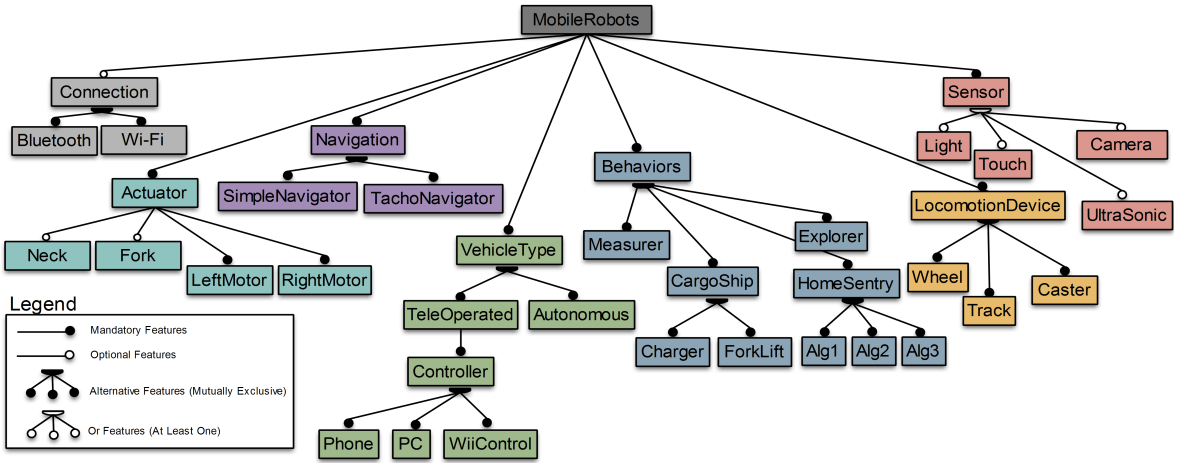


Figure 2. Feature Model of Mobile Robots

In Figure 3 we present the F2MoC's metamodel which was obtained based on the guidelines previously defined. As we can see, colors were used to facilitate the visualization and the understanding of how the mapping between the feature model was performed for the metamodel. For instance, the features that represents sensors (Figure 2), i.e., the "red" ones, were represented in the metamodel by metaclasses using the same color.

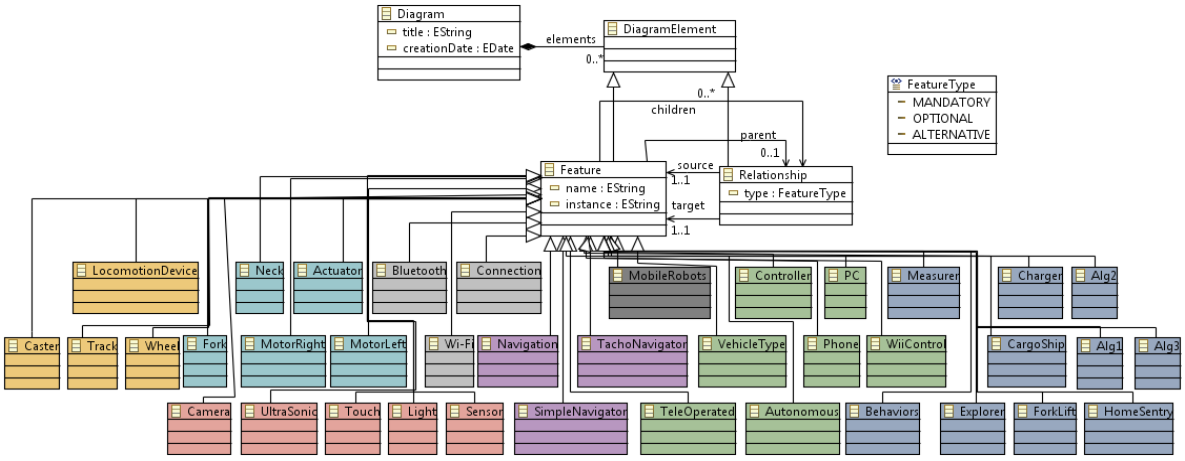


Figure 3. MetaModel's F2MoC (*genmodel*)

After developing the F2MoC metamodel, the concrete syntax of the DSL must be developed as show in Figure 1 (*graphical definition*). In Figure 4.(a) and (b) shows the file *gmfgraph*, which defines the visual aspects of the F2MoC. This file was developed using the guidelines previously defined in Section 3. For instance, all subclass of Features were represented using a "rounded rectangle" (Figure 4.(a)) and the all RelationShip were represented by a "polyline" (Figure 4.(b)).

According to Figure 1 the next file that must be developed is the *gmftool*. The Figure 4.(c) and (d) depicts how the *gmftool* was developed. As can be seen for each Features and RelationShip were created a menu, just like previously defined in Section 3. The next file developed was the *gmfmap* which is depicts in Figure 4.(e) and (f).

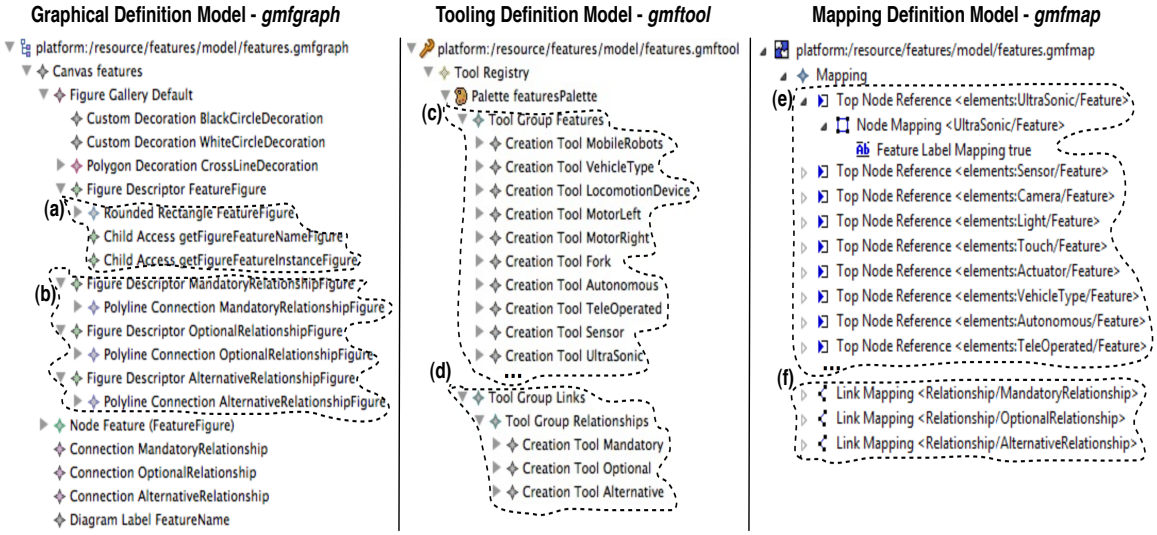


Figure 4. Arquivos necessario para o desenvolvimento da DSL

As can be seen in Figure 1 the fifth guideline is create the templates. Thus, we have developed such templates, in Figure 5 we show an example of the sort of code template that we have developed following the guidelines previously defined.

```
package com.br.ufscar.lejos.sensors.builder;

<c:iterate select="sensors/features" var="f">
   import lejos.nxt.<c:get select="$f/@name">;
</c:iterate>
import lejos.nxt.SensorPort;

public class Builder<c:get select="$f/@name"> extends
BuilderSensor{

   public void buildSensor(String sensorPort) {
        SensorPort port = null;
                if(sensorPort.equals("S1")){
                        port = SensorPort.S1;
                }else if (sensorPort.equals("S2")){
                        port = SensorPort.S2;
                }else if (sensorPort.equals("S3")){
                        port = SensorPort.S3;
                }
                else{
                        port = SensorPort.S4;
                }
            this.sensor = new <c:get select="$f/@name">(port);
        }
}
```

Figure 5. Chunk of template code

In this chunk of code (Figure 5), all occurrences of ***<c:get select="$f/@name">*** will be replaced by the name of all features related to sensors, e.g., *UltraSonicSensor*, *TouchSensor*, *Camera* and *LightSensor*.

4. F2MoC

This section presents an overview of the F2MoC that we have developed. Figure 6 depicts an overview of instantiation of a member of SPL presented in [Durelli et al. 2010].

As we can see the F2MoC comes with a unified IDE for DSL and Java, imple-

mented over Eclipse IDE [7]. In Figure 6.(a) depicts the view of the application engineer when he is assembling the feature models. These feature models are assembled using the action drop-and-drag of the components that are available at the palette, this palette can be seen at Figure 6.(c). Figure 6.(b) depicts all projects related to the DSL F2MoC.

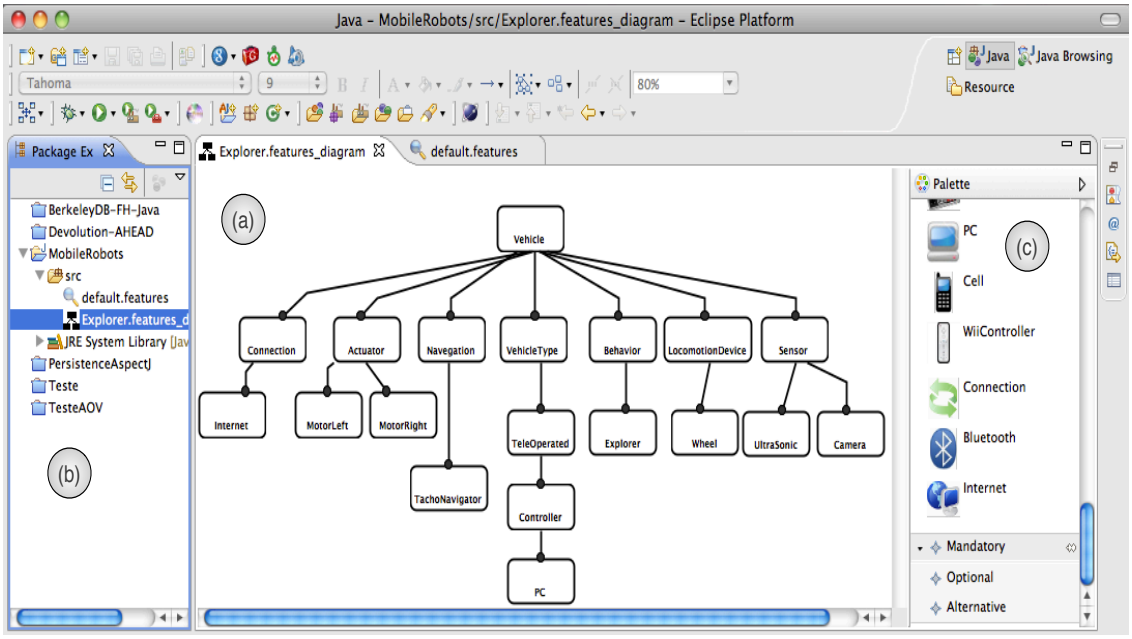


Figure 6. Overview of the F2MoC

Afterward assembled the feature model it is possible perform transformations model-to-code (M2C). In Figure 7 we show an example of chunck of code generated. This chunk of code has been generated based on sort of code template shown in Figure 5. As can be seen, all occurrences of ***<c:get select='$f/@name'>*** are replaced by the name "*UltraSonicSensor*".

```
package com.br.ufscar.lejos.sensors.builder;

import lejos.nxt.UltraSonicSensor
import lejos.nxt.SensorPort;

public class BuilderUltraSonicSensor extends BuilderSensor{

   public void buildSensor(String sensorPort) {
        SensorPort port = null;
                if(sensorPort.equals("S1")){
                        port = SensorPort.S1;
                }else if (sensorPort.equals("S2")){
                        port = SensorPort.S2;
                }else if (sensorPort.equals("S3")){
                        port = SensorPort.S3;
                }
                else{
                        port = SensorPort.S4;
                }
            this.sensor = new UltraSonicSensor(port);
        }
}
```

Figure 7. Chunk of generated code

[7] www.eclipse.org

After performing the transformation the code can be embedded in the mobile robot.

5. Concluding Remarks

Systematic techniques of reusing already established in software engineering such as SPL and MDD, have as main objective provide highest level of reuse and ease development. Several studies in the literature indicate that MDD can be used together with SPL to assist and speed up the development of SPL's members. Thus, this paper has presented step-by-step how to build a DSL called F2MoC which provides support in the instantiation of members of an SPL by M2C transformations.

F2MoC is an straightforward development process for SPL members, through which the application engineer can assemble a SPL's member just by dragging-and-dropping features. An intrinsic advantage of F2MoC is that the feature model devised by developers is used to automatically generated source code (i.e., M2C).

Long term future work involves conducting experiments to evaluate the level of reuse provided by F2MoC and creating a repository with versioning strategies.

Acknowledgements

The authors would like to thank the financial support provided by FAPESP (grant number 2009/00632-1) and CAPES (grant number 0340-11-1).

References

Avila-García, O., García, A. E., and Rebull, E. V. S. (2007). Using software product lines to manage model families in model-driven engineering. *Proceedings of the 2007 ACM symposium on Applied computing*, 2(1):1006–1011.

Czarnecki, K., Antkiewicz, M., Kim, C. H. P., Lau, S., and Pietroszek, K. (2005). Model-driven software product lines. *ACM*, 19(5):126–127.

Durelli, R., Conrado, D., Ramos, R., Pastor, O., Camargo, V., and Penteado, R. (2010). Indentifying features for ground vehicles software product lines by means of annotated models. *ACM/IEEE International Conference on Model Driven Engineering, Languages and Systems*, 13:160–165.

Fowler, M. (2009). A pedagogical framework for domain-specific languages. *IEEE Softw.*, 26:13–14.

Freeman, G., Batory, D., and Lavender, G. (2008). Lifting transformational models of product lines: A case study. *Springer-Verlag*, 15(2):16–30.

Iris, G., Holger, P., and Markus, V. (2007). Integrating model-driven development and software product line engineering. *Proceedings of the 2007 ACM*, 3(2):120–124.

P. Clements, L. N. (2001). *Software product lines: practices and patterns*. Addison-Wesley Longman Publishing.

Polzer, A., Kowalewski, S., and Botterweck, G. (2009). Applying software product line techniques in model-based embedded systems engineering. *Communications of the ACM*, 22:2–10.

Stahl, T., Voelter, M., and Czarnecki, K. (2006). *Model-Driven Software Development: Technology, Engineering, Management*.

Trujillo, S., Azanza, M., and Diaz, O. (2007). Generative metaprogramming. *ACM*, 13(3):105–114.

van Deursen, A., Klint, P., and Visser, J. (2000). Domain-specific languages: an annotated bibliography. *SIGPLAN Not.*, 35:26–36.

Anexo 3: Comprovante da publicação no *Simpósio Brasileiro de Engenharia de Software* (SBES'12)

Cópia do email de aceitação (Somente a primeira página)

It is a pleasure to inform you that your paper "Model-Based Reuse for Crosscutting Frameworks: Assessing Reuse and Maintainability Effort" (id: 99070) submitted to CBSoft 2012 - SBES has been accepted.

Your paper will be published at IEEEs digital library, and authors of the 5 best papers (to be announced at SBES 2010) will be invited to submit extended versions of their papers to an special issue of a journal.

You can find the reviews at
.https://submissoes.sbc.org.br/PaperShow.cgi?m=99070

While preparing the camera-ready version of the paper please carefully observe the following guidelines:
- the camera-ready version and the copyright form must be submitted until July 9 (URL to be informed soon)
- it must be formatted according to the IEEE format available at
http://www.ieee.org/portal/cms_docs/pubs/confpubcenter/pdfs/samplems.pdf
- the final version must have at most 10 pages (please, do not compress any items, specially pictures).
- if written in Portuguese, papers must contain, First, a Portuguese resumo, and Second, an English abstract.
- the final version must be in .pdf format, strictly following the IEEE guidelines for PDF generations
(please carefully read http://www.ieee.org/portal/cms_docs/pubs/confstandards/pdfs/IEEE-PDF-SpecV401.pdf)
- please make sure to remove typos, orthographic and grammatical errors.
- make sure that figures are readable in black and white. Color figures should be avoided as the proceedings will be printed in black and white.
- PLEASE OBSERVE AND CAREFULLY ADDRESS THE REMARKS MADE BY THE REFEREES while preparing the camera-ready copy.

COPYRIGHT FORM:
- the copyright form can be found at: http://www.ieee.org/portal/cms_docs_iportals/iportals/publications/rights/IEEECopyrightForm.pdf
- It must be filled, signed, and scanned

In addition to the requirements established above, at least one of the authors must register and attend CBSoft 2012 - SBES in order to present the paper.

If you have any question please let us know. We look forward seeing you in Natal.

Marcio Delamaro
PC Chair of CBSoft 2012 - SBES

- Uma cópia do artigo é apresentado a seguir (a partir da próxima página).

Model-Based Reuse for Crosscutting Frameworks: Assessing Reuse and Maintainability Effort

Thiago Gottardi*, Rafael Serapilha Durelli†, Oscar Pastor López‡ and Valter Vieira de Camargo*
*Departamento de Computação, Universidade Federal de São Carlos,
Caixa Postal 676 – 13.565-905, São Carlos – SP – Brazil
Email: {thiago_gottardi,valter}@dc.ufscar.br
†Instituto de Ciências Matemáticas e Computação, Universidade de São Paulo,
Av. Trabalhador São Carlense, 400, São Carlos – SP – Brazil
Email: rsdurelli@icmc.usp.br
‡Universidad Politecnica de Valencia, Camino de Vera s/n, Valencia, Spain
Email: opastor@dsic.upv.es

***Abstract*—Over the last years a number of Crosscutting Frameworks (CFs) have been developed employing white-box strategies. This strategy requires significant technical knowledge to reuse these frameworks, such as, knowledge in specific programming languages, architectural details and also about the framework nomenclature. Besides, the reuse process can only be initiated when the development reaches the implementation phase, avoiding starting the reuse process in early development phases. In this paper we present a model-based technique for reusing CFs that improves the productivity by allowing the application engineer to concentrate on what is really important during the reuse process. We also present the foundations of our approach and also the result of two experiments that uses two versions of a Persistence CF; the original and the model-based. The results were promising regarding the effort employed to conduct the reuse process, but almost no difference was noticed concerning the effort in conducting maintenance activities.**

I. Introduction

Model-Driven Development (MDD) consists of the combination of generative programming, domain-specific languages and model transformations. MDD aims to reduce the semantic gap between the program domain and the implementation, using high-level models that shield software developers from complexities of the underlying implementation platform [1].

On the other hand, Aspect-Oriented Programming (AOP) is a programming paradigm that overcomes the limitations of Object-Orientation by providing abstractions able to modularize crosscutting concerns (CC) such as persistence, security and distribution. Among these abstractions, Pointcuts are expressions used to capture join-points of an application, e.g., method calls and executions and variable accesses. By capturing these join-points, it is possible to write code to be executed upon a Pointcut occurrence in a modular fashion. AspectJ is one of the AOP languages that implement these abstractions [2]. It is also the language employed in our work.

Since the advent of AOP, several researchers have investigated how its abstractions and concepts impact reuse methodologies, like product lines [3] and frameworks [4]. Many of these researchers investigated how to design a CC in a generic way to enhance their reusability [5]–[15]. Several terms are used to represent this kind of design, e.g., reusable aspects [5], aspect-oriented frameworks and aspect libraries [12]. Because the absence of a taxonomy for this kind of design, we have defined and employed the term "Crosscutting Framework" (CF) to represent a specific kind of abstract aspect-oriented framework implementation of a single CC [9].

Most of the CFs found in literature apply white-box reuse strategies in their instantiation process, relying on writing source code to reuse the framework [5]–[15]. White-box strategy makes application engineers to worry about low level implementation details during the reuse process, leading to the following problems: (*i*) to get know coding details regarding the programming paradigm employed in the framework, making the learning curve steeper; (*ii*) coding mistakes are more likely to happen when the reuse code is created manually; (*iii*) several lines of code must be written for the definitions of small number of hooks, impacting development productivity and (*iv*) reuse process can only be started during implementation phase, as there is no source code in earlier phases.

To overcome these problems, in this paper, we present an approach for supporting the reuse of CFs. The approach is based on two models: Reuse Requirements Model (RRM) and Reuse Model (RM). The RRM documents all the features and variabilities of a CF and is a partial replacement for cookbooks. Based on the RRM, the application engineer can then select just the desired features, building a more specific model, referred as RM. The application engineer has the opportunity to conduct the reuse process in our model-driven approach by filling the fields of this model, which is also used for code generation.

We also present the results of two experiments. In both experiments we have used the same Persistence CF [9]. Our approach showed benefits for the instantiation time, however, no differences were identified regarding the maintenance effort. Therefore, the main contributions of this paper are: (*i*) presenting a model-based approach for CFs, (*ii*) presenting the results of two experiments and (*iii*) the problems of the presented approach can be generalized to other model-based framework reuse processes.

In Section II the notion related to CF, their details and a CF's description that is used in this paper are showed; both the proposed approach and an example of instantiation related to a member of persistence CF are presented in Section III; in Section IV, an empirical evaluation is presented; in Section V, there are related works and in Section VI, there are the conclusions.

II. Crosscutting Frameworks

Crosscutting Frameworks (CF) encapsulate the generic behavior of a single crosscutting concern [9], [13]–[15]. There are CFs developed for persistence [7], [9], security [6], cryptography [10], distribution [7] and other concerns [5]. Their main objective is to make the reuse of such concerns easier during the development of an application.

As well as other types of frameworks, CFs also need information regarding the base application in order to be reused correctly and work properly. We named these information "Reuse Requirements" (RR). For instance, the RR for an Access Control CF includes: 1) the application methods that need to have their access controlled; 2) which are the roles played by users; 3) how many times a user is allowed get an incorrect password. This information is commonly documented in manuals known as "Cookbooks".

Unlike application frameworks, which are used to generate a whole new application, a CF needs to be coupled to a base application in order to become functional. The standard process to reuse a CF is composed by two activities: instantiation and composition. The instantiation is when the application engineer is choosing variabilities and implementing hooks, while the coupling is when he/she is providing composition rules to couple the chosen variabilities to a base code. During the composition activity, pointcuts and composition rules are defined, unifying the chosen variabilities and the base code.

Applications developed with CFs are composed by three types of code modules: base, reuse and framework. The "base code" represents code of the base application. In the "framework code" there is the code of the CF, which is untouched during the reuse process. The "reuse module" is the connection between the base application and a framework. Each final application can be composed by several CFs, each one coupled by a reuse module. The code that was created specifically to reuse an CF is referred here as "reuse code" and applications which were developed based on CFs is referred here as CF-based Applications.

In a previous work we have developed a Persistence CF [16], which is used as case study in this paper. This CF has some features, for instance, both "Persistence" and "Connection" are mandatory features. The first one, aims to introduce a set of persistence operations (e.g., store, remove, update, etc) into applications persistence classes. The second feature, is related to the database connection and identifies points in the application code where the connection should be opened and closed. This feature has variabilities, as for example, the Database Management System (e.g., MySQL, SyBase, Native and Interbase). The CF also has a set of optional features such as "Caching", which is used to improve performance by keeping copies of the data in local memory, and "Pooling" which represents a number of active database connections.

III. Model-Based Reuse Approach

In order to assist the instantiation and composition of members of a CF we have put forward two new models, "Reuse Requirements Model" (RRM) and "Reuse Model" (RM). These models have been devised on top of Eclipse Modeling Framework and Graphical Modeling Framework [17] The formal definition of both models is specified by a single metamodel, which is shown in Figure 1. This metamodel is a set of enumerations and metaclasses, which are either concrete or abstract.

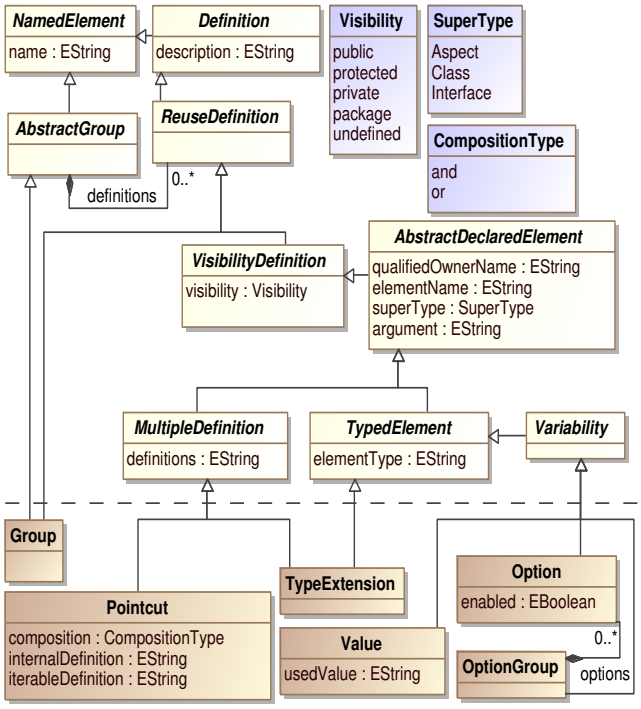


Fig. 1. Metamodel of the proposed models

The metamodel was built based on the vocabulary commonly used in the context of CFs. Among these concepts, there are pointcuts, classifier extensions, method overriding to return values and variabilities selections. These concepts were then mapped into concrete metaclasses, which are visible under the dashed line in Figure 1.

Above the dashed line in Figure 1, there are also the following enumerations: "Visibility", "SuperType" and "CompositionType", which are sets of literals used as properties of the metaclasses. The other elements above the line are abstract metaclasses. They were created after performing an analysis to generalize the properties of the concrete metaclasses. These abstract metaclasses can be applied in similar approaches and

are also important to improve modularity and avoid code replication of the reuse code generator.

In Figure 2 there is an overview of our tool which is used to edit both of our proposed models. On the right of Figure 2, there is a "Palette" with the possible elements that can be inserted into these models. These elements are instances of the concrete metaclasses of the metamodel, visible under the dashed line in Figure 1. Their uses are: (*I*) "Group": an element to group any element visible in the models, including child groups; (*II*) "Pointcut": employed to override abstract pointcuts which represent join-points of the base application code that should be affected by the CF; (*III*) "TypeExtension": elements used to represent types found in the base application that must extend or implement classes, aspects or interfaces found in the CF; (*IV*) "Value": elements used to override methods to return any numeric or textual values that must be informed while reusing the CF; (*V*) "Option": defines a selectable variability of the framework and (*VI*) "OptionGroup": group selectable variabilities of the framework. The last two elements are used to represent the variabilities provided by the CF that may be chosen by the application engineer. The "Group" element is also employed to support feature hierarchy, however, details on feature selection are not shown on this paper. Nevertheless, more details related to both feature selection and the tool can be seen in another paper [18]. This tool provides fully computational environment to the approach herein described.

Both of our proposed models have identical appearance, however, they are employed in different moments. The first proposed model, the RRM, is a graphical documentation regarding the Reuse Requirements, which are related to the information needed to couple the CF to a base application, which is conventionally part of "cookbook". This model contains all of information regarding all CF features and should be provided by a framework engineer. The second model, the RM, is a subset of the RRM that only contains the features selected for reuse. Since both models share the same metamodel, it is possible to employ a direct model transformation to instantiate a RM from a RRM by selecting a valid set of features. Both of our models are represented as forms that contain boxes, as seen in Figure 2. Each box is an instance of a concrete metaclass element and represent a reuse requirement. They also contain three lines, the first line contains a icon of the element type, which is the same type visible in the "Palette", and a name for the reuse requirement. The second line shows a description to facilitate the comprehension of the application engineer and the last line is filled by the application engineer to provide the information regarding the base application. Note that the last line is only used in RMs.

By analyzing the RRM, an application engineer should be able to learn which informations are required by the framework during the reuse process. This model also represents the variabilities provided by a framework that must be chosen by an application engineer. In order to instantiate a framework, the RRM may indicate the need of informing join-points of the base code where crosscutting behavior would be applied to, as well as classes, interfaces or aspect names that would be affected. Framework variabilities that must be chosen during reuse process are also visible. For example, to be able to instantiate a persistence CF, the application engineer must specify methods from base application that should be executed after a database connection is opened and before it is closed. It is also needed to specify methods that represent data base transactions, and the variabilities must be chosen, e.g., the driver which should be used to connect to the database system.

The other model, the RM, is shown in Figure 2. It supports the reuse process of a crosscutting framework. It is intended that the reuse process can be completely executed by completing the third line of the boxes of this model. Therefore, it should be used by the application engineer in order to reuse a framework. For instance, the value "*base.Customer.opening()*" is a method of the base application and was inserted by the application engineer in the third line of box "Connection Opening".

After the application engineer fills in the RM with the information needed by an member of a CF, it is possible to generate the final reuse code. To illustrate the use of these models we have used the "persistence" CF described in Section II.

A. Reuse Example

In this section, we briefly show an example of how to use our models and generate code in order to reuse a CF. To reuse the "persistence" CF, the application engineer must specify explicitly which features will be used in the base application. This is important because usually the CFs have a great deal of features that probably will not be used in the application base. Therefore, we developed an environment to facilitate the feature selection in order to instantiate a member. This environment also provides a way to validate the combinations of features. Further details of this environment are out of the scope of this paper.

In Figure 2, there is our model editor. By using this editor, it is possible to model RRMs and RMs. In this example there is a RM related to the "persistence" CF being completed with information of a base application. The pointcuts "Connection Opening", "Connection Closing" and "Transaction Methods" are intended to capture specific join-points of the base application, e.g. names of methods of the base application that will be affected by the framework. The first two represent, respectively, method executions that should occur after a database connection is open or before it is closed. The last pointcut represents methods that encapsulate data transactions.

The "Persistent Objects" is a type extension definition, then, it may represent either a class or an interface that should be extended or implemented by a base class or interface. In this case, the application engineer must supply names of classes (or their super-types) which represent objects that should be persisted on the database.

"Dirty Objects Controller" is a boolean value which is used to define if the dirty objects controller should be active. This is used to update the database records automatically as soon as

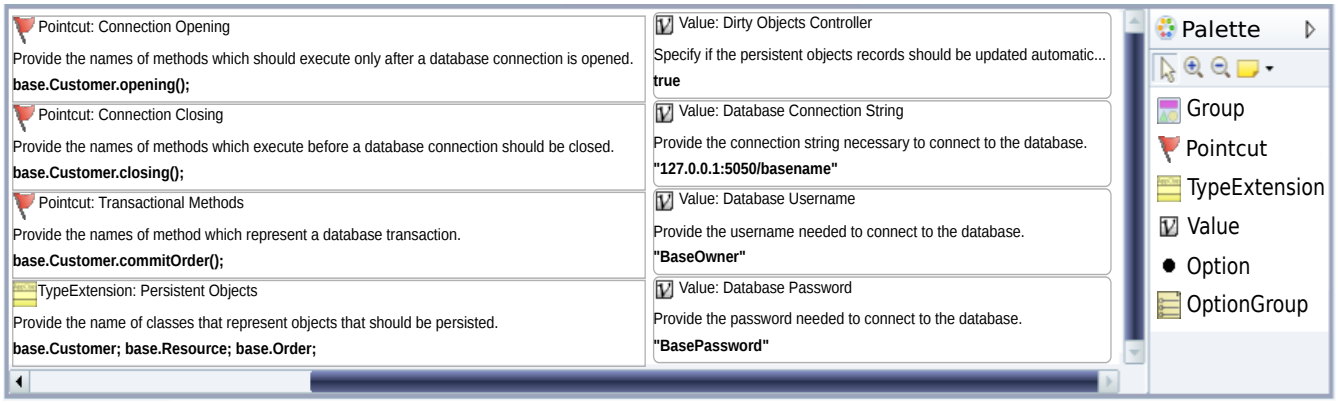


Fig. 2. Reuse Requirements Models and Reuse Models editor

any attribute belonging to a persistent object is changed by a set method. "Database Username" and "Database Password" are string values that are used to define the username and password needed to log into the database system. "Database Connection String" is a string value which should be used to specify the database connection details, i.e., the database system address, port and database name.

After completing the Reuse Model, it is possible to execute a code generator, which is a model to code transformation tool capable of generating reuse code in AspectJ, illustrated on Figure 3, which allows coupling the base application to the framework in a separate module. The final software is the composition of base application code, reuse code for each reused framework and the code of reused frameworks.

```
public aspect ConnectionCompositionReuse
        extends ConnectionComposition {
        public pointcut openConnection():
            execution  ( * base.Customer.opening());
        public pointcut closeConnection():
            execution  ( * base.Customer.closing());
        public pointcut transactional():
            execution  ( * base.Customer.commitOrder());
}
public aspect OORelationalMappingReuse
        extends OORelationalMapping {
        declare parents: base.Customer
                implements PersistentRoot;
        declare parents: base.Resource
                implements PersistentRoot;
        declare parents: base.Order
                implements PersistentRoot;
}
public aspect ConnectionValues {
        public String SelectedManager.setDSN() {
                return "127.0.0.1:/basename";
        }
        public String SelectedManager.setUsername() {
                return "BaseOwner";
        }
        public String SelectedManager.setPassword() {
                return "BasePassword";
        }
}
```

Fig. 3. Reuse Code Fragment

The first code of Figure 3 contains a new aspect which was created by generating code for the three pointcuts of the RM. This aspect extends an abstract aspect of the framework with the supplied information. In the second code, the type extension is implemented, then the classes written in the RM, "Customer", "Resource" and "Order", receive an interface of the framework, which is used to apply crosscutting behavior. In the third code, the value definitions are set by overriding methods of the framework.

IV. Evaluation

Two experiments were conducted to compare our reuse tool with the conventional technique. The first experiment is called "Reuse Study" and was planned to identify the gains in productivity when reusing a framework, which is the main objective of our tool. The second experiment is referred as "Maintenance Study" and was planned to identify whether the reuse models help or not the maintenance of an application that uses a CF and needs modifications. This second study is important because maintenance activities are usually performed more times than the reuse process. Each experiment was applied twice. In this paper, the first execution is referred as "Primary" and the second execution is referred as "Secondary". Since there are two executions for each experiment, we present four study executions in this section.

A. Reuse Study Definition

The objective is to compare the effort of reusing frameworks by using a conventional technique with by using a model-based technique. The Persistence CF briefly presented in Section II played the role of "study subject" and it was used in both reuse techniques (conventional and model-based). The quantitative focus was determined considering the time spent in conducting the reuse process and the qualitative focus was to determine which technique takes less effort during reuse process. This experiment was conducted from the perspective of application engineers reusing CFs and the study object is the 'effort' to perform a CF reuse.

B. Maintenance Study Definition

The objective was to compare the effort in modifying a CF-based application by editing the reuse code (conventional

technique) with by editing the RM. The Persistence CF shown in Section II was again used in the two maintenance exercises. The quantitative focus was measured by means of the time spent in the maintenance tasks and the qualitative focus was to determine which artifact takes less effort to edit during maintenance. This experiment was conducted from the perspective of application engineers who intend to maintain CF-based applications. The study object is the 'effort' to maintain a CF-based application.

C. Study Planning

The first experiment was planned considering the following question: "Which reuse technique takes less effort to reuse a CF?"; The second experiment was planned considering the question: "Which artifact takes less effort to edit during maintenance, reuse model or reuse code?"; We gathered and analyzed the timings taken to complete the process for each activity.

1) Context Selection: Both studies were conducted with students of Computer Science, in this section, they are referred as participants. Sixteen participants took part on the experiments, eight of these were undergraduate students and the other eight were post graduate students. Every participant had prior AspectJ experience.

2) Formulation of Hypotheses: The Table I contains our formulated hypotheses for the reuse study, which are used to compare the productivity of our tool and the conventional ad-hoc process. Both of these processes can be used to successfully reuse a CF and couple it to an application that has no reuse code.

TABLE I
HYPOTHESES FOR THE REUSE STUDY

$H0_r$	There is no difference between using our tool and using an ad-hoc reuse process in terms of productivity (time) to successfully couple a CF with an application. Then, the techniques are equivalent. Tc_r - $Tm_r \approx 0$
Hp_r	There is a positive difference between using our tool and using an ad-hoc reuse process in terms of productivity (time) to successfully couple a CF with an application. Then, the conventional technique takes more time than the model-based tool. Tc_r - $Tm_r > 0$
Hn_r	There is a negative difference between using our tool and using an ad-hoc reuse process in terms of productivity (time) to successfully couple a CF with an application. Then, the conventional technique takes less time than the model-based tool. Tc_r - $Tm_r < 0$

There are two variables shown on the table: "Tc_r" and "Tm_r". "Tc_r" represents the overall time to reuse the framework using the conventional technique while "Tm_r" represents the overall time to reuse the framework using the model-based tool. There are three hypotheses shown on the table: "$H0_r$", "Hp_r" and "Hn_r". The "$H0_r$" hypothesis is true when both techniques are equivalent; then, the time spent using the conventional technique minus the time spent using the model-based tool is approximately zero. The "Hp_r" hypothesis is true when the conventional technique takes longer than the model-based tool; then, the time spent to use the conventional technique minus the time of the model-based tool is positive. The "Hn_r" hypothesis is true when the conventional technique takes longer than the model-based tool; then, the time taken to use the conventional technique minus the time taken to use the model-based tool is negative. As these hypotheses consider different ranges of a single resulting real value, then, they are mutually exclusive and exactly one of them is true.

The formulated hypotheses for the maintenance study are listed on Table II. These hypotheses consider the outcome of comparing the edition of the reuse code (conventional technique) with our approach (model-based).

TABLE II
HYPOTHESES FOR THE MAINTENANCE STUDY

$H0_m$	There is no difference between using editing a reuse model and editing the reuse code in terms of productivity (time) when maintaining an application that reuses a CF. Then, it is equivalent to edit any of the artifacts. Tc_m - $Tm_m \approx 0$
Hp_m	There is a positive difference between using editing a reuse model and editing the reuse code in terms of productivity (time) when maintaining an application that reuses a CF. Then, editing the reuse code takes more time than editing a reuse model during maintenance. Tc_m - $Tm_m > 0$
Hn_m	There is a negative difference between using editing a reuse model and editing the reuse code in terms of productivity (time) when maintaining an application that reuses a CF. Then, editing the reuse code takes less time than editing a reuse model during maintenance. Tc_m - $Tm_m < 0$

The Table II also contains three variables. "Tc_m" represents the overall time to edit a reuse code during maintenance while "Tm_m" represents the overall time to edit the reuse model during maintenance. The "$H0_m$" hypothesis is true when the edition of both artifacts is equivalent. The "Hp_m" hypothesis is true when the edition of the reuse code takes longer than editing the RM. The "Hn_m" hypothesis is true when the edition of the reuse code takes less time than editing the RM. These hypotheses are also mutually exclusive and exactly one of them is true.

3) Variable Selection: The dependent variables are those which we analyze in this work. For each study, we provide analysis of the "time spent to complete the process". The independent variables are controlled and manipulated, for example, "Base Application", "Technique" and "Execution Types".

4) Participant selection criteria: The participants were selected through a non probabilistic approach by convenience, i. e., the probability of all population elements belong to the same sample is unknown.

5) Design of the studies: The participants were divided into two groups. Each group was composed by four post graduate students and four undergraduate students. Each group was also balanced considering a characterization form and their results from the pilot study. On Table III, there are the phases planned for both studies.

6) Instrumentation for the Reuse Study: Base applications were provided along with two documents. The first document is a manual regarding the current reuse technique, and the second document is a list of details, which describes the

TABLE III
STUDY DESIGN

Phase	Group 1	Group 2
General Training	Reuse and Maintenance Training	
	Repair Shop	
1^{st} Reuse Pilot Phase	Conventional	Models
	Hotel Application	
2^{nd} Reuse Pilot Phase	Models	Conventional
	Library Application	
1^{st} Primary Reuse Phase	Conventional	Models
	Deliveries Application	
2^{nd} Primary Reuse Phase	Models	Conventional
	Flights Application	
1^{st} Secondary Reuse Phase	Conventional	Models
	Medical Clinic Application	
2^{nd} Secondary Reuse Phase	Models	Conventional
	Restaurant Application	
1^{st} Primary Maintenance Phase	Conventional	Models
	Deliveries Application	
2^{nd} Primary Maintenance Phase	Models	Conventional
	Flights Application	
1^{st} Secondary Maintenance Phase	Conventional	Models
	Medical Clinic Application	
2^{nd} Secondary Maintenance Phase	Models	Conventional
	Restaurant Application	

classes, methods and values regarding the application to be coupled which are needed when reusing the framework.

The applications provided had the same reuse complexity, then, in order to reuse each application, the participants had to specify four values, twelve methods and six classes. Each phase row of the Table III is divided into the name of the application and the technique employed to reuse the framework. For instance, during the 1^{st} Primary Reuse Phase, the participants of the first group coupled the framework to the "Deliveries Application" by using the conventional technique, while the participants of the second used the model-based tool to perform the same exercise.

7) Instrumentation for the Maintenance Study: The base applications provided for the second study were modified versions of the same applications supplied during the first study. These applications were provided with incorrect reuse codes (conventional) and reuse models (model-based), which should be fixed by the participants. The participants received a manual regarding generic errors that could happen when the reuse code or model is incorrectly defined. It is important to point that the manual did not have details regarding the base applications, then, the participants had to find the errors by themselves by browsing the source code.

The applications provided had the same reuse complexity and the reuse codes and models had the same amount of errors. Then, in order to fix each CF coupling, the participants had to fix three outdated class names, three outdated method names and three mistyped characters. It is also important to point that errors specific to manual edition of reuse code were not inserted in this study. The phases are also listed on Table III. That table contains the name of the application and the technique employed during maintenance. For instance, during the 1^{st} Primary Maintenance Phase, the participants of the first group had to fix the reuse code of the "Deliveries Application" , while the participants of the second had to fix the reuse model to perform the same exercise.

D. Operation

1) Preparation: At first, every student was introduced to the tool and was taught how to edit reuse codes and reuse models. During each phase of the reuse study, the students were required to reuse the CF with a provided application. During the maintenance study, the students had to fix a reuse code or reuse model to complete the process. Every participant had to reuse and maintain every application by using only one of the techniques in equal numbers. Also, at any moment of the experiment, each group was using a different technique than the other group.

2) Execution: Initially, the participants signed a consent form and then answered a characterization form. The characterization form had questions regarding knowledge about AspectJ constructs, Eclipse IDE and Crosscutting Frameworks.

After concluding the characterization forms, participants were trained on how to reuse the supplied CF with the model-based reuse tool and then conventionally. It is important to note that every participant already had a basic experience with AspectJ and the conventional reuse of crosscutting frameworks.

Following the training, the pilot experiment was executed. The participants were split into two groups considering the results of the characterization forms. The pilot experiment was intended to simulate the real experiments, except that the applications were different, but equivalent. During the pilot experiment, the participants were allowed to ask questions about any issues they did not understand during the training. This could affect the validity, then, the data from this activity was only used to rebalance the groups.

During the real experiments, the participants had to work with two applications starting with a different technique for each group. The secondary executions were replications of the primary executions with another two applications. They were created in order to avoid the risk of getting unbalanced results during the primary execution, since some data gathered during the pilot were rendered invalid.

3) Data Validation: The forms filled by the participants were confirmed with preliminary data gathered during the pilot study. The researchers also watched the information system notifications to confirm if the participants had concluded and the captured data.

4) Data Collection: The recorded timings during the reuse processes with both techniques are listed on the Table IV. The timings for the maintenance study are found on Table V. There are five columns in each of these tables, "G" stands for the group of the participant during the activity; "A" stands for the application being reused; "T" stands for the reuse technique which is either "C" for conventional or "M" for model-based tool; "P" column lists an identifying code of the participants (students), whereas the least eight values are allocated to post-graduate students and the rest are undergraduate students; "Time" column lists the time the participant spent to complete each phase.

The information system employed to gather the experiment data stored the timings with milliseconds precision considering both the server and clients system clocks. However, the values presented in this paper only consider the server time, then, the delay of transmission by the computers are not considered, which are believed to be insignificant in this case, because preliminary calculations considering the client clocks did not change the order of results.

That system was able to gather the timings and supplied information transparently. The participants only had to execute the start time, which was supervised, and work on the processes by themselves. Once the test case provided had successful results, which meant that the framework was correctly coupled, the finish time was automatically submitted to the server before notifying the success to the participant.

TABLE IV
REUSE PROCESS TIMINGS

Real Study					Spare Study				
G	A	T	P	Time	G	A	T	P	Time
1	F	M	15	04:19.952015	2	C	M	10	02:59.467569
1	F	M	13	04:58.604963	1	R	M	13	03:56.785359
1	F	M	8	05:18.346829	1	R	M	15	04:23.629206
2	D	M	11	05:24.249952	2	C	M	11	04:25.196135
2	D	M	5	05:31.653952	1	R	M	8	04:33.954349
2	D	M	9	05:45.484577	2	C	M	9	04:41.254920
2	D	M	3	06:16.392424	1	R	M	12	05:05.524264
2	D	M	10	06:45.968790	2	C	M	3	05:45.333167
2	D	M	14	07:05.858718	2	C	M	14	05:57.009310
2	D	M	6	07:39.300214	2	C	M	5	06:31.365498
2	D	M	2	08:02.570996	2	C	M	2	06:59.967490
1	F	M	1	08:38.698360	2	R	C	2	07:18.927029
2	F	C	2	08:42.389884	2	C	M	6	07:45.403075
1	F	M	16	10:18.809487	2	R	C	10	08:56.765163
1	D	C	13	10:25.359836	1	C	C	16	09:20.284593
2	F	C	9	10:51.761493	1	R	M	7	09:23.574403
1	F	M	7	10:52.183247	1	R	M	4	09:25.089084
2	F	C	10	10:52.495216	2	R	C	14	09:27.112225
1	D	C	8	11:39.151434	2	R	C	3	09:55.736324
1	D	C	15	12:03.519008	1	C	C	15	10:25.475603
1	F	M	4	12:17.693128	2	R	C	5	10:37.460834
2	F	C	3	12:26.993837	2	R	C	9	10:49.014842
2	F	C	14	12:49.585392	1	R	M	16	10:56.743477
2	F	C	11	13:04.272941	1	C	C	13	11:04.485390
1	D	C	4	13:16.470523	1	C	C	4	12:06.690347
1	D	C	1	15:47.376327	1	C	C	8	13:38.014602
1	D	C	16	18:02.259692	1	C	C	12	14:37.197260
1	F	M	12	20:03.920754	1	R	M	1	17:09.073104
2	F	C	5	21:32.272442	2	R	C	11	17:11.980052
2	F	C	6	23:10.727760	1	C	C	7	19:35.816561
1	D	C	7	23:20.991158	2	R	C	6	28:02.391335
1	D	C	12	41:29.414342	1	C	C	1	28:18.301114

E. Data Analysis and Interpretation

The data of the first study is found on Table IV, which is ordered by the time taken to complete the process. The first notorious information found on this table is that the model-based reuse tool, which is identified by the letter 'M', is found on the first twelve results. The conventional process, which is identified by the letter 'C', got the last four results.

The timings data of Table IV is also represented graphically in a bar graph, which is plotted on Figure 4. The same identifying code for each participant and the elapsed time in seconds are visible on the graph. The bars for conventional technique and model tool use are paired for each participant, allowing easier visualization of the amount of time taken by each of them.

The second important information found on the first study is that is not a single participant that could reuse the framework faster by using the conventional process in the same activity than by using the reuse tool.

TABLE V
MAINTENANCE PROCESS TIMINGS

Real Study					Spare Study				
G	A	T	P	Time	G	A	T	P	Time
2	F	C	10	02:30.944685	2	C	M	5	01:43.801965
2	F	C	9	02:54.232578	2	C	M	3	02:17.158954
1	D	C	8	03:02.751342	1	C	C	8	02:34.248260
2	F	C	2	03:11.695431	1	C	C	14	02:57.405545
1	D	C	15	03:31.801582	2	R	C	2	03:01.547524
2	F	C	12	03:45.692316	2	R	C	10	03:09.169865
2	F	C	3	05:09.817914	2	C	M	2	03:25.640129
2	F	C	5	05:44.462030	2	R	C	3	03:39.443080
1	F	M	8	05:53.407296	1	C	C	7	04:28.998071
2	F	C	11	07:08.687074	2	R	C	6	04:35.517498
2	F	C	6	07:38.576312	2	R	C	12	04:41.052812
1	F	M	4	07:53.595699	2	R	C	11	04:46.028085
1	F	M	14	08:14.148937	1	R	M	8	04:51.290971
2	D	M	3	08:27.092566	2	C	M	6	04:53.800449
1	D	C	1	08:37.138931	1	R	M	15	04:58.094389
1	F	M	13	08:50.185469	1	C	C	15	05:21.846560
1	F	M	1	09:15.253791	2	R	C	5	05:42.389865
2	D	M	5	09:15.934211	2	C	M	10	07:18.533351
1	D	C	14	09:32.031612	1	R	M	14	07:24.342788
1	D	C	7	10:04.694800	1	C	C	16	07:37.332151
1	F	M	15	11:07.617639	1	R	M	1	07:44.516376
2	D	M	6	11:32.482992	2	C	M	11	08:08.144168
2	D	M	2	11:49.247460	2	R	C	9	08:13.115942
1	D	C	16	12:12.576158	1	R	M	13	08:32.056119
1	F	M	7	12:27.297563	1	R	M	16	11:28.592180
1	D	C	13	12:49.443610	1	R	M	7	11:45.459699
2	D	M	11	13:00.604583	2	C	M	9	12:42.958789
1	D	C	4	13:25.433748	1	R	M	4	13:57.879299
2	D	M	9	15:51.117061	1	C	C	1	14:46.465482
2	D	M	12	15:56.048486	1	C	C	4	17:55.176353
2	D	M	10	21:23.533192	1	C	C	13	18:02.486509
1	F	M	16	32:32.875079	2	C	M	12	25:54.176697

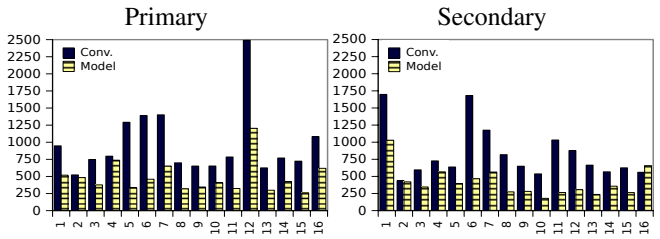


Fig. 4. Reuse Process Timings Bars Graph

The data of the second study is found on Table V. This study has provided similar results. The first eleven values were scored by using the model-based tool while the last four were scored by using the conventional tool. Only the participant number 16 was able to reuse the framework faster by using the conventional process, which contradicts the results taken from the same participant in the previous study. There is also a bar graph for this study in Figure 4.

The plots for the maintenance study are found on Figure 5, which also follow the same guidelines used while plotting the graphs for the previous study. Considering the timings of the maintenance study, the reuse model edition does not provide advantage in terms of productivity when maintaining an application that reuses a CF, since most of participants took longer to edit the model than the reuse code.

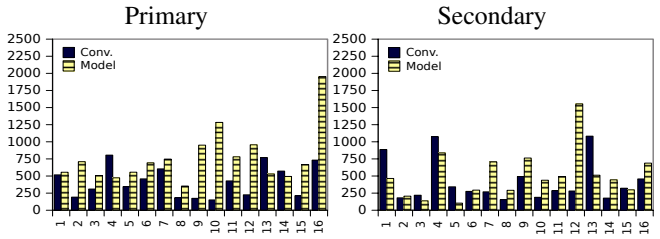


Fig. 5. Maintenance Process Timings Bars Graph

On Table VI there are average timings and their proportions. By considering the average time the participants of both groups needed to complete the processes, the conventional technique took approximately 97.64% longer than the model-based tool.

TABLE VI
AVERAGE TIMINGS

A.	Tech.	Avg.	Sum of Avg.	Percents
Reuse Study				
Primary	Conv.	16:13.44008	30:03.79341	66.7766%
Secondary		13:50.35333		
Primary	Model	08:04.980525	14:57.441176	33.2234%
Secondary		06:52.460651		
Total		45:01.234586		100.0000%
Maintenance Study				
Primary	Conv.	06:57.498758	13:55.762733	39.5521%
Secondary		06:58.263975		
Primary	Model	12:43.152626	21:17.305521	60.4479%
Secondary		08:34.152895		
Total		35:13.068254		100.0000%

F. Hypotheses Testing

In this section, we present statistical calculations to evaluate the data of both studies. We applied Paired T-Tests for each execution and another T-Test after removing eight outliers for each study. The seconds spent were processed using the statistic computation environment "R" [19]. Considering the reuse study, the results of the T-Tests are shown on Table VII. For the maintenance study, the same operation was executed and the results of the T-Test are shown on Table VIII.

The first columns of these tables contain the type of T-Test, the second columns indicate the source of the data, the "Means" columns indicate the resultant mean, which is the mean of the differences for an paired T-Test and one mean for each set for the other T-Test, which represent the conventional and the model-based tool means, respectively. The "d.t." columns stand for the degree of freedom; "t" and "p" are variables considered in the hypothesis testing.

The Paired T-Test is used to compare the the differences between two samples related to each participant, in this case, the time difference of every participant is considered individually, and then, the means of the differences are calculated. In the "Two-Sided" T-Tests, which are unpaired, the means are calculated for the entire group, because a participant may be an outlier in a specific technique, which breaks the pairs. It is referred as two-sided because the two sets have the same number of elements, since the same number of outliers were removed from each group.

TABLE VII
REUSE STUDY T-TEST RESULTS

T-Test	Data	Means	d.f.	t	p
Paired	Real	488.4596	15	5.841634	$3.243855 \cdot 10^{-05}$
Paired	Spare	417.8927	15	5.285366	$9.156136 \cdot 10^{-05}$
Two-Sided	Both	771.4236 409.4295	43.70626	6.977408	$1.276575 \cdot 10^{-08}$

The "Chi-squared test" was applied on both studies in order to detect the outliers that were removed when calculating the unpaired T-Test, which is refered as "Two-sided". The results of the "Chi-squared test" for the reuse study are found on Table IX and the results of the same test for the maintenance study are found on Table X. The 'M' in the techniques column indicates the use of our tool while 'C' indicates the conventional technique, the group column indicates the number of the group; the X^2 indicates the result of an comparison to the variance of the complete set and the position column indicates their position on the set, i.e., highest or lowest. The outlier column shows the timings in seconds that were considered abnormal.

TABLE VIII
MAINTENANCE STUDY T-TEST RESULTS

T-Test	Data	Means	d.f.	t	p
Paired	Real	-345.6539	15	-3.971923	0.001227479
Paired	Spare	-95.88892	15	-1.191781	0.2518624
Two-Sided	Both	431.3323 641.0024	24.22097	-2.662684	0.0135614

TABLE IX
CHI-SQUARED TEST FOR OUTLIER DETECTION APPLIED ON REUSE STUDY

Study	T.	G.	X^2	p	position	outlier
Real	C	1	5.104305	0.02386654	highest	2489.414342
		2	2.930583	0.08691612	highest	1390.72776
	M	1	4.091151	0.04310829	highest	1203.920754
		2	2.228028	0.1355267	highest	482.570996
Spare	C	1	4.552248	0.03287556	highest	1698.301114
		2	5.013908	0.02514448	highest	1682.391335
	M	1	3.917559	0.04778423	highest	1029.073104
		2	2.943313	0.08623369	lowest	179.467569

TABLE X
CHI-SQUARED TEST FOR OUTLIER DETECTION APPLIED ON MAINTENANCE STUDY

Study	T.	G.	X^2	p	position	outlier
Real	C	1	2.350449	0.1252469	lowest	182.751342
		2	2.152789	0.1423112	highest	458.576312
	M	1	5.788559	0.0161308	highest	1952.875079
		2	3.598538	0.05783041	highest	1283.533192
Spare	C	1	1.771974	0.183138	highest	1082.486509
		2	4.338041	0.03726978	highest	493.115942
	M	1	2.422232	0.1196244	highest	837.879299
		2	4.87366	0.02726961	lowest	1554.176697

In order to achieve better visualization of the outliers, we also provide two plots of the data sets. In Figures 6 and 7 there are line graphs which may be used to visualize the dispersion of the timing records. In these plots, the timings for each technique are ordered independently, therefore, the participant numbers in these plots are not related to their identification codes.

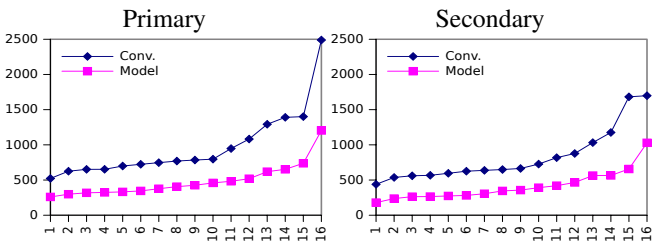


Fig. 6. Reuse Process Timings Bars Graph

Considering the reuse study and according to the analysis from Table VII, since all p-values are less than the margin

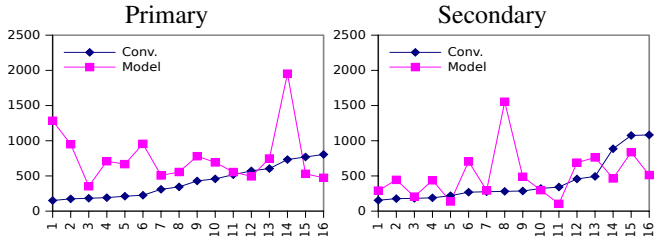


Fig. 7. Maintenance Process Timings Bars Graph

of error (0.01%), which corresponds to the established significance level of 99.99%, then, statistically, we can reject the "H0$_r$" hypothesis that states the techniques are equivalent. Since every t-value is positive, we can accept the "Hp$_r$" hypothesis, which considers that the conventional technique takes more time than our tool.

Considering the maintenance study and according to the analysis from Table VIII, since all p-values are bigger than the margin of error (0.01%), which corresponds to the established significance level of 99.99%, then, statistically, we cannot reject the "H0$_m$" hypothesis that states the techniques are equivalent. Therefore, statistically, we can assume that the effort needed to edit a reuse code and a reuse model is approximately equal.

G. Threats to Validity

Internal Validity:

- Experience Level of Participants: the varied participant knowledge that could affect the collected data. To mitigate this threat, we divided the participants in two balanced groups considering the experience level and rebalanced the groups considering the preliminary results. Also, the participants had prior experience on how to reuse the CF conventionally. During the training, the participants were trained on how to reuse the CF with the model-based tool and then again on how to reuse it conventionally, which could cause the participants to have more experience with the conventional technique.
- Productivity under evaluation: there is a possibility that this might influence the experiment results because students often tend to think they are being evaluated by experiment results. In order to mitigate this, we explained to the students that no one was being evaluated and their participation was considered anonymous.
- Facilities used during the study: different computers and installations could affect the recorded timings. However, the different groups used the same configuration, make, model and operating system in equal numbers and the participants were not allowed to change their machines during in the same activity, which means that a participant could not reuse a framework conventionally by using a different computer that was used to reuse it with our tool.

Validity by Construction:

- Hypothesis expectations: the participants already knew the researchers and knew that the model-based tool was supposed to ease the reuse process, which reflects one of our hypothesis. Both of these issues could affect the collected data and cause the experiment to be less impartial. In order to avoid impartiality, we enforced that the participants had to keep a steady pace during the whole study.

External Validity:

- Interaction between configuration and treatment: it is possible that the reuse exercises are not accurate for every reuse of a crosscutting framework for real world applications. Only a single crosscutting framework was considered and the base applications had the same complexity. To mitigate this threat, the exercises were designed considering applications based on the real world.

Conclusion Validity:

- Measure reliability: it refers to metrics used to measuring the reuse effort. To mitigate this threat we have used only the time taken which was captured by an information system in order to allow greater precision;
- Low statistic power: the ability of a statistic test in reveal reliable data. To mitigate we applied three T-Tests to statistically analyze the experiment data.

V. Related Work

The approach proposed by Cechticky *et al.* [20] allows object-oriented application framework reuse by using a tool called OBS Instantiation Environment. That tool supports graphical models do define the settings of the expected application to be generated. The model to code transformation generates a new application that reuses the framework.

The proposal found in this paper differs from their approach on the following topics: 1) their approach is restricted to frameworks known during the development of the tool; 2) it does not use aspect-orientation; 3) the reuse process is applied on application frameworks, which are used to create new applications.

Another approach was proposed by Oliveira *et al.* [21]. Their approach can be applied to a greater number of object oriented frameworks. After the framework development, the framework developer may use the approach to ease the reuse by writing the cookbook in a formal language known as Reuse Definition Language (RDL) which also can be used to generate the source code. This process allows to select the variabilities and resources during reuse, as long as the framework engineer specifies the RDL code correctly.

These approaches were created to support the reuse during the final development stages. Therefore, the approach proposed in this paper differs from others by the supporting earlier development phases. This allows the application engineer to initiate the reuse process since the analysis phase while developing an application compatible to the reused frameworks. Although the approach proposed by Cechticky *et al.* [20] is specific for only one framework, its can be employed since the design phase. The other related approach can be employed in a higher number of frameworks, however it is used in a lower

abstraction level, and does not support the design phase. Other difference is the generation of aspect-oriented code, which improves code modularization.

VI. Conclusions

In this paper, a model-based process was presented, which raises abstraction levels of CF reuse. It serves as a graphical view that replaces textual cookbooks and is used to perform the reuse in a model driven approach. From our proposed model-based approach, a new reuse process was delineated, which employs the forms during the development of a new application, allowing engineers to start the reuse since earlier software development phases and reduce the time to reuse a CF. With this, application developers do not need to worry about reuse coding issues nor how the framework was implemented, allowing to focus on the reuse requirements in a higher abstraction level.

Our approach was evaluated in two experiments that could answer the questions of the study planning, which indicate their conclusive success. The links for the gathered data can be accessed on *http://www2.dc.ufscar.br/~valter/*. The results regarding the productivity of reuse process were promising. However, the results of the maintenance study showed that our technique has no disadvantages in maintenance effort.

Furthermore, we have identified some limitations related to our research project. Once the models have been devised on top of the Eclipse Modeling Project, they can not be used in another environment. Furthermore, the code generator only generates code for Java and AspectJ, therefore, only frameworks developed in these languages are currently supported.

It is also important to point that our tool is part of a project to develop an integrated development environment for CF, which currently supports CF feature subset selection and a CF repository service. It is important to note that our tool also supports CFs that do not employ feature selection, in these cases, the RRM and RMs would be exactly equal.

However, we have not yet evaluated how to deal with coupling multiple CFs to a single base application. Despite this functionality already being supported, some frameworks may conflict with each other and lead to unwanted results.

The code generated is based on AspectJ and it was not evaluated if it supports every CF without modifications. Although not stated, we have also worked on selecting subsets of features of the framework.

Long term future works regard: (*i*) carry out a experiment using other CF, for verifying if the models proposed assist both the reuse and to maintain a reuse code; (*ii*) execute a experiment to verify whether the abstraction of the elements related to the models are sufficiently ideal; (*iii*) evaluate the standpoint of the domain engineers/frameworks, (*iv*) improve the elements of the models, i.e., better them graphically and (*v*) analyze the reusability of the abstract metamodel's metaclasses.

Acknowledgments

The authors would like to thank CNPq for funding (Processes 132996/2010-3 and 560241/2010-0) and for the Universal Project (Process Number 483106/2009-7) in which this paper was created. Thiago Gottardi would also like to thank FAPESP (Process 2011/04064-8).

References

[1] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *2007 Future of Software Engineering*, ser. FOSE 07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 37–54.

[2] AspectJ Team, "The AspectJ(tm) programming guide," 2003.

[3] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, 3rd ed. Addison-Wesley Professional, 2001.

[4] M. Fayad and D. C. Schmidt, "Object-oriented application frameworks," *Commun. ACM*, vol. 40, pp. 32–38, October 1997. [Online]. Available: http://doi.acm.org/10.1145/262793.262798

[5] M. Mortensen and S. Ghosh, "Creating pluggable and reusable non-functional aspects in AspectC++," in *Proceedings of the Fifth AOSD Workshop on Aspects, Components, and Patterns for Infrastructure Software*, 2006.

[6] V. Shah and V. Hill, "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, March, 23 2004.

[7] S. Soares, E. Laureano, and P. Borba, "Distribution and persistence as aspects," *Software: Practice and Experience*, vol. 33, no. 7, pp. 711–759, 2006.

[8] U. Kulesza, E. Alves, R. Garcia, C. J. P. D. Lucena, and P. Borba, "Improving extensibility of object-oriented frameworks with aspect-oriented programming," in *Proc. of the 9th Intl Conf. on Software Reuse (ICSR'06)*, 2006, pp. 231–245.

[9] V. Camargo and P. Masiero, "Frameworks orientados a aspectos," in *Anais Do 19º Simpósio Brasileiro De Engenharia De Software (SBES'2005), Uberlândia-MG, Brasil, Outubro.*, 2005.

[10] M. Huang, C. Wang, and L. Zhang, "Towards a reusable and generic aspect library," in *Workshop of the Aspect Oriented Software Development Conference at AOSDSEC'04*, Lancaster, UK, March, 23 2004.

[11] I. Zanon, V. V. Camargo, and R. A. D. Penteado, "Reestructuring an application framework with a persistence crosscutting framework," *INFOCOMP*, vol. 1, pp. 9–16, 2010.

[12] M. Bynens, D. Landuyt, E. Truyen, and W. Joosen, "Towards reusable aspects: The mismatch problem," in *Workshop on Aspect, Components and Patterns for Infrastructure Software (ACP4IS'10)*, 2010, pp. 17–20.

[13] D. Sakenou, K. Mehner, S. Herrmann, and H. Sudhof, "Patterns for re-usable aspects in object teams," in *Net Object Days*, Erfurt, 2006.

[14] C. Cunha, J. Sobral, and M. Monteiro, "Reusable aspect-oriented implementations of concurrency patterns and mechanisms," in *Aspect-Oriented Software Development Conference (AOSD'06)*, Bonn, Germany, 2006.

[15] N. Soudarajan and R. Khatchadourian, "Specifying reusable aspects," in *Asian Workshop on Aspect-Oriented and Modular Software Development (AOAsia'09)*, 2009.

[16] V. V. de Camargo and P. C. Masiero, "An approach to design crosscutting framework families," in *Proceedings of the 2008 AOSD workshop on Aspects, components, and patterns for infrastructure software*. New York, NY, USA: ACM, 2008, pp. 3:1–3:6.

[17] Eclipse Consortium, *Graphical Modeling Framework, version 1.5.0*, http://www.eclipse.org/modeling/gmp/, Graphical Modeling Project Std., 2011.

[18] R. Durelli, T. Gottardi, and V. Camargo, "Crossfire: An infrastructure for storing crosscutting framework families and supporting their model-based reuse," in *CBSoft2012 - Tools 2012*, September 2012.

[19] Free Software Foundation, Inc. , "R," http://www.r-project.org/, December 2011.

[20] V. Cechticky, P. Chevalley, A. Pasetti, and W. Schaufelberger, "A generative approach to framework instantiation," in *Proceedings of the 2nd international conference on Generative programming and component engineering*, ser. GPCE '03. New York, NY, USA: Springer-Verlag New York, Inc., 2003, pp. 267–286. [Online]. Available: http://portal.acm.org/citation.cfm?id=954186.954203

[21] T. C. Oliveira, P. Alencar, and D. Cowan, "Reusetool-an extensible tool support for object-oriented framework reuse," *J. Syst. Softw.*, vol. 84, no. 12, pp. 2234–2252, Dec. 2011. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2011.06.030

Anexo 4: Comprovante da publicação no *Simpósio Brasileiro de Engenharia de Software - Sessão de Ferramentas* (SBES-Tools'12)

Cópia do email de aceitação (Somente a primeira página)

Dear Ms. Rafael Durelli,

On behalf of the CBSoft Tools 2012 Program Committee, I am delighted to inform you that your paper "CrossFIRE: An Infrastructure for Storing Crosscutting Framework Families and Supporting their Model-Based Reuse" has been accepted to appear at the conference.

The Program Committee worked very hard to thoroughly review all the submitted papers. Please repay their efforts, by following their suggestions when you revise your paper.

Please send an email to rmfl@cin.ufpe.br with the final manuscript by July 10, 2012.

The reviews and comments are attached below. Again, try to follow their advice when you revise your paper.

Congratulations on your fine work.

If you have any additional questions, please feel free to get in touch.

Best Regards,
Ricardo Massa F. Lima

- Uma cópia do artigo é apresentado a seguir (a partir da próxima página).

CrossFIRE: An Infrastructure for Storing Crosscutting Framework Families and Supporting their Model-Based Reuse

Rafael S. Durelli[1], Thiago Gottardi[2] and Valter V. de Camargo[2]

[1]Computer Systems Department University of São Paulo
São Carlos, SP, Brazil.

[2]Computing Departament
Federal University of São Carlos (UFSCAR)
São Carlos, SP, Brazil.

rdurelli@icmc.usp.br[1], {thiago_gottardi, valter}@dc.ufscar.br[2]

Abstract. *A Crosscutting Framework (CF) is an abstract design of a single Crosscutting Concern (CC) which was designed for being reused. A Crosscutting Framework Family (CFF) is a set of features whose composition results in a CF. As CFs encapsulate just one CC, their usefulness occurs when there are a number of them available for being reused. However, despite the number of CFs proposed in the last years, there is no a suitable repository in which they can be uploaded and made available. Furthermore, most of them employ white-box strategies in their reuse process, requiring significant technical knowledge to reuse them. To cope with these problems, we put forward the CrossFIRE, which is an infrastructure that allows to store, search, view and reuse CFs. Thus, domain engineers can make their CFFs available and the application engineers can search this repository, select the CFF features and reuse them in a model-based fashion.*

1. Introduction

Crosscutting Framework (CF) is a term we have proposed in a previous work to represent an abstract aspect-oriented design of a crosscutting concern (CC), e.g., persistence, security and distribution [Camargo and Masiero, 2005]. The reuse process of CFs has two steps; instantiation and composition. The instantiation involves variability selection and hook implementation, while the composition involves providing composition rules to couple the chosen variabilities to a base code. Another important term in this paper is Crosscutting Framework Families (CFFs) [Camargo and Masiero, 2008]. A CFF is a set of features in which their composition results in a CF, that is, a member of this family is a CF. The difference between CF and CFF is that, in the later, there are several features which can be composed before starting the reuse process, while in the former there is no feature selection - the framework is ready to be used.

Most of the CFs found in literature apply white-box reuse strategies, relying on writing source code to reuse the framework [Bynens et al., 2010; Camargo and Masiero, 2005; Cunha et al., 2006; Kulesza et al., 2006; Sakenou et al., 2006]. White-box strategy makes application engineers to worry about low level implementation details during the reuse process, leading to the following problems: (*i*) the need to know coding details regarding the programming paradigm employed in the framework, making the learning

curve steeper; (*ii*) coding mistakes are more likely to happen when the reuse code is created manually; (*iii*) several lines of code must be written for the definitions of small number of hooks, impacting development productivity and (*iv*) reuse process can only be started during implementation phase, as there is no source code in earlier phases.

To overcome the described problems, we put forward an infrastructure which supports (*i*) the storage of CFFs/CFs, (*ii*) the feature selection and (*iii*) the model-based reuse process of CFs. In our infrastructure, domain engineers can make their CF/CFF available and application engineers can use one or more CFs in their application development in a model-based fashion.

There are two models that support the reuse process: Reuse Requirements Model (RRM) and Reuse Model (RM). The RRM documents all the information needed to perform the composition of a CF. The RM is instantiated along with the framework member, thus, it only contains the composition information related to the selected features. This model is also employed to conclude the reuse process in our model-driven approach by filling the fields of this model, which is also used for code generation. The motivation is the following: (*i*) there is neither a repository, in which the CFs/CFFs can be stored by domain engineer in order to share it, nor an infrastructure where the CFs can be reused as easy as possible in order to disseminate them; (*ii*) most of the CFs found apply white-box reuse strategies - thus it is important provides a way to assist the instantiation of these CFs using models. This paper is organized as followed: Section 2 provides information related to CrossFIRE - Section 3 the architecture of CrossFIRE is depicted - in Section 4 there are related works and in Section 5 we conclude the paper with some remarks and future directions.

2. CrossFIRE

In this section the Crosscutting Framework Integrated Reuse Environment (CrossFIRE) is presented. Figure 1 depicts a screenshot of CrossFIRE.

All artifacts of the CF/CFF must be developed before one uses the CrossFIRE. Thus, it is worth highlighting that CrossFIRE only supports the Application Engineering (AE) phase. Having this in mind, the domain engineer has to use a traditional approach in order to create the artifacts in the Domain Engineering (DE) phase. In the DE phase, three artifacts must be created: (*i*) source code of the whole CF, (*ii*) the RRM and (*iii*) the feature models of the CFF. These feature models have to be devised using the FeatureIDE[1], which is a tool that provides a graphical way to assemble feature models. Afterwards, the engineer can use the CrossFIRE in order to uploaded these artifacts into a repository, as is shown in Figure 1(A).

In the AE phase, an application engineer can browse the repository in order to check whether there are CFs that can be reused in order to implement concerns of the planned application. In order to assist this activity, CrossFIRE provides a visualization, which depicts all CFs/CFFs that have been uploaded by the domain engineers. Figure 1(B) shows the visualization of all CFs/CFFs available. As can be seen, there are six different CFFs - persistence, security, distribution, concurrency, logging and Business, respectively. In addition, CrossFIRE also shows descriptions for each of the selected CFFs by clicking

[1] `http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide/`

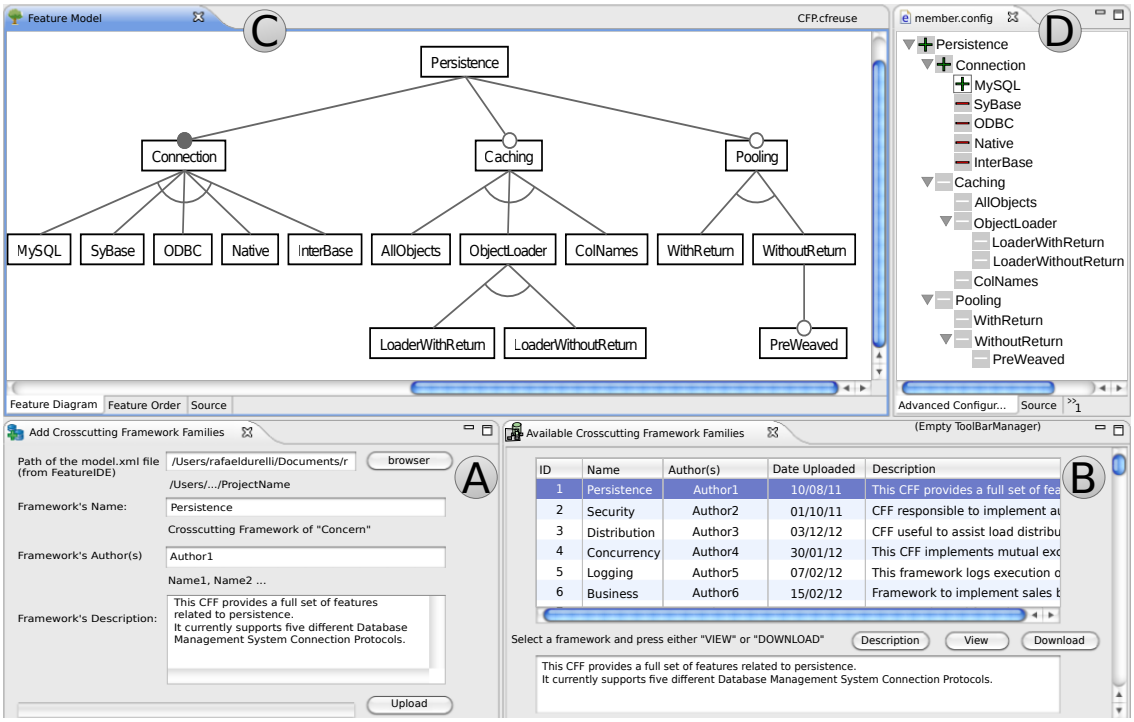


Figure 1. Screenshot of CrossFIRE

on the button “Description”. Nevertheless, if this description is not enough to help the engineer takes a decision on reusing the CF/CFF, CrossFIRE supplies a way to visualize the feature model related to selected CF/CFF by clicking on the button “View”, Figure 1(C). As is shown in Figure 1(B), the “persistence” CF is highlighted to indicate that it has been chosen. Next, the “Download” button has to be clicked to transfer the feature model belonging to the CF chosen from the remote repository to the computer of the application engineer.

Furthermore, to reuse the CFF, its features must be chosen by the engineer aiming at specifying explicitly which features will be reused in the base application. To assist this activity, CrossFIRE uses the “configuration file” of FeatureIDE. By using this “configuration file”, features can be chosen by the application engineer. The graphical notation of the “configuration file” is shown in Figure 1(D), which represents all features of the “persistence” CF. Moreover, FeatureIDE validates if the selected features match a valid combination for the instantiation of a member of the CFF. As shown in Figure 1(D), once the application engineer has chosen the features (represented by “+”), the resulting variant and constraints are generated automatically (represented by “-”).

The application engineer should provide the selected features by using the “configuration file” to repository server, which will carry out an algorithm. This algorithm aims to extract two artifacts. The first extracted artifact only contains code related to the selected features. The second extracted artifact is a RM derived from a RRM, by removing unrelated requirements. The RM is used to support the reuse process of a CF. After that, these artifacts are sent to the application engineer’s computer.

To reuse a member of the CFF or any other CF persisted in the repository, the ap-

plication engineer may use the RM. It is graphically represented as a form which contains fields that should be filled with information regarding the base application. By completing this form, the code needed to couple the CF to the base application can be generated. It is possible to see our model editor in Figure 2. The "Palette" on the right of the figure contains the elements of the RMs. They are: "Group": an element to group any element visible in the models; "Pointcut": represents join-points of the base application; "TypeExtension": represents types found in the base application; "Value": represents any numeric or textual values that must be informed while reusing the CF; "Option": defines a selectable variability of the framework and "OptionGroup": group selectable variabilities of the framework.

Each box contains a name and a description for the required information. The last line should be filled to provide the information regarding the base application. Note that the last line is only used in RMs. For example, to be able to instantiate a persistence CF, the application engineer must specify methods from base application that should be executed after a database connection is opened and before it is closed. Then, the box "Connection Opening" in Figure 2 represents names of methods that need an open database connection. It is also needed to specify methods that represent data base transactions, and the variabilities must be chosen, e.g., the driver which should be used to connect to the database system. After filling the fields of the RM, a model transformation generates the code needed to couple the CF, and then, the reuse process is complete. Our tool also provides validation of the information filled into RM models. At the moment this paper was written, only AspectJ CFs are supported.

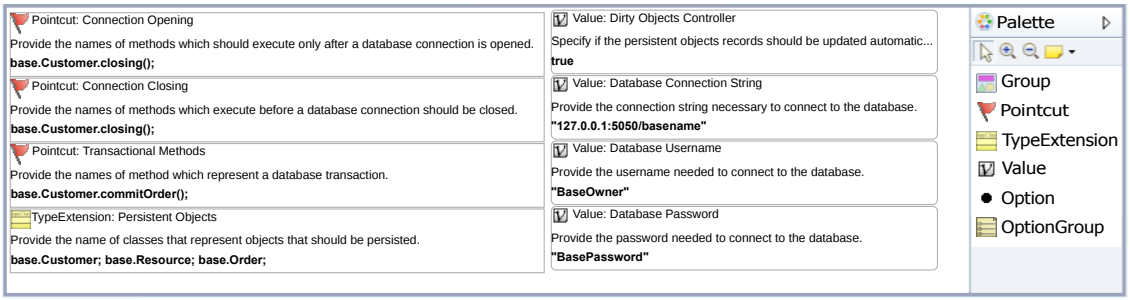


Figure 2. Reuse Model Editor

3. Architecture

In Figure 3 is depicted the architecture of the CrossFIRE. As shown in this figure, we devised the CrossFIRE on the top of the Eclipse Platform. On the other hand, both the RRM and RM have been developed using Eclipse Modeling Framework and Graphical Modeling Framework[2]. Moreover, we have studied FeatureIDE's source code and extended it in order to use for CF/CFF, its feature model editor and its configuration file, which is used to extract member of a CFF.

As can be seen in Figure 3, all artifacts (source code, feature model, RM and RRM) that the CrossFIRE provides are persisted in a database. These artifacts are persisted in a remote server, available to be reused in the AE phase. This remote server is a

[2]http://www.eclipse.org/modeling/gmp/

physical computer, which is dedicated to run the RESTful API. Therefore, to send these artifacts by the server we have used this API as web service to cache the representation of all artifacts. This server receives requests of the CrossFIRE through RESTful, processes database queries and sends a response to the CrossFIRE by using RESTful as well. Furthermore, we have used Java Persistence API (JPA) 2.0 to deal with the way relational data is mapped to Java objects. To implement the database of the server, the MySQL was chosen.

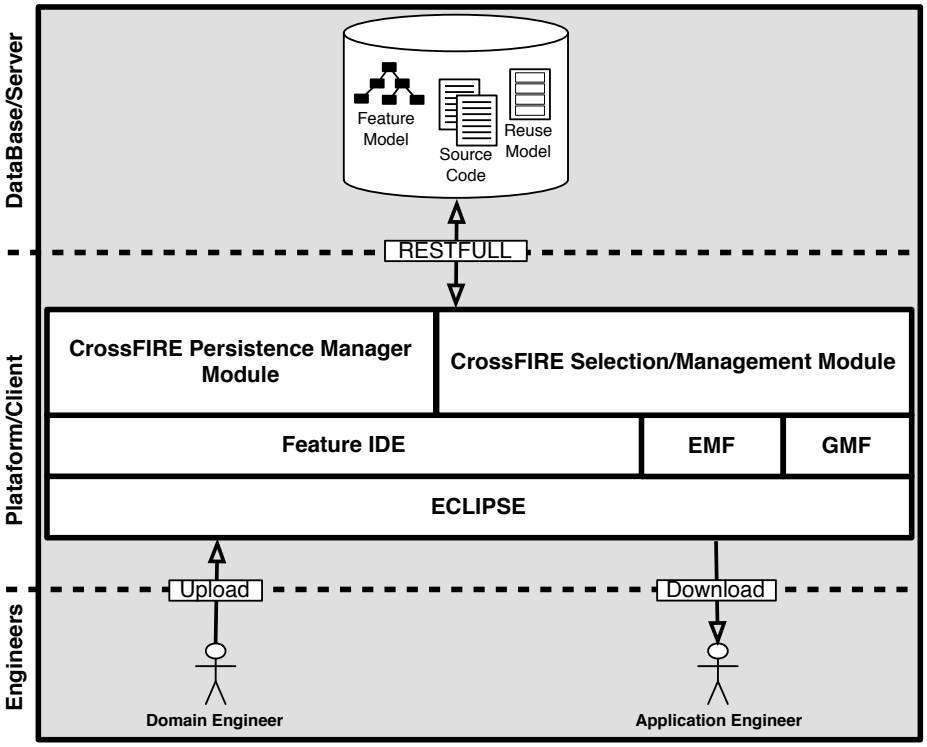


Figure 3. Architecture of CrossFIRE

4. Related Work

The approach proposed by Cechticky *et al.* [Cechticky et al., 2003] allows object-oriented application framework reuse by using a tool called OBS Instantiation Environment. That tool supports graphical models do define the settings of the expected application to be generated. The model to code transformation generates a new application that reuses the framework. The proposal found in this paper differs from their approach on the following topics: (*i*) their approach is restricted to frameworks known during the development of the tool; (*ii*) it does not use aspect orientation; (*iii*) the reuse process is applied on application frameworks, which are used to create new applications.

Another approach was proposed by Oliveira *et al.* [Oliveira et al., 2011]. Their approach can be applied to a greater number of object oriented frameworks. After the framework development, the framework developer may use the approach to ease the reuse by writing the cookbook in a formal language known as Reuse Definition Language (RDL) which also can be used to generate the source code. This process allows to select the variabilities and resources during reuse, as long as the framework engineer specifies the RDL code correctly.

5. Concluding Remarks

In the DE phase, CrossFIRE provides an infrastructure in which all of the CFF artifacts is developed. Afterwards, the CrossFIRE allows the engineers share these artifacts. As a result, these artifacts will be stored in a remote repository to be reused in the AE phase.

In the AE phase the CrossFIRE shows a list of all available CFFs that have been shared. Therefore, the engineer can pick out which CFF(s) can be reused in his base application. Next, the CrossFIRE provides a way to perform the download of the feature models related to each CFF. Through these feature models the engineer can pick out which features his base application really requires. As a consequence, the CrossFIRE downloads two artifacts, the necessaries chunks of code and a RM. Thus, the application engineer fills in the RM with the information needed by an member of a CFF's, and after that, it is possible to generate the final reuse code.

We believe that CrossFIRE increases the degree of reuse and allows the engineer to avoid dead code in his base application. Moreover, this infrastructure aims make the reuse activities easier. Long term future work involves conducting experiments to evaluate the level of reuse provided by CrossFIRE. It is worth highlighting that CrossFIRE is open source and it can be downloaded at *www.dc.ufscar.br/∼valter/crossfire*.

6. Acknowledgements

The authors would like to thank CNPq for Processes 132996/2010-3, 560241/2010-0 and 483106/2009-7 and FAPESP for Process 2011/04064-8.

References

Bynens, M., Landuyt, D., Truyen, E., and Joosen, W. (2010). Towards reusable aspects: The mismatch problem. In *Workshop on Aspect, Components and Patterns for Infrastructure Software (ACP4IS'10)*, pages 17–20.

Camargo, V. and Masiero, P. (2005). Frameworks orientados a aspectos. In *Anais Do 19º Simpósio Brasileiro De Engenharia De Software (SBES'2005), Uberlândia-MG, Brasil, Outubro.*

Camargo, V. V. and Masiero, P. C. (2008). An approach to design crosscutting framework families. In *Proceedings of the 2008 AOSD workshop on Aspects, components, and patterns for infrastructure software*, pages 3:1–3:6, New York, NY, USA. ACM.

Cechticky, V., Chevalley, P., Pasetti, A., and Schaufelberger, W. (2003). A generative approach to framework instantiation. In *Proceedings of the 2nd international conference on Generative programming and component engineering*, GPCE '03, pages 267–286, New York, NY, USA. Springer-Verlag New York, Inc.

Cunha, C., Sobral, J., and Monteiro, M. (2006). Reusable aspect-oriented implementations of concurrency patterns and mechanisms. In *Aspect-Oriented Software Development Conference (AOSD'06)*, Bonn, Germany.

Kulesza, U., Alves, E., Garcia, R., Lucena, C. J. P. D., and Borba, P. (2006). Improving extensibility of object-oriented frameworks with aspect-oriented programming. In *Proc. of the 9th Intl Conf. on Software Reuse (ICSR'06)*, pages 231–245.

Oliveira, T. C., Alencar, P., and Cowan, D. (2011). Reusetool-an extensible tool support for object-oriented framework reuse. *J. Syst. Softw.*, 84(12):2234–2252.

Sakenou, D., Mehner, K., Herrmann, S., and Sudhof, H. (2006). Patterns for re-usable aspects in object teams. In *Net Object Days*, Erfurt.

Anexo 5: Comprovante da publicação no *ACM SAC 2013* (SAC'13)

Cópia do email de aceitação (Somente a primeira página)

Dear Mr. Rafael Durelli:

On behalf of the SAC 2013 Program Committee, I am delighted to inform you that the following submission has been accepted to appear at the conference:

A Systematic Review on Mining Techniques for Crosscutting Concerns

The Program Committee worked very hard to thoroughly review all the submitted papers. Please repay their efforts, by following their suggestions when you revise your paper.

To upload your final manuscript, please visit the following site:

https://www.softconf.com/d/sac2013/

and, on the left-hand side of the page, enter the passcode associated with your submission. Your passcode is as follows:

1557X-J9E3J3J7J7

Alternatively, you can click on the following URL, which will take you directly to a form to submit your final paper:

https://www.softconf.com/d/sac2013/cgi-bin/scmd.cgi?scmd=aLogin&passcode=1557X-J9E3J3J7J7

The reviews and comments are attached below. Again, try to follow their advice when you revise your paper.

Congratulations on your fine work. If you have any additional questions, please feel free to get in touch.

Best Regards,

Eric Wong, John Kim, Chris Sung
SAC 2013, SE - Software Engineering

SAC 2013

The 28th ACM SYMPOSIUM ON APPLIED COMPUTING

José Carlos Maldonado and Sung Y. Shin – *Conference Chairs*
Rui P. Rocha – *Conference Vice-Chair*
Hisham M. Haddad - *Conference Treasurer and Registrar*
Chih-Cheng Hung and Jiman Hong – *Program Chairs*

SAC 2013

March 18 – 22, 2013 – Coimbra, Portugal

CERTIFICATE OF PARTICIPATION

SAC 2013 Organizing Committee, is hereby pleased to certify

that

Rafael Serapilha Durelli

has attended the 28th ACM SIGAPP Symposium on Applied Computing and presented paper titled "**A Systematic Review on Mining Techniques for Crosscutting Concerns**" in the Software Engineering Track.

Sincerely,

The Conference Organizing Committee

Hisham Haddad

SAC 2013

The 28th ACM Symposium on Applied Computing
March 18 – 22, 2013
Coimbra, Portugal

SAC 2013

Official Registration Receipt

SAC 2013 received from

RAFAEL SERAPILHA DURELLI

INSTITUTO DE CIÊNCIAS MATEMÁTICA e de COMPUTAÇÃO
ICMC/USP)

the sum of US$ 610,00 in payment of registration fee for the 28th ACM Symposium on Applied Computing, held in Coimbra, Portugal, March 18 – 22, 2013.

Hisham Haddad

SAC 2013 Treasurer
Hisham M. Haddad
hhaddad@kennesaw.edu

3/19/2013

Date

SAC 2013

The 28th ACM SYMPOSIUM ON APPLIED COMPUTING

José Carlos Maldonado and Sung Y. Shin – *Conference Chairs*
Rui P. Rocha – *Conference Vice-Chair*
Hisham M. Haddad - *Conference Treasurer and Registrar*
Chih-Cheng Hung and Jiman Hong – *Program Chairs*

SAC 2013

March 18 – 22, 2013 – Coimbra, Portugal

CERTIFICATE OF PARTICIPATION

SAC 2013 Organizing Committee, is hereby pleased to certify

that

RAFAEL SERAPILHA DURELLI
(Participant's title and full name)

of

USP ICMC - UNIVERSITY OF SÃO PAULO
(Participant's institution)

has attended the 28th ACM SIGAPP Symposium on Applied Computing and participated in the conference events.

Sincerely,
The Conference Organizing Committee

Hisham Haddad

- Uma cópia do artigo é apresentado a seguir (a partir da próxima página).

A Systematic Review on Mining Techniques for Crosscutting Concerns

Rafael S. Durelli
ICMC - USP
São Carlos, SP, Brazil
rdurelli@icmc.usp.br

Daniel S. M. Santibáñez
DC - UFSCar
São Carlos, SP, Brazil
daniel.santibanez@dc.ufscar.br

Nicolas Anquetil
RMoD Team - INRIA
Lille, France
Nicolas.Anquetil@inria.fr

Márcio E. Delamaro
ICMC - USP
São Carlos, SP, Brazil
delamaro@icmc.usp.br

Valter Vieira de Camargo
DC - UFSCar
São Carlos, SP, Brazil
valter@dc.ufscar.br

ABSTRACT

Background: The several maintenance tasks a system is submitted during its life usually cause its architecture deviates from the original conceivable design, ending up with scattered and tangled concerns across the software. The research area named concern mining attempts to identify such scattered and tangled concerns to support maintenance and reverse-engineering. Objectives: The aim of this paper is threefold: (*i*) identifying techniques employed in this research area, (*ii*) extending a taxonomy available on the literature and (*iii*) recommending an initial combination of some techniques. Results: We selected 62 papers by their mining technique. Among these papers, we identified 18 mining techniques for crosscutting concern. Based on these techniques, we have extended a taxonomy available in the literature, which can be used to position each new technique, and to compare it with the existing ones along relevant dimensions. As consequence, we present some combinations of these techniques taking into account high values of precision and recall that could improve the identification of both Persistence and Observer concerns. The combination that we recommend may serve as a roadmap to potential users of mining techniques for crosscutting concerns.

Categories and Subject Descriptors

D.2 [**Software Engineering**]: Miscellaneus

Keywords

Systematic Review, Concern Mining, Aspect Mining, Crosscutting Concerns.

1. INTRODUCTION

A possible definition for "software concern" is anything which stakeholders regard as a conceptual unit [11]. Examples of common software concerns include Persistence, Caching, Synchronization among others [5]. Developers and architects are continuously in need of up-to-date knowledge about the concerns currently implemented in their legacy system, and about the location of these concerns throughout the code. For example, during maintenance and reengineering, when there are bugs to be fixed, the maintenance task affects the whole implementation of a concern, and possibly to other concerns with which the fixed concern interacts.

Mining techniques for crosscutting concerns are indispensable for software maintenance, reverse engineering, reengineering and even for re-documentation [19]. However, manually applying a mining technique for crosscutting concern is difficult and error-prone. This came about because, legacy systems tend to: (*i*) have complex architectures with several clones spread out throughout the source code, (*ii*) involve several kinds of crosscutting concerns, e.g., patterns, architectural styles, business rules and non-functional properties and (*iii*) be very large, making the manual mining impractical. Thus, there is a need to use techniques and fully or semi-automated tools, which can aid software engineers to locate crosscutting concern into the legacy systems. In this context, the research area which aims to investigate techniques and tools to improve the mining of crosscutting concerns is known as "concern mining".

The aim of this paper is threefold. First, we aim to identify techniques employed in the research area herein described. Therefore, we have carried out a systematic review identifying mining techniques for crosscutting concerns. Second, we intent to extend the taxonomy presented by Kellens et al. [15]. Thus, we selected 62 papers and among them, we identified 18 mining techniques for crosscutting concerns. This taxonomy was proposed in 2007 and it needs to be updated because we found 7 new techniques developed in the past few years. Third, we recommend possible combinations of these techniques/tools that might improve recall and precision metrics for both Persistence and Observer concerns. We recommend four combinations of techniques discovered herein in order to improve recall and precision for Persistence concern. Similarly, we proposed two initial combinations of techniques identified to make better recall and precision metrics for Observer concern. This combination can be a motivation for potential users to improve the identification of well-known concerns.

SAC'13 March 18-22, 2013, Coimbra, Portugal.

This paper is organised as follows: In Section 2 presents how we have planned, conducted, reported and validated the systematic review. In addition, in this section there is also an extend taxonomy which was firstly proposed by Kellens et al. [15]. In Section 3 there are the threats to validity of our study. Section 4 presents a related work. Concluding remarks are made in Section 5.

2. THE SYSTEMATIC REVIEW

This study has been undertaken as a systematic review based on the guidelines proposed by Kitchenham and Brereton [16]. According to them, in order to conduct a systematic review, it is advisable to follow three main phases: (*i*) planning the review, (*ii*) conducting the review and (*iii*) reporting the review. Furthermore, in this paper we have used Visual Text Mining (VTM) technique to support the studies selection [18]. VTM uses text mining algorithms and methods combined with interactive visualisations. Therefore, it can help the user making sense of a collection of primary studies, without actually reading all of them. In this case the studies were reading partially or full. The following sections present details on how each phase was carried out.

2.1 Planning the Systematic Review

In this phase we have defined the review protocol. This protocol contains: (*i*) the research questions, (*ii*) the search strategy, (*iii*) the inclusion and exclusion criteria and (*iv*) the data extraction and synthesis method.

Research questions must embody the review study purpose. Moreover, these questions reflect the general scope of the review study. The scope is comprised of population (i.e., population group observed by the intervention), intervention (i.e., what is going to be observed in the context of the planned review study), and outcomes of relevance (i.e., the results of the intervention). Furthermore, during the conduction of this step, it was also necessary to establish the scope of the review study. According to the systematic review process [16], the scope has to be established using the PICO criteria. Thus, herein our **Population** is published scientific literature reporting on some existing mining technique for crosscutting concerns. The **Intervention** is published scientific literature interested with mining technique for crosscutting concerns. The **Comparison** is not applied herein. Finally, the **Outcomes of relevance** is an overview of the studies that have been conducted in the field of crosscutting concern mining, emphasizing primary studies that report on the techniques used in the research area, from observing such an aggregated data set, we also intend to provide insight into the frequencies of publication over time to inspect trends.

As described before, the objective of this review is to find out **which techniques are employed in mining techniques for crosscutting concerns. Moreover, we intent to extend the the taxonomy presented by Kellens et al. [15]. Finally, we also want to recommend possible combination of the identified techniques in order to identify remaining open research questions and possible avenues for future research**. In order to achieve such objectives we worked out five research questions. The questions are:

RQ$_1$: What are the mining techniques that are currently explored in the literature?

RQ$_2$: Which assessment techniques have been employed to evaluate these techniques and what are the results for common concerns?

RQ$_3$: Is there any difference in the precision and recall metrics when different techniques are used for mining the same concern?

RQ$_4$: Given a set of concerns, which are the most indicated techniques for performing the mining?

RQ$_5$: How can someone combine the techniques for improving the precision and recall metrics?

Afterwards, we have defined the search string and chosen the electronic databases. The search string was created based upon the following keywords: *approach, method, technique, methodology, aspect oriented, aspect-oriented, aspect mining, concern mining, coding mining, code mining, crosscutting concerns, cross-cutting concerns, Separation of Concern, SoC*. A sophisticated search string was constructed using boolean operators i.e., *AND*, *OR* and *NOT*. Figure 1 shows the search string elaborated. The search have encompassed electronic databases which are deemed as the most relevant scientific sources [9] and therefore likely to contain important primary studies. We have used the search string on the following electronic databases: *ACM* (portal.acm.org), *IEEE* (ieeexplore.ieee.org), *Scopus* (scopus.com) and *Springer* (springer.com/lncs).

*(("aspect oriented") **OR** ("aspect-oriented")) **AND** (("aspect mining") **OR** ("concern mining") **OR** ("coding mining") **OR** ("code mining")) **AND** (("crosscutting concerns") **OR** ("cross-cutting concerns") **OR** ("Separation of Concern") **OR** ("SoC")) **AND NOT** (("data mining") **OR** ("early aspects") **OR** ("product lines") **OR** ("mining of web"))*

Figure 1: Search String.

Then, in order to determine which primary studies are relevant to answer our research questions, we have applied a set of inclusion and exclusion criteria. Inclusion criteria devised and applied are:

(a) **The primary study presents at least one mining technique for crosscutting concern:** the encountered techniques must assist the software engineer in the crosscutting concern mining.

(b) **The primary study presents at least one type of evaluation technique for mining techniques for crosscutting concern:** without the results of the evaluation we neither would be able to make comparisons desired nor would we propose a set of combination of the identified techniques.

Not all of these criteria must be present for every primary study. However, at least the former (a) must be present. If all criteria were mandatory, the number of selected techniques would decrease significantly.

Exclusion criteria devised and applied are:

(a) **The primary study presents data mining technique. Nevertheless, such technique is applied to databases and not for crosscutting concern mining:** techniques which are applied to databases were not included, since this sort of techniques is outside the scope of this paper.

(b) **The primary study is a short paper:** papers with two pages or less were not considered herein, since we considered that this kind of study do not own sufficient information.

We devised data extraction forms to accurately record the information obtained by the researchers from the primary studies. The form for data extraction provides some standard information, such as (*i*) name of the techniques identified, (*ii*) date of data extraction, (*iii*) title, authors, journal, publication details and (*iv*) a list of each conclusion and statement encountered for each sub-question.

During the extraction process, the data of each primary study were independently gathered by two reviewers. The review was performed in August, 2012 by a M.Sc. and a Ph.D. students; the achieved results were crossed and then validated. All the results of the search process are documented in the web material[1]. Therefore, it is clear to others how thorough the search was, and how they can find the same documents.

2.2 Conducting the Systematic Review

In this phase, firstly we identified primary studies in the digital libraries. The digital libraries Scopus has returned more primary studies than the others (262), i.e., IEEE, ACM and Springer have returned 215, 202 and 127, respectively. Possibly, this came about because this digital library indexes studies of others libraries, such as IEEE and Springer. Summing up, we have gotten 802 primary studies. Afterwards we have selected the primary studies by means of reading the titles and abstracts and the application of the inclusion and exclusion criteria. As a result, we have gotten a total of 124 primary studies that were read entirely, so the upshot obtained were 62 studies.

Among these 62 primary studies we have identified 18 mining techniques for crosscutting concern. Therefore, each included primary study was assigned to one or more techniques. In the following we outline each of the techniques:

- *Execution Patterns (EP)*: This technique analyses program traces reflecting the run-time behavior of a system in search of recurring execution patterns. Thus, their mining algorithm discovers concern candidates based on recurring patterns of method invocations [3].

- *Dynamic Analysis (DA)*: This technique applies formal concept analysis (FCA) to execution traces in order to discover possible concerns. A version of the system is executed on a number of use cases. The obtained execution traces are analyzed using FCA for identifying methods and classes. Thus, methods belonging to more than one class may indicate presence of scattering code. If different methods from same class are specific to more than one use-case may indicate presence of tangling code [6].

- *Identifier Analysis (IA)*: This technique performs an identifier analysis using FCA algorithm. The assumption behind this approach is that relevant concerns in the source code are reflected by the use of naming conventions in the classes and methods of the system. As input to FCA algorithm, the classes and methods are used as objects and substrings generated from the classes and methods names are used as attributes. The resulting concepts consists out of maximal groups of classes and methods which share a maximal number of substrings [26].

- *Language Clues (LC)*: The approach uses natural language processing for mining crosscutting concern. The input is a collection of words from the source code and the output, chains of words which are semantically strongly related calculated with an algorithm. In order to mine for crosscutting concerns, they apply the chaining algorithm to the comments, method names, field names and class names of the system. A manual inspection to the resulting chains is needed in order to select possible concerns [25].

- *Method Clustering (MC)*: This technique starts by putting each method in a separate cluster and then, recursively, merges clusters by similarities in method names [1].

- *Call Clustering (CC)*: This technique starts from the assumption that if the same methods are called frequently from within different modules, then, they are closely related and must be clustered [30].

- *Fan-In analysis (FI)*: It is an approach that involves looking for methods that are called from many different call sites. Through fan-in metric, this approach can measure the number of methods that call some other methods. High fan-in metric values may indicate presence of a crosscutting concern [30].

- *AST-Based Clone Detection (ACD)*: This technique takes the Abstract Syntax Tree (AST) of the source code into account. The output is a number of clone classes, i.e. groups of code fragments which are considered to be clones of each other [4].

- *Token-Based Clone Detection (TCD)*: This technique is based on lexical analysis of the source code. The output is a number of clone classes, i.e., groups of code fragments which are considered to be clones of each other [4].

- *History Based (HB)*: This technique intends to discover crosscutting concerns by analyzing the changes made in the source code along the time by using software repositories like revision control systems, files and databases [21].

- *Information Retrieval (IR)*: This technique tries to identify concerns. It is based on the similarity between terms used in the concern descriptions and in the program elements, e.g., element names, variable names. The results are ranked and a manual inspection is performed [10].

- *Parser-Based (PB)*: This technique performs a lexical or syntactic analysis of the source code to locate crosscutting concerns. It is based on the premise that code fragments which share concerns are likely to refer to readily identifiable shared entities such as identifiers and libraries [12].

- *PrefixSpan (PS)*: It is a data mining technique used to identify coding patterns in source code. Each method

[1]http://tinyurl.com/99spmaz

in a program is translated to a sequence that comprises method call elements and control elements. The algorithm searches for repetitive subsequences that could form a pattern [14].

- *Concern-Peers (CP)*: This technique identifies certain groups of code units that potentially share some crosscutting concerns. These code units, called concern peers, are detected based on their similar interactions (similar calling relations in similar contexts, either internally or externally). The algorithm scan for candidates, i.e., methods with similar code and names, then scan for peers and rank it to recommend possible concerns [22].

- *Method Call Tree (MCT)*: It uses method call tree to generate method call traces. These traces are then investigated for recurring method patterns based on different constraints, such as, the requirement that the patterns exist in always the same composition and in different calling contexts in the method call trace [24].

- *Data-Flow Concern Identification (DF)*: It is a semi-automated approach for concern identification specifically designed to support software understanding. It starts from a set of related variables and uses static dataflow information to determine the concern skeleton, a data-oriented abstraction of a concern [27].

- *Random Walks (RW)*: A random walk is a mathematical formalization of a trajectory that consists of taking successive random steps. This technique performs a random walks on the coupling graphs extracted from the program sources. The algorithm reflects the degrees of "popularity and significance" for each of the program elements on the coupling graphs. Filtering techniques, exploiting both types of ranks, are applied to produce a final list of candidate crosscutting concerns [29].

- *Model-Driven (MD)*: This technique is a model driven approach for concern mining and their separation, which automatically identifies desirable candidate concerns, without requiring input from the user. The concern miner acts as a model transformer converting the source code to a concern-oriented model [23].

The taxonomy proposed by Kellens et al. [15], takes into account 3 dimensions: *(i)* static or dynamic analysis; if the technique does a static analysis of the code or dynamic information which is obtained by executing the program or both. *(ii)* Token-Based or structural/behavioral analysis; lexical analysis like sequences of characters, regular expression or abstract syntax trees, type information, message sends, etc. *(iii)* Granularity: The level of granularity of the technique, method level or more fine-grained.

In this context, we have extended the taxonomy proposed by Kellens et al. [15] by means of adding the new identified mining techniques. In Figure 2, it is depicted our extended taxonomy. The small rectangles in the middle of the figure represent all of the techniques: the previous ones, proposed by Kellens et al. [15] and the new ones proposed by us, marked with an asterisk in Figure 2. More specifically, from the 18 techniques identified herein, 7 of them are new and were added to the taxonomy. The new techniques are:

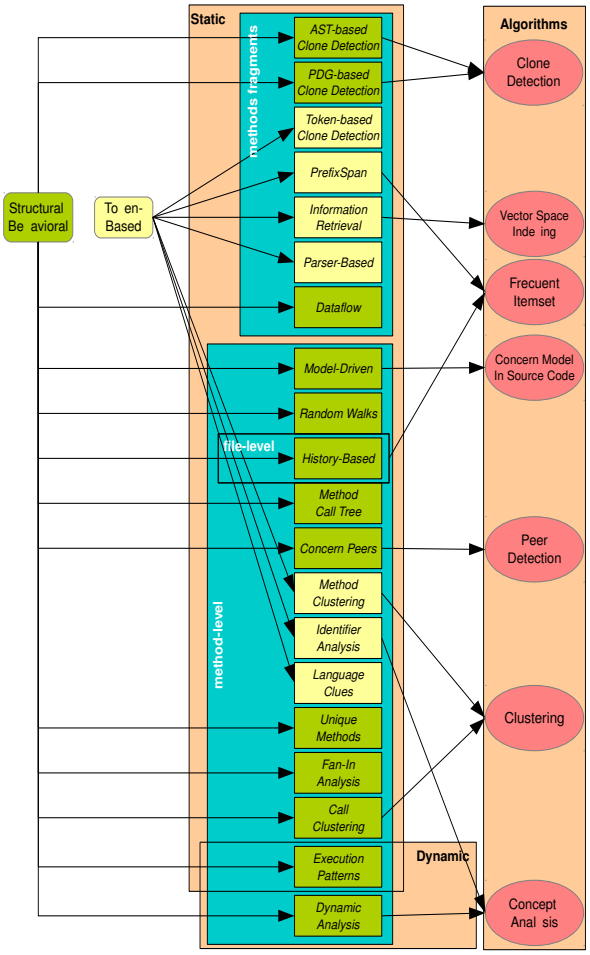


Figure 2: The Extended Taxonomy (Adapted from *Kellens et al.* [15]).

PrefixSpan, Information Retrieval, Dataflow, Model-Driven, Random Walks, History-Based, Concern Peers. The details on each of the identified new techniques and algorithms are outside the scope of this paper. Also, the fact of adding new techniques asked for the inclusion of new algorithms in the taxonomy as well. So, we added four algorithms, they are: Vector Space Indexing, Frecuent Itemset, Concern Model, Peer Detection. Finally, a new granularity level was added because the history-based technique can search for crosscutting concerns into source code repositories.

2.3 Validation

In validation phase an approach that uses VTM technique and the associated tool - Projection Explorer (PEx) - were applied to support the inclusion and exclusion decisions [18].

Figure 3 presents a document map generated using PEx. This map is composed of 802 primary studies analysed in this review, highlighting them using different shades of gray to differentiate in which of the stages a study was removed from the review. White points are studies excluded in first stage, gray points are the studies excluded in second stage and the black points are the included. The exploration of a document map is conducted in two steps: (*i*) firstly, a clustering algorithm is applied to the document map, cre-

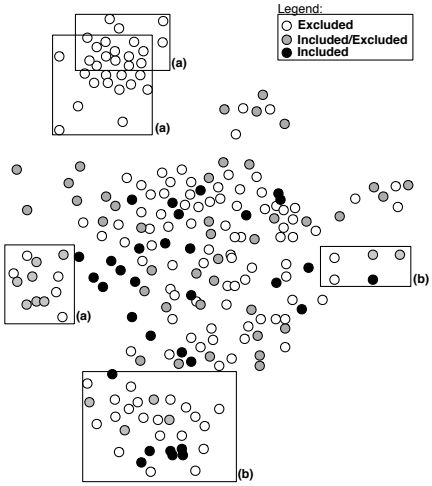


Figure 3: Document map colored with the history of the inclusions and exclusions of the studies.

ating groups of highly related documents; (*ii*) secondly, the resulting clusters are analysed in terms of: **Pure Clusters** - all documents belonging to a cluster have the same classification (all included or excluded, regardless of exclusion stage). Normally, in this case do not need to be reviewed; and **Mixed Clusters** - which represent documents with different classification on the same cluster. These cases are hints to the reviewer, and the estuaries grouped should be reviewed following the traditional method. To facility the visualisation, in Figure 3 just five clusters generated by PEx are depicted. Examples of pure clusters (all excluded) are identified in Figure 3 using label "(a)" and therefore do not needed to be reviewed. Mixed clusters (clusters containing black (included) and white or gray (excluded) studies) are identified using label "(b)" and they were reviewed by the authors of this paper. At the end, we kept the initial classifications conducted manually, but this technique contributed to a review of studies that could have been wrongly excluded or included previously.

2.4 Reporting the Systematic Review

The focus of this section is to present the broad overview of research within crosscutting concern mining we have acquired after classifying and categorizing primary studies. Moreover, we have used information drawn from this overview to answer this review study's research questions.

Aiming to show the frequencies of publication of all identified techniques for mining techniques for crosscutting concerns mining we have plotted a bubble plot, which is depicted in Figure 4. Bubble plots are essentially two x-y scatter plots with bubbles in category intersections. The size of each bubble is determined by the number of primary studies that have been classified as belonging to the categories corresponding to the bubble coordinates. This visual summary provides a bird's-eye view that enables one to pinpoint which categories have been emphasized in past research along with gaps and opportunities for future research.

In Figure 4 the facets we have used for organizing the map are the crosscutting concern mining techniques and year of publication. Based in this figure it is evident from observing it that we have found out 18 mining techniques for crosscutting concerns, as result we have answered the first part of the RQ_1. Based upon this bubble plot, we argue that the answer to second part of the RQ_1 is that Fan-In Analysis, Identifier Analysis and Dynamic Analysis are the techniques most used and Program Analysis Based, XScan-Concern-Peers, Data-Flow and Model Driven are the least used. More precisely, among the 62 primary studies included herein, 27 describe Fan-In Analysis, Identifier Analysis or Dynamic Analysis, respectively. In other hands, the techniques with less studies available in literature are Program Analysis Based, XScan-Concern-Peers, Data-Flow and Model Driven. Furthermore, it is argued that Fan-In Analysis, Identifier Analysis and Dynamic Analysis are evidence clusters (i.e., where there may be scope for more complete literature reviews to be undertaken). In contrast, Program Analysis Based, XScan-Concern-Peers, Data-Flow and Model Driven can be deemed as "evidence desert" (i.e., wherein better or new research is required).

The majority of selected primary studies are published by IEEE, i.e., 20 primary studies. The others primary studies have been selected from ACM, Scopus and Springer, 18, 16 and 8, respectively. As for the publication types, we have selected primaries studies that have been published in conferences, workshops and journals. The majority of the primary studies are conference paper (37), followed by workshop (16) and journal (9).

The way in which a technique is evaluated provides researchers and practitioners with useful information on the approach's quality, effectiveness, robustness, and practical applicability. Evaluating crosscutting concern mining techniques is difficult because defining the program elements that are relevant to a concern may be subjective. Despite this difficulty, researchers have devised some empirical strategies to assess them.

Empirical strategies of software engineering techniques are classified as experiment, case study and none [28]. In this context, we attempted to answer the RQ_2 by analyzing individually the 62 primary studies focus on gather which empirical strategies they have employed to validate the crosscutting concern mining techniques. In Figure 5 is depicted a pie chart wherein we have plotted the collected data.

As can be seen, among the 62 primary studies, 52 have carried out at least one empirical strategy. More specifically, 28 have carried out experiments to validate their crosscutting concern mining techniques and 24 primary studies have employed some sort of case studies. Only 10 primary studies neither have carried out experiments/case studies nor have made evident the use of specific evaluation strategies in order to validate their mining crosscutting concern techniques. Among the 52 primary studies, i.e., studies which carried out some evaluation techniques, we identified 31 studies using JHotDraw [2], a widely used framework to validate their mining crosscutting concern techniques. It is worth highlighting that from the universe of 31 studies using JHotDraw, 6 of them were evaluated by using two most well-known relevance-based measures of effectiveness, recall and precision. Table 1 presents the techniques and tools of these 6 studies with the precision and recall metrics for a particular JHotDraw version and where some crosscutting concerns were mentioned.

The data shown in Table 1 have useful information about how is the behavior in terms of precision and recall for different techniques regarding to JHotDraw concern. Note that

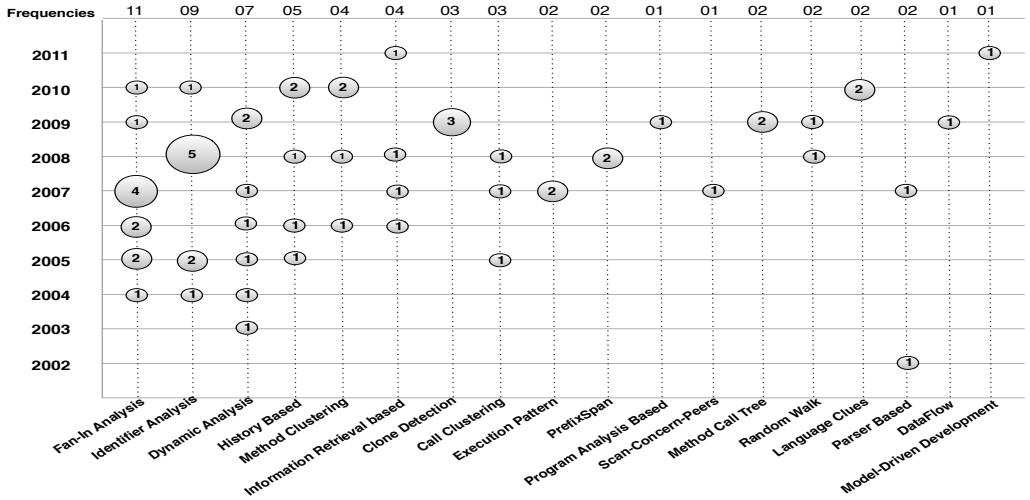


Figure 4: Map containing a year-wise distribution of publications on the all techniques found.

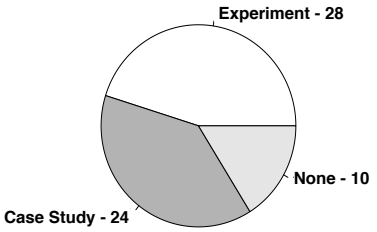


Figure 5: Which empirical strategies have been employed.

there are some missing recall values because they were not reported. Recall is the proportion of relevant concern candidates that were discovered out of all concern candidates present in the source code. Thus a problem with calculating this metric, in a program under analysis, it is not known what the relevant concerns and code fragments are, except in an ideal case. In order to respond the RQ_3 take the persistence concern into account. It has a good precision value in most of the cases, that is, the percentage of relevant concern candidates in the set of all candidates reported was high. However, the recall value is uneven and this is a strong evidence that the universe of concern candidates used by each technique is not standardized. In other words, as shown in Table 1 usually there is a difference in the precision and recall metrics when different sorts of concerns are mined.

Based on the Table 1 we have answered the RQ_4. This table give us evidence that some techniques must be addressed to deal with certain kind of concerns instead of others. For instance, as shown in Table 1, the best techniques to mine Persistence concern are XScan and CBFA, as they have a precision of 100% - recall of 93% and precision of 80% - recall of 100%, respectively. Similarly, the best techniques to mine Iterator concern are CBFA and XScan, since the former has both Precision and Recall of 100% and the latter has a Precision of 100% and Recall of 89%.

In order to answer the RQ_5 we devised Table 2 and Table 3. Those tables show some candidate combinations of techniques described in Table 1. Combining these techniques

Table 1: Precision and Recall for JHotDraw

Dynamic Analysis					
Ref.	Tool	Precision	Recall	Concern	Version
[7]	Dynamo	64% 100%	49% 26%	Undo Persistence	5.4
Concern Peers					
Ref.	Tool	Precision	Recall	Concern	Version
[22]	XScan	90% 100% 100% 97% 100%	93% 89% 93% 100% 100%	Undo Iterator Persistence Observer Visitor	6
Method Clustering + Fan-In					
Ref.	Tool	Precision	Recall	Concern	Version
[22]	CBFA	100% 100% 80% 86% 86%	86% 100% 100% 80% 100%	Undo Iterator Persistence Observer Visitor	6
Information Retrieval					
Ref.	Tool	Precision	Recall	Concern	Version
[13]	MSAM	5%	100%	Undo Persistence Observer Command	-
Method Clustering					
Ref.	Tool	Precision	Recall	Concern	Version
[8]	Clustering Algorithms	87.5%	-	Observer *Consistent Behaviour* *Contract Enforcement* Command	5.2
Call Clustering					
Ref.	Tool	Precision	Recall	Concern	Version
[17]	SOM	51%	-	Observer *Consistent Behaviour* *Contract Enforcement* Command	5.4
Fan-In					
Ref.	Tool	Precision	Recall	Concern	Version
[20]	FINT	30%	-	*Consistent Behavior* *Contract Enforcement*	5.4

*The values for MSAM were calculated using two thresholds, 0.4 for precision and 0.5 for recall.

can improve precision and recall metrics. We consider that someone could implement and/or reuse the best of several techniques to create a better mining technique for crosscutting concerns. For instance, as we have stated earlier, XScan is the best technique to mine Persistence concern, since it has a precision of 100%, but it has a recall of 93%. Therefore, maybe a solution to improve such recall, could be to combine other technique such as CFBA, which has a more reliable recall (100%) as shown in Table 1, i.e., first combi-

nation.

Also we stated that another solution to improve precision and recall metrics of techniques could be to combine the best techniques. For example, the second and the third combination illustrated in Table 2 represent this solution. We established that maybe the percentage of Dynamo will be improved if someone combine it with either CBFA or XScan. Finally, the last alternative is to combine Dynamo and MSAM, since according to the Table 1, the former has a good precision but it does not have a good recall, on the other side, the latter has a bad Precision but it has a good Recall. In the same way, Table 3, shows two candidate combinations in order to improve precision and recall for Observer concern.

Table 2: Combination for Persistence

N	Combined Technique
1st	CFBA + XScan
2nd	Dynamo + CBFA
3rd	Dynamo + XScan
4th	MSAM + Dynamo

Table 3: Combination for Observer

N	Combined Technique
1st	MSAM + CBFA
2nd	MSAM + Clustering Algorithms

3. THREATS TO VALIDITY

Primary studies selection. Aiming at ensuring an unbiased selection process, we defined research questions in advance and devised inclusion and exclusion criteria we believe are detailed enough to provide an assessment of how the final set of primary studies was obtained. However, we cannot rule out threats from a quality assessment perspective, we simply selected studies without assigning any scores. In addition, we wanted to be as inclusive as possible, thus no limits were placed on date of publication and we avoided imposing many restrictions on primary study selection since we wanted a broad overview of the research area.

Missing important primary studies. The search for primary studies was conducted in several search engines, even though it is rather possible we have missed some primary studies. Nevertheless, this threat was mitigated by selecting search engines which have been regarded as the most relevant scientific sources [9] and therefore prone to contain the majority of the important studies.

Reviewers reliability. All the reviewers of this study are researchers in the software reuse field, focused on the aspect-oriented programming, software testing and software product line, and none of the techniques and tools developed by us. Therefore, we are not aware of any bias we may have introduced during the analyses.

Data extraction. Another threat for this review refers to how the data were extracted from the digital libraries, since not all the information was obvious to answer the questions and some data had to be interpreted. Therefore, in order to ensure the validity, multiple sources of data were analyzed, i.e. papers, technical reports, white papers. Furthermore, in the event of a disagreement between the two primary reviewers, a third reviewer acted as an arbitrator to ensure full agreement was reached.

4. RELATED WORK

Closely related work to this review is a survey with aspect mining techniques [15], which details and compares a large selection of automated techniques and aspect mining tools. The goal is to present a comparative framework for distinguishing aspect mining techniques, and assess known techniques against this framework.

5. CONCLUDING REMARKS

In this paper we presented a systematic review of mining techniques for crosscutting concern, following the process described by Kitchenham [9]. Through a examination of 62 primary studies encompassing techniques to mine crosscutting concern, this review has presented 18 techniques. Researchers can use this review as a basis for advancing the field, while practitioners can use it to identify techniques that are well-suited to their needs. This systematic review should serve not only academic researchers but also industrial professionals, aiming at adopting some techniques to mine crosscutting concern within their organizations. The review described in this paper reveals that the most mentioned mining techniques for crosscutting concern are Fan-In Analysis, Identifier Analysis and Dynamic Analysis. In contrast, Program Analysis Based, XScan-Concern-Peers, Data-Flow and Model Driven can be deemed as "evidence desert".

Based on the identified techniques we have extended the taxonomy proposed by Kellens et al. [15]. This new taxonomy contains 7 new mining techniques for crosscutting concerns. By using this taxonomy we hold that this taxonomy could serve as an initial roadmap to crosscutting concern researchers. Moreover, this extended taxonomy could be relevant for tool developers who might have knowledge about the best aspect indicators to use or who may have certain demands about the granularity of the results.

The main future directions that emerged from this review are the need for empirical, comparative evaluations and the opportunity for developing combined techniques. Indeed, since every technique relies on different assumptions and uses different underlying analysis techniques, the found techniques are highly complementary, which suggests the possibility of several useful combinations. Thus, through the results obtained in this review we argue that if one pretends to devise a new mining techniques for crosscutting concerns to mine either Persistence or Observer, a good initial point is to take into consideration the combination herein illustrated in Table 2 and 3 but more studies are needed because the combinations proposed did not take into consideration the versions of the system, so we intend to analyze this in future works.

6. ACKNOWLEDGMENTS

Rafael Durelli would like to thank the financial support provided by FAPESP, process number 2012/05168-4. Daniel Santibáñez would like to thank the financial support provide by CAPES. Valter Camargo would like to thank CNPQ, process number 560241/2010-0. We also thank Thiago Gottardi for his proofreading of this paper.

7. REFERENCES

[1] M. L. Bernardi and G. A. Di Lucca. Conan: A tool for the identification of crosscutting concerns. In *16th Working Conference on Reverse Engineering*, Washington, DC, USA, 2009. IEEE Computer Society.

[2] J. Brant. *HotDraw*. Masters thesis, 1995.

[3] S. Breu and J. Krinke. Aspect mining using event traces. In *Proceedings of the 19th IEEE international conference on Automated software engineering*, Washington, DC, USA, 2004. IEEE Computer Society.

[4] M. Bruntink, A. van Deursen, R. van Engelen, and T. Tourwe. On the use of clone detection for identifying crosscutting concern code. *IEEE Trans. Softw. Eng.*, 31:804–818, October 2005.

[5] V. Camargo, R. Ramos, and P. Masiero. Implementacao de variabilidades em frameworks orientados a aspectos desenvolvidos em aspectj. *1st Brazilian Workshop on Aspect-Oriented Software Development (WASP)*, 2004.

[6] M. Ceccato. Automatic support for the migration towards aspects. In *Proceedings of the 2008 12th European Conference on Software Maintenance and Reengineering*, pages 298–301, Washington, DC, USA, 2008. IEEE Computer Society.

[7] M. Ceccato and P. Tonella. Dynamic aspect mining. *Software, IET*, 3(4):321 –336, august 2009.

[8] G. Cojocar and G. Czibula. On clustering based aspect mining. In *Intelligent Computer Communication and Processing, 2008. ICCP 2008. 4th International Conference on*, pages 129 –136, aug. 2008.

[9] T. Dyba, T. Dingsoyr, and G. K. Hanssen. Applying systematic reviews to diverse study types. In *International Symposium on Empirical Software Engineering and Measurement*, ESEM '07, pages 225–234, Washington, DC, USA, 2007. IEEE Computer Society.

[10] M. Eaddy, A. V. Aho, G. Antoniol, and Y.-G. Guéhéneuc. Cerberus: Tracing requirements to source code using information retrieval, dynamic analysis, and program analysis. In *Proceedings of the 2008 The 16th IEEE International Conference on Program Comprehension*, pages 53–62, Washington, DC, USA, 2008. IEEE Computer Society.

[11] M. Eaddy, T. Zimmermann, K. D. Sherwood, V. Garg, G. C. Murphy, N. Nagappan, and A. V. Aho. Do crosscutting concerns cause defects? *IEEE Trans. Softw. Eng.*, 34:497–515, July 2008.

[12] W. G. Griswold, Y. K. Y, and J. J. Yuan. Aspect browser: Tool support for managing dispersed aspects. In *Separation of Concerns in Object-oriented Systems*, 1999.

[13] J. Huang, Y. Lu, and J. Yang. Aspect mining using link analysis. In *5th International Conference on Frontier of Computer Science and Technology*, pages 312–317, Washington, DC, USA, 2010. IEEE Computer Society.

[14] T. Ishio, H. Date, T. Miyake, and K. Inoue. Mining coding patterns to detect crosscutting concerns in java programs. In *15th Working Conference on Reverse Engineering*, pages 123–132, Washington, DC, USA, 2008. IEEE Computer Society.

[15] A. Kellens, K. Mens, and P. Tonella. Transactions on aspect-oriented software development iv. chapter A survey of automated code-level aspect mining techniques, pages 143–162. Berlin, Heidelberg, 2007.

[16] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman. Systematic literature reviews in software engineering. *Information and Software Technology*, 51:7–15, January 2009.

[17] S. G. Maisikeli. *Aspect mining using self-organizing maps with method level dynamic software metrics as input vectors*. PhD thesis, 2010.

[18] V. Malheiros, E. Hohn, R. Pinho, M. Mendonca, and J. C. Maldonado. A visual text mining approach for systematic reviews. In *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*, ESEM '07, pages 245–254, Washington, DC, USA, 2007. IEEE Computer Society.

[19] M. Marin, A. V. Deursen, and L. Moonen. Identifying crosscutting concerns using fan-in analysis. *ACM Trans. Softw. Eng. Methodol.*, 17:3:1–3:37, December 2007.

[20] M. Marin, L. Moonen, and A. van Deursen. A common framework for aspect mining based on crosscutting concern sorts. In *13th Working Conference on Reverse Engineering*, pages 29–38, Washington, DC, USA, 2006. IEEE Computer Society.

[21] F. Mulder and A. Zaidman. Identifying cross-cutting concerns using software repository mining. In *International Workshop on Principles of Software Evolution*, pages 23–32, New York, NY, USA, 2010. ACM.

[22] T. T. Nguyen, H. V. Nguyen, H. A. Nguyen, and T. N. Nguyen. Aspect recommendation for evolving software. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 361–370, New York, NY, USA, 2011. ACM.

[23] B. Nora and S. Ghoul. A model-driven approach to aspect mining. In *27th International Conference on Software Engineering*, pages 361–370, New York, NY, USA, 2006. ACM.

[24] L. Qu and D. Liu. Aspect mining using method call tree. In *Proceedings of the 2007 International Conference on Multimedia and Ubiquitous Engineering*, pages 407–412, Washington, DC, USA, 2007. IEEE Computer Society.

[25] D. Shepherd, L. Pollock, and T. Tourwé. Using language clues to discover crosscutting concerns. In *Proceedings of the 2005 workshop on Modeling and analysis of concerns in software*, pages 1–6, New York, NY, USA, 2005. ACM.

[26] T. Tourwe and K. Mens. Mining aspectual views using formal concept analysis. In *Source Code Analysis and Manipulation, Fourth IEEE International Workshop*, pages 97–106, Washington, DC, USA, 2004. IEEE Computer Society.

[27] M. Trifu. Using dataflow information for concern identification in object-oriented software systems. In *Proceedings of the 2008 12th European Conference on Software Maintenance and Reengineering*, pages 193–202, Washington, DC, USA, 2008. IEEE Computer Society.

[28] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[29] C. Zhang and H.-A. Jacobsen. Mining crosscutting concerns through random walks. *IEEE Transactions on Software Engineering*, 2011.

[30] D. Zhang, Y. Guo, and X. Chen. Automated aspect recommendation through clustering-based fan-in analysis. In *Automated Software Engineering, 2008.*, pages 278 –287, 2008.

Anexo 6: Comprovante da publicação no *Latin-American Workshop on Aspect-Oriented Software Development 2012* (LA-WASP'12)

Cópia do email de aceitação (Somente a primeira página)

Dear Ms. Rafael Durelli:

Congratulations - your paper "A Systematic Review on Mining Techniques
for Crosscutting Concerns" for CBSoft2012 - LAWASP has been accepted in
the Posters track.

The reviews are below or can be found at
https://submissoes.sbc.org.br/PaperShow.cgi?m=104677

Regards,
Conference Chairs

- Uma cópia do artigo é apresentado a seguir (a partir da próxima página).

A Systematic Review on Mining Techniques for Crosscutting Concerns

Rafael Serapilha Durelli†, Daniel San Martín Santibáñez*, Márcio Eduardo Delamaro† and Valter Vieira de Camargo*
*Departamento de Computação, Universidade Federal de São Carlos,
Caixa Postal 676 – 13.565-905, São Carlos – SP – Brazil
Email: {daniel.santibanez,valter}@dc.ufscar.br
†Instituto de Ciências Matemáticas e Computação, Universidade de São Paulo,
Av. Trabalhador São Carlense, 400, São Carlos – SP – Brazil
Email: {rdurelli, delamaro}@icmc.usp.br

***Abstract*—Background: The several maintenance tasks a system is submitted during its life usually causes its architecture deviates from the original conceivable design, ending up with scattered and tangled concerns across the software. Mining techniques for crosscutting concern attempt to identify such scattered and tangled concerns in a program to support maintenance, reverse-engineering or the first step of a reestructuration process towards an aspect-oriented software. Objectives: Conducting a systematic review aiming at identifying the main tools and techniques employed in this process and also verify the existence of combined mining approaches. Methods: We have carried out a systematic review based on searching of major electronic databases. Results: We selected and classified 62 papers by their mining technique, type and year of publication. Among these papers were identified 18 mining techniques for crosscutting concern. Furthermore, we believe strongly that the results of this review may serve as a roadmap to potential users of this type of techniques, e.g., helping future researchers in selecting an appropriate technique. Conclusions: Our review shows that in the literature there is a considerable number of mining techniques for crosscutting concern. It is interesting to note that a large amount of this techniques are intended to identify several types of crosscutting concerns but there is a lack of research to say which technique is most suitable for identifying a specific crosscutting concern. Also, we believe it is necessary to perform studies showing the pros and cons of using combinations of several mining techniques for crosscutting concern in a unique software environment.**

I. Introduction

Mining techniques for crosscutting concern is indispensable for software maintenance, reverse engineering, re-engineering and even for re-documentation. However, manually applying mining techniques for crosscutting concern is difficult and error-prone [1], once, this sort of system tends to (*i*) have complex architectures with several clones spread out through the source code; (*ii*) involve several kinds of crosscutting concerns, such as patterns, architectural styles, business rules and non-functional properties; (*iii*) be very large, making the manual mining impractical. Thus, there is a need to use techniques and automated or semi-automated tools, which can aid software engineers to locate crosscutting concern into legacy systems. In this context, the research area which aims to investigate techniques and tools to increase the mining of crosscutting concerns is known as "concern mining" [2].

In this paper we present a systematic review with the aim of identifying a large amount of techniques related to crosscutting concern mining. As a consequence, we have discovered that in the last years a considerable number of mining techniques for crosscutting concern are available within academic context. Furthermore, this systematic review shows that there are a lot of interesting and important research topics that could be investigated yet.

II. The Systematic Review

This study has been undertaken as a systematic review based on the guidelines proposed by Kitchenham and Brereton [3]. According to them, in order to conduct a systematic review it is advisable to follow three main phases: (*i*) planning the review – (*ii*) conduction the review – and (*iii*) reporting the review. Following sections present slightly details on how each phase was carried out.

A. Planning the Systematic Review

In this phase we have defined the review protocol. This protocol contains: (*i*) the research questions, (*ii*) the search strategy, (*iii*) the inclusion and exclusion criteria and the (*iv*) data extraction and synthesis method. In this setting, three research questions were devised, they are:

- **RQ_1:** Which techniques for mining crosscutting concerns are used within academic contexts? Furthermore, which ones are more and less explored?
- **RQ_2:** Which evaluation techniques (e.g., case study, empirical strategies, comparative experiments) have been employed to evaluate these techniques?
- **RQ_3:** Considering the techniques that we found, which ones have automated support?

Afterwards, we have defined the search string and the electronic databases. Figure 1 shows the search string which we have used. We have used the search string on the following electronic databases: *ACM* (www.portal.acm.org), *IEEE* (www.ieeexplore.ieee.org), *Scopus* (www.scopus.com) and *Springer* (www.springer.com/lncs). No limits were placed

on date of publication with a view to not restrict the review study scope.

*(("approach") **OR** ("method") **OR** ("technique") **OR** ("methodology")) **AND** (("aspect oriented") **OR** ("aspect-oriented")) **AND** (("aspect mining") **OR** ("concern mining") **OR** ("coding mining") **OR** ("code mining")) **AND** (("crosscutting concerns") **OR** ("cross-cutting concerns") **OR** ("Separation of Concern") **OR** ("SoC")) **AND NOT** (("data mining") **OR** ("early aspects") **OR** ("product lines") **OR** ("mining of web"))*

Fig. 1. Search String.

The Inclusion criteria devised and applied are: $\mathbf{IC}_1$**:** the primary study presents at least one mining technique for crosscutting concern and $\mathbf{IC}_2$**:** the primary study presents at least one type of evaluation technique for crosscutting concern. Similarly, the exclusion criteria are: $\mathbf{EC}_1$**:** the primary study is not about mining techniques for crosscutting concern – $\mathbf{EC}_2$**:** the primary study presents data mining technique, however, such technique is applied to databases and not for crosscutting concern mining – $\mathbf{EC}_3$**:** the primary study is a short paper (papers with twos pages or less).

B. Conducting the Systematic Review

The database Scopus has returned more primary studies than the others databases (262), i.e., IEEE, ACM and Springer have returned 215, 202 and 127, respectively. Possibly, this came about because this database indexes studies of others databases, such as IEEE and Springer. Summing up, we have gotten 806 primary studies. After, we read the the titles and abstracts and applied the inclusion and exclusion criteria. As a result, we have gotten a total of 124 primary studies that were read entirely, so the upshot obtained were 62.

C. Reporting the Systematic Review

Aiming to answer the RQ_1 we have read the 62 primary studies, and thereby we have identified 18 techniques of crosscutting concern mining. Therefore, each included primary study was assigned to one or more techniques. Additionally, we classify those techniques in two major categories: (*i*) Generative approach and (*ii*) Query-based. The first ones aims at extracting and generating aspect seeds automatically using structural information of the program source code while the second ones utilizes manual input such as textual pattern.

Figure 2 illustrates the frequency of primary studies related to techniques which were found. It may be seem clearly that Fan-In Analysis followed by Identifier Analysis are the more explored. Similarly, Program Analysis Based, XScan-Conern-Peers, Data-Flow Concern Identification and Model-Driven are less explored.

Aiming to answer the RQ_2 we have analyzed individually the 62 primary studies focus on gather which empirical strategies them have employed to validate the crosscutting concern mining techniques. Thus, among the 62 primary studies, 28 have been carried out experiments to validate their crosscutting concern mining techniques. Similarly, 24 primary studies have been employed some sort of case studies, and 17 neither have carried out experiments nor case studies to validate their techniques for mining crosscutting concern.

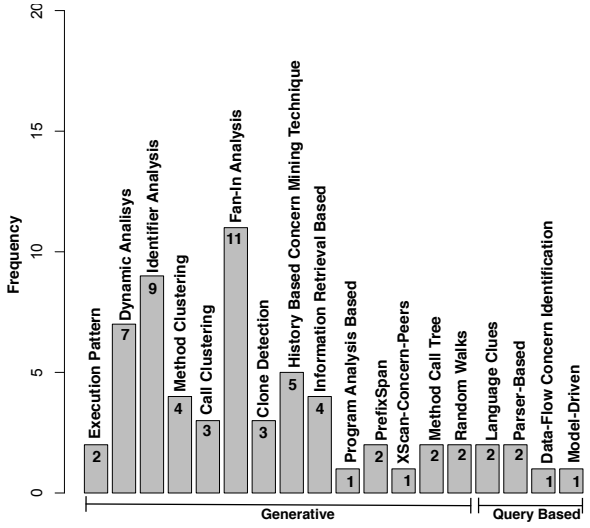


Fig. 2. Frequency of studies in each category.

As for answering the RQ_3 we have read the 62 primary studies, as consequence, we found 33 tools. They are: DynAMiT, Dynamo, CERBERUS, AspectRT, DelfStof, FindConcept tool, Lexical-Chaining tool, ConAn, AMAV, SOM, CBFA, FAN-in metric tool, Fint, CBFA, PDG-DUP, ccdiml, CCFinder, COMMIT, HAM, line co-change, tool-chain, MSAM, Aspect Browser, Grep, JQuery, FEAT, DRACO-PUC, AMT, FUNG, Xscan, RRAM, CoDEx and Prism CC.

III. Concluding Remarks

The main contribution of this paper is to provide a basic knowledge and state of art of several techniques found in the literature. It can be used to introduce new researchers in concern mining research area.

Although there are some tools which combine concern mining techniques, it is important to perform studies showing the pros and cons of using combinations of several mining techniques for crosscutting concern in a unique software environment. Another interesting issue is to know which technique is most suitable for identifying a specific crosscutting concern.

Acknowledgment

The authors would like to thanks the financial support provide by CAPES. Rafael S. Durelli also would like to thanks the financial support provided by FAPESP (Process 2012/05168-4).

References

[1] M. Eaddy, A. Aho, and G. C. Murphy, "Identifying, assigning, and quantifying crosscutting concerns," in *Proceedings of the First International Workshop on Assessment of Contemporary Modularization Techniques*. Washington, DC, USA: IEEE Computer Society, 2007.

[2] B. Adams, Z. M. Jiang, and A. E. Hassan, "Identifying crosscutting concerns using historical code changes," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE 10. New York, NY, USA: ACM, 2010, pp. 305–314.

[3] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering - a systematic literature review," *Information and Software Technology*, vol. 51, pp. 7–15, January 2009.

Anexo 7: Historico Escolar

Janus - Sistema Administrativo da Pós-Graduação

Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação

FICHA DO ALUNO

55134 - 6713611/2 - Rafael Serapilha Durelli

Email:	rdurelli@icmc.usp.br
Data de Nascimento:	13/07/1987
Cédula de Identidade:	RG - 340631377 - SP
Local de Nascimento:	Estado de São Paulo
Nacionalidade:	Brasileira
Graduação:	Bacharel em Ciência da Computação - Centro Universitário Eurípides de Marília - Fundação de Ensino "Eurípides Soares da Rocha" - São Paulo - Brasil - 2009
Mestrado:	Mestre em Ciência da Computação (1) - Universidade Federal de São Carlos - São Paulo - Brasil - 2011

Curso:	Doutorado
Programa:	Ciências de Computação e Matemática Computacional
Data de Matrícula:	02/03/2012
Início da Contagem de Prazo:	08/08/2011
Data Limite:	08/04/2016
Orientador:	Prof(a). Dr(a). Márcio Eduardo Delamaro - 02/03/2012 até o presente. E.Mail: delamaro@icmc.usp.br
Proficiência em Línguas:	Inglês, Aprovado em 17/10/2012
Data de Aprovação no Exame de Qualificação:	Aprovado em 11/12/2012
Data do Depósito do Trabalho:	
Título do Trabalho:	
Data Máxima para Aprovação da Banca:	
Data de Aprovação da Banca:	
Data Máxima para Defesa:	
Data da Defesa:	
Resultado da Defesa:	
Histórico de Ocorrências:	Ingressou no Doutorado em 02/03/2012 Matrícula de Acompanhamento em 18/02/2013

Aluno matriculado nas normas vigentes a partir de 01/07/2009
Última ocorrência: Matrícula de Acompanhamento em 18/02/2013
Impresso em: 13/03/13 10:00:49

Janus - Sistema Administrativo da Pós-Graduação

Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação

FICHA DO ALUNO

55134 - 6713611/2 - Rafael Serapilha Durelli

Sigla	Nome da Disciplina	Início	Término	Carga Horária	Cred.	Freq.	Conc.	Exc.	Situação
SSC5906-2/1	Engenharia de Software Experimental (2)	08/08/2011	30/09/2011	90	6	100	A	N	Concluída
SSC5905-2/1	Revisão Sistemática em Engenharia de Software (2)	03/10/2011	02/12/2011	90	6	100	A	N	Concluída
SSC5877-4/1	Validação e Teste de Software	12/03/2012	29/06/2012	180	12	100	A	N	Concluída
SMA5839-15/2	Preparação Pedagógica	12/03/2012	29/06/2012	60	4	100	A	N	Concluída
SSC5793-2/2	Especificação formal de software	06/08/2012	07/12/2012	180	12	80	A	N	Concluída

	Créditos mínimos exigidos		Créditos obtidos
	Para exame de qualificação	Para depósito de tese	
Disciplinas:	24	36	40
Estágios:			
Total:	24	36	40

Créditos Atribuídos à Tese: 144

Observações:

1) Curso com validade nacional, de acordo com o disposto na Portaria nº 524, de 29.04.2008.

2) Disciplina(s) cursada(s) isoladamente e aceita(s) pelo(a) orientador(a) do(a) candidato(a).

Conceito a partir de 02/01/1997:

A - Excelente, com direito a crédito; B - Bom, com direito a crédito; C - Regular, com direito a crédito; R - Reprovado; T - Transferência.

Um(1) crédito equivale a 15 horas de atividade programada.

Última ocorrência: Matrícula de Acompanhamento em 18/02/2013

Impresso em: 13/03/13 10:00:49