UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MINERAÇÃO DE INTERESSES NO PROCESSO DE MODERNIZAÇÃO DIRIGIDA A ARQUITETURA

DANIEL GUSTAVO SAN MARTÍN SANTIBÁÑEZ

ORIENTADOR: PROF. DR. VALTER VIEIRA DE CAMARGO

São Carlos – SP

Agosto/2013

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MINERAÇÃO DE INTERESSES NO PROCESSO DE MODERNIZAÇÃO DIRIGIDA A ARQUITETURA

DANIEL GUSTAVO SAN MARTÍN SANTIBÁÑEZ

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

Orientador: Prof. Dr. Valter Vieira de Camargo

São Carlos – SP

Agosto/2013

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

S235mi

Santibáñez, Daniel Gustavo San Martín.
Mineração de interesses no processo de modernização dirigida a arquitetura / Daniel Gustavo San Martín Santibáñez. -- São Carlos : UFSCar, 2013.
104 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2013.

1. Ciência da computação. 2. Modelos. 3. Knowledge Discovery Metamodel - KDM. 4. Architecture-Driven Modernization-ADM. I. Título.

CDD: 004 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Mineração de Interesses no Processo de Modernização Dirigida a Arquitetura"

Daniel Gustavo San Martín Santibañez

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

Membros da Banca:

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

Prof. Dr. Daniel Lucrédio
(DC/UFSCar)

Prof. Dr. Eduardo Magno Lages Figueiredo
(UFMG)

São Carlos
Agosto/2013

A meus pais.

AGRADECIMENTOS

A realização desta Dissertação de Mestrado só foi possível graças à Deus, pela força espiritual que inspira, protege e ilumina constantemente a minha vida e à colaboração e ao contributo, de forma direta ou indireta, de várias pessoas e instituições, às quais gostaria de exprimir algumas palavras de agradecimento e profundo reconhecimento, em particular:

- *A Jesus Cristo, porque sem ele eu não estaria aqui*;
- *Aos meus amados pais, Gustavo e Patricia por me tornarem quem eu sou, pelo amor, pela dedicação, por todos os sacrifícios para que eu chegasse até aqui, mesmo quando a minha distância os fazia sofrer*;
- *A minha irmã, por me incentivar no caminho do saber*;
- *A minha amada e prometida Irene, cúmplice e companheira de sempre, que com seu amor incondicional me impulsionaram a buscar mais uma realização, mais uma conquista, e por despertar em mim uma admiração crescente e constante*;
- *A meu amigo Victor, por ser meu companheiro dos momentos felizes e infelizes em nossa estada em São Carlos*;
- *A meu orientador Valter, pelo incentivo, confiança e pela parceria nas idéias e discussões, além de aguentar meu português ruim*;
- *A Rafael pela sua contribuição na revisão deste trabalho*;
- *Ao Professor Prado e à CIN por ter disponibilizado o ProgradWeb*;
- *Ao Departamento de Computação da UFSCar, técnico e acadêmico durante minha estada*; *e*
- *A todos aqueles que torceram e acreditaram em mim e neste trabalho*.

Portanto eu lhes digo: não se preocupem com suas próprias vidas, quanto ao que comer ou beber; nem com seus próprios corpos, quanto ao que vestir. Não é a vida mais importante do que a comida, e o corpo mais importante do que a roupa?. Observem as aves do céu: não semeiam nem colhem nem armazenam em celeiros; contudo, o Pai celestial as alimenta. Não têm vocês muito mais valor do que elas?. Quem de vocês, por mais que se preocupe, pode acrescentar uma hora que seja à sua vida?. Por que vocês se preocupam com roupas? Vejam como crescem os lírios do campo. Eles não trabalham nem tecem. Contudo, eu lhes digo que nem Salomão, em todo o seu esplendor, vestiu-se como um deles. Se Deus veste assim a erva do campo, que hoje existe e amanhã é lançada ao fogo, não vestirá muito mais a vocês, homens de pequena fé?. Portanto, não se preocupem, dizendo: "Que vamos comer?" ou "que vamos beber?" ou "que vamos vestir?". Pois os pagãos é que correm atrás dessas coisas; mas o Pai celestial sabe que vocês precisam delas. Busquem, pois, em primeiro lugar o Reino de Deus e a sua justiça, e todas essas coisas lhes serão acrescentadas. Portanto, não se preocupem com o amanhã, pois o amanhã se preocupará consigo mesmo. Basta a cada dia o seu próprio mal.

Mateus 6 : 25 – 34

RESUMO

Sistemas de software são considerados legados quando foram desenvolvidos há muitos anos com tecnologias obsoletas e seu processo de manutenção consome uma quantidade de recursos além da desejada. Uma das causas desses problemas é a modularização inadequada de seus interesses transversais. Quando se encontram nessa situação, uma alternativa é modernizar o sistema para novas linguagens que forneçam melhor suporte à modularização desse tipo de interesse. A ADM (Architecture-Driven Modernization) é uma proposta do OMG para a modernização orientada a modelos de sistemas legados, sendo composta por um conjunto de metamodelos, em que o principal é o KDM (Knowledge Discovery Metamodel), que permite representar todas as particularidades de um sistema. O processo de modernização inicia-se com a engenharia reversa, em que o sistema legado é inteiramente representado em KDM. Depois disso, pode-se aplicar refatorações nesse modelo e gerar o código modernizado. Entretanto, a proposta atual da ADM não inclui suporte para modularizar interesses transversais de um sistema. Isso ocorre porque o primeiro passo desse processo é minerar e encontrar os elementos que contribuem para a implementação de um dado interesse, e isso não é fornecido pela ADM. Nesse sentido, nesta dissertação é apresentada uma abordagem para mineração de interesses no metamodelo KDM, estabelecendo o primeiro passo para um processo de modernização dirigido a interesses. A abordagem de mineração proposta atua com uma combinação de duas técnicas; uma biblioteca de interesses e um algoritmo modificado *K*-means para agrupar strings similares. A abordagem inclui quatro passos onde a entrada é um modelo KDM e o resultado é o mesmo modelo KDM com os interesses anotados e mais alguns arquivos de registro. Além disso, desenvolveu-se um plugin chamado CCKDM para o ambiente Eclipse que implementa a abordagem. Uma avaliação foi realizada envolvendo três sistemas de software. Os resultados da avaliação mostraram que para sistemas que utilizam APIs para implementar seus interesses a técnica desenvolvida é efetiva para a identificação deles, atingindo bons valores de precisão e cobertura.

Palavras-chave: Mineração de Interesses, Clustering, ADM, KDM, Modelos, MDA

ABSTRACT

Software systems are considered legacy when they were developed many years ago with outdated technologies and their maintenance process consumes a large amount of resources. One cause of these problems is the inadequate modularization of its crosscutting concerns. In this situation, an alternative is to modernize the system with a new language to provide better support for concern modularization. ADM (Architecture-Driven Modernization) is an OMG model-driven proposal to modernize legacy systems and consist of a set of metamodels in which the main metamodel is KDM (Knowledge Discovery Metamodel), which allows to represent all the characteristics of a system. The modernization process begins with reverse engineering to represent the legacy system in a KDM model. Thereafter, refactorings can be applied to the model and then generate the modernized code. However, the current proposals do not support crosscutting concerns modularization. This occurs because the first step is to identify the elements which contribute with the implementation of a particular concern and it is not supplied by ADM. In this sense, this dissertation presents an approach for mining crosscutting concerns in KDM models, thus establishing the first step towards to a Concern-Driven modernization. The approach is a combination of two techniques, a concern library and a modified K-means clustering algorithm, which comprises four steps where the input is a KDM model and the result is the same KDM model with annotated concerns and some log files. In addition, we developed an Eclipse plugin called CCKDM to implement the approach. An evaluation was performed involving three software systems. The results show that for systems using APIs to implement their concerns the developed technique is an effective method for identifying them, achieving good values of precision and recall.

Keywords: Concern Mining, Clustering, ADM, KDM, Models, MDA

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS

LISTA DE ALGORITMOS

Capítulo 1

INTRODUÇÃO

1.1 Contexto

Os sistemas de software são considerados "legados" quando possuem elevados custos de manutenção mas ainda são úteis para as organizações. Eles incorporam uma grande quantidade de conhecimento adquirido pelos anos de manutenção e não podem ser substituídos facilmente. Porém, manter esses sistemas não é uma tarefa trivial porque problemas relacionados com sua má estruturação e organização do código fonte, bem como a falta de documentação, resultam em outros como dificuldade de compreensão e evolução (BENNETT; RAJLICH, 2000).

A reengenharia tem sido aceita como uma alternativa para resolver os problemas supracitados. Ela visa realizar modificações profundas nos sistemas, como por exemplo alterar sua plataforma, linguagem de programação ou paradigma em que foram desenvolvidos (CHIKOFSKY; CROSS, 1990; LEITE *et al.*, 1994). Nesse contexto, o OMG propôs a ADM (Architecture-Driven Modernization), como uma alternativa orientada a modelos para a reengenharia tradicional. O objetivo é realizar todo o processo de modernização com apoio de metamodelos padrões. A ADM utiliza a metodologia MDA (*Model-Driven Arquitecture*) (IZQUIERDO; MOLINA, 2010; ULRICH; NEWCOMB, 2010) e abrange duas etapas. Inicialmente, uma engenharia reversa é realizada e um modelo é gerado (PSM (Platform Specific Model)). Então sucessivos refinamentos (transformações) são aplicados nesse modelo inicial até a obtenção de um modelo mais abstrato (PIM (Platform Independent model) ou CIM (Common Information Model)), chamado KDM (*Knowledge Discovery Metamodel*). Sobre esse modelo é possível realizar refatorações, otimizações e modificações para resolver os problemas encontrados no sistema legado. Na segunda etapa, se realiza uma engenharia avante onde o código fonte é gerado novamente com as modernizações correspondentes. O artefato mais importante fornecido pela ADM é o metamodelo KDM o qual é um metamodelo padrão ISO que representa todas

as características existentes em uma arquitetura de software, independentemente da linguagem de programação e plataforma (OMG, 2013). A ideia principal 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 poderia ter o poder de reestruturar um sistema, independentemente da linguagem de programação que foi utilizada em seu desenvolvimento, já que as refatorações ocorrem nos modelos.

Por outro lado, um dos problemas existentes em sistemas legados é a modularização inadequada de interesses transversais. Interesses transversais são aqueles cuja implementação com técnicas tradicionais, como a orientação a objetos, se torna espalhada e entrelaçada pelos módulos do sistema, levando a problemas de manutenção e reuso (KICZALES *et al.*, 1997a; CAMARGO; MASIERO, 2005; MARIN *et al.*, 2006). Nesse sentido, uma possível modernização é modularizar de forma mais adequada os interesses transversais de um sistema legado. Entretanto, para isso o primeiro passo é identificar onde os interesses transversais se encontram no sistema; essa é uma atividade conhecida como mineração de interesses e é uma atividade importante dentro de qualquer processo de modernização porque possibilita a identificação de interesses que estejam implementados de forma espalhada e entrelaçada no sistema. Um ponto a ressaltar é que pouca pesquisa tem sido feita para investigar mineração em artefatos além do código fonte (DURELLI *et al.*, 2013).

A maior parte dos sistemas legados existentes não foram projetados e implementados com técnicas adequadas para modularizar seus interesses transversais. Dessa forma, considera-se bastante satisfatório e provável, a necessidade de se modernizar um sistema legado com o objetivo de modularizar de forma mais adequada seus interesses. Argumenta-se que a elaboração de uma nova abordagem que contemple a identificação de interesses e que esteja inserida no contexto da modernização-orientada a arquitetura, permitirá uma independência das linguagens de programação na procura de interesses, graças as abstrações fornecidas pelo metamodelo KDM.

1.2 Motivação

A principal motivação é que a ADM não contempla uma forma de fazer mineração de interesses no KDM. Embora o metamodelo KDM provê as estruturas para modernizar sistemas identificando elementos de programação, ele não fornece técnicas nem ferramentas para identificar interesses.

O segundo fator motivador foi a criação de uma técnica de mineração que seja independente

da linguagem de programação. Isso é uma vantagem porque uma mesma ferramenta de mineração poderia identificar interesses em sistemas desenvolvidos com diferentes linguagens. Dessa forma, as refatorações aplicadas ao modelo KDM seriam realizadas por algum parser criado para modularizar os interesses identificados. Na literatura encontra-se vários trabalhos em que parsers foram criados no contexto do KDM (DELTOMBE *et al.*, 2012; PéREZ-CASTILLO *et al.*, 2012; MAINETTI *et al.*, 2012).

Outro fator motivador é que grande parte das técnicas de mineração existentes são baseadas em uma única forma ou algoritmo para a identificação dos interesses (MENS *et al.*, 2008). Algumas delas não atingem bons níveis de cobertura e precisão (DURELLI *et al.*, 2013). Assim, o terceiro fator motivador para este projeto foi investigar a combinação de duas técnicas de mineração para melhorar a identificação dos interesses em termos de precisão e cobertura.

1.3 Objetivos

O objetivo é a criação de uma abordagem para viabilizar a realização de modernizações com enfoque em modularização de interesses. Espera-se que essa abordagem sirva de ponto de partida para outros pesquisadores que tenham interesse em estender a ADM ou mesmo utilizar a abordagem aqui proposta. Também espera-se que a ferramenta desenvolvida possa ser usada para apoiar modernizações reais de sistemas.

1.4 Contribuições deste Projeto de Mestrado

Com a realização deste projeto de mestrado, as seguintes contribuições podem ser destacadas:

- Apresentação de uma abordagem para mineração de interesses transversais no contexto ADM. A abordagem possui quatro passos onde dois deles são opcionais. A entrada ao processo é um modelo KDM e as saídas são o mesmo modelo KDM com anotações dos interesses identificados e alguns arquivos de registros (logs). A identificação é realizada por meio de uma biblioteca de interesses em conjunto com o repositório de elementos KDM. A abordagem é explicada de forma que pode ser replicada por outros pesquisadores que tenham interesse em modificá-la e/ou estendê-la.

- Disponibilização de uma ferramenta implementada na forma de um Plug-in para o ambiente Eclipse que dá apoio ao processo de mineração; CCKDM é o nome da ferramenta

desenvolvida para apoiar a abordagem de mineração de interesses para ADM. Utiliza o modelo KDM como fonte de entrada e a saída é o mesmo modelo com os interesses anotados. Essa ferramenta está disponível sob a licença Apache para que outros pesquisadores possam analisá-la e estendê-la.

- Desenvolvimento de um algoritmo de *clustering* que utiliza o valor da distância de *Levenshtein* para agrupar os strings.

- Uma avaliação empírica utilizando dois sistemas bem conhecidos para determinar a eficácia da abordagem proposta, em termos de precisão e cobertura. Os resultados mostram que a técnica de *clustering* atinge bons resultados para certos valores *Levenshtein* e a técnica combinada obtém valores maiores que 90% para a precisão e cobertura.

1.5 Organização da dissertação

Além desse capítulo, de um glossário e anexos, este trabalho é composto pelos seguintes capítulos:

- Capítulo 2: apresenta a fundamentação teórica da modernização dirigida a arquitetura;
- Capítulo 3: apresenta a fundamentação teórica da mineração de interesses e uma revisão sistemática das técnicas de mineração de interesses existentes na literatura até o ano 2012;
- Capítulo 4: apresenta a abordagem criada para a mineração de interesses em um processo ADM;
- Capítulo 5 descreve CCKDM, uma ferramenta que implementa a abordagem apresentada no Capítulo 4;
- Capítulo 6 apresenta a avaliação da abordagem apresentada no Capítulo 4;
- Capítulo 7 apresenta trabalhos relacionados com a mineração de interesses em código fonte e modelos;
- Capítulo 8 apresenta as conclusões, enumera as contribuições e propõe trabalhos futuros.

Capítulo 2

MODERNIZAÇÃO DIRIGIDA À ARQUITETURA

O projeto realizado e apresentado nesta dissertação enquadra-se no contexto de modernização dirigida à arquitetura, pois apresenta-se uma abordagem para mineração de interesses como passo inicial desse processo. Assim, para uma compreensão mais adequada da ADM, neste capítulo discorre-se mais profundamente sobre esse assunto destacando suas principais características, como por exemplo o metamodelo KDM.

2.1 Modernização Dirigida a Arquitetura (ADM)

Em 2003 o OMG iniciou esforços no sentido de padronizar o processo de modernização de sistemas legados utilizando modelos por meio da Architecture-Driven Modernization Task Force (ADMTF) (OMG, 2013).

O objetivo da ADM é revitalizar as aplicações existentes melhorando ou adicionando funcionalidades, aproveitar os padrões existentes de modelagem do OMG e da iniciativa MDA (Model-Driven Architecture) e consolidar as melhores práticas que conduzem à modernização bem sucedida. Em outras palavras, o OMG por meio da ADMTF teve a iniciativa de padronizar os processos da reengenharia de software para aumentar o êxito nos projetos dessa natureza (OMG, 2013). É importante ressaltar que já existiam processos de modernização de sistemas mas não padronizados.

De acordo com o OMG (2013), a ADM resolve os problemas da reengenharia tradicional por meio da utilização dos processos de reengenharia juntamente com os princípios da orientação a modelo.

O fluxo de um processo de modernização apoiado pela ADM pode ser visto na Figura 2.1. Como pode ser observado na Figura 2.1, a ADM possui três fases e é semelhante ao contorno

de uma ferradura, são elas: Engenharia reversa, reestruturação de processes de negócios e engenharia avante. Partindo do lado inferior esquerdo, na parte da engenharia reversa, o conhecimento é extraído do sistema legado e um modelo PSM (*Platform Specific Model*) é gerado. O modelo PSM serve como base para a geração de um modelo PIM (*Platform Independent Model*), que serve como base para a geração do modelo CIM (*Computing Independent Model*). Ou seja, durante a fase de engenharia reversa, transformações são feitas com o intuito de se obter uma representação de alto nível do software, independentemente da plataforma utilizada anteriormente.

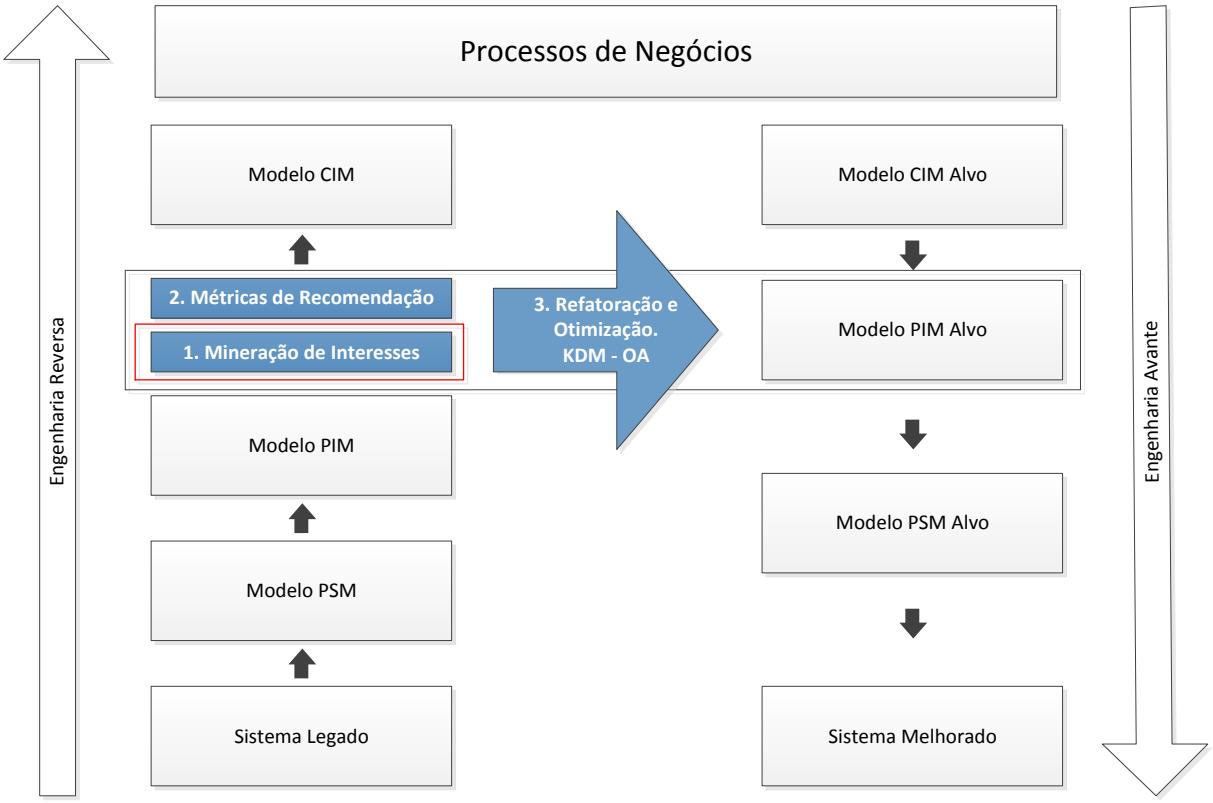


Figura 2.1: Fluxo do processo de modernização apoiado pela ADM (PÉREZ-CASTILLO *et al.*, 2011)

O modelo PSM é um modelo específico de uma plataforma, ou seja, nele existem metadados relacionados a uma implementação em particular. O modelo PIM, representado por modelos KDM, possui um nível de abstração maior, pois é um modelo de representação do sistema independente da implementação. Por último, têm-se o modelo CIM, também representado por modelos KDM, que é o modelo de mais alto nível no processo de modernização, esse modelo representa uma representação do sistema independente de computação.

Na fase de reestruturação, refatorações, melhorias e novas regras de negócios podem ser introduzidas no sistema. O resultado continua sendo uma representação independente de plataforma do software modernizado. Segue-se para a fase de engenharia avante, em que os modelos

são novamente submetidos a uma série de transformações para chegar ao nível de código fonte.

O trabalho apresentado nesta dissertação está focado na mineração de interesses denotado com o número 1 na Figura 2.1. Essa mineração é feita sobre um modelo PIM, onde algumas métricas poderiam ser aplicadas para recomendar as possíveis transformações, e esta denotado com o número 2. Um possível tipo de modularização representado pelo número 3, poderia ser feito utilizando por exemplo, orientação a aspectos.

Segundo Pérez-Castillo *et al.* (2011), Sadovykh *et al.* (2009), as refatorações e melhorias podem ser realizadas nos três níveis de abstração: PSM, PIM e CIM, ou seja, existem cenários em que é interessante realizar um processo de modernização somente a partir de modelos PSM ou PIM, dependendo da necessidade.

Para dar suporte ao processo de modernização, foi criado em 2006 um metamodelo que possibilita a comunicação entre diferentes plataformas e linguagens, e foi chamado pela ADMTF de KDM.

O KDM é um metamodelo de representação intermediária comum para sistemas existentes e seus ambientes operacionais. Utilizando essa representação para sistemas existentes é possível trocar representações do sistema em modelo entre plataformas e linguagens com a finalidade de analisar, padronizar e transformar os sistemas existentes (OMG, 2013).

O KDM consiste de vários modelos arquiteturais do sistema que são definidos 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 extração de conhecimento, de análises e de transformações (NORMANTAS *et al.*, 2012). De acordo com Ulrich e Newcomb (2010) o KDM permite representar um sistema como ele está (*as-is*[1]) a partir das informações que podem ser derivadas de seus artefatos disponíveis.

O desenvolvimento do KDM baseou-se em um número de requisitos chave e suas principais características são:

- Representa os principais artefatos do software usando entidades, relacionamentos e atributos;
- Mapeia artefatos externos com o qual o software interage;

[1] As-is é uma terminologia utilizada pelo OMG para se referir ao sistema legado bruto, ou seja, como ele se encontra antes do processo de modernização.

- Possui um núcleo que gera uma representação independente de plataforma ou linguagem, e é extensível para dar apoio a outras tecnologias;
- Define uma terminologia unificada para a descoberta de conhecimento dos artefatos de software existente;
- Está representado usando diagramas de classe da UML (Unified Modeling Language);
- Utiliza XML Metadata Interchange (XMI) como formato padrão de intercâmbio para desenvolvedores importarem e exportarem de suas ferramentas específicas;
- Dá apoio à representação de todo tipo de plataforma e de linguagem e descreve tanto as estruturas físicas quanto as lógicas de arquiteturas de software;
- Facilita o rastreamento de artefatos que se encontram em diferentes níveis de abstração, partindo de sua estrutura lógica até sua representação física.

O metamodelo KDM representa artefatos físicos e lógicos dos sistemas legados em diferentes níveis de abstração e contém doze pacotes organizados em quatro camadas, são elas: infraestrutura (*infrastructure*), elementos do programa (*program elements*), recursos de tempo de execução (*runtime resources*) e abstração (*abstractions*). Na Figura 2.2 está representada a arquitetura do KDM ilustrando a forma como as camadas se relacionam. As camadas estão organizadas em pacotes que definem um conjunto de elementos do metamodelo e cada camada adiciona um nível de abstração maior com o propósito de representar características específicas de um sistema (ULRICH; NEWCOMB, 2010).

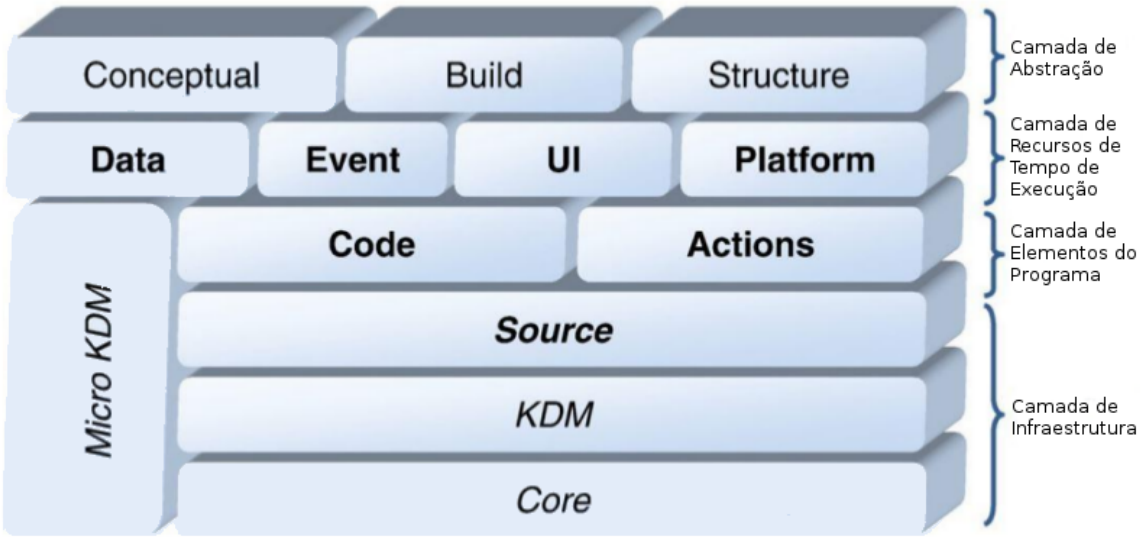


Figura 2.2: Arquitetura do KDM (PÉREZ-CASTILLO *et al.*, 2011)

É importante ressaltar que as contribuições do grupo AdvanSE concentram-se no lado esquerdo da ferradura (engenharia reversa) na Figura 2.1, usando o modelo PIM (metamodelo KDM) como fonte de entrada. Este trabalho se encaixa na mineração de interesses e é o primeiro trabalho desenvolvido pelo grupo no escopo da ADM.

2.2 Camadas do metamodelo KDM

2.2.1 Micro KDM

A especificação KDM define o pacote micro KDM objetivando refinar a semântica definida na ontologia[2]. O pacote micro KDM é principalmente aplicado nos elementos de ação, encontrados no pacote action. Micro KDM é um conjunto de regras de conformidade, orientações adicionais para a construção e interpretação de visões de alta fidelidade KDM, adequados para a realização de análise estática. A Tabela 2.1 apresenta as ações definidas pelo pacote micro KDM (PÉREZ-CASTILLO *et al.*, 2011).

Categoría	**Descrição**	**Micro ações**
Comparison	Representa ações relacionadas a comparações entre expressões de valor.	Equals, NotEquals, LessThanOrEqual, LessThan,GreaterThan, GreaterThanOrEqual, Not, And, Or, Xor.
Numerical datatypes	Representa micro ações relacionados aos tipos de dados.	Add, Multiply, Negate, Subtract, Divide, Remainder, Successor.
Bitwise operation	Representa ações relacionadas às operações bit a bit em tipos de dados primitivos.	BitAnd, BitOr, BitNot, BitXor, LeftShift, RightShift, BitRightShif.
Control	Esta categoria representa micro ações que representam o fluxo de controle.	Assign, Condition, Call, MethodCall, PtrCall, VirtualCall, Return, Nop, Goto, Throw, Incr, Decr, Swith, Compound.
Data Access	Representa ações relacionadas ao acesso a dados.	FieldSelect, FieldReplace, ChoiceSelect, ChoiceReplace, Ptr, PtrSelect, PtrReplace, ArraySelect, ArrayReplace, MemberSelect, MemberReplace, New, NewArray.
Conversions	representa as ações relacionadas às conversões de tipo	Sizeof, Instanceof, DynCast, TypeCast.
String operations	Representa as ações relacionadas a operações do tipo string.	IsEmpty, Head, Tail, Empty, Append.
String type operations	Representa as ações relacionadas para a um conjunto de tipo de operações. O espaço de valor de um conjunto de elementos é o conjunto de todos os subconjuntos do espaço de valor do tipo de dados de elemento.	IsIn, Subset, Difference, Union, Intersection, Select, IsEmpty, Empty.
Set type operations	Representa as ações relacionadas para a um conjunto de tipo de operações. O espaço de valor de um conjunto de elementos é o conjunto de todos os subconjuntos do espaço de valor do tipo de dados de elemento.	IsIn, Subset, Difference, Union, Intersection, Select, IsEmpty, Empty.
Sequence type operations	representa as ações relacionadas às operações do tipo de seqüência. O valor de um elemento de sequência é uma sequência ordenada de valores do tipo de dados.	IsEmpty, Head, Tail, Empty, Append.
Big type operations	representa as ações relacionadas a uma bolsa de operações de tipo. O valor do elemento bolsa é uma coleção de instâncias de valores do tipo de dados do elemento.	IsEmpty, Select, Delete, Empty, Insert, Serialize.

Tabela 2.1: Micro ações definidas pelo pacote micro KDM (ISO, 2012)

[2]Ontologia - Nas Ciências e Tecnologias de Informação, as ontologias são classificações. São usadas como um meio para categorizar ou agrupar as informações em classes.

As micro ações definidas pelo catálogo do pacote micro KDM contribuem com as macro ações apresentadas pelos elementos de ações do pacote *action*. Dessa forma, os fluxos de controle e de dados das representações KDM são mais precisos (PÉREZ-CASTILLO *et al.*, 2011). Cada micro ação consiste em quatro elementos; *(i)* o tipo de ação, especificando a natureza da operação realizada, que é representada por meio do tipo de atributo da macro ação; *(ii)* entradas, demonstrando os relacionamentos de leitura das saídas, que representa os argumentos da micro ação; e, *(iii)* saídas, representando os relacionamentos de escrita de saída, que representa o resultado da ação; *(iv)* a parte de controle, representando o fluxo de controle dos relacionamentos de saída.

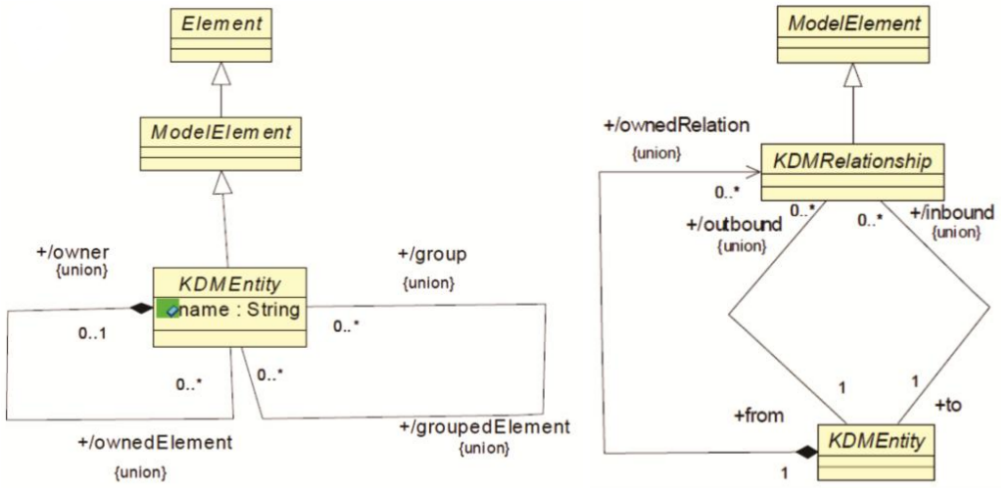
2.2.2 Camada de Infraestrutura

A camada de infraestrutura, que está no nível mais baixo de abstração, define um conjunto de conceitos utilizados ao longo de toda especificação KDM, fornecendo um núcleo comum para todos os outros pacotes. Existem três pacotes nesta camada, o core, o KDM e o source (PÉREZ-CASTILLO *et al.*, 2011).

O pacote core define a estrutura básica para a criação e representação dos elementos de metamodelo em todos os pacotes KDM. Sendo assim determina a estrutura dos modelos KDM e define os padrões e as restrições fundamentais, implementados por todos os outros pacotes KDM (PÉREZ-CASTILLO *et al.*, 2011).

Todo elemento de metamodelo definido em qualquer outro pacote deve ser uma subclasse de uma das classes principais do KDM. As duas classes fundamentais do pacote core são *KDMEntity* e *KDMRelationship* (PÉREZ-CASTILLO *et al.*, 2011). O *KDMEntity* é uma abstração de um elemento de um sistema existente, que possui dados distintos e independentes, podendo ser referenciado como uma unidade. Já o *KDMRelationship* representa uma associação semântica entre os elementos de um sistema existente (ISO, 2012). Para facilitar o entendimento do metamodelo KDM, está representado na Figura 2.3 os diagramas de classes dos principais elementos do pacote core.

Na Figura 2.3(a) são observadas três classes abstratas: a Element, o *ModelElement* e o *KDMEntity*. A Classe *Element* é a classe pai de todos os elementos do metamodelo KDM e, hierarquicamente, é o metaelemento do topo do KDM. A classe *ModelElement* é um elemento que representa alguma qualidade do sistema existente, sendo também a base para todos os meta-elementos do KDM. Com isso, todos os outros meta-elementos são subclasses diretas ou indiretas de *ModelElement*. Já a classe *KDMEntity* representa um artefato do software existente.



(a) *KDMEntities* (b) *KDMRelationship*

Figura 2.3: Diagramas de classes *KDMEntities* e *KDMRelationship* (ISO, 2012)

Um *KDMEntity* pode ser um elemento atômico[3] ou um grupo de elementos, em que o primeiro é um contêiner para *KDMEntities* e o segundo é um grupo de *KDMEntities* (ISO, 2012).

Na Figura 2.3(b) são descritas três classes abstratas: *ModelElement*, *KDMRelationship* e *KDMEntity*. *KDMRelationship* é uma subclasse de *ModelElement*, e é um elemento do modelo que representa uma associação semântica entre duas entidades. Os relacionamentos concretos do KDM entre *KDMEntities*, sob a perspectiva do KDM, são instâncias das subclasses concretas do *KDMRelationship*. Cada instância do *KDMRelationship* tem exatamente um alvo e uma origem, onde as subclasses concretas de *KDMRelationship* define um tipo aceitável dos elementos que ele relaciona. O *KDMEntity*, nesse contexto, recebe uma propriedade adicional que é *ownedRelation* representando simplesmente o ato de ter uma relação. Do ponto de vista de infraestrutura, a associação *ownedRelation* é necessária para gerenciar elementos de relacionamento. Já com relação ao metamodelo, cada relacionamento é uma classe de associação independente, com as propriedades *from-* e *to-*, que armazenam as informações do relacionamento (ISO, 2012).

Outro pacote da camada de infraestrutura é o KDM. O referido pacote também define elementos de metamodelo constituindo a infraestrutura para outros pacotes e definindo os elementos que formam a estrutura de cada representação KDM. Essa estrutura define a estrutura física de uma representação KDM, conhecida como *KDMinstances*, em que cada representação KDM consiste em um ou mais elementos que possuem diversos modelos KDM (PÉREZ-CASTILLO *et al.*, 2011).

[3]Um elemento atômico representa o próprio elemento. Diferente do grupo de elementos que pode representar dois ou mais elementos.

Existem nove tipos de modelos sendo, cada um deles, descrito por um ou mais pacotes KDM correspondendo a um domínio KDM. Da perspectiva arquitetural, cada pacote KDM define uma representação diferente da arquitetura do sistema, evidenciando o modelo KDM como mecanismo chave para organizar artefatos de software legado em representações da arquitetura do sistema. Já no contexto da infraestrutura, um modelo KDM é um contêiner tipado para instâncias de um elemento do metamodelo, ou seja, uma coleção de informações sobre o sistema legado organizada em uma representação da arquitetura do sistema. Da perspectiva do metamodelo, cada modelo KDM é representado por um pacote KDM separado que define uma coleção de elementos de metamodelos, os quais podem ser usados nas representações das informações de um sistema existente em uma representação arquitetural (ISO, 2012).

A estrutura do KDM define um modelo de superclasse comum para todos os modelos, a classe *KDMModel*. A especificação KDM utiliza o termo "KDM model" para referenciar um elemento de um metamodelo correspondente para um tipo de modelo particular, servindo também para referenciar uma instância de tal elemento em uma representação concreta de algum sistema existente (ISO, 2012).

O ultimo pacote da camada de infraestrutura a ser discutido é o *source*. O referido pacote define o primeiro e mais básico modelo do KDM: modelo de inventário. Este modelo enumera os artefatos físicos de um sistema legado (arquivos fonte, imagens, arquivos de configuração, arquivos de recursos e outros) como entidades KDM. O pacote *source* define um mecanismo de rastreabilidade para ligar as entidades KDM a suas representações originais no código-fonte legado, a fim de que a representação KDM seja criada. Outros modelos KDM e outros pacotes *KDM* usam este mecanismo para se referirem a artefatos de software físicos (PÉREZ-CASTILLO *et al.*, 2011).

A ISO/IEC (ISO, 2012) relata que, quanto a sua organização, o pacote *source* depende dos pacotes *core* e *KDM* consistindo em um diagrama com cinco classes: *InventoryModel*, *InventoryInheritances*, *InventoryRelations*, *SourceRef* e *ExtendedInventoryElements*, que podem ser visualizadas na Figura 2.4

O *InventoryModel* tem vários *AbstractInventoryElements* que por sua vez, possui vários *AbstracInventoryRelationships*, que são subclasses de *KDMEntity* e *KDMRelationship* do pacote core. A classe *AbstractInventoryElement* é especializada em concretizar elementos do metamodelo para representar diferentes artefatos físicos, como observado pelas demais classes do diagrama. *AbstractInventoryElement* também é especializada em elementos contêiner tais como diretórios (*Directory*) e projetos (*Projects*) que agrupam outros elementos de inventário (PÉREZ-CASTILLO *et al.*, 2011).

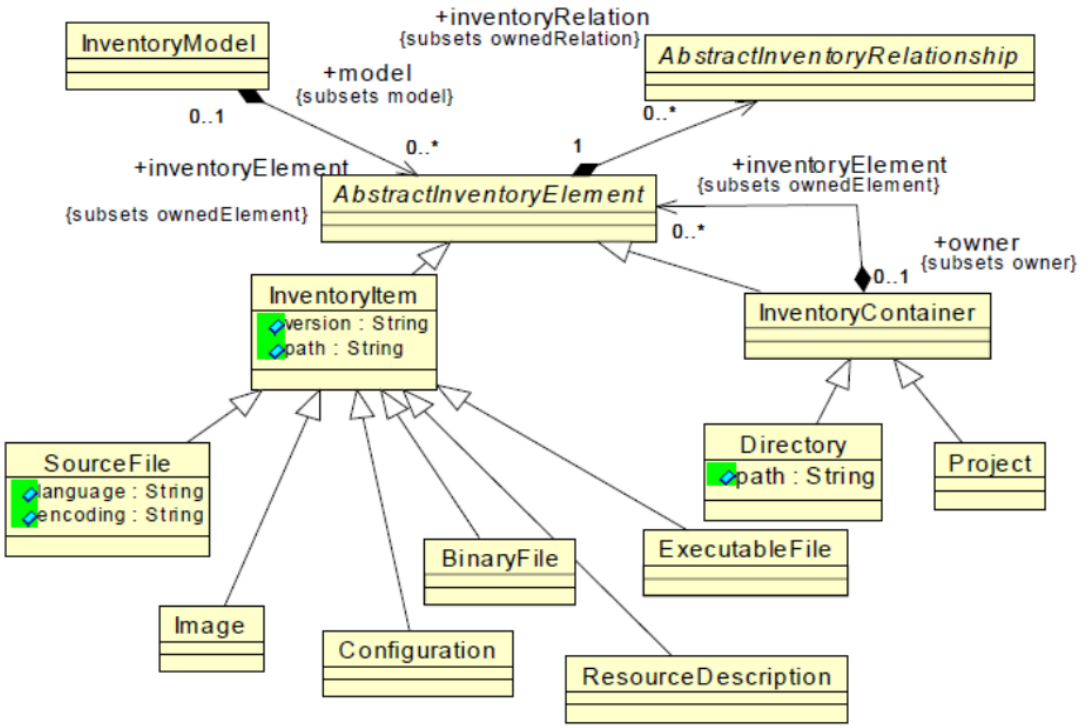


Figura 2.4: Diagrama de classe *InventoryModel* (ISO, 2012)

2.2.3 Camada de Elementos do Programa

A camada de elementos do programa oferece um amplo conjunto de elementos de metamodelo, a fim de fornecer uma representação intermediária independente de linguagem para várias construções definidas por linguagens de programação comuns (BRUNELIERE *et al.*, 2010). Esta camada representa elementos de programa em nível de implementação e suas associações, dessa forma, a camada de elementos do programa representa a visão lógica de um sistema legado. A camada elementos do programa tem dois pacotes, o *code* e o *action*, em que ambos definem um único modelo chamado de *CodeModel* (PÉREZ-CASTILLO *et al.*, 2011).

O primeiro pacote da camada de elementos do programa é o code. Ele define um conjunto de elementos *CodeItem* que representa os elementos comuns nomeados no código-fonte apoiado por diferentes linguagens de programação tais como tipos de dados, classes, procedimentos, métodos, templates e interfaces. O pacote code torna possível a representação estrutural dos relacionamentos entre os elementos do código (PÉREZ-CASTILLO *et al.*, 2011).

Já o pacote *action* é responsável por estender o modelo de code para representar a descrição do relacionamento, do comportamento e dos fluxos de controle e de dados entre os elementos do código. O pacote *action* adiciona dois elementos chave: o *ActionElement* e o *AbstractActionRelationship* (PÉREZ-CASTILLO *et al.*, 2011).

Os elementos do metamodelo *ActionElement* descrevem uma unidade básica de comportamento e representa algumas construções de linguagens de programação tais como declarações, operadores e condições. A funcionalidade característica deste elemento especifica a semântica exata da ação de acordo com um conjunto finito de tipos de ação definidas no Micro KDM (Seção 2.2.1), categorizando as possíveis ações em um sistema legado, fornecendo uma semântica adicional para cada ação. Os *ActionElements* podem também ser ligados à representação original no código legado por meio da classe *SourceRef* do pacote *source* (PÉREZ-CASTILLO *et al.*, 2011).

2.2.4 Camada de Recursos de Tempo de Execução

A camada de recursos de tempo de execução permite a representação do alto valor de conhecimento sobre sistemas legados e seu ambiente operacional, ou seja, foca no que não está contido no código fonte. A camada de recursos de tempo de execução representa os recursos gerenciados pela plataforma de tempo de execução, fornecendo ações de recursos abstratos para gerenciar os demais recursos. Cada pacote desta camada define entidades e containers específicos para representar os recursos do sistema legado e também relacionamentos estruturais específicos entre os recursos (PÉREZ-CASTILLO *et al.*, 2011).

A camada de recursos de tempo de execução estabelece quatro pacotes: *data*, *event*, *UI* e *plataform*. Estes pacotes definem respectivamente quatro modelos KDM (PÉREZ-CASTILLO *et al.*, 2011).

O pacote data define a representação de vários recursos de organização de dados. A representação do pacote *data* está relacionada com as características de como os dados do sistema legado estão persistidos. Os dados da aplicação como variáveis de entrada e saída ou parâmetros em unidades que podem ser chamadas são representados pelo pacote *code*. Este pacote torna possível representar repositórios de dados complexos como banco de dados relacionais, arquivos de registro, esquemas XML e documentos (PÉREZ-CASTILLO *et al.*, 2011).

O pacote *event* define um conjunto de elementos do metamodelo cujo propósito é representar o comportamento de alto nível das aplicações, em particular transições de estados orientados a eventos. O pacote *event* representa dois tipos de estados: os concretos e os abstratos. Os estados concretos são explicitamente apoiados por um estado de máquina de uma estrutura baseada em tempo de execução ou uma linguagem de programação de alto nível. Já o modelo de eventos que representa estados abstratos são associados com um algoritmo em particular, um recurso ou uma interface de usuário (ISO, 2012).

O pacote *UI* oferece um metamodelo para a representação de recursos relacionados aos aspectos de interface do usuário, tais como sua composição, sequencia de operações e seus relacionamentos para um sistema legado. O pacote *UI* também define um modelo específico por meio do elemento *UIModel* com o objetivo de representar, estaticamente, os principais componentes da interface de usuário do sistema legado. O elemento *UIModel* consiste em um conjunto de elementos representado por *UIResources*, que por sua vez, é especializado nos elementos *UIDisplay*, *UIField*, *UIEventm*, entre outros (ISO, 2012).

O pacote plataform define um conjunto de metamodelos, cujo propósito é representar o ambiente operacional de tempo de execução dos softwares existentes (ISO, 2012). Os elementos do modelo plataform descrevem o contexto da execução do código legado, uma vez que o sistema legado não é determinado apenas pela linguagem de programação do código-fonte, mas também pela plataforma de tempo de execução selecionada (ISO, 2012).

O pacote plataform define o elemento *PlataformModel* como uma agregação de *AbstractPlataformElements*, restringindo a elementos específicos como *Process*, *Thread*, *DataManager*, *FileResource*, *StreamResource*, *PlatformEvent*, *NamingResource* e outros. Dessa forma, este modelo constitui apenas um relacionamento de plataforma nomeado BindTo, definindo uma associação entre dois *ResourcesType*. Na Figura 2.5 estão representadas as classes e associações do diagrama de classes *PlataformRelations*, em que a classe *ResourceType* é a fonte do relacionamento e a classe *KDMEntity* é a entidade cujo qual o recurso atual está vinculado (PÉREZ-CASTILLO *et al.*, 2011).

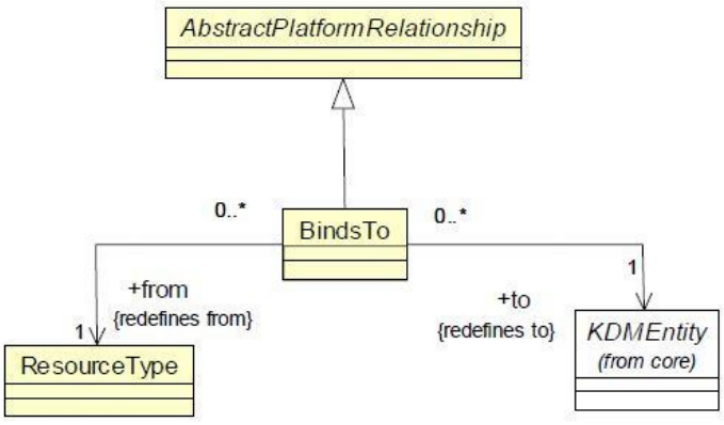


Figura 2.5: Diagrama de classe *PlataformRelations* (ISO, 2012)

2.2.5 Camada de Abstração

A camada de abstração define um conjunto de elementos de metamodelo, os quais representam abstrações de domínios e aplicações específicas, bem como artefatos relacionados ao

processo de construção do sistema existente. Os elementos do metamodelo da camada de abstrações fornecem vários containers e grupos para outros elementos de metamodelos (ISO, 2012). A camada de abstrações define três modelos KDM, são eles: *Structure*, *Conceptual* e *Build*.

O pacote *structure* define elementos de metamodelo que representam componentes de arquitetura de sistemas de software existentes, tais como subsistemas, camadas, pacotes, etc. e define a rastreabilidade desses elementos para outras informações KDM para o mesmo sistema (ISO, 2012). O pacote *structure* define uma representação da arquitetura do sistema para o domínio da estrutura. As representações arquiteturais do sistema baseadas na representação definida pelo modelo de estrutura apresentam como os elementos estruturais do sistema estão relacionados com os módulos definidos nas visões do código, correspondente à representação arquitetural do código definido pelo modelo *Code* (PÉREZ-CASTILLO *et al.*, 2011).

O pacote conceptual representa a informação do domínio específico do sistema legado em um modelo conceitual e pode representar um grafo de comportamento com caminhos da aplicação lógica e as condições associadas. Este grafo de comportamento é construído a partir dos elementos do pacote *action* na camada de elementos do programa, embora os elementos do *action* estejam enriquecidos com semânticas do domínio (PÉREZ-CASTILLO *et al.*, 2011).

Os principais interesses do pacote conceptual estão relacionados: aos termos de domínio implementados pelo sistema, aos comportamentos dos elementos do sistema, às regras de negócio implementadas e aos cenários apoiados pelo sistema (ISO, 2012).

O pacote *build* apresenta os artefatos relacionados com a visão da engenharia do sistema legado quanto aos papéis, desenvolvimento de atividades ou resultados gerados pelo processo de construção. Este pacote define o *BuildModel* com um modelo que representa essa informação, definindo um conjunto de elementos *build* tais como um fornecedor e ferramenta, e fornecendo uma gama de recursos como *BuildProduct*, *BuildComponent*, *BuildStep*, BuildDescription, entre outros (PÉREZ-CASTILLO *et al.*, 2011).

2.3 Considerações Finais

Neste capítulo se apresentaram definições e conceitos relacionados com a modernização de sistemas, bem como os detalhes do metamodelo KDM fornecido pela ADM. Mostrou-se a construção do metamodelo KDM, as camadas e pacotes que formam parte da especificação e suas inter-relações. Especificamente, neste trabalho é utilizado a camada de elementos de programa, os pacotes *Code* e *Actions* para a procura dos interesses transversais.

Capítulo 3

MINERAÇÃO DE INTERESSES

Esse capítulo discorre sobre o tema de mineração de interesses. Apresenta-se o conceito de interesse e sua relação com a fase de manutenção e os problemas que pode ocasionar. Por fim, esse capítulo apresenta uma revisão sistemática realizada que ajudou a direcionar os esforços de pesquisa deste trabalho.

3.1 Interesses em Sistemas de Software

Segundo Dijkstra (1982), um interesse ou *concern* é uma funcionalidade ou requisito (funcional ou não funcional) em um sistema de software e pode ser considerado como uma unidade conceitual passível a ser modularizada. Essa foi possivelmente a primeira publicação científica que trata a questão da separação dos interesses, que tem como objetivo separar as características relevantes de um programa de software para obter uma melhor compreensão do mesmo (READE, 1989). Na literatura podem-se encontrar outras definições, por exemplo, os autores Brito e Moreira (2004) definem um *concern* como sendo uma questão ou assunto de interesse no sistema de software, exemplificando ainda que *concern* pode ser um objetivo ou um conjunto de propriedades que o sistema deve satisfazer. Segundo Tekinerdogan (2003) um *concern* é um subproblema. Para Milli (2004) um *concern* é um conjunto de requisitos relacionados. Para Silva e Leite (2008) um *concern* é uma característica do sistema ou do domínio que seja interessante analisar isoladamente ou em conjunto com outras.

Típicos interesses nos projetos de software são requisitos não funcionais, implementações de padrões de projeto e requisitos funcionais. Para facilitar a evolução do código base, os engenheiros de software geralmente tentam manter os interesses separados quando se está desenvolvendo o sistema. Infelizmente, na prática não é possível separar todos os interesses de um sistema por muitas razões, por exemplo, requisitos mal projetados, limitações das linguagens

de programação, surgimento de novos interesses à medida que o software evolui e a degradação das estruturas de código pelas contínuas mudanças (KICZALES *et al.*, 1997b; LEHMAN *et al.*, 1997). Como resultado, muitos interesses acabam espalhados e entrelaçados com outros no código-fonte. Os interesses espalhados geram um problema para o desenvolvedor porque ele deve procurar por trechos de código que implementam um determinado interesse e isso pode ser uma tarefa intratável quando o sistema é muito grande em termos de linhas de código. Um entendimento incompleto de um interesse pode causar que o desenvolver faça uma modificação incorreta ou ineficiente (SOLOWAY *et al.*, 1988).

Um interesse transversal ou *crosscutting-concern* é definido como um comportamento que atravessa e afeta vários módulos de um sistema (KICZALES; MEZINI, 2005). A programação orientada a objetos (POO) e a programação procedural não proporcionam alguns mecanismos para capturar decisões de projeto que o programa deveria implementar. Isso traz como consequência tomar decisões de projeto pouco efetivas relacionadas com a implementação de certos interesses. O resultado final será código espalhado e entrelaçado onde os interesses são difíceis de se identificar gerando dificuldades na hora de desenvolver ou manter o software. A Figura 3.1 representa um sistema com seus interesses espalhados e entrelaçados entre si. A primeira etapa consiste na identificação desses interesses e a segunda etapa em refatorá-los.

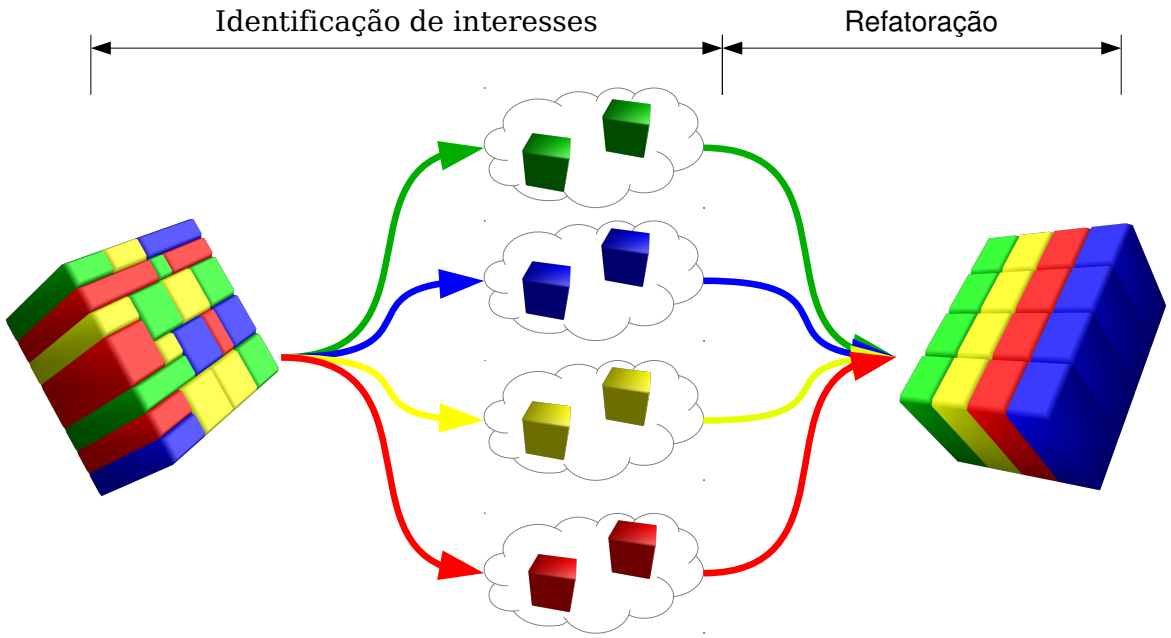


Figura 3.1: Interesses transversais espalhados e entrelaçados

Alguns exemplos tradicionais de interesses transversais são: persistência, segurança, auditoria e tratamento de exceções.

3.1.1 Programação Orientada a Aspectos

O paradigma da programação orientada a aspectos (POA) foi proposta com a intenção de modularizar as implementações dos interesses transversais que se encontram espalhados e entrelaçados no código fonte (KICZALES *et al.*, 1997b). Essas novas modularizações são os *aspectos* de um sistema de software que afetam outros interesses, ou seja, são interesses que são transversais a outros interesses.

Assim como a POO não descarta as idéias de dados e operações do paradigma estrutural, a POA não rejeita a tecnologia existente. O paradigma orientado a aspectos complementa o paradigma orientado a objetos a fim de auxiliar na separação daquelas preocupações que a POO manipula deselegantemente (não possuindo recursos para fazer tal tarefa, necessitando de outras formas que tornam o código mais difícil de entender e manutenir). Para isso, a POA provê mecanismos para localizar e compor os interesses transversais com as unidades que eles afetam de maneira não invasiva, ou seja, evitando que fiquem explicitamente espalhados nos artefatos de software ou misturados com as preocupações específicas desses artefatos

Existem várias implementações de linguagens relacionadas com POA. A mais popular é o AspectJ, que funciona com uma extensão à linguagem POO Java, adicionando-lhe conceitos práticos de POA e permitindo a implementação modular dos interesses transversais (KICZALES *et al.*, 2001). Acrescentando novos conceitos, o join point e alguns construtores: pointcuts, advice afetam dinamicamente o fluxo normal do programa e as declarações inter-type que alteram estaticamente a hierarquia de classes do programa e aspectos que encapsulam todos os novos construtores.

Como benefícios da utilização do Paradigma Orientado a Aspectos pode-se enumerar:

1. Melhora a compreensão: a separação modular dos interesses permite efetuar uma análise e leitura do código escrito para que o software possa evoluir com maior facilidade. A adição de módulos ou redefinição de regras em um sistema de software é simplificada e as alterações mais facilmente localizáveis;

2. Facilidade na adição de novos requisitos: a separação dos interesses permite a alteração de código fonte, com o mínimo de repercussões ao nível da estrutura do sistema de software. Modificações, para melhorar a performance ou adicionar/alterar regras de negócio, são mais facilmente acopladas ao sistema;

3. Reúso de código: a eliminação do efeito de intrusão e espalhamento permite que os módulos sejam mais facilmente acoplados a outros sistemas de software;

3.1.2 Mineração de Interesses

Os engenheiros mantenedores de software frequentemente devem recuperar a rastreabilidade dos requisitos no código-fonte legado. Em um programa bem projetado com poucas manutenções, cada funcionalidade deveria ser facilmente localizada no código fonte. Porém, em sistemas legados existem mais probabilidades de ter as funcionalidades espalhadas em vários módulos (ROBILLARD; MURPHY, 2007). Isso ocorre pois os engenheiros mantenedores geralmente não são os mesmos que projetaram os sistemas, então eles tendem a identificar e interpretar erroneamente as funcionalidades procuradas no código (SOLOWAY *et al.*, 1988).

Para resolver os problemas apresentados, foram desenvolvidas técnicas de mineração de código-fonte ou também chamadas de técnicas de mineração de interesses. A utilização dessas técnicas é o passo inicial para identificar os interesses que possivelmente serão modularizados, por meio de ferramentas automáticas ou semi-automáticas.

Segundo Demeyer *et al.* (2002), existem duas abordagens para o estudo do código-fonte; *bottom-up* e *top-down*. A abordagem *top-down* primeiramente analisa as representações de mais alto nível (ex., documentos de requisitos, projeto, etc) e verificar se são correspondente com as de mais baixo nível (ex., código-fonte). Diferentemente, a abordagem *bottom-up* analisa as representações de baixo nível, essas representações são filtradas utilizando algum critério e são transformadas em representações de alto nível.

Na literatura é possível encontrar várias técnicas de mineração de código-fonte, as quais possuem diferentes abordagens de implementações. Por exemplo Khatoon *et al.* (2011) classifica as técnicas de mineração em três tipos; Baseadas em mineração de regras, baseadas em detecção de clones e pelo uso de certas APIs. As técnicas baseadas em mineração de regras, utilizam uma análise estática do código-fonte para identificar certas regras referentes a um padrão para poder detectar violações dessas regras. As técnicas baseadas em detecção de clones tentam identificar trechos de código que são semelhantes sintaticamente no programa ou têm pequenas variações. As técnicas baseadas no uso de APIs tentam identificar certos interesses no código-fonte dependendo do uso de certas bibliotecas de programação.

Khatoon *et al.* (2011) ressalta que não existe uma técnica superior a outra, uma vez que cada técnica possui capacidades e limitações específicas, para atender diferentes contextos. Porém, a combinação dessas técnicas pode permitir a identificação de vários tipos de interesse e assim, obter melhores resultados quanto ao entendimento dos mesmos.

Na literatura também é possível encontrar outras classificações das técnicas de mineração. Hassan e Xie (2010) classificam as técnicas mineração de código fonte em três tipos; *(i)*

análise de tipos e texto; *(ii)* análise por grafos; e *(iii)* análise de sequência ou exploratório. Na análise por grafos encontra-se a ferramenta *FINT* (MARIN *et al.*, 2007a). Considerando a análise exploratória pode-se identificar a ferramenta *FEAT* (ROBILLARD; MURPHY, 2003). No análise textual encontra-se a ferramenta *ComSCID* (JUNIOR *et al.*, 2012).

3.2 Revisão Sistemática

Nesta seção, é apresentada a revisão sistemática sobre técnicas de mineração de interesses transversais. Essa revisão foi proposta com o intuito de ter uma visão do estado da arte das pesquisas na área de mineração de interesses até o começo do ano 2012, sendo o ponto inicial para desenvolver este trabalho. Segue-se os passos propostos por Kitchenham *et al.* (2009) para a produção de uma revisão imparcial e reproduzível: *(i)* planejamento da revisão, *(ii)* condução da revisão e *(iii)* sintetização de dados e descrição da revisão.

3.2.1 Planejamento

Nesta primeira fase de planejamento, definiu-se o Protocolo de Revisão, informando o propósito e os objetivos da revisão sistemática; as fontes, estratégias e questões de pesquisa; os critérios e procedimentos de seleção dos resultados; e a forma de extração dos dados destes resultados. Por meio deste protocolo, fica descrito todo o processo de revisão, permitindo assim, que outros pesquisadores repitam esta mesma pesquisa e obtenham resultados similares.

3.2.1.1 Objetivos

Para essa revisão sistemática foram definidos os seguintes objetivos:

I. Obter as técnicas de mineração mais nomeadas na literatura até o ano 2012.

II. Estender a taxonomia proposta por Kellens *et al.* (2007).

III. Recomendar possíveis combinações de técnicas de mineração.

3.2.1.2 Questões de Pesquisa

A partir dos objetivos estabelecidos anteriormente, foram elaboradas 5 questões de pesquisa:

Q1. Quais são as técnicas de mineração encontradas na literatura?

Q2. Quais técnicas de avaliação foram empregadas para avaliar as técnicas de mineração de interesses e quais são os resultados obtidos por essas técnicas na procura dos interesses mais comuns?

Q3. Existe alguma diferença nas métricas de precisão e cobertura quando são usadas diferentes técnicas na mineração de um mesmo interesse?

Q4. Para um conjunto de interesses, qual é a técnica mais adequada para a mineração deles?

Q5. Como se poderiam combinar as técnicas para melhorar as métricas de precisão e cobertura?

Existem ainda, como citado em (KITCHENHAM *et al.*, 2009), alguns outros itens relacionados ao escopo e especificidades das questões de pesquisa que merecem destaque:

- **Intervenção**: Artigos científicos publicados relacionados com mineração de interesses transversais.
- **Controle**: Jornais e artigos levantados que abordam o tema de mineração de interesses transversais em código fonte.
- **População**: Estudos e pesquisas abordando a mineração de interesses transversais.
- **Resultados**: Dificuldades, problemas, boas práticas, ferramentas, processos e metodologias específicas à mineração de interesses.
- **Aplicação**: Estudos posteriores sobre este mesmo tema, bem como ferramentas, processos e metodologias específicas para a melhora da procura de interesses.

3.2.1.3 Estratégias de Busca

Esta seção descreve onde e como foram realizadas as buscas, bem como as palavras-chave que geraram a string de busca para os resultados selecionados.

- Fontes: Foram selecionadas como fontes de pesquisa deste trabalho bases de dados indexados, máquinas de busca eletrônica e bibliotecas digitais.
  - *ACM* (portal.acm.org)

- *IEEE* (ieeexplore.ieee.org)
- *Scopus* (scopus.com)
- *Springer* (springer.com/lncs)

- Palavras-Chave: Baseando-se nas questões de pesquisa levantadas anteriormente, foram definidas as seguintes palavras-chave: *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*.

- Strings de busca: Visando abordar todas as palavras-chave anteriores e eliminando o máximo possível de resultados não desejados a Figura 3.2 apresenta a string de busca utilizada a qual é uma combinação de várias palavras-chave unidos por operadores AND e OR.

*(("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"))*

Figura 3.2: String de Busca

Estas strings poderão ser adaptadas de acordo com o mecanismo de busca de cada Fonte, porém sempre de forma que não altere o seu sentido lógico.

- Línguas: Inglês, por ser a língua padrão para publicações internacionais.

3.2.1.4 Critérios de Seleção e Extração dos Dados

Depois de obtidos os resultados das buscas, estes foram analisados quanto à sua relevância, de forma a determinar se deveriam ser considerados como estudos primários ou desconsiderados. Esta relevância foi analisada de acordo com os seguintes critérios de inclusão e exclusão:

- Critérios de Inclusão:
  - O estudo primário apresenta pelo menos uma técnica de mineração de interesses: As técnicas devem apoiar aos engenheiros de software na mineração de interesses transversais.
  - O estudo primário apresenta pelo menos uma técnica para avaliar as técnicas de mineração de interesses: Sem técnicas de avaliação não é possível realizar comparações das técnicas.

- Critérios de Exclusão:

  - O estudo primário apresenta uma técnica de mineração de dados. Porém, a técnica é aplicada em base de dados e não na mineração de interesses transversais.
  - O estudo primário é um artigo com menos de três páginas.

Dos estudos primários foram extraídas as seguintes informações: *(i)* nome das técnicas identificadas, *(ii)* data da extração dos dados, *(iii)* título, autores, revista, detalhes da publicação e *(iv)* uma lista de sentencias com o intuito de responder as questões.

3.2.2 Validação

A validação foi realizada por meio da técnica *Visual Text Mining* (VTM) (MALHEIROS *et al.*, 2007). Essa técnica permite agrupar estudos primários que estão fortemente correlacionados. A ferramenta *Projection Explorer* (PEx) visualiza gráficamente os grupos formados pelos estudos primários e mostra o grau de relacionamento entre os distintos estudos de um mesmo grupo. Dessa forma, lendo um artigo de um grupo homogêneo e for excluído, então todos os artigos desse grupo serão excluídos. Pelo contrario, lendo um artigo de um grupo homogêneo e for incluído, então todos os artigos desse grupo serão excluídos. Para grupos que não são homogêneos e estudos que não foram agrupados, deve-se seguir o método tradicional de validação, ou seja, ler todos esses estudos.

3.2.3 Condução

Nesta fase, primeiro foram identificados os estudos primários das bibliotecas digitais. A Scopus foi a biblioteca que mais estudos primários retornou com um total de 262 artigos, a IEEE com 215, a ACM com 202 e a Springer com 127 dando um total de 802 estudos primários. Depois, levando em conta o título, o resumo e as respostas dos critérios de inclusão e exclusão se excluíram um total de 678 estudos primários deixando 124. Finalmente, foram lidos os 124 estudos dos quais 62 foram excluídos ficando um total de 62 estudos.

Dos 62 estudos primários, foram identificados 18 técnicas de mineração de interesses transversais. Dessa forma, cada estudo primário pode-se encaixar em uma o mais técnicas. A seguir descreve-se cada uma das técnicas identificadas:

- *Execution Patterns (EP)*: Essa técnica analisa traços do programa em tempo de execução para procurar certos padrões. Com os padrões identificados, o algoritmo identifica possíveis interesses nas invocações a métodos (BREU; KRINKE, 2004);

- *Dynamic Analysis (DA)*: Essa técnica utiliza o método de analises de dados *Formal Concept Analysis* (FCA) para a identificação de interesses. Uma versão do sistema é executada e direcionada por vários casos de usos. Os traços de execução são analisados por meio do FCA para a identificação de métodos e classes. Métodos pertencentes a mais de uma classe pode indicar presença de interesses espalhados. Se métodos de uma mesma classe estão especificados em mais de um caso de uso então pode indicar presença de interesses entrelaçados com outros interesses (CECCATO, 2008);

- *Identifier Analysis (IA)*: A técnica utiliza um algoritmo FCA e assume que o código fonte segue uma convenção de nomes para métodos e classes. Por meio da relação entre nomes de métodos e classes e os substrings desses métodos e classes o algoritmo tenta identificar padrões e assim interesses (TOURWE; MENS, 2004);

- *Language Clues (LC)*: A abordagem utiliza uma técnica de processamento de linguagem natural para a mineração de interesses. A entrada do algoritmo é uma coleção de palavras do código fonte e a saída são cadeias de palavras semanticamente relacionadas. O algoritmo é aplicado nos comentários, métodos, classes e atributos do código fonte. Uma inspeção manual é feita sobre o resultado do algoritmo para identificar possíveis interesses (SHEPHERD *et al.*, 2005b);

- *Method Clustering (MC)*: Essa técnica agrupa métodos similares (similaridade das cadeias de strings) em distintos conjuntos de forma recursiva. A premissa é que métodos com nomes similares implementam mesmos interesses (BERNARDI; Di Lucca, 2010);

- *Call Clustering (CC)*: A técnica assume que se mesmos métodos são chamados frequentemente de diferentes classes, então esses métodos estão fortemente relacionados e devem ser agrupados (ZHANG *et al.*, 2008a);

- *Fan-In analysis (FI)*: A técnica observa todas as chamadas dos métodos. Por meio da métrica de fan-in, a técnica calcula a quantidade de chamadas dos métodos do sistema. Valores altos de fan-in podem indicar presença de interesses transversais (ZHANG *et al.*, 2008a);

- *AST-Based Clone Detection (ACD)*: A entrada dessa técnica é a arvore sintática (AST) do código fonte e a saída são grupos de fragmentos de código considerados clones uns com outros. O código clonado pode indicar presença de interesses transversais (BRUNTINK *et al.*, 2005);

- *Token-Based Clone Detection (TCD)*: Essa técnica é baseada na analise léxica do código fonte. A saída da técnica são grupos de fragmentos de código considerados clones uns dos outros (BRUNTINK *et al.*, 2005);

- *History Based (HB)*: Essa técnica analisa as mudanças ocorridas no código fonte ao longo do tempo por meio de sistemas de controle de versão, arquivos ou banco de dados (MULDER; ZAIDMAN, 2010);

- *Information Retrieval (IR)*: Essa técnica é baseada na similaridade de termos nos elementos de um programa. Os resultados são classificados e é realizada uma inspeção manual para identificar interesses (EADDY *et al.*, 2008);

- *Parser-Based (PB)*: Essa técnica realiza uma análise léxica e sintática do código fonte para identificar interesses transversais. Baseia-se na premissa em que fragmentos de código que compartilham algum interesse podem ser identificados por alguma característica, por exemplo alguma biblioteca ou algum tipo de identificador (GRISWOLD *et al.*, 1999);

- *PrefixSpan (PS)*: É uma técnica que vem da mineração de dados e foi aproveitada para identificar interesses no código fonte. Cada método do código é transformado em uma sequência de chamadas a métodos e elementos de controle. O algoritmo procura por sequências repetitivas que poderiam ser um padrão (ISHIO *et al.*, 2008);

- *Concern-Peers (CP)*: Essa técnica visa identificar unidades de código que provavelmente compartilham interesses transversais e suas recomendações para a criação ou atualização de aspectos. Essas unidades de código são chamadas *concern peers* e são detectadas por meio das interações similares que possam ter. Essas interações podem ser internas (dentro do corpo do método) ou externas (chamadas para métodos de outras classes) que estejam relacionadas em contextos similares. A proposta está sustentada em definições matemáticas baseadas na teoria de conjuntos (NGUYEN *et al.*, 2011);

- *Method Call Tree (MCT)*: Essa técnica utiliza uma arvore para representar traços de código relacionados com chamadas a métodos. Os traços são analisados para procurar certos padrões e assim interesses transversais (QU; LIU, 2007);

- *Data-Flow Concern Identification (DF)*: É uma técnica semi-automática para a identificação de interesses. A identificação é feita analisando um conjunto de variáveis relacionadas e usa informações do fluxo de dados estática para determinar indícios de interesses (TRIFU, 2008).

- *Random Walks (RW)*: É uma formalização matemática que consiste em tomar sucessivos passos aleatórios. Essa técnica realiza passeios aleatórios sobre grafos que representam o código fonte. Após disso os caminhos são comparados e filtrados para a busca de certos padrões. Esses padrões podem conter interesses transversais candidatos (ZHANG; JACOBSEN, 2011).

- *Model-Driven (MD)*: Essa técnica identifica interesses candidatos por meio de um metamodelo criado pelos autores chamado **concern miner**. Esse metamodelo transforma o código fonte em um modelo orientado a interesses para facilitar a identificação dos interesses (NORA; GHOUL, 2006).

A taxonomia proposta por Kellens *et al.* (2007), leva em conta 3 dimensões: *(i)* Analise estática ou dinâmica; se a técnica realiza uma analise estática do código fonte ou uma analise dinâmica por meio de traços de execução do programa ou ambas. *(ii)* Analise de tokens ou analise estrutural ou de comportamento; analise léxica de sequencias de caracteres, expressões regulares, arvores sintáticas, o tipo de informação, massagens enviadas, etc. *(iii)* Granularidade; o nível de granularidade da técnica, no nível de métodos ou uma granularidade mais fina.

Nesse contexto, a taxonomia proposta por Kellens *et al.* (2007) foi estendida adicionando as novas técnicas identificadas. A Figura 3.3 apresenta a taxonomia estendida. Os retângulos no meio da figura representam todas as técnicas: as propostas por Kellens *et al.* (2007) e as novas identificadas pela revisão sistemática marcadas com um asterisco. Das 18 técnicas identificadas, 7 são novas e adicionadas na taxonomia. Essas novas técnicas são: PrefixSpan, Information Retrieval, Dataflow, Model-Driven, Random Walks, History-Based, Concern Peers. Além disso, pelo fato de adicionar as novas técnicas, foram adicionados 4 algoritmos: Vector Space Indexing, Frecuent Itemset, Concern Model, Peer Detection. Finalmente, um novo nível de granularidade foi adicionado chamado *file-level* por causa da técnica *History Based (HB)*.

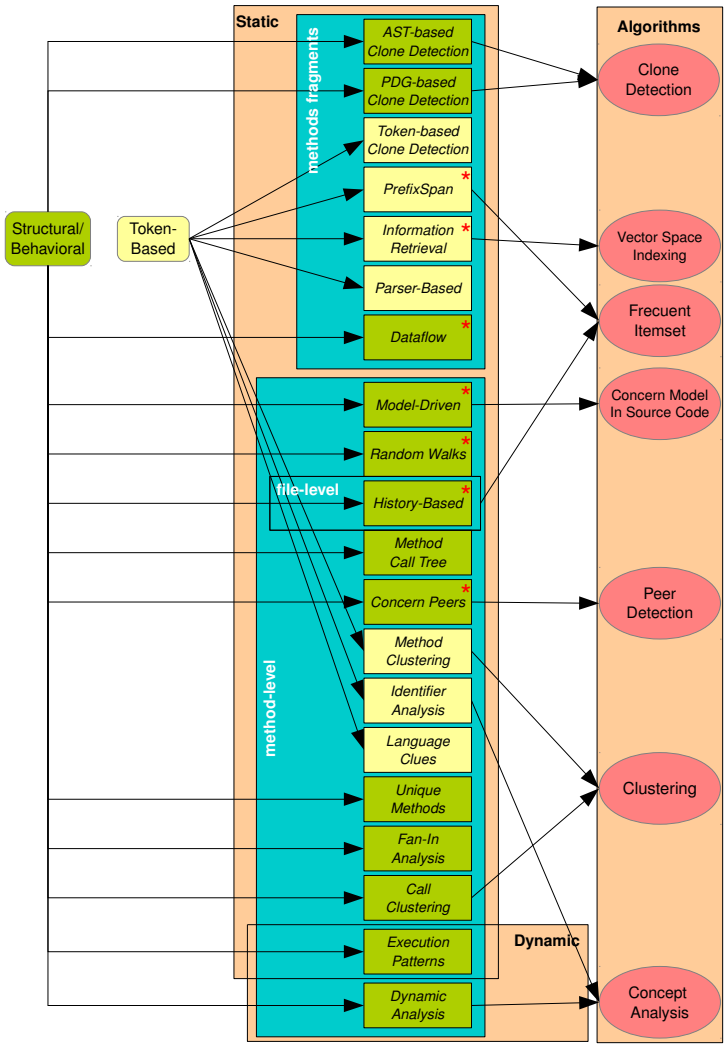


Figura 3.3: A taxonomia estendida (Adaptada de Kellens *et al.* (2007))

3.2.4 Análise dos Resultados

Nesta fase da revisão, os trabalhos colhidos foram analisados de acordo com as questões que estes contemplaram. Desta forma, as questões de pesquisa serão novamente retomadas e agora respondidas de acordo com o resultado extraído da análise dos artigos.

A Figura 3.4 apresenta um gráfico de bolhas onde o eixo *X* representa as técnicas de mineração identificadas e o eixo *Y* representa os anos. A intersecção entre os dois eixos é indicado por uma bolha a qual mostra a quantidade de estudos primários referentes a uma determinada técnica e um determinado ano. Baseados nessa figura, responde-se a questão Q1.

Q1. Quais são as técnicas de mineração encontradas na literatura?
R: Foram obtidas 18 técnicas de mineração de interesses, e as técnicas mais estudadas são; *Fan-In Analysis*, *Identifier Analysis* e *Dynamic Analysis*. As técnicas menos estuda-

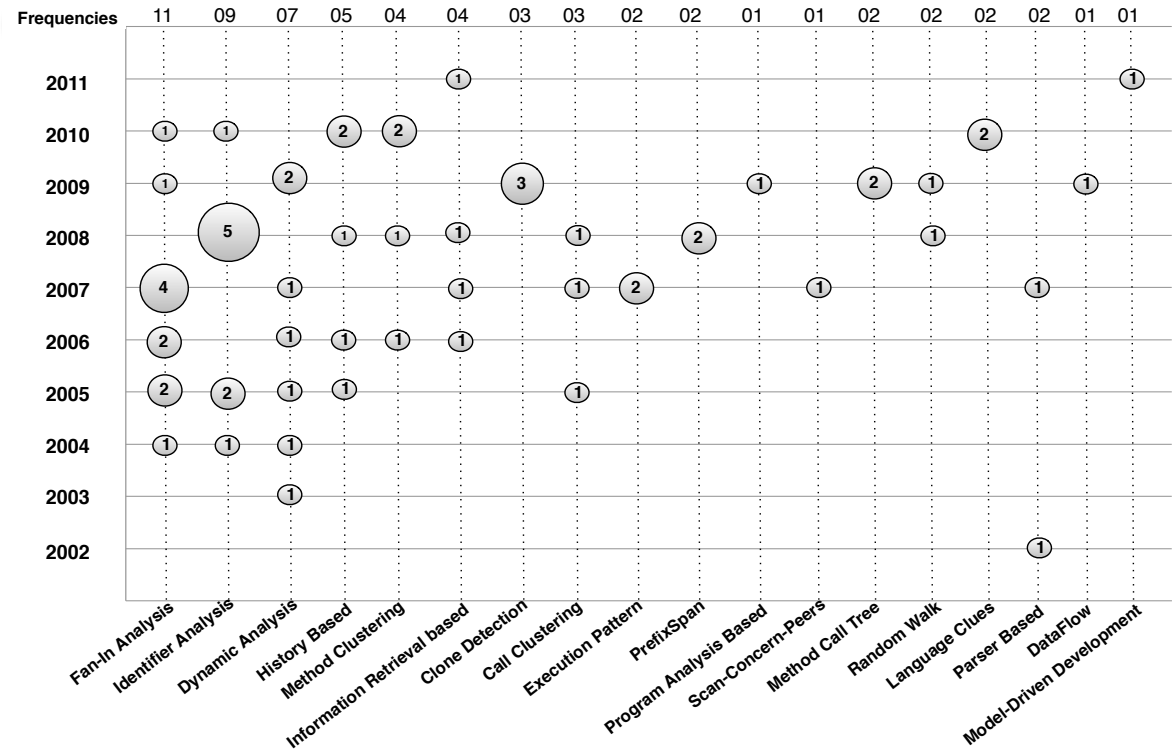


Figura 3.4: Mapa das publicações distribuídas por ano com as técnicas de mineração.

das são; *Program Analysis Based*, *XScan-Concern-Peers*, *Data-Flow* e *Model Driven*.

Segundo Wohlin *et al.* (2000), as estratégias empíricas na engenharia de software são classificadas como; experimento (*Experiment*), casos de estudo (*Case Study*) ou nenhum (*none*). Nesse contexto, para responder a questão Q2 foram analisados individualmente os 62 estudos primários. A Figura 3.5 apresenta um gráfico de pizza onde se mostram os dados coletados.

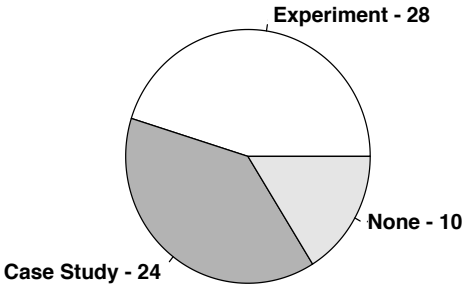


Figura 3.5: Quantidade de estratégias empíricas utilizadas

Q2. Quais técnicas de avaliação foram empregadas para avaliar as técnicas de mineração de interesses e quais são os resultados obtidos por essas técnicas na procura dos interesses mais comuns?

R: Dos 62 estudos primários, 52 tem algum tipo de avaliação. 28 estudos foram avaliados por meio de experimentos e 24 estudos usaram algum tipo de estudo de caso. 10 estudos não apresentam avaliação empírica. Além disso, dos 52 estudos que apresentam algum tipo de avaliação empírica, 31 deles utilizaram JHotDraw (BRANT, 1995), um framework desenvolvido em Java comumente utilizado para avaliar técnicas de mineração de interesses. Finalmente, dos 31 estudos que usaram JHotDraw, 6 deles usaram métricas bem conhecidas para avaliar técnicas na área de mineração de interesses; precisão e cobertura.

Para responder a questão Q3 foi criada a Tabela 3.1 que apresenta as técnicas e ferramentas dos 6 estudos que usaram as métricas de precisão e cobertura em distintas versões de JHotDraw (MARIN *et al.*, 2007b).

Q3. Existe alguma diferença nas métricas de precisão e cobertura quando são usadas diferentes técnicas na mineração de um mesmo interesse?
R: Com os valores da Tabela 3.1, pode-se verificar que a Persistência apresenta porcentagens similares e altos para a métrica de precisão na maioria dos casos. Porém, a métrica de cobertura apresenta valores diferentes na maioria dos casos. Isso pode indicar que o universo de interesses candidatos utilizado por cada técnicas não é padronizado então, existem diferencias nas métricas de precisão e cobertura quando diferentes tipos de interesses são minerados.

Q4. Para um conjunto de interesses, qual é a técnica mais adequada para a mineração deles?
R: Segundo o apresentado pela Tabela 3.1 e levando em conta o interesse de Persistencia, as técnicas que possuem melhores valores de precisão e cobertura são XScan e CBFA.

Q5. Como se poderiam combinar as técnicas para melhorar as métricas de precisão e cobertura?
R: Baseada na Tabela 3.1, foram criadas a Tabela 3.2 e a Tabela 3.3 as quais apresentam as combinações recomendadas para melhorar a mineração de interesses baseada nas métricas de precisão e cobertura para os interesses de Persistência e Observador. Por exemplo, para Persistência uma boa recomendação seria combinar XScan com CBFA porque XScan tem um bom valor para precisão e CBFA um bom valor para cobertura pelo qual poderiam se complementar. O mesmo caso ocorre com Observador, onde MSAM poderia-se combinar com CBFA para atingir melhores valores nas métricas de precisão e cobertura.

Dynamic Analysis					
Ref.	Ferramenta	Precisão	Cobertura	Interesse	Versão
(CECCATO; TONELLA, 2009)	Dynamo	64% 100%	49% 26%	Undo Persistence	5.4
Concern Peers					
Ref.	Ferramenta	Precisão	Cobertura	Interesse	Versão
(NGUYEN *et al.*, 2011)	XScan	90% 100% 100% 97% 100%	93% 89% 93% 100% 100%	Undo Iterator Persistence Observer Visitor	6
Method Clustering + Fan-In					
Ref.	Ferramenta	Precisão	Cobertura	Interesse	Versão
(NGUYEN *et al.*, 2011)	CBFA	100% 100% 80% 86% 86%	86% 100% 100% 80% 100%	Undo Iterator Persistence Observer Visitor	6
Information Retrieval					
Ref.	Ferramenta	Precisão	Cobertura	Interesse	Versão
(HUANG *et al.*, 2010)	MSAM	5%	100%	Undo Persistence Observer Command	-
Method Clustering					
Ref.	Ferramenta	Precisão	Cobertura	Interesse	Versão
(COJOCAR; CZIBULA, 2008)	Clustering Algorithms	87.5%	-	Observer *Consistent Behaviour* *Contract Enforcement* Command	5.2
Call Clustering					
Ref.	Ferramenta	Precisão	Cobertura	Interesse	Versão
(MAISIKELI, 2010)	SOM	51%	-	Observer *Consistent Behaviour* *Contract Enforcement* Command	5.4
Fan-In					
Ref.	Ferramenta	Precisão	Cobertura	Interesse	Versão
(MARIN *et al.*, 2006)	FINT	30%	-	*Consistent Behavior* *Contract Enforcement*	5.4

Tabela 3.1: Precisão e Cobertura para JHotDraw

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

Tabela 3.2: Combinação para Persistência

N	Técnicas Combinadas
1st	MSAM + CBFA
2nd	MSAM + Clustering Algorithms

Tabela 3.3: Combinação para Observador

Se observa que a técnica mais combinada é a técnica de *clustering* pela sua natureza adaptável, dependendo das necessidades que exige o problema. *Clustering* é utilizada em distintos cenários para a procura dos interesses, sendo bem adaptável aos problemas que aborda quando é utilizada na área de mineração de interesses (TRIBBEY; MITROPOULOS, 2012; MCFADDEN; MITROPOULOS, 2012; SERBAN; MOLDOVAN, 2006). Na literatura existem várias formas de detectar e analisar *clusters*. Por exemplo Everitt *et al.* (2009) apresenta-se uma breve

classificação das técnicas de *clustering*; *(i)* Detecção de *clusters* gráficamente; *(ii)* Detecção de *clusters* por medição de proximidade; *(iii)* Agrupamento hierárquico e *(iv)* Mistura finita de densidades.

3.2.5 Conclusões

A revisão sistemática revela que as técnicas de mineração de interesses mais mencionadas são; *Fan-In Analysis*, *Identifier Analysis* e *Dynamic Analysis*. Em contraste, *Program Analysis Based*, *XScan-Concern-Peers*, *Data-Flow* e *Model Driven* podem ser considerados como técnicas "pouco investigadas". Outro ponto em destaque é que a taxonomia estendida com 7 novas técnicas pode servir como um roteiro para pesquisadores na área de mineração de interesses. Nesse sentido, a revisão ajudou a direcionar os esforços de pesquisa deste trabalho os quais serão detalhados no Capítulo 4 e Capítulo 5.

Os trabalhos futuros a serem realizados devem ser encaminhados na realização de novas avaliações comparativas das técnicas levando em conta a precisão e a cobertura para outros interesses, e assim criar novas técnicas combinadas que sejam complementares. Outro ponto importante a realizar é o desenvolvimento das ferramentas para apoiar as combinações apresentadas na Tabela 3.2 e na Tabela 3.3 e assim verificar se realmente essas combinações atingem os resultados esperados.

3.3 Considerações Finais

A revisão sistemática apresentada na seção 3.2 forneceu algumas diretrizes para o desenvolvimento da abordagem apresentada no Capítulo 4; *(i)* a necessidade de combinar técnicas de mineração com o intuito de melhorar a identificação dos interesses e *(ii)* indicar quais técnicas atingem bons resultados na procura de interesses em termos de precisão e cobertura. Dessa forma, a revisão sistemática deu luzes de qual técnica é a mais adequada para complementar a técnica de biblioteca de interesses a qual é baseada no trabalho de Junior *et al.* (2012) do grupo AdvanSE. Essa técnica pode-se categorizar como *Parser-Based* de acordo com o framework estendido apresentado na revisão sistemática. Segundo a revisão sistemática, a técnica de *clustering* atinge bons resultados na identificação de interesses, combinada com outras técnicas ou por si só (ZHANG *et al.*, 2008a; COJOCAR; CZIBULA, 2008). De fato, a Tabela 3.2 e a Tabela 3.3 mostram que a técnica CBFA é uma boa candidata para ser combinada com outras técnicas. CBFA é uma técnica que combina *clustering* de strings e *Fan-In* de métodos.

Capítulo 4

MINERAÇÃO DE INTERESSES EM UM PROCESSO ADM

Esse capítulo apresenta a abordagem de mineração de interesses proposta neste trabalho. A abordagem faz identificação de interesses transversais em modelos KDM e é uma combinação de duas técnicas; uma biblioteca de interesses e um algoritmo de agrupamento (clustering) de strings. O capítulo detalha os sub-processos envolvidos no processo de mineração de interesses, as suas entradas e saídas e as decisões que foram tomadas para desenvolver a proposta.

4.1 Visão Geral

A Figura 4.1 retrata esquematicamente o processo de mineração proposto em alto nível de abstração, o qual é composto por 4 estágios indicados com as letras *A*, *B*, *C* e *D*.

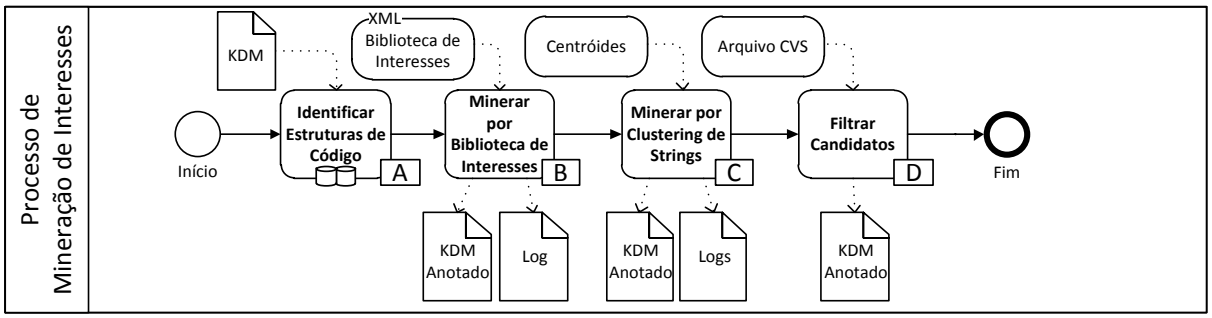


Figura 4.1: Processo de mineração de interesses em ADM em alto nível

O primeiro estágio *A* consiste na identificação das estruturas do modelo KDM que representam elementos do código fonte. Esses elementos são pacotes, classes, métodos, atributos/-variáveis entre outras onde os nomes desses elementos são armazenados em um repositório de

elementos da aplicação.

O segundo estágio *B* consiste na identificação de interesses utilizando uma biblioteca de interesses. A biblioteca armazena um conjunto de termos que são comumente usados para representar determinados interesses. Esses elementos são mapeados para os elementos estruturais (métodos, atributos, etc) identificados no estágio anterior com o objetivo de identificar quais desses elementos contribuem para a implementação de um determinado interesse. As saídas deste estágio são o arquivo KDM anotado com os interesses identificados e um arquivo log.

O terceiro estágio *C* consiste na mineração de interesses por meio da técnica de *clustering*. Essa técnica tem como objetivo complementar a anterior e tenta identificar elementos que não foram identificados previamente, isto é, para os métodos e atributos que não foram associados a nenhum interesse, ela tenta encontrar uma associação. Para isso, a técnica de *clustering* agrupa os nomes dos métodos e atributos/variáveis com algum elemento identificado pela biblioteca de interesses dependendo da similaridade dos strings. A entrada deste estágio são os centróides (elementos identificados no estágio anterior) e as saídas são arquivos dois arquivos logs e o modelo KDM com anotações de interesses identificados.

Finalmente o estágio *D*, corresponde à filtragem manual de interesses identificados pela técnica de *clustering*, ação realizada pelo usuário mediante marcações em um arquivo CSV. A saída e o arquivo KDM com as novas anotações.

A Figura 4.2, apresenta o processo de mineração em um nível mais baixo de abstração, detalhando cada um dos estágios. As subseções seguintes explicam cada um dos passos e usam essa figura como base para a explicação.

4.2 Sub-Processo A: Recuperação de Estruturas da Aplicação

Na Figura 4.2 o Sub-Processo-[A] apresenta o primeiro subprocesso da técnica combinada de mineração de interesses. O processo começa com a ação do usuário escolhendo o código fonte (contido em um projeto Java para Eclipse™) ou por meio de um arquivo XMI que representa o modelo KDM do sistema a ser minerado. Se o código fonte é escolhido como fonte de entrada para o processo, então esse deve ser transformado em um modelo KDM pelo framework Modisco™ (BRUNELIERE *et al.*, 2010).

O Modisco™ é uma solução de engenharia reversa de código aberto que utiliza princípios e técnicas da engenharia dirigida a modelos (MDE) e em específico a ADM . Assim, ele fornece as

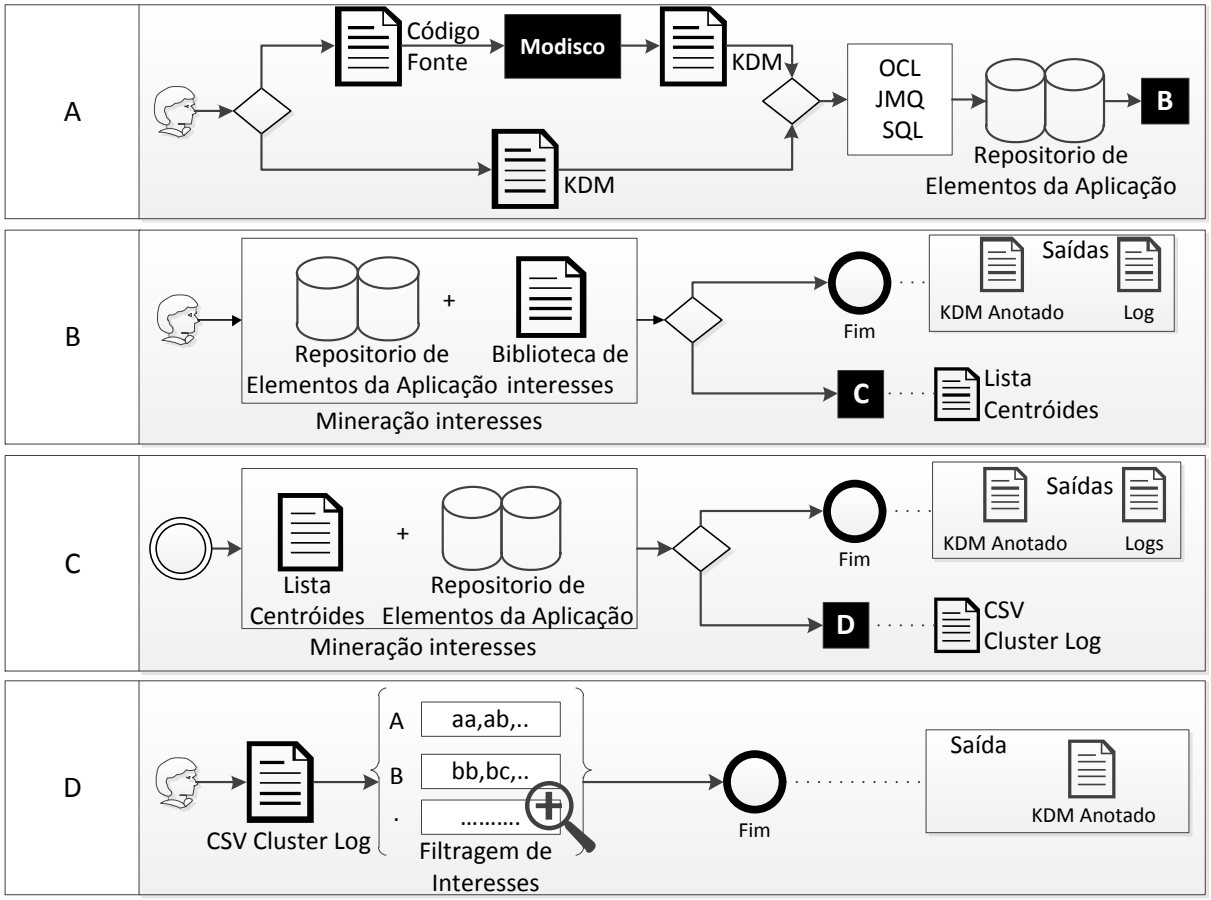


Figura 4.2: Processo de mineração de interesses em ADM

APIs (*Discoverers*) necessárias para transformar o código fonte em modelos KDM. Até a escrita deste trabalho o Modisco^{TM} era a melhor opção para apoiar o processo de transformação para modelos KDM por ter uma boa documentação, ser de código aberto e ser desenvolvido como um plugin para o Eclipse^{TM}. Porém, uma limitação é que ele permite realizar a engenharia reversa do código para os pacotes das camadas de infraestrutura, camada de elementos de programa, alguns pacotes da camada de recursos de tempo de execução. Além disso, as APIs fornecidas fazem transformações só de código Java.

Depois de transformar o código fonte em um modelo KDM ou selecionar diretamente o arquivo XMI, o modelo é carregado em memória pelo *Eclipse Modeling Framework* (EMF) do Eclipse^{TM}. Em seguida, são executadas consultas em OCL (*Object Constraint Language*) para obter a estrutura da aplicação. A linguagem OCL é uma linguagem de texto que possibilita a especificação de restrições em um modelo orientado a objeto mas também pode ser utilizada como uma linguagem de consulta. Neste trabalho, a OCL foi utilizada como uma linguagem para extrair elementos específicos do modelo. Para refinar as consultas feitas com OCL utilizou-se JMQ (*Java Model Query*) descrita neste capítulo mais adiante, em combinação com SQL.

Outra característica da abordagem proposta é que o nível de granularidade é bem fina pelas estruturas do KDM levadas em conta. As consultas OCL executadas sobre o modelo KDM são as seguintes;

- *Package.allInstances()*: recupera todos os pacotes do sistema;
- *Class.allInstances()*: recupera todas as classes do sistema;
- *Method.allInstances()*: recupera todos os métodos do sistema;
- *Property.allInstances()*: recupera todos os atributos/variáveis do sistema.

Depois que as quatro consultas OCL são executadas, utiliza-se JMQ proporcionado pela Modisco^{TM} para recuperar informações de mais baixo nível, tais como as chamadas a métodos, os parâmetros dos métodos, o tipo dos métodos e atributos, a visibilidade dos métodos e atributos/variáveis e o tipo de retorno dos métodos. JMQ é uma forma de navegar e extrair dados em modelos por meio da linguagem Java, usando as classes que implementam um determinado metamodelo (metaclasses). Note-se que é possível realizar as consultas no modelo utilizando só OCL ou só com JMQ mas cada uma têm suas vantagens e desvantagens. A grande vantagem de utilizar só OCL é que se precisaria de pouco código Java para obter as informações retornadas pelas consultas e em geral poucas linhas de código para implementar essas consultas. Porém, a grande desvantagem é que algumas consultas podem ficar complexas devido ao nível de granularidade requerido, o que dificultaria possíveis modificações se as consultas precisassem ser atualizadas por alguém que não é especialista em OCL. Por outro lado, a grande vantagem de JMQ consiste em que as consultas são escritas em código Java, uma linguagem bem conhecida e portanto não precisaria de um especialista para sua manutenção. Porém, sua desvantagem reside na quantidade de linhas de código para implementar uma consulta. Então decidiu-se utilizar OCL para as consultas globais e JMQ para as consultas mais específicas.

O Código 4.1 apresenta um exemplo de código utilizando o gerenciador de consultas de Modisco^{TM} para executar uma consulta OCL. Neste caso a consulta OCL executada que obtém todas as instâncias de métodos do modelo KDM é *Method.allInstances()*. A linha 2 cria um catálogo de consultas vazio de Modisco^{TM} . A linha 5 obtém um catálogo de consultas de nome *QuerySetKDM* a qual contém as 4 consultas OCL. As linhas 10 e 12 procuram a consulta *getMethodsOCL* que será executada. A linha 28 executa a consulta e finalmente a linha 35 cria um iterador para navegar pelas instâncias de cada método do sistema obtido pela consulta OCL.

```
// Create the model query set catalog.
ModelQuerySetCatalog catalog = ModelQuerySetCatalog.getSingleton();

// Get the query set named "QuerySetKDM".
ModelQuerySet modelQuerySet = catalog.getModelQuerySet("QuerySetKDM");

// Select in the "QuerySetKDM" query set a query named "getMethodsOCL".
// modelQueryDescription is a model element.
ModelQuery modelQueryDescription = null;
for (ModelQuery modelQuery : modelQuerySet.getQueries())
{
        if (modelQuery.getName().equals("getMethodsOCL"))
        {
                modelQueryDescription = modelQuery;
                break;
        }
}
if (modelQueryDescription == null)
{
        throw new Exception();
}

//Get a java instance of the querySet
AbstractModelQuery myModelQuery = catalog.getModelQueryImpl(modelQueryDescription);


//the model query set evaluation
ModelQueryResult result = myModelQuery.evaluate(context);
if (result.getException() != null)
{
        throw new Exception();
}

@SuppressWarnings("unchecked")
Iterator<MethodUnit> it = ((HashSet<MethodUnit>) result.getValue()).iterator();
```

Código 4.1: Código Java para executar uma consulta OCL

O Código 4.2, é a continuação do código anterior. Utiliza-se JMQ para navegar pelos métodos e assim extrair as informações para persisti-las no repositório de elementos da aplicação. São 5 métodos *Java* utilizando JMQ com codificações similares para obter as informações da aplicação.

```
1  while (it.hasNext()){
2          method = (MethodUnit) it.next();
3          if (method.getAttribute().size() > 0){
4                  if (method.getAttribute().get(0).getTag().equals("export")){
5                          String name = method.getName();
6                          String module = getContainer(method.eContainer());
7                          String type = method.getKind().getName();
8                          String visibility = method.getExport().getName();
9                          String rtn = "";
10
11                         EList<AbstractCodeElement> elements = method.getCodeElement();
12                         for (int i=0 ; i<elements.size() ; i++)
13                                 if (elements.get(i) instanceof Signature )
14                                         signature = (Signature) elements.get(i);
15
16                         String sign = "";
17                         EList<ParameterUnit> parameterUnit = signature.getParameterUnit();
18                         for (int i=0 ; i<parameterUnit.size() ; i++){
19                                 if (parameterUnit.get(i) instanceof ParameterUnit ){
20                                         paramUnit = (ParameterUnit) parameterUnit.get(i);
21                                         if (paramUnit.getKind().getName().equals("return"))
22                                                 rtn = paramUnit.getType().getName();
23
24                                         if(paramUnit.getName() != null){
25                                                 sign = sign + paramUnit.getType().getName() + "-" + paramUnit.getName()
                                                           + "-";
26                                         }
27                                 }
28                         }
29                         if (sign.length() == 0)
30                                 sign = "void";
31
32                         paramsMethod.add(name + "|" + module + "|" + sign + "|" + type + "|" + visibility + "|" + rtn);
33                 }
34         }
35 }
36 try{
37         ConnectionDB.getInstance(projectName).addBatch(Query.getQuery("insertMethod"), paramsMethod);
38 }
39 catch (SQLException e){
40         e.printStackTrace();
41 }
42 paramsMethod.clear();
```

Código 4.2: Código JMQ para obter os elementos de um método

A linha 1 do Código 4.2 apresenta um iterador que contém todos os métodos do sistema. Então para cada método, serão obtidas as suas informações. Na linha 2 a metaclasse *MethodUnit* de KDM especifica um método no código fonte, então a variável *method* vai conter um método do modelo por cada iteração. Com as linhas 5, 6, 7 e 8 obtém-se o nome do método, o nome da

classe onde está contido o método, o tipo (abstrato, construtor, virtual, destrutor) e a visibilidade (privada, publico, protegida). As linhas $11-25$ obtém a assinatura do método com os nomes dos parâmetros e o nome do tipo de retorno do método. Todas essas informações são adicionadas na variável *paramsMethod* da linha 32 para logo serem persistidas no repositório de estruturas na linha 37. Para persistir as estruturas relacionadas com pacotes, classes e atributos/variáveis a lógica seguida é similar aos códigos apresentados.

A Tabela 4.1 apresenta um resumo do tipo de consulta e os nomes de elementos de código recuperados. Por exemplo, os pacotes do sistema foram recuperados com OCL e os nomes desse pacotes foram recuperados com JMQ.

	Elementos			
OCL	Pacotes	Classes	Métodos	Atributos/Variáveis
JMQ	Nome do pacote	Nome da classe Nome do pacote contendor Nome da classe aninhada Nome do import	Nome do método Nome da classe contendora Nome do tipo de método Visibilidade Assinatura Nome do tipo de retorno	Nome do atributo/variável Nome da classe contendora Nome do método contenedor Âmbito Visibilidade Nome do tipo

Tabela 4.1: Resumo do tipo de consulta com os elementos recuperados

A Figura 4.3 apresenta o modelo de dados EER (Extend Entity Relationship) criado para persistir as informações recuperadas sobre a estrutura da aplicação. Esse modelo foi desenvolvido para dar suporte ao processo de mineração e é criado automaticamente quando o processo é executado. A tabela *Module* contém informação sobre as classes e interfaces da aplicação; o seu nome e o seu tipo (classe abstrata ou não). Note-se que uma classe pode ser aninhada com outra classe, então por isso que tem um relacionamento consigo mesma. A tabela *Import* persiste os imports da aplicação. Existe um relacionamento $n : m$ entre essa tabela e a tabela *Module*, então a tabela *Module_has_Import* é criada. A classe pacote persiste as informações dos pacotes do sistema e pode estar contida em outro pacote. A tabela *Method* persiste as informações dos métodos; nome, assinatura, tipo, visibilidade e o tipo do retorno. Uma classe pode ter atributos os quais são persistidos na tabela *ModuleProperty*. As variáveis locais estão persistidas na tabela *MethodProperty* es está relacionada com a tabela *Method*. Outro ponta a ser ressaltado é que a tabela *Method* está relacionada consigo mesma $n : m$ porque pode chamar a outros métodos. Com isso pode se calcular a métrica *Fan-in* dos métodos.

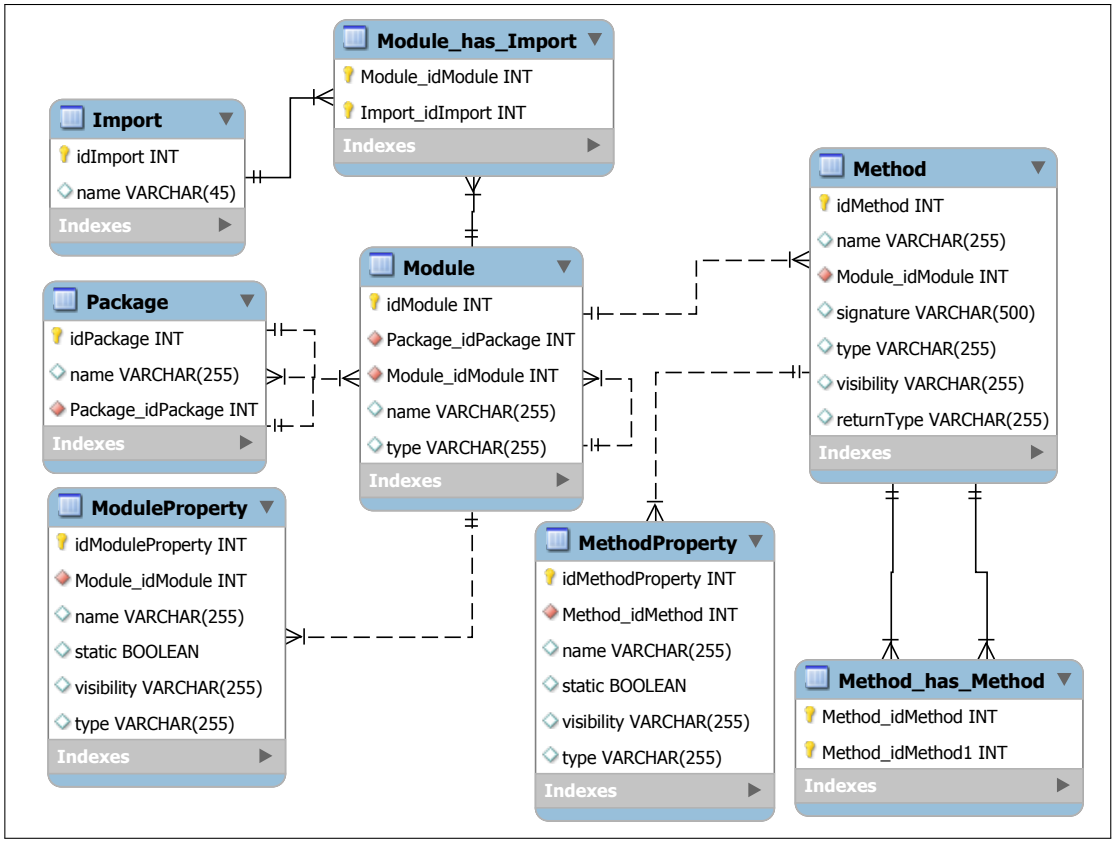


Figura 4.3: Modelo de dados EER para o repositório de elementos da aplicação

4.3 Sub-Processo B: Mineração por Biblioteca de Interesses

Após persistir as informações do sistema legado no banco de dados, inicia-se a mineração pela biblioteca de interesses, Figura 4.2 Sub-Processo-[B]. O principal fundamento desta técnica é que os interesses existentes nas aplicações podem ser deduzidos a partir das APIs usadas por essas aplicações. As APIs são amplamente utilizadas no desenvolvimento de sistemas porque além de disponibilizar os seus serviços, ocultam os detalhes das suas implementações. Isto fornece algumas vantagens como, por exemplo, *(i)* rapidez no desenvolvimento porque o programador não precisa preocupar-se em implementar certas características que fogem do escopo do domínio do negócio, e *(ii)* robustez porque geralmente as APIs são soluções testadas por muitos usuários, então pode-se ter certeza dos resultados esperados. Dessa forma, pode-se encontrar diferentes tipos de APIs para auxiliar aos programadores em diferentes âmbitos como por exemplo, APIs que implementam banco de dados, tratamento de exceções, etc.

A técnica de biblioteca de interesses tem por objetivo identificar variáveis/atributos cujo tipo é uma classe/interface de API e o tipo de retorno dos métodos. Para atingir esse objetivo a biblioteca é composta de regras que são adicionadas pelo usuário. Essas regras são compostas por um nome (interesse a ser procurado) e os nomes dos pacotes e classes que implementam

uma determinada API. Na literatura pode-se encontrar ferramentas que se baseiam em técnicas similares (BREU *et al.*, 2006; GRISWOLD *et al.*, 1999; HANNEMANN; KICZALES, 2001). O Código 4.3 apresenta um exemplo de biblioteca de interesses a qual está composta por três interesses; *Persistence*, *Logging* e *Authentication*. Cada pacote é implementado por várias classes que serão finalmente procuradas no repositório de elementos da aplicação para identificar presença de interesses em métodos e atributos/variáveis. Um dos pontos a favor de utilizar essa técnica é que a biblioteca pode ser modificada, isto é, adicionar novas definições de interesses, removê-las ou atualizá-las o que a faz uma técnica flexível e que poderia ser utilizada em outros contextos, como por exemplo em linhas de produtos para a procura de *features*. A principal limitação é que o usuário deve ter conhecimento dos interesses a ser procurados. Outra coisa em destaque é que o armazenamento em um arquivo XML facilita a navegação pela estrutura hierárquica da informação, é um formato bem estruturado e possibilita o intercâmbio de arquivos facilmente.

```
<ConcernLibrary>
        <Concern name='Persistence'>
                <Package name='java.sql'>
                        <Element>Connection</Element>
                        <Element>PreparedStatement</Element>
                        <Element>ResultSet</Element>
                        <Element>Statement</Element>
                        <Element>DriverManager</Element>
                        <Element>BatchUpdateException</Element>
                        <Element>SQLException</Element>
                        <Element>SQLWarning</Element>
                </Package>
        </Concern>
    <Concern name='Logging'>
            <Package name='java.util.logging'>
                    <Element>Filter</Element>
                    <Element>LoggingMXBean</Element>
                    <Element>Logger</Element>
                    <Element>LoggingPermission</Element>
                    <Element>LogManager</Element>
                    <Element>LogRecord</Element>
                    <Element>XMLFormatter</Element>
            </Package>
    </Concern>
    <Concern name='Authentication'>
            <Package name='javax.security.auth'>
                    <Element>Destroyable</Element>
                    <Element>Refreshable</Element>
                    <Element>AuthPermission</Element>
                    <Element>DestroyFailedException</Element>
                    <Element>RefreshFailedException</Element>
            </Package>
    </Concern>
</ConcernLibrary>
```

Código 4.3: Exemplo de biblioteca de interesses

Vale ressaltar novamente que a mineração é feita no modelo de dados. O Código 4.4 SQL realiza a identificação de métodos que poderiam implementar um interesse. Os pacotes (*Pac-*

kages) de um determinado interesse da biblioteca de interesses mostrados na Figura 4.3, são passados como parâmetro para procurar quais classes estão usando essas APIs. A outra condição obtém os métodos onde o tipo de retorno pertence a algum elemento da biblioteca para um determinado interesse. A tabela temporal *ELEMENTS* contém as classes de um determinado pacote-interesse dependendo do interesse procurado. Para a identificação de atributos/variáveis que poderiam implementar um interesse, a lógica é a mesma.

```
1 SELECT MO.NAME AS MODULE_NAME,
2            M.NAME AS METHOD_NAME,
3            M.TYPE AS METHOD_TYPE,
4            M.RETURNTYPE AS METHOD_RETURN
5 FROM IMPORT I INNER JOIN MODULE_HAS_IMPORT MI ON I.IDIMPORT = MI.IMPORT_IDIMPORT
6                           INNER JOIN MODULE MO ON MI.MODULE_IDMODULE = MO.IDMODULE
7                           INNER JOIN METHOD M ON M.MODULE_IDMODULE = MO.IDMODULE
8 WHERE I.NAME LIKE ? AND M.RETURNTYPE IN (SELECT ELEMENT FROM ELEMENTS)
```

Código 4.4: Identificação de métodos que implementam algum interesse

Se o usuário não deseja executar Sub-Processo-[C] no caso que ele precisar maior precisão nos resultados que cobertura, o Sub-Processo-[B] termina. Por default, após ter executado o Sub-Processo-[B], executa-se o Sub-Processo-[C] correspondente a técnica de *clustering* de strings e é detalhada na seção seguinte.

As saídas para o Sub-Processo-[B], correspondentes à mineração com a biblioteca de interesses, são um arquivo log e o KDM anotado.

A Figura 4.4 apresenta um extrato do um log de saída após ter sido executado o Sub-Processo-[B] e encontra-se marcada com letras para uma melhor explicação. O quadro *A* mostra o nome do interesse encontrado. Pode-se observar que o log apresenta um interesse, *Persistence*. A letra *B*, mostra o pacote (API) da biblioteca de interesses que foi encontrada no sistema. A letra *C*, mostra 4 elementos. O primeiro elemento é a variável *resposta* pertencente ao método *getCommunicationChannel* da classe *PersistenceMechanism*. O segundo elemento é a variável *conexoesLivres* da classe *conexoesLivres*. O terceiro elemento é o método *ComplaintRepositoryRDB* da classe *ComplaintRepositoryRDB* e finalmente o quarto elemento é a variável *conexoesCriadas* da classe *PersistenceMechanism*. Os 4 elementos encontrados são anotados no modelo KDM.

A Figura 4.5 apresenta um pequeno extrato de um modelo KDM com duas anotações com *Persistence*.

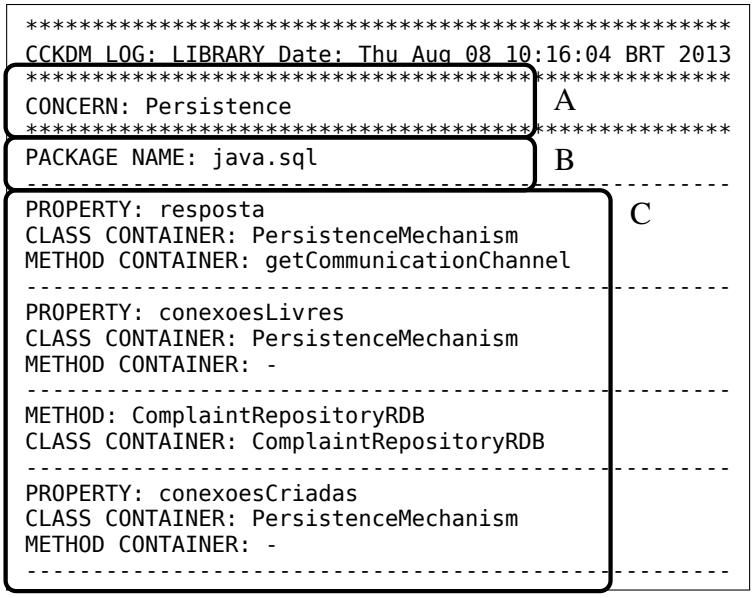


Figura 4.4: Arquivo log após mineração pela biblioteca de interesses

```
@codeElement.2/@codeElement.0"/>
                    </codeElement>
                  </codeElement>
                  <codeElement xsi:type="action:ActionElement" name="variable declaration" kind="variable
declaration">
                    <source language="java">
                      <region file="/0/@model.2/@inventoryElement.32" language="java"/>
                    </source>
                    <codeElement concern="Persistence" xsi:type="code:StorableUnit" name="symptom"
type="/0/@model.0/@codeElement.0/@codeElement.2/@codeElement.1/@codeElement.7" kind="local">
                      <attribute tag="export" value="none"/>
                      <source language="java">
                        <region file="/0/@model.2/@inventoryElement.32" language="java"/>
                      </source>
                    </codeElement>
                  </codeElement>
                  <codeElement xsi:type="action:TryUnit" name="try" kind="try">
                    <source language="java">
                      <region file="/0/@model.2/@inventoryElement.32" language="java"/>
                    </source>
                    <codeElement xsi:type="action:BlockUnit">
                      <source language="java">
                        <region file="/0/@model.2/@inventoryElement.32" language="java"/>
                      </source>
                      <codeElement xsi:type="action:ActionElement" name="variable declaration"
kind="variable declaration">
                        <source language="java">
                          <region file="/0/@model.2/@inventoryElement.32" language="java"/>
                        </source>
                        <codeElement concern="Persistence" xsi:type="code:StorableUnit" name="stmt"
type="/0/@model.1/@codeElement.0/@codeElement.5/@codeElement.3" kind="local">
                          <attribute tag="export" value="none"/>
                          <source language="java">
                            <region file="/0/@model.2/@inventoryElement.32" language="java"/>
                          </source>
                          <codeRelation xsi:type="code:HasValue" to="/0/@model.1/@codeElement.131"
from="/0/@model.0/@codeElement.0/@codeElement.1/@codeElement.2/@codeElement.1/@codeElement.5/
@codeElement.1/@codeElement.4/@codeElement.0/@codeElement.0/@codeElement.0"/>
                        </codeElement>
                      </codeElement>
                      <codeElement xsi:type="action:ActionElement" name="expression statement"
kind="expression statement">
                        <source language="java">
                          <region file="/0/@model.2/@inventoryElement.32" language="java"/>
                        </source>
                        <codeElement xsi:type="action:ActionElement" name="ASSIGN" kind="assignment">
                          <source language="java">
                            <region file="/0/@model.2/@inventoryElement.32" language="java"/>
                          </source>
                          <codeElement xsi:type="action:ActionElement" name="method invocation"
kind="method invocation">
```

Figura 4.5: Interesses anotados no modelo KDM

4.4 Sub-Processo C: Identificação de Interesses por Clustering de Strings

Neste seção é apresentada a técnica de *clustering* de strings desenvolvida para complementar a mineração baseada em biblioteca de interesses. A técnica de *clustering* visa agrupar elementos que compartilham características em comum, em específico para esse trabalho deseja-se agrupar nomes de métodos e atributos/variáveis que tenham similaridade entre si.

Os programadores tendem a utilizar identificadores em elementos de programa (classes/-variáveis/métodos etc.) com nomes similares quando estão implementando determinados interesses, mesmo que em contextos diferentes (BINKLEY *et al.*, 2013). Dessa forma, variáveis e métodos que possivelmente implementam algum interesse, mas que não foram identificados pela técnica anterior podem agora ser agrupados em torno de algum elemento que foi reconhecido no SubProcesso-[B] pela similaridade dos seus nomes. Assim, a técnica de *clustering* complementa a técnica da biblioteca de interesses.

A técnica de *clustering* implementada é um algoritmo *K-means* modificado (BERKHIN, 2006). É modificado porque o algoritmo só faz uma iteração para agrupar os strings. O algoritmo original faz múltiplas iterações até encontrar um agrupamento ótimo de acordo com a métrica utilizada para agrupar. Para atingir esse objetivo os *clusters* são desagrupados e agrupados novamente com outros elementos porque os centróides mudam. No caso da técnica desenvolvida, os *clusters* não são desagrupados porque os centróides não mudam e assim a métrica utilizada sempre resulta no mesmo valor (similaridade de strings).

O algoritmo *K-means* divide o conjunto de dados em K grupos onde cada grupo contém elementos similares. Os K elementos iniciais do algoritmo são chamados centróides e para o caso da técnica implementada, são todos os candidatos que implementam algum interesse e foram identificados no Sub-Processo-[B]. Assim, o agrupamento dos elementos identificados pela técnica de *clustering* são agrupados em torno aos centroides segundo o valor da métrica distância *Levenshtein* (LEVENSHTEIN, 1966). A distância *Levenshtein* é uma métrica que mede a diferença entre duas sequências de strings e possui aplicações nas áreas de pesquisa tais como mineração de texto e bioinformática (EIBLER *et al.*, 2011; CHOI, 2013). Na sequencia há um exemplo de como calcula-se a distância *Levenshtein* entre as palavras inglesas *meeting* e *feeling*. A distância entra as duas palavras é 2, já que com apenas 2 edições consegui-se transformar uma palavra na outra:

1. meeting

2. feeting (substituição de “m” por “f”)

3. feeling (substituição de “t” por “l”)

O algoritmo 1 apresenta a função de *clustering* simplificada.

Algorithm 1 Algoritmo modificado *K-means* para cluster de strings

Entrada: c: Nome do interesse, t: Valor *Levenshtein*
Saída: $matrix(nxm)$: matriz com n centróides e m strings

```
1:  function CLUSTERING(c, t)
2:      methodCentroid[] = getMethodByLibrary(c);
3:      propertyCentroid[] = getPropertyByLibrary(c);
4:      P = join(methodCentroid[],propertyCentroid[]);
5:      eliminateDuplicates(P);
6:      matrix(K,T);
7:      for each p into P do
8:          values[] = levenshtein(p , propertyMethodList[]);
9:          i = 0;
10:         for each v in values[] do
11:             if v ≥ t then
12:                 vector[i] = v;
13:             else
14:                 vector[i] = 0;
15:             end if
16:             i = i + 1;
17:         end for
18:         matrix.addRow(vector[]);
19:     end for
20:     return matrix;
21: end function
```

A função recebe como parâmetros de entrada o nome do interesse c e uma valor límite t e a saída é uma matriz(nxm). As linhas $2-3$ do algoritmo criam duas listas que são os centroides iniciais. Nas linhas $4-5$ as listas são combinadas e os valores duplicados são removidos. A linha 8 calcula a distância *Levenshtein* para cada centroide inicial p com todos os nomes de métodos e variáveis do sistema. O calculo da distância *Levenshtein* é realizado pela biblioteca *SimMetrics*[1] (*Similarity Metric Library*), a qual contém vários tipos de métricas para strings. A linha 11 compara o valor *Levenshtein* v com o valor limite t. Se $v \geq t$ então o valor *Levenshtein* v é adicionado em um vetor $vector$ na posição i. O algoritmo que implementa a distância *Levenshtein* encontra-se normalizado em um intervalo de $(0,1)$. Se o valor limite t for mais perto de 1.0 o algoritmo irá agrupar pequenas quantidades de strings mas com uma maior similaridade. No caso contrário, se o valor limite t for mais perto de 0.0 então o algoritmo irá agrupar

[1] http://sourceforge.net/projects/simmetrics/

maiores quantidades de strings mas com uma menor similaridade. Na linha 18, o vetor *vector* é adicionado em uma matriz *matrix* com *K* filas representando a quantidade de centróides e *T* colunas representando que é quantidade de total de nomes de métodos e variáveis não duplicados. Na linha 20 a matriz é retornada onde os valores mais altos de cada coluna indicam a que centróide o string deve ser agrupado. É importante ressaltar que a implementação do algoritmo não leva em conta mais de uma iteração, ou seja os grupos formados não sofrem novas mudanças. É por isso que a implementação é um *K-means* modificado. O Sub-Processo-[C] termina com alguns logs de saída e um arquivo contendo o modelo KDM do sistema com as anotações dos interesses identificados. As saídas para o Sub-Processo-[C], correspondente a mineração com a técnica de *clustering*, são 2 arquivos logs e o KDM anotado. A Figura 4.6 apresenta o primeiro log deste sub processo.

```
**************************************************
CCKDM LOG: K-MEANS Date: Thu Aug 08 10:16:04 BRT 2013
**************************************************
CONCERN: Persistence | Threshold: 0.4
**************************************************
**************************************************
CLUSTER CENTROID: resposta
--------------------------------------------------
PROPERTY: response | L.V.: 0,62
CLASS CONTAINER: ComplaintRepositoryRDB
METHOD CONTAINER: exists
--------------------------------------------------
PROPERTY: repEsp | L.V.: 0,5
CLASS CONTAINER: GetDataForSearchBySpeciality
METHOD CONTAINER: execute
--------------------------------------------------
**************************************************
CLUSTER CENTROID: conexoesLivres
--------------------------------------------------
METHOD: closeQueries | L.V.: 0,43
CLASS CONTAINER: HTMLCode
--------------------------------------------------
**************************************************
CLUSTER CENTROID: ComplaintRepositoryRDB
--------------------------------------------------
PROPERTY: complaintRep | L.V.: 0,5
CLASS CONTAINER: ComplaintRecord
METHOD CONTAINER: -
--------------------------------------------------
PROPERTY: complaintType | L.V.: 0,41
CLASS CONTAINER: ComplaintRepositoryRDB
METHOD CONTAINER: deepInsertCommon
--------------------------------------------------
**************************************************
CLUSTER CENTROID: conexoesCriadas
--------------------------------------------------
PROPERTY: conexoesAlocadas | L.V.: 0,75
CLASS CONTAINER: PersistenceMechanism
METHOD CONTAINER: -
--------------------------------------------------
```

A
B
C

Figura 4.6: Extrato de arquivo log após mineração pela técnica de *clustering*

A letra *A* mostra o interesse minerado com o valor da distância *levenshtein* em 0.4. A letra *B* apresenta o primeiro centróide, *resposta*. Para esse string foram encontrados dois strings simila-

res, letra *C* a variável *response* pertencente ao método *exists* da classe *ComplaintRepositoryRDB* e a variável *repEsp* pertencente ao método *execute* da classe *GetDataForSearchBySpeciality*. Da mesma forma se observa que os centróides *conexoesLivres*, *ComplaintRepositoryRDB* e *conexoesCriadas* agruparam strings. Então, para a técnica de *clustering* os elementos anotados com persistência são; *response*, *repEsp*, *closeQueries*, *complaintRep*, *complaintType* e *conexoesAlocadas*. O objetivo deste log não é para ser lido por um humano, de fato com as informações do arquivo log da Figura 4.6 é gerado um arquivo CSV que serve para realizar as marcações manuais no Sub-Processo-[D]. A Figura 4.7 apresenta um exemplo de arquivo CSV con dois centróides para o interesse de Persistência, a variável *con* (linha 3) e a variável *resposta* (linha 12). Nesse casso, as variáveis das linhas 7 até 11 agrupadas por *con* foram marcadas com X. Da mesma forma, as variáveis das linhas 13 até 20 agrupadas por *resposta* também fora marcadas com X. Isso significa que essa variáveis serão desmarcadas com o interesse de Persistência do modelo KDM. Quando o arquivo CSV é carregado com o aplicativo Excel, tem a mesma aparência que o arquivo da Figura 4.8.

```
1   ,,,,,,,
2   ** Persistence **,,,,,,,
3   con,0.4,0.5,0.6,0.7,0.8,0.9,
4   ,|command|getCommand|HWServlet,|command|getCommand|HWServlet,|command|getCommand|HWServlet,|command|getCommand|HWServlet,|
    command|getCommand|HWServlet,|command|getCommand|HWServlet,
5   ,|command|handleRequest|HWServlet,|command|handleRequest|HWServlet,|command|handleRequest|HWServlet,|command|handleRequest|
    HWServlet,|command|handleRequest|HWServlet,|command|handleRequest|HWServlet,
6   ,|command|retry|HWServlet,|command|retry|HWServlet,|command|retry|HWServlet,|command|retry|HWServlet,|command|retry|HWServlet,|
    command|retry|HWServlet,
7   ,,|code|-|Address,|code|-|Address,|code|-|Address,|code|-|Address,|code|-|Address,X
8   ,,|code|-|ComplaintState,|code|-|ComplaintState,|code|-|ComplaintState,|code|-|ComplaintState,|code|-|ComplaintState,X
9   ,,|code|-|DiseaseType,|code|-|DiseaseType,|code|-|DiseaseType,|code|-|DiseaseType,|code|-|DiseaseType,X
10  ,,|code|-|HealthUnit,|code|-|HealthUnit,|code|-|HealthUnit,|code|-|HealthUnit,|code|-|HealthUnit,X
11  ,,|code|-|Situation,|code|-|Situation,|code|-|Situation,|code|-|Situation,|code|-|Situation,X
12  resposta,0.4,0.5,0.6,0.7,0.8,0.9,
13  ,,|request|-|ServletRequestAdapter,|request|-|ServletRequestAdapter,|request|-|ServletRequestAdapter,|request|-|
    ServletRequestAdapter,|request|-|ServletRequestAdapter,X
14  ,,,|response|-|ServletResponseAdapter,|response|-|ServletResponseAdapter,|response|-|ServletResponseAdapter,|response|-|
    ServletResponseAdapter,X
15  ,,,|response|search|ComplaintRepositoryArray,|response|search|ComplaintRepositoryArray,|response|search|
    ComplaintRepositoryArray,|response|search|ComplaintRepositoryArray,X
16  ,,,|response|search|DiseaseTypeRepositoryArray,|response|search|DiseaseTypeRepositoryArray,|response|search|
    DiseaseTypeRepositoryArray,|response|search|DiseaseTypeRepositoryArray,X
17  ,,,|response|search|EmployeeRepositoryArray,|response|search|EmployeeRepositoryArray,|response|search|EmployeeRepositoryArray,|
    response|search|EmployeeRepositoryArray,X
18  ,,,|response|search|HealthUnitRepositoryArray,|response|search|HealthUnitRepositoryArray,|response|search|
    HealthUnitRepositoryArray,|response|search|HealthUnitRepositoryArray,X
19  ,,,|response|search|SpecialityRepositoryArray,|response|search|SpecialityRepositoryArray,|response|search|
    SpecialityRepositoryArray,|response|search|SpecialityRepositoryArray,X
20  ,,,|response|search|SymptomRepositoryArray,|response|search|SymptomRepositoryArray,|response|search|SymptomRepositoryArray,|
    response|search|SymptomRepositoryArray,X
21  ,,,|response|exists|ComplaintRepositoryRDB,|response|exists|ComplaintRepositoryRDB,|response|exists|ComplaintRepositoryRDB,|
    response|exists|ComplaintRepositoryRDB,
22  ,,,|response|exists|DiseaseTypeRepositoryRDB,|response|exists|DiseaseTypeRepositoryRDB,|response|exists|
    DiseaseTypeRepositoryRDB,|response|exists|DiseaseTypeRepositoryRDB,
23  ,,,|response|exists|EmployeeRepositoryRDB,|response|exists|EmployeeRepositoryRDB,|response|exists|EmployeeRepositoryRDB,|
    response|exists|EmployeeRepositoryRDB,
24  ,,,|response|exists|HealthUnitRepositoryRDB,|response|exists|HealthUnitRepositoryRDB,|response|exists|HealthUnitRepositoryRDB,|
    response|exists|HealthUnitRepositoryRDB,
25  ,,,|response|exists|SpecialityRepositoryRDB,|response|exists|SpecialityRepositoryRDB,|response|exists|SpecialityRepositoryRDB,|
    response|exists|SpecialityRepositoryRDB,
26  ,,,|response|exists|SymptomRepositoryRDB,|response|exists|SymptomRepositoryRDB,|response|exists|SymptomRepositoryRDB,|response|
    exists|SymptomRepositoryRDB,
27  ,,,|response|update|ComplaintRepositoryRDB,|response|update|ComplaintRepositoryRDB,|response|update|ComplaintRepositoryRDB,|
    response|update|ComplaintRepositoryRDB,
```

Figura 4.7: Exemplo de arquivo CSV

4.5 Sub-Processo D: Filtragem Manual de Interesses

O Sub-Processo-[D] é opcional e possibilita ao usuário filtrar candidatos manualmente e assim ter controle das marcações no modelo KDM dos elementos identificados pela técnica de *clustering*. Um dos arquivos de saída que fornece a técnica proposta é um arquivo CSV (*Comma-Separated Values*) contendo todos os strings identificados. A Figura 4.8 apresenta um exemplo de arquivo CSV onde a técnica de *clustering* identificou que a variável *code* declarada em várias classes, faz parte da implementação do interesse de Persistência porque foi agrupada com a variável *con*. O X marcada no final de cada fila indica que a variável deve ser desmarcada do modelo KDM. Da mesma forma, a variável *response* agrupada com o centróide *resposta*, será desmarcada.

** Persistence **				
con	0.7	0.8	0.9	
	|command|getCommand|HWServlet	|command|getCommand|HWServlet	|command|getCommand|HWServlet	
	|command|handleRequest|HWServlet	|command|handleRequest|HWServlet	|command|handleRequest|HWServlet	
	|command|retry|HWServlet	|command|retry|HWServlet	|command|retry|HWServlet	
	|connect|IPersistenceMechanism	|connect|IPersistenceMechanism	|connect|IPersistenceMechanism	
	|connect|IteratorRMISourceAdapter	|connect|IteratorRMISourceAdapter	|connect|IteratorRMISourceAdapter	
	|connect|PersistenceMechanism	|connect|PersistenceMechanism	|connect|PersistenceMechanism	
	|connect|RMIServletAdapter	|connect|RMIServletAdapter	|connect|RMIServletAdapter	
	|context|readFile|Library	|context|readFile|Library	|context|readFile|Library	
	|code|-|Address	|code|-|Address	|code|-|Address	X
	|code|-|ComplaintState	|code|-|ComplaintState	|code|-|ComplaintState	X
	|code|-|DiseaseType	|code|-|DiseaseType	|code|-|DiseaseType	X
	|code|-|HealthUnit	|code|-|HealthUnit	|code|-|HealthUnit	X
	|code|-|Situation	|code|-|Situation	|code|-|Situation	X
	|code|-|Symptom	|code|-|Symptom	|code|-|Symptom	X
	|code|execute|InsertDiseaseType	|code|execute|InsertDiseaseType	|code|execute|InsertDiseaseType	
	|code|execute|InsertHealthUnit	|code|execute|InsertHealthUnit	|code|execute|InsertHealthUnit	
	|code|execute|InsertMedicalSpeciality	|code|execute|InsertMedicalSpeciality	|code|execute|InsertMedicalSpeciality	
	|code|execute|InsertSymptom	|code|execute|InsertSymptom	|code|execute|InsertSymptom	
	|code|getComplaintList|ComplaintRepositoryRDB	|code|getComplaintList|ComplaintRepositoryRDB	|code|getComplaintList|ComplaintRepositoryRDB	
	|fone|search|AddressRepositoryRDB	|fone|search|AddressRepositoryRDB	|fone|search|AddressRepositoryRDB	
	|count|insert|AddressRepositoryRDB	|count|insert|AddressRepositoryRDB	|count|insert|AddressRepositoryRDB	
	|count|insert|ComplaintRepositoryRDB	|count|insert|ComplaintRepositoryRDB	|count|insert|ComplaintRepositoryRDB	
resposta	0.7	0.8	0.9	
	|repEsp|execute|GetDataForSearchBySpeciality	|repEsp|execute|GetDataForSearchBySpeciality	|repEsp|execute|GetDataForSearchBySpeciality	
	|repEsp|execute|SearchSpecialtiesByHealthUnit	|repEsp|execute|SearchSpecialtiesByHealthUnit	|repEsp|execute|SearchSpecialtiesByHealthUnit	
	|request|-|Command	|request|-|Command	|request|-|Command	
	|request|-|ServletRequestAdapter	|request|-|ServletRequestAdapter	|request|-|ServletRequestAdapter	
	|response|-|Command	|response|-|Command	|response|-|Command	
	|response|-|ServletResponseAdapter	|response|-|ServletResponseAdapter	|response|-|ServletResponseAdapter	
	|response|exists|ComplaintRepositoryRDB	|response|exists|ComplaintRepositoryRDB	|response|exists|ComplaintRepositoryRDB	
	|response|exists|DiseaseTypeRepositoryRDB	|response|exists|DiseaseTypeRepositoryRDB	|response|exists|DiseaseTypeRepositoryRDB	
	|response|exists|EmployeeRepositoryRDB	|response|exists|EmployeeRepositoryRDB	|response|exists|EmployeeRepositoryRDB	
	|response|exists|HealthUnitRepositoryRDB	|response|exists|HealthUnitRepositoryRDB	|response|exists|HealthUnitRepositoryRDB	
	|response|exists|SpecialityRepositoryRDB	|response|exists|SpecialityRepositoryRDB	|response|exists|SpecialityRepositoryRDB	
	|response|exists|SymptomRepositoryRDB	|response|exists|SymptomRepositoryRDB	|response|exists|SymptomRepositoryRDB	
	|response|search|ComplaintRepositoryArray	|response|search|ComplaintRepositoryArray	|response|search|ComplaintRepositoryArray	X
	|response|search|DiseaseTypeRepositoryArray	|response|search|DiseaseTypeRepositoryArray	|response|search|DiseaseTypeRepositoryArray	X
	|response|search|EmployeeRepositoryArray	|response|search|EmployeeRepositoryArray	|response|search|EmployeeRepositoryArray	X
	|response|search|HealthUnitRepositoryArray	|response|search|HealthUnitRepositoryArray	|response|search|HealthUnitRepositoryArray	X
	|response|search|SpecialityRepositoryArray	|response|search|SpecialityRepositoryArray	|response|search|SpecialityRepositoryArray	X
	|response|search|SymptomRepositoryArray	|response|search|SymptomRepositoryArray	|response|search|SymptomRepositoryArray	X
	|response|update|ComplaintRepositoryRDB	|response|update|ComplaintRepositoryRDB	|response|update|ComplaintRepositoryRDB	

Figura 4.8: Exemplo de arquivo CSV marcado pelo usuário

4.6 Anotação de Interesses no Modelo KDM

Os sub-processos e o processo em geral têm como saídas o **modelo KDM anotado** e **logs** que registraram os elementos identificados. Nesta seção detalha-se como é anotado o modelo KDM e as tecnologias envolvidas neste processo.

O arquivo gerado pelo Modisco^{TM} contendo o modelo KDM é um arquivo XMI. O arquivo XMI é um padrão do OMG para troca de informações baseado em XML. Então dadas as características dos arquivos baseados em XML, tomou-se a decisão de utilizar XQuery [2] para fazer as consultas e as anotações. XQuery é um linguagem que provê mecanismos para consultar diferentes tipos de fonte de dados XML, semânticamente similar a SQL, incluindo capacidades de programação. Para realizar a marcação no modelo KDM foi introduzido um novo atributo chamado *"concern"* onde métodos e atributos/variáveis são anotados com o interesse que colaboram com a implementação na seguinte forma; concern="CONCERN". A Figura 4.9 apresenta um exemplo de dois interesses anotados no modelo KDM; *Persistence* e *Logging* para um método (*MethodUnit*) e uma variável (*StorableUnit*) respectivamente.

```
<codeElement concern="Persistence"
             xsi:type="code:MethodUnit"
             ….>
<codeElement concern="Logging"
             xsi:type="code:StorableUnit"
             …..>
```

Figura 4.9: Interesses anotados no modelo KDM

O novo atributo adicionado não faz parte do metamodelo KDM, porém é uma modificação leve que não afeta o funcionamento do metamodelo e portanto continua sendo padrão. Além disso, o KDM fornece os mecanismos para poder estendê-lo facilmente e assim ferramentas que lêem o metamodelo poderão identificar o novo atributo. Só é possível realizar uma marcação para cada método ou atributo, isto é porque a biblioteca de interesses define o tipo de interesse que pode implementar um determinado método ou atributo. Depois, para o *clustering* os centróides direcionam qual interesse implementa um determinado elemento. Para descobrir se um determinado elemento pode implementar mais de um interesse se precisa utilizar outras técnicas complementares para conhecer as relações entre vários elementos. O Código 4.5 XQuery é encarregado de anotar atributos/variáveis no modelo KDM. As linhas $1-11$ declaram *namespaces* e algumas variáveis. A linha 13 indica que sejam procurados elementos dentro do pacote *CodeModel*. As linhas $14-16$ indicam em que classe, método e variável deve ser anotado o interesse. Por fim, a linha 17 faz a marcação no modelo.

[2] http://www.w3.org/TR/xquery/

```
1  declare namespace xmi="http://www.omg.org/XMI";
2  declare namespace xsi="http://www.w3.org/2001/XMLSchema−instance";
3  declare namespace action="http://www.eclipse.org/MoDisco/kdm/action";
4  declare namespace code="http://www.eclipse.org/MoDisco/kdm/code";
5  declare namespace kdm="http://www.eclipse.org/MoDisco/kdm/kdm";
6  declare namespace source="http://www.eclipse.org/MoDisco/kdm/source";
7  declare variable $concern as xs:string external;
8  declare variable $class as xs:string external;
9  declare variable $method as xs:string external;
10 declare variable $property as xs:string external;
11 declare variable $kind as xs:string external;
12
13 for $a in //kdm:Segment/model[@xsi:type="code:CodeModel" and @name != "externals"]
14             //codeElement[@xsi:type="code:ClassUnit" and @name=$class]
15             //codeElement[@xsi:type="code:MethodUnit" and @name=$method]
16             //codeElement[@xsi:type="code:StorableUnit" and @name=$property and @kind=$kind and empty(@concern)]
17 return insert node attribute concern {$concern} as last into $a
```

Código 4.5: Anotação do modelo KDM com XQuery

4.7 Considerações finais

Nesse capítulo apresentou-se a técnica combinada para a mineração de interesses transversais a qual consiste em uma biblioteca de interesses e um algoritmo de *clustering* de strings *K-means* modificado. É importante destacar que embora OCL é uma linguagem poderosa utilizada em modelos ou metamodelos com especificações MOF (*Meta-Object Facility*), escrever código eficiente (em termos de clareza e expressividade) na OCL é geralmente uma tarefa difícil porque apesar da sua aparente natureza "declarativa", pode ser considerada como uma linguagem de navegação de baixo nível devido a que todos os detalhes da navegação através dos modelos têm que ser especificados (VAZIRI; JACKSON, 2000; CUADRADO *et al.*, 2009).

Outro ponto a ser comentado é que pelas características dos arquivos KDM, não foi possível apresentar um inteiro. Para sistemas pequenos em torno de 5K s arquivos KDM podem ter um tamanho superior a 5 megabytes com mais de 2000 páginas

A técnica proposta foi implementada em um plug-in para Eclipse^{TM} framework e é descrita no Capítulo 5. A avaliação da técnica é apresentada no Capítulo 6.

Capítulo 5

SUPORTE PARA A MINERAÇÃO DE INTERESSES EM ADM: CCKDM

Esse capítulo apresenta o CCKDM, um plug-in desenvolvido em Eclipse™ para implementar a técnica combinada de biblioteca de interesses e clustering de strings descrita no Capítulo 4. Descreve-se a arquitetura do plug-in e as funcionalidades fornecidas.

5.1 Tecnologias Envolvidas

CCKDM foi implementado como um plug-in Eclipse™. É apoiado por três plugins que são detalhados em seguida:

1. Modisco™ é um framework estensível para desenvolver ferramentas dirigidas a modelos com o intuito de modernizar sistemas legados. Fornece vários *discoverers* que são APIs que geram modelos do código fonte. Por exemplo, é capaz de gerar modelos UML, modelos a partir de código JSP e modelos KDM, entre outros.

2. *Eclipse Modeling Framework* (EMF) é um framework de modelamento fornecido pelo Eclipse™. Permite navegar nos modelos KDM que foram gerados pelo Modisco™ (BRUNELIERE *et al.*, 2010).

3. *Model Development Tools* (MDT) provê ferramentas para implementar e desenvolver modelos. Especificamente, para esse trabalho foi usado OCL.

Além dos plugins utilizados, os algoritmos de CCKDM foram implementados mediante a combinação de duas linguagens de programação; Java e Groovy [1]. CCKDM utiliza intensiva-

[1] http://groovy.codehaus.org/

mente arquivos XML e CSV. Essa é a razão da escolha de utilizar Groovy pela simplicidade da sintaxe e completamente integrada com Java. O Código 5.1 apresenta código Groovy para manipular a biblioteca de interesses. O método *getConcerns* obtém uma lista de nome de interesses e o método *getPackages* obtém pacotes da biblioteca de interesses.

```
1  public ArrayList<String> getConcerns(String path)
2  {
3          ArrayList<String> concernName = new ArrayList<String>();
4          GPathResult library = new XmlSlurper().parse(new File(path + "/ConcernLibrary.xml"));
5          def concerns = library.'Concern'.findAll();
6          for (int i= 0; i< concerns.size(); i++)
7                  concernName.add(concerns[i].@name.toString());
8          return concernName;
9  }
10
11 public ArrayList<String> getPackages(String path, String concern)
12 {
13         ArrayList<String> packageName = new ArrayList<String>();
14         GPathResult library = new XmlSlurper().parse(new File(path + "/ConcernLibrary.xml"));
15         def packages = library.Concern.find{it.@name == concern}.Package;
16         packages.each{ it →
17
18                 packageName.add(it.@name.toString());
19         }
20         return packageName;
21 }
```

Código 5.1: Código Groovy

Como se indicou no Capítulo 4, a biblioteca de interesse é implementada mediante um arquivo XML. Por exemplo, o método *getConcerns* retorna uma lista de interesses e o método *getPackages* retorna uma lista de pacotes para um determinado interesse. Pode-se ver que os códigos apresentados são pequenos e fáceis de entender.

Outra tecnologia utilizada foi BaseX a qual é um motor de banco de dados XML e Processador XPath/XQuery 3.0, leve, de alto desempenho, escalável e de código livre além de proporcionar as APIs para java. Cada vez que um modelo KDM é vai ser marcado o modelo é carregado pelo BaseX criando um banco de dado com o modelo. As operações realizadas com XQuery são executadas sobre o motor de BaseX utilizando as APIs para Java. Por último, o motor de banco de dados SQL utilizado para persistir as estruturas de código fonte do modelo KDM é um Apache Derby embutido. Finalmente, a interface de usuário foi desenvolvida utilizando o framework SWT. Consiste em uma janela com 4 abas e as opções correspondentes para realizar a mineração de interesses.

5.2 Arquitetura

A Figura 5.1 apresenta a visão lógica da arquitetura de CCKDM. Pode-se identificar duas camadas; a camada de interface de usuário e a camada do modelo.

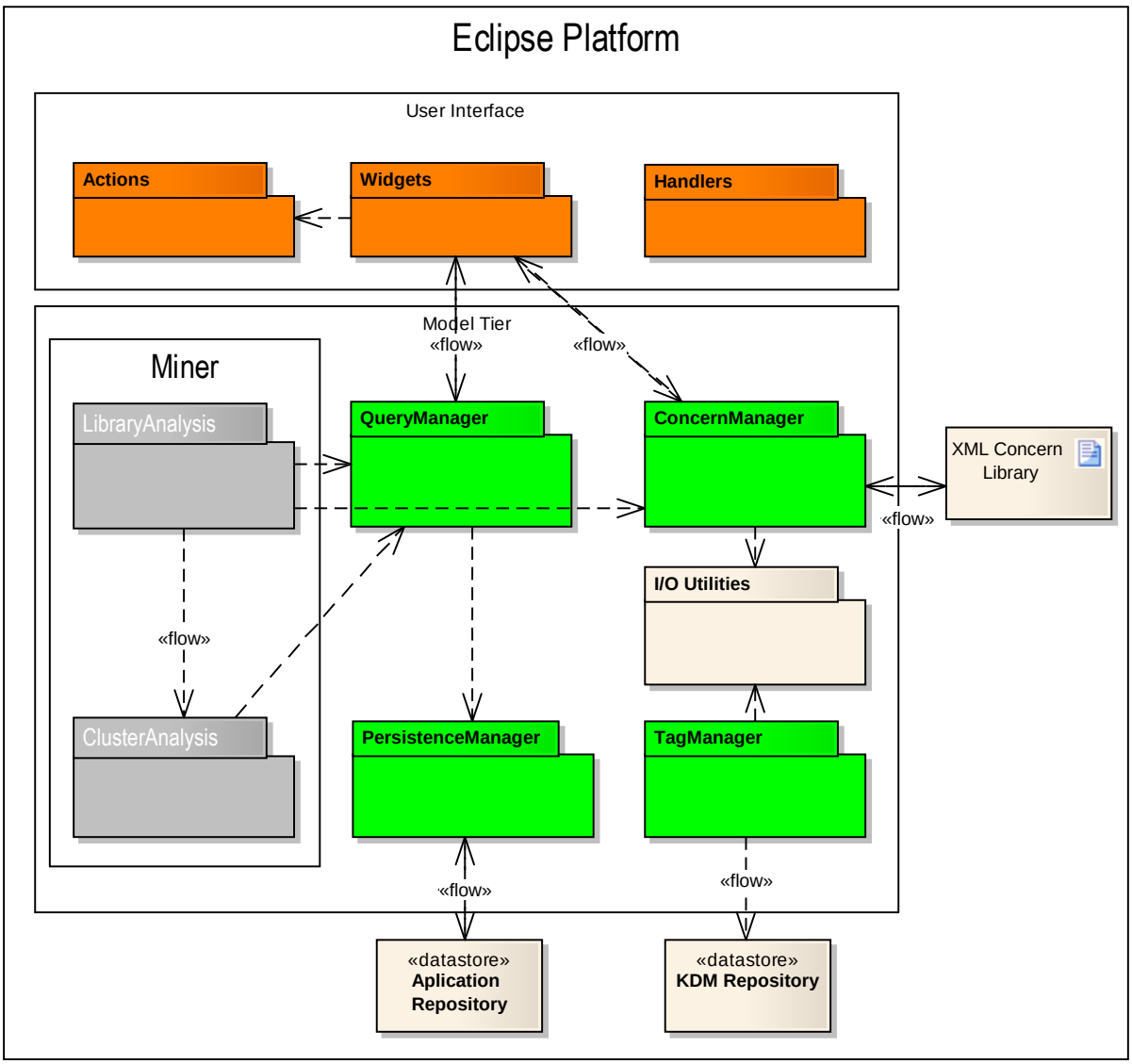


Figura 5.1: Visão lógica da arquitetura CCKDM

A camada de interface de usuário está composta por três módulos; *Actions*, *Widgets* e *Handlers*. O módulo *Actions* implementa a funcionalidade do menu *popup* pelo qual o usuário começa o processo de mineração. O módulo *Handlers* implementa a funcionalidade para carregar um arquivo KDM e o módulo *Widgets* implementa a janela principal com os elementos necessários para que o usuário possa interagir com o plugin. Esse módulo é dependente do módulo *Actions*.

A camada do modelo realiza as operações de mineração de interesses e a comunicação com

a biblioteca de interesses e os repositórios. Os módulos encarregados da mineração são dois; *LibraryAnalysis* e *ClusterAnalysis*. *LibraryAnalysis* faz a mineração pela biblioteca de interesses a qual depende do gerenciador de consultas *QueryManager* e da biblioteca de interesses que é gerenciado pelo módulo *ConcernManager*. As saídas do módulo *LibraryAnalysis* são as entradas para o modulo *ClusterAnalysis* o qual implementa o algoritmo de *clustering*. O módulo *ClusterAnalysis* também depende do módulo *QueryManager*. O módulo *PersistenceManager* gerencia as conexões com o banco de dados para persistir os elementos da aplicação. O módulo *TagManager* realiza as marcações dos interesses nos arquivos KDM utilizando um repositório de dados XML. Por fim, o módulo *I/O Utilities* realiza operações de entrada e saída com arquivos.

5.3 Processo de Uso do CCKDM

Para começar o processo com CCKDM, precisa-se de um projeto Java para Eclipse™ carregado no *Package Explorer*. O projecto deve conter o código fonte ou um arquivo XMI com o modelo KDM. A Figura 5.2 apresenta a forma de carregar um modelo KDM. Depois de carregado, um novo projeto Java automaticamente é criado com o novo arquivo.

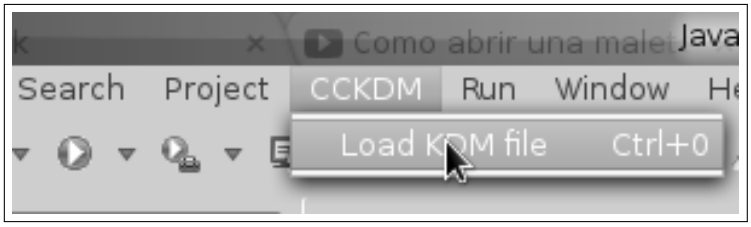


Figura 5.2: Carregar um modelo KDM

A Figura 5.3 mostra como se inicia o processo de mineração. O usuário deve selecionar o projeto no *Package Explorer* com o segundo botão para abrir a janela pop-up que tem a opção para iniciar o processo de mineração de interesses.

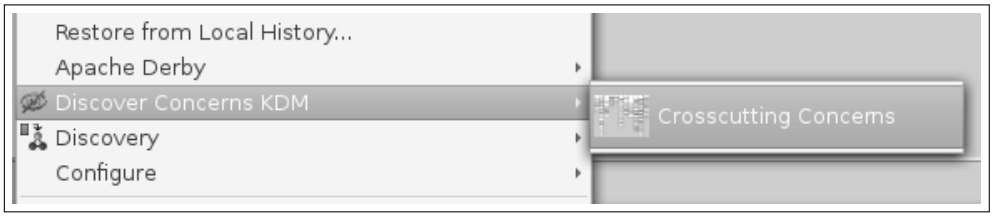


Figura 5.3: Iniciar o processo CCKDM

Após começar o processo, se for a primeira vez que o projeto é selecionado, então CCKDM vai gerar o metamodelo KDM (só se o projeto contém código fonte), a biblioteca de interesses

que vem por default (*Persistênce*, *Logging* e *Authentication*) e identificar estruturas relacionadas com código fonte do modelo KDM para serem persistidos no banco de dados. Para que o usuário saiba a ação que o CCKDM está realizando na etapa de coleta de dados, uma barra de progresso com informações é apresentada. A Figura 5.4 apresenta essa barra de progresso.

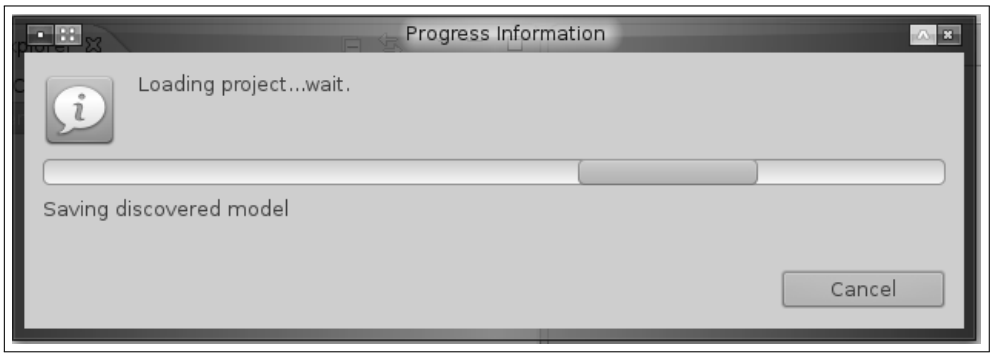


Figura 5.4: Barra de progresso com informações das ações sendo executadas

A Figura 5.5 apresenta a janela principal mostrada para o usuário após carregar as informações do modelo KDM. A Figura foi marcada com os números 1,2 e 3 para facilitar sua explicação.

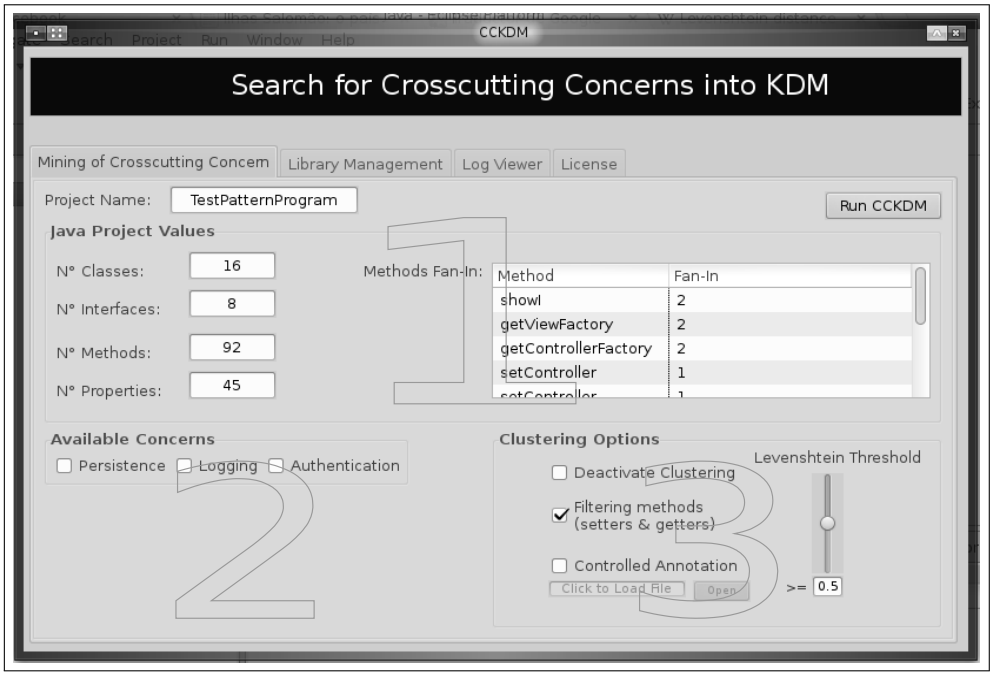


Figura 5.5: Janela principal

A parte esquerda da região 1 apresenta as informações do projeto carregado tais como a quantidade de classes, interfaces, métodos, atributos e variáveis. Com essa informação o usuário ter uma ideia do tamanho do sistema a ser analisado. A parte direita da região 1 apresenta o valor fan-in dos métodos do sistema, onde os valores altos podem indicar presença de inte-

resses transversais. A região 2, apresenta as caixinhas de seleção que representam os interesses carregados no CCKDM. O usuário deve selecionar pelo menos um interesse para começar o processo de mineração. Quando novas regras são adicionadas na biblioteca de interesses, a região automaticamente é atualizada. A região 3 apresenta opções relacionadas com a técnica de *clustering*. Com a primeira opção o usuário pode desativar ou ativar a mineração de interesses utilizando a técnica de *clustering*. Com a segunda opção o usuário pode filtrar os métodos *getters* e *setters*. Com a terceira opção o usuário ativa ou desativa o sub processo de seleção manual de interesses procurados pela técnica de *clustering*. Para isso o usuário deve selecionar o arquivo CSV que contém os elementos identificados pelo algoritmo de *clustering*. A Figura 5.6 mostra como é feita a carga do arquivo.

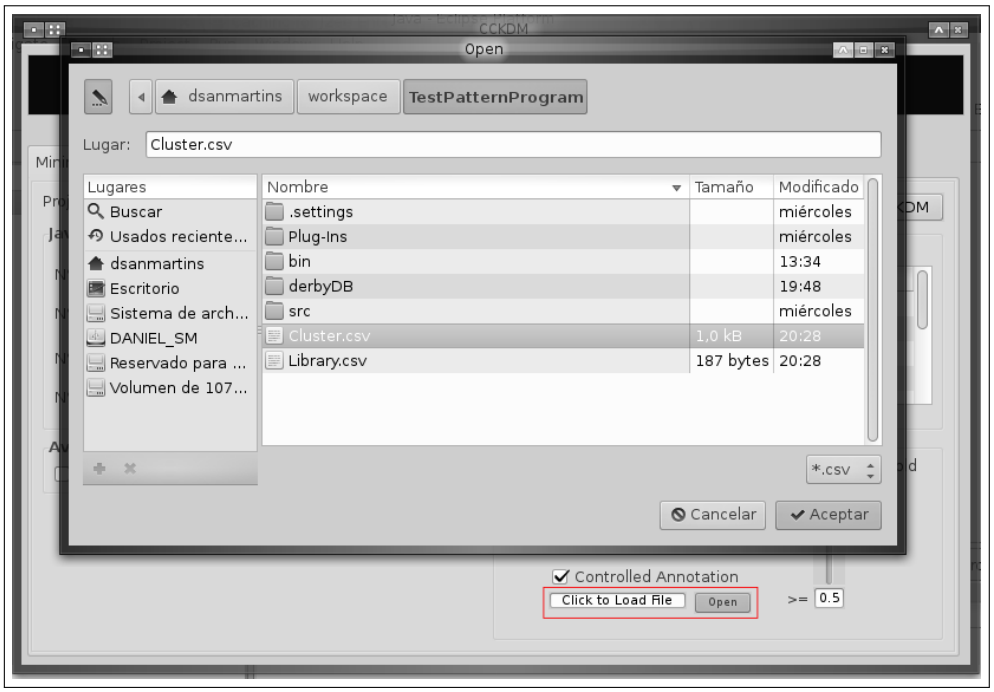


Figura 5.6: Carregando o arquivo CSV com os resultado do *clustering*

Finalmente, o usuário pode indicar o valor da distância *Levenshtein*, ou seja o nível de similaridade dos strings agrupados pela técnica de *clustering*.

A Figura 5.7 apresenta o gerenciamento da biblioteca de interesses. Na parte esquerda da janela o usuário pode inserir ou editar regras de interesses. Para inserir uma regra o usuário deve seguir o seguinte padrão; [*nome do interesse*],[*pacote*],[*elementos*..N] por linha. Podem-se inserir várias regras ao mesmo tempo em um só passo o que facilita a experiência do usuário e é uma das melhoras em relação ao COMSCID.

O árvore do lado direito mostra as regras carregadas no CCKDM. Para eliminar uma regra o usuário deve selecionar um interesse e abrir uma janela pop-up com o segundo botão e clicar

em eliminar. Esse árvore corresponde a o arquivo XML apresentado na Figura 4.3.

Cada vez que uma regra é adicionada ou eliminada, automaticamente a região 2 da Figura 5.5 é atualizada.

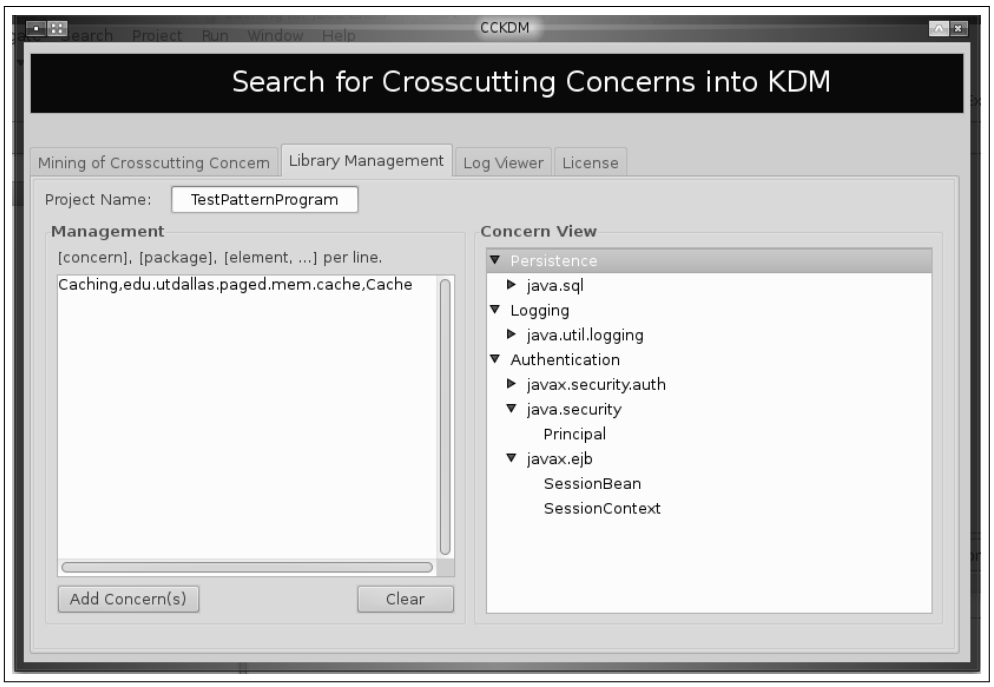


Figura 5.7: Biblioteca de interesses

A Figura 5.8 apresenta os logs gerados pelo CCKDM onde o usuário pode visualizar por meio da ferramenta os elementos que possivelmente implementam algúm interesse. O lado esquerdo apresenta os elementos identificados pela biblioteca de interesses e o lado direito os elementos identificados pela técnica de *clustering*.

Finalmente, a Figura 5.9 apresenta todos os elementos gerados pelo CCKDM enquadrados em vermelho. O nome do projeto analisado é *TestPatternProgram*. Pode-se ver que o CCKDM gerou dois modelos KDM; o modelo *TestPatternProgram0_KDM.xmi* e o modelo *TestPatternProgram_KDM.xmi*. O modelo *TestPatternProgram0_KDM.xmi* é o KDM original e *TestPatternProgram_KDM.xmi* é o KDM com as anotações. Além disso, CCKDM gerou o banco de dados *derbyDB*, a biblioteca de interesses *ConcernLibrary.xml* e 4 arquivos de registros (arquivos CSV e arquivos logs).

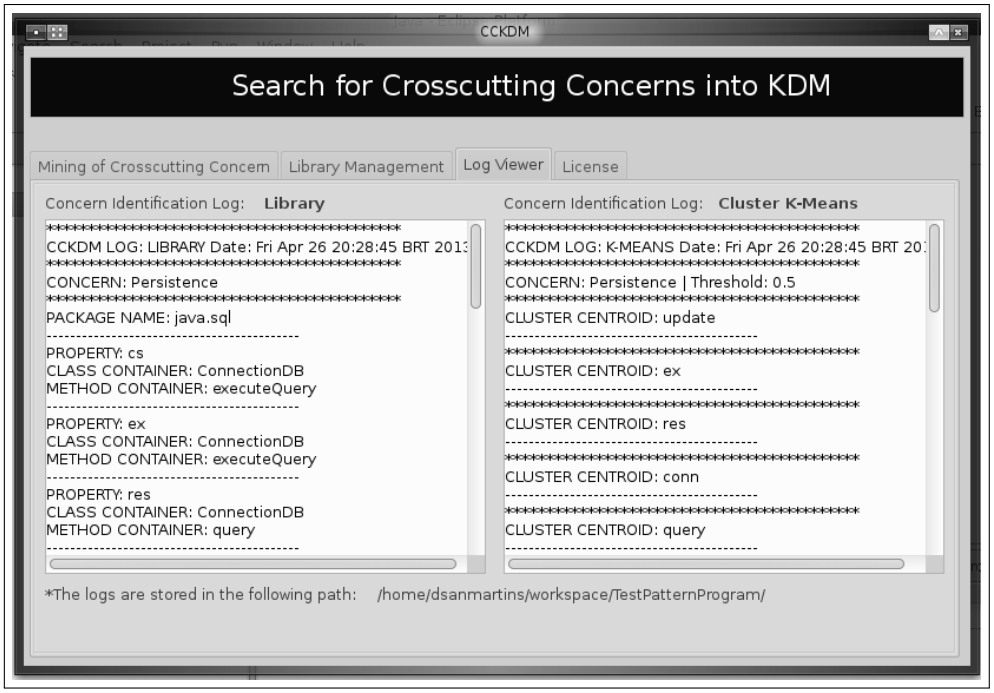


Figura 5.8: Logs gerados pelo CCKDM

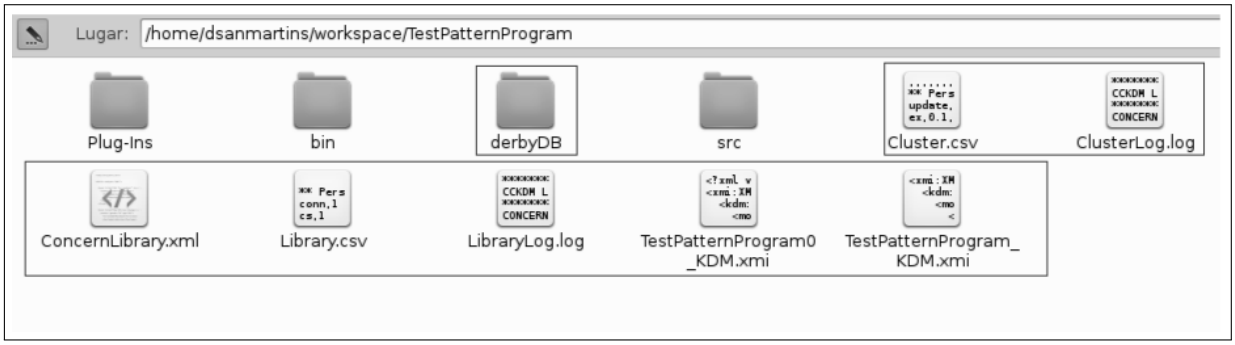


Figura 5.9: Elementos gerados pelo CCKDM

5.4 Considerações finais

Neste capítulo foi apresentado o plug-in para Eclipse™ CCKDM. Ele implementa a técnica combinada de mineração de interesses em modelos KDM e pode-se descarregar no seguinte endereço web: `http://sourceforge.net/projects/cckdm/`. O plug-in tem licença Apache versão 2.0 e é livre para ser usado e modificado.

Capítulo 6

AVALIAÇÃO

Neste capítulo, apresenta-se a avaliação da técnica proposta no Capítulo 4. O primeiro passo consistiu em analisar se existiam diferenças significativas ao utilizar diferentes valores Levenshtein. Em seguida, com o resultado dessa análise, realizou-se uma avaliação em que se compararam valores de precisão e cobertura com outras técnicas de mineração de interesses.

6.1 Considerações Iniciais

Segundo Travassos (2002), existem duas técnicas para avaliar uma abordagem: experimentos e estudos de caso. A realização de experimentos provê uma maneira de avaliação baseada em comparação direta, permitindo aos pesquisadores investigar qual o impacto da tecnologia no processo de maneira detalhada. O estudo de caso está preocupado em avaliar os benefícios de uma abordagem para verificar se as mudanças no processo proporcionarão o resultado esperado.

A técnica escolhida para esse trabalho foi o estudo caso. Ao contrário de outros estudos de caso empíricos de mineração de interesses, que descrevem com detalhes os tipos de candidatos descobertos por suas ferramentas, este trabalho concentra-se em comparar valores de precisão e cobertura (MANNING *et al.*, 2008). A razão da escolha dessas métricas é porque são amplamente aceitadas na avaliação de técnicas de mineração de interesses (DURELLI *et al.*, 2013).

Antes de começar com o estudo de caso, foi realizada uma avaliação da métrica distância de *Levenshtein*. A razão disso foi conhecer se existiam diferenças significativas na precisão e cobertura utilizando 5 valores *Levenshtein*. A avaliação é detalhada na Seção 6.3.

Em seguida definem-se as métricas utilizadas para avaliar o caso de estudo:

- Precisão (*precision*) é a porcentagem que define a quantidade de elementos encontrados pela técnica e que verdadeiramente implementam um determinado interesse sobre o total de elementos encontrados pela técnica. Define-se a fórmula em seguida:

$$P = \frac{VerdadeirosPositivos}{VerdadeirosPositivos + FalsosPositivos}$$

- Cobertura (*recall*) é a porcentagem que define a quantidade de elementos encontrados pela técnica e que verdadeiramente implementam um determinado interesse sobre o total de elementos que implementam o interesse. Define-se a fórmula em seguida:

$$C = \frac{VerdadeirosPositivos}{VerdadeirosPositivos + FalsosNegativos}$$

6.2 Sistemas em estudo

Na tabela 6.1 encontram-se os sistemas que foram usados para realizar a avaliação da métrica distância de *Levenshtein* e o caso de estudo. Pode-se observar que o tamanho dos sistemas é razoável (KLOCK $\leq$ 9K), porque os torna adequados para realizar uma análise manual de interesses e assim poder calcular a cobertura além da precisão. Como o tamanho dos sistemas é similar e o interesse de Persistência está implementado da mesma forma, é factível poder realizar as comparações correspondentes.

	Sistemas	LOC	Classes	Métodos	Propriedades
1	HealthWatcher v10	8K	118	894	1290
2	PetStore v1.3.2	9K	228	1917	3002
3	ProgradWeb	5K	133	254	4182

Tabela 6.1: Sistemas em estudo

O *HealthWatcher* é um sistema de reclamações desenvolvida para instituições de saúde com o intuito de melhorar a qualidade dos seus serviços. Utiliza uma interface web para registrar as reclamações além de realizar outras operações. O *PetStore* é um sistema que permite comprar bens pela internet e o *ProgradWeb* é um sistema de gerenciamento de informações acadêmicas da Universidade Federal de São Carlos.

6.3 Avaliação dos Valores Levenshtein

O objetivo desta seção é determinar quais são os valores *Levenshtein* mais adequados para agrupar strings com o algoritmo de *clustering* e assim estabelecer um precedente ao analisar

sistemas na procura de interesses. É claro que os valores extremos não podem ser levados em conta porque o algoritmo de *clustering* agruparia strings muito diferentes em grandes quantidades aumentando o número de falsos positivos ou pelo contrario, agrupar strings bem similares aumentando o número de falsos negativos.

Os métodos estatísticos multivariados estudam o comportamento de três ou mais variáveis simultaneamente. Eles são utilizados principalmente para pesquisar quais são as variáveis menos representativas e assim eliminá-las. Dessa forma, os modelos são simplificados para compreender a relação entre vários grupos de variáveis (MORRISON, 1976).

A Tabela 6.2 foi gerada após analisar manualmente os sistemas da Seção 6.2 para o interesse de Persistência. O primeiro passo consistiu em aplicar a mineração pela biblioteca de interesse para obter os centróides. No segundo passo aplicou-se o algoritmo de *clustering* utilizando os centróides do primeiro passo para agrupar os strings. Por fim, o terceiro passo consistiu em analisar se os strings agrupados implementavam o interesse de Persistência para obter a precisão e a cobertura de cada conjunto agrupado (ver Anexo A, Anexo B e Anexo C). Os valores apresentados na Tabela 6.2 são as médias ponderadas de precisão e cobertura (as que agruparam mais quantidade de strings têm maior peso) para 5 valores *Levenshtein* (entre um intervalo de valores de 0.1 - 0.9) nos 3 sistemas. Dessa forma, realizou-se uma MANOVA que é uma forma generalizada do método de análise de variância (ANOVA) e abrange os casos em que existe mais de uma variável dependente.

	Valores Levenshtein para o Algoritmo de Clustering									
	0.3		0.4		0.5		0.6		0.7	
Sistemas	P	C	P	C	P	C	P	C	P	C
HealthWatcher v10	34,31%	100%	41,53%	58,03%	46,68%	52,23%	46,02%	46,45%	46,74%	43,76%
PetStore v1.3.2	37,35%	100%	38,08%	68,29%	41,14%	50,88%	50,43%	47,82%	36,00%	8,60 %
ProgradWeb	49,03%	100%	28,47%	47,24%	29,23%	36,77%	25,58%	6,18 %	45,45%	6,64 %

Tabela 6.2: Precisão e cobertura para Persistência

É importante notar que esses valores são dependentes dos termos cadastrados na biblioteca e poderiam melhorar se os termos da biblioteca foram melhorados.

6.3.1 Pergunta de Pesquisa

Nesta seção se apresenta a formulação da pergunta de pesquisa que é contestada por meio de um análise estatístico nas seguintes seções. A razão desta pergunta, é para conhecer se a escolha de um determinado valor *Levenshtein* afeta significativamente ou não os valores de precisão e cobertura em conjunto. Em seguida define-se a pergunta:

RQ1: Existem diferenças significativas de precisão e cobertura para os valores *Levenshtein*

da Tabela 6.2?

6.3.2 Hipóteses

A pergunta **RQ1** foi formalizada nas seguintes hipóteses de pesquisa a fim de realizar o teste estatístico:

Hipóteses nula, $\mathbf{H}_0$ não existem diferenças significativas de precisão e cobertura para os valores *Levenshtein* da Tabela 6.2 (as médias são iguais), pode ser formalizado como:

$\mathbf{H}_0 : \mu_{0.3} = \mu_{0.4} = \mu_{0.5} = \mu_{0.6} = \mu_{0.7}$

Hipóteses alternativa, $\mathbf{H}_1$ existem diferenças significativas de precisão e cobertura para os valores *Levenshtein* da Tabela 6.2 (as médias não são iguais), pode ser formalizado como:

$\mathbf{H}_1 : \mu_{0.3} \neq \mu_{0.4} \neq \mu_{0.5} \neq \mu_{0.6} \neq \mu_{0.7}$

6.3.3 Teste de Hipóteses

Para realizar o teste de hipóteses, foi utilizado o software R[1], um software estatístico *open source*. Foram aplicados 4 testes estatísticos para o conjunto de dados e os resultados são sumarizados na Tabela 6.3.

Test	Value	F-Value	num Df	den DF	Pr(>F)
Wilks' Lambda	**0.49045**	**0.9628**	**8**	**18**	**0.4935**
Pillai's Trace	0.53649	0.91645	8	20	0.5232
Hotelling-Lawley	0.98399	0.98399	8	16	0.4828
Roy's Greatest	0.92457	2.3114	4	10	0.1288

Tabela 6.3: Resultados do teste MANOVA para os valores Levenshtein da Tabela 6.2

Normalmente, não há nenhuma diferença no resultado ao aplicar os testes apresentados na Tabela 6.3, porém *Wilks Lambda* é uma boa escolha em geral. *Wilks Lambda* é um F-test (Fischer) e é interpretado da mesma forma como é interpretada para a versão univariada. Pode-se observar que o valor $F - Value$ é de 0.9628 com 8 (numerador) e 18 (denominador) graus de liberdade. O valor $p - value$ ($Pr(>F)$) é >0.4935 o que não é um valor significativo. Por essa razão, de forma estatística a **hipóteses nula não é rejeitada**, ou seja não há diferenças significativas de precisão e cobertura nos valores *Levenshtein* (nas médias) da Tabela 6.2, embora a tabela apresente alguns valores que poderiam indicar o contrário. Então dessa forma, não se

[1] http://www.r-project.org/

precisa realizar um análise para determinar qual variável dependente (precisão ou cobertura) é a mais afetada pela escolha de um determinado valor *Levenshtein*.

Para validar graficamente os resultados, realiza-se uma segunda análise multivariada; análise canônica. A análise canônica é utilizada para determinar relações entre grupos de variáveis de um conjunto de dados. Os dados são divididos em 2 grupos, por exemplo um grupo X e um grupo Y. Logo esse análise encontra as relações entre X e Y, ou seja como dados do grupo X podem representar dados do grupo Y (HOTELLING, 1936). Na Figura 6.1, apresenta-se o gráfico da análise canônica.

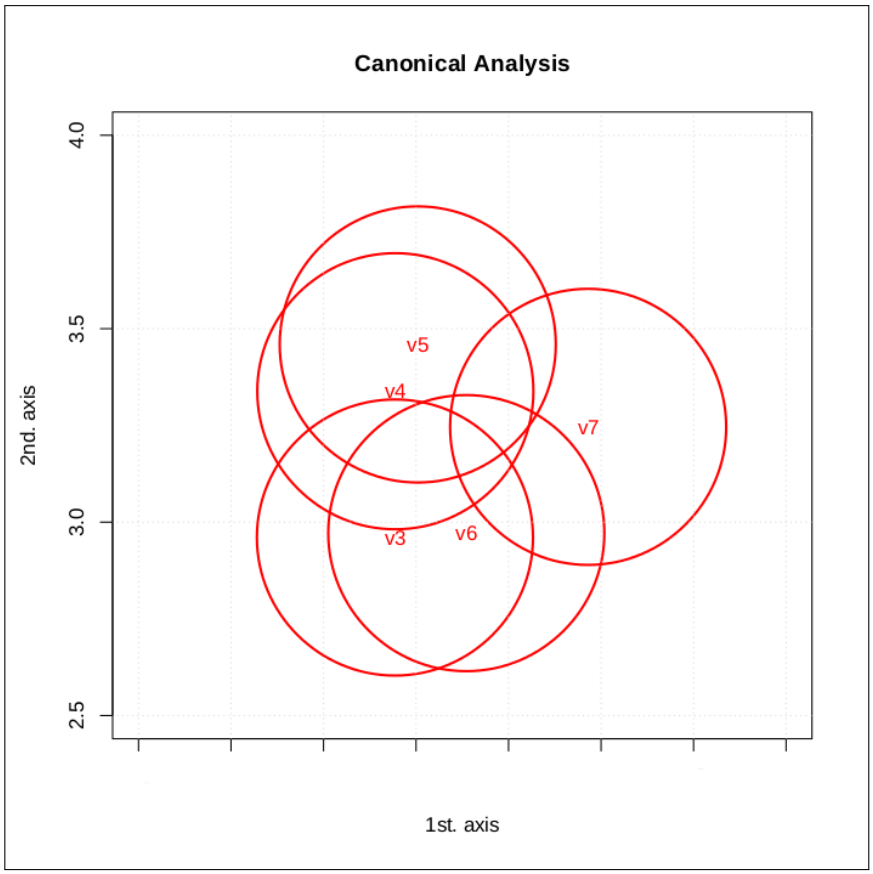


Figura 6.1: A representação das distâncias médias entre indivíduos

O primeiro eixo (1st axis) pode-ser interpretada como os 5 valores *Levenshtein* e a evolução da precisão e a cobertura sobre sobre esses valores. Pode-se observar três grupos; o primeiro grupo esta conformado por $v4$ e $v5$ onde suas áreas se sobrepõem em uma grande extensão. O segundo grupo esta conformado por $v3$ e $v6$ onde suas áreas também se sobrepõem em uma grande extensão e o terceiro grupo esta conformado por $v7$. Se observa que entre esses grupos existe uma sobreposição das áreas dos círculos porque as distâncias dos centros dos círculos não é significativa. Então graficamente esse análise apóia o analise anterior que determina que não existem variações significativas nas médias dos grupos. Finalmente, a Figura 6.2 apresenta

um diagrama de caixa com a média da precisão e cobertura como variável de estudo para os 5 valores *Levenshtein*.

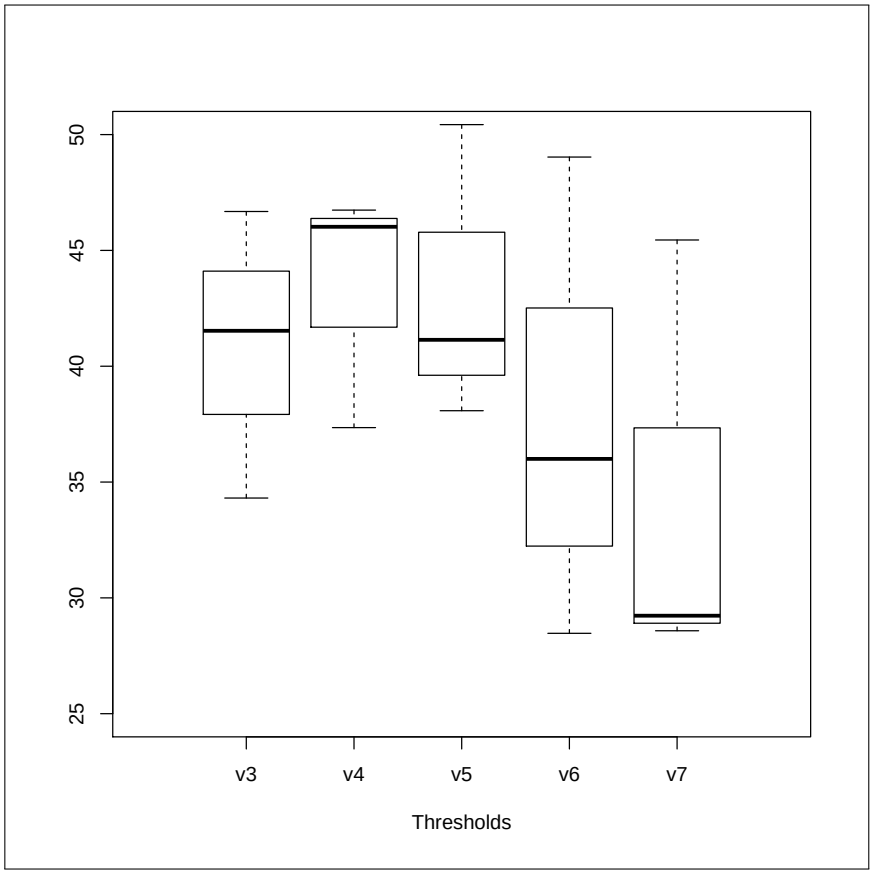


Figura 6.2: Diagrama box-plot de precisão e cobertura para os 5 valores *Levenshtein*

Embora a análise estatística mostra que não existem variabilidades significativas de precisão e cobertura em conjunto para os distintos valores *Levenshtein*, é possível observar algumas variações nas medianas. Por exemplo, as médias para 0.3, 0.4 e 0.5 têm valores mais altos que 0.6 e 0.7. Da mesma forma, é possível observar que a variabilidade dos valores *Levenshtein* 0.6 e 0.7 são maiores que o resto. Como se precisa que os valores de precisão e cobertura sejam mais homogêneos então, são levadas em conta os valores precisão e cobertura para 0.3, 0.4 e 0.5 para realizar as comparações com outras técnicas/ferramentas de mineração de interesses.

6.4 Avaliação Empírica

A avaliação empírica realizada consistiu em duas análises. A primeira foi em comparar a técnica de *clustering* para os valores *Levenshtein* 0.3, 0.4 e 0.5 com as técnicas *XS-can* (NGUYEN *et al.*, 2011) e Timna (SHEPHERD *et al.*, 2005a) porque são técnicas que pelo menos apresentam um valor de avaliação para PetStore e HealthWatcher de cobertura ou preci-

são. A segunda foi comparar a técnica combinada (biblioteca de interesses mais *clustering*) com os sistemas mencionados na Tabela 6.2. *XScan* é uma técnica de mineração de interesses que utiliza uma variação da técnica de mineração de interesses por Fan-in e *Timna* é um framework que utiliza 5 técnicas de mineração não combinadas que são utilizadas dependendo de algumas marcações manuais no código fonte. As técnicas utilizadas pelo Timna são *Fan-in*, *Pairings*, *Code clone* e *No parameters*. Nessa avaliação comparou-se o valor de precisão da Timna para o interesse de Persistência produto da análise realizada para o sistema *PetStore* e foram utilizadas as 5 técnicas que ele implementa.

A Tabela 6.4 apresenta as comparações da técnica de *clustering* com as técnicas descritas anteriormente.

	Cluster-0.3		Cluster-0.4		Cluster-0.5		XScan		Timna	
Sistemas	P	C	P	C	P	C	P	C	P	C
HealthWatcher v10	34,31%	100%	41,53%	58,03%	46,68%	52,23%	-	100%	-	-
PetStore v1.3.2	37,35%	100%	38,08%	68,29%	41,14%	50,88%	-	-	93.80%	-

Tabela 6.4: Comparações de precisão e cobertura para Persistência com *clustering*

XScan apresenta somente o valor de cobertura para *HealthWatcher* e *Timna* apresenta só o valor de precisão para *PetStore*. Pode-se observar que o algoritmo de *clustering* obtém 100% de cobertura para os dois sistemas mas com um nível baixo de precisão para uma distância *Levenshtein* de 0.3. Isso significa que neste nível não se tem falsos negativos mas uma grande quantidade de falsos positivos.

Para *HealthWatcher* os melhores valores de precisão e cobertura foi utilizando o valor *Levenshtein* 0.5 porque foi em torno do 50%. Para *PetStore* o melhor valor *Levenshtein* foi o 0.4 porque a cobertura foi alta (38.29%) e o valor da precisão é similar a precisão utilizando o valor *Levenshtein* 0.5. É claro que os valores de precisão e cobertura ainda são baixos se foram comparados com *XScan* e *Timna* porque á técnica não está combinada, então não leva em conta os centróides iniciais identificados pela técnica de biblioteca de interesses.

A Tabela 6.5 apresenta as comparações da técnica CCKDM com as técnicas descritas anteriormente.

	CCKDM-0.3		CCKDM-0.4		CCKDM-0.5		XScan		Timna	
Sistemas	P	C	P	C	P	C	P	C	P	C
HealthWatcher v10	100%	100%	100%	80%	100%	76,11%	-	100%	-	-
PetStore v1.3.2	95%	100%	95,73%	84,15%	98,79%	75,44%	-	-	93.80%	-

Tabela 6.5: Comparações de precisão e cobertura para Persistência com a técnica combinada

Pode-se observar que os valores de precisão e cobertura aumentaram significativamente ao utilizar a técnica combinada CCKDM para os dois sistemas analisados. É importante destacar

os benefícios de utilizar a técnica de *clustering* porque por exemplo, foi capaz de detetar *PersistenceMechanismException* e *PersistenceMechanism* pertencentes a APIs que não formam parte da API de Java, então a biblioteca de interesses por si só não poderia identificá-las.

6.5 Conclusões Finais

Neste capítulo se apresentou uma avaliação da técnica apresentada no Capítulo 4 levando em conta o interesse de Persistência. Primeiro se realizou um análise estatístico para determinar se existiam diferenças significativas para cinco valores *Levenshtein* levando em conta duas variareis independentes, precisão e cobertura para a técnica de clustering de strings. A análise determinou que não há diferenças significativas nas médias dos cinco conjuntos, porém os primeiros três conjuntos apresentam uma menor variabilidade.

Então, levando em conta a técnica combinada e comparando os resultados de precisão e cobertura (para valores *Levenshtein* 0.3, 0.4 e 0.5) com duas técnicas encontradas na literatura que apresentam valores de precisão e cobertura para PetStore e HealthWatcher, os resultados indicam que a técnica combinada atinge valores acima de 90% para a precisão e cobertura na mineração do interesse de Persistência, atingindo a mesma efetividade que as técnicas utilizadas para comparar.

Capítulo 7

TRABALHOS RELACIONADOS

Neste capítulo apresentam-se os trabalhos encontrados na literatura que foram considerados diretamente relacionados ao tema deste projeto. Como não foram encontradas propostas que realizam mineração de interesses diretamente no metamodelo KDM, considerou-se inicialmente trabalhos que realizam mineração de interesses em modelos e depois os que realizam mineração em código-fonte.

7.1 Mineração de Interesses em Modelos

Zhang *et al.* (2008b) apresentam uma investigação sobre o aumento dos benefícios na utilização de técnicas de mineração de aspectos em níveis de abstração mais altos mediante algoritmos de mineração de aspectos em modelos. A principal contribuição é a capacidade de identificar interesses transversais nos estágios iniciais do desenvolvimento, para dar apoio na modularização deles por meio de aspectos na etapa do projeto de software.

Os autores realizam a identificação dos interesses em modelos construídos hierarquicamente com *The Generic Modeling Environment*(GME) (KARSAI *et al.*, 2004; GRAY *et al.*, 2001). GME permite configurar e adaptar metadados bem como validar as suas relações para a implementação de um determinado modelo. Um átomo (*atom*) é considerado a entidade mais básica e não pode possuir estruturas internas. Um modelo (*model*) é um outro tipo de entidade e pode conter outros tipos de entidades. Uma conexão (*connection*) representa a relação entre entidades. Os atributos (*attributes*) são usados para registrar informações relacionadas com as entidades do modelo. GME é um ambiente de modelagem multipropósito onde tem sido desenvolvidos modelos relacionados com diferentes domínios. A Figura 7.1 apresenta um modelo organizado hierarquicamente em vários sub-modelos com interesses transversais espalhados nesses sub-modelos. A setas representam que existem interesses que devem ser modularizados.

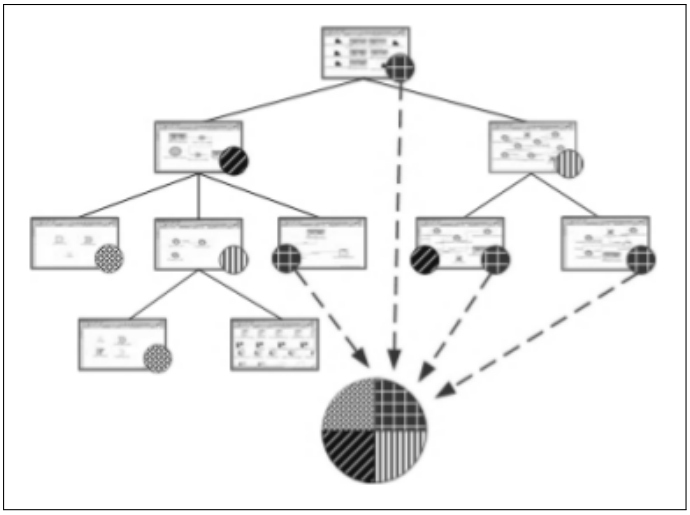


Figura 7.1: Interesses transversais espalhados em um modelo hierárquico

Para realizar a mineração dos modelos, os autores desenvolveram uma técnica baseada em detecção de clones. A ideia é identificar fragmentos do modelo que sejam similares em toda a hierarquia do modelo. A identificação dos clones é realizada com base nos conceitos de similaridade dos metadados apresentada na Tabela 7.1.

	Átomo	**Modelo**	**Conector**
Nível 1	Tipo	Tipo Elementos	Tipo Fonte Alvo
Nível 2	Tipo Nome	Tipo Nome Elementos	Tipo Nome Fonte Alvo
Nível 3	Tipo Nome Atributos	Tipo Nome Atributos Elementos	Tipo Nome Atributos Fonte Alvo

Tabela 7.1: Três níveis de similaridade (Traduzida de (ZHANG *et al.*, 2008b))

Define-se um átomo do modelo como uma combinação de; *tipo*, *nome* e *um conjunto de atributos*. Um modelo é uma combinação de elementos atômicos e conectores. Assim, por exemplo para o Nível 1, dois átomos são considerados clones quando os seus tipos são iguais e dois modelos são iguais se os seus tipos e todos os seus elementos são iguais. Pode-se observar que o Nível 1 é menos restritivo e o Nível 3 é mais restritivo. A técnica consiste em 4 passos e se detalham em seguida:

1. Pre-processamento do metamodelo: O primeiro passo consiste em avaliar os modelos para reduzir o número de comparações. Os modelos são particionados em diferentes grupos de entidades formando um conjunto de pares da seguinte forma:

   ```
   {Type-model}:{Type-element}
   ```

   onde *Type-model* é uma coleção de tipos cujas instâncias do modelo incluem alguns elementos comuns, e *Type-element* é uma coleção de elementos do modelo que compartilham *Type-model*. Por exemplo na Figura 7.2, *ModelA* e *ModelB* compartilham o elemento *AtomAB*. *ModelB* e *ModelC* ambos contém *AtomBC*. Neste caso, a partição do fragmento do metamodelo é a seguinte:

   ```
   {ModelA, ModelB} : {AtomAB}
   {ModelB, ModelC} : {AtomBC}
   ```

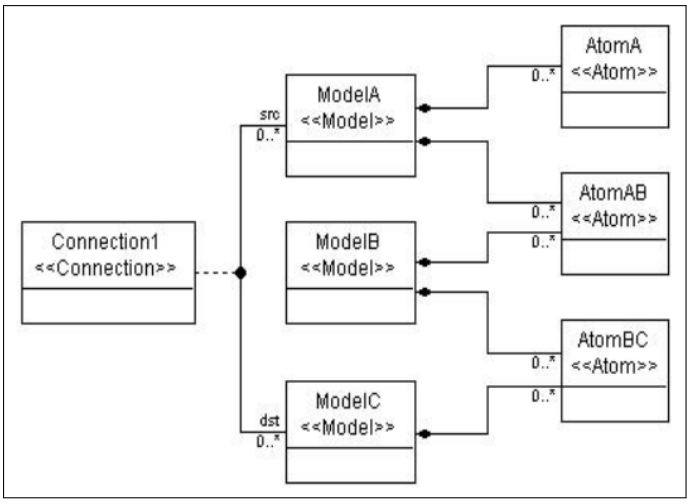


Figura 7.2: Fragmento de um metamodelo (ZHANG *et al.*, 2008b)

   Assim modelos que compartilham o mesmo tipo ou estejam no mesmo grupo são comparados.

2 Comparação de fragmentos de modelo: O segundo passo determina se os elementos de um par de sub-modelos são clones, realizando as comparações segundo a Tabela 7.1. Começa-se pela raiz de um sub-modelo, cada sub-modelo é comparado com outros que tenham o mesmo tipo ou sejam parte do passo 1.

3 Agrupamento de elementos similares: Todos os elementos clones são agrupados em um processo iterativo.

4 Filtro de interesses: Os elementos agrupados poderiam conter falsos positivos, então precisa-se filtrar os interesses que não são interesses.

Os autores realizaram um estudo de caso analisando um sistema embarcado projetado com GME relacionado com a aviação. Os interesses identificados estão relacionados com elementos do domínio do problema que podem ser interpretados por usuários com alguns conhecimentos do sistema.

As principais diferenças da abordagem apresentada no Capítulo 4 com a abordagem apresentada por Zhang *et al.* (2008b), são:

- Os autores procuram por certos padrões os quais podem ou não ser interesses transversais por meio de uma técnica de detecção de clones. Ao invés, a abordagem apresentada no Capítulo 4 procura por indícios de interesses transversais conhecidos segundo a biblioteca de interesses;
- Os autores não realizam marcações nos modelos e não especificam como é feita a modularização dos interesses.

Lengyel *et al.* (2009) apresentam uma técnica para identificar regras de restrição em modelos gerados pelo framework VMTS (*Visual Modeling and Transformation System*). VTMS é um ambiente de metamodelagem e modelagem específico de domínio que permite criar regras de restrição escritas em OCL. VTMS também permite criar regras de transformação entre modelos com suas respectivas regras de verificação. Porém, é frequente que as regras de verificação sejam repetidas e aplicadas em vários lugares em uma transformação, assim as regras de restrição entrecortam as regras de transformação e a gestão delas torna-se impraticável segundo os autores.

O processo para identificar as restrições nos modelos é o seguinte:

1. Coletar todas as restrições que aparecem nas transformações;
2. Identificar as restrições repetidas.
3. Nem todas as restrições são restrições transversais. Então, para cada restrição que apareça repetida o algoritmo decide se a restrição entrecorta ou não a transformação.
4. Modularizar as restrições extraídas.

O passo 3 é realizado por um algoritmo que atribui cores às restrições. Cada cor representa um interesse e o ideal é que cada restrição tenha uma só cor atribuída, isso significa que a restrição não entrecorta ao modelo. Por exemplo:

```
context Class inv NonAbstract: not self.abstract
```

A restrição *NonAbstract* representa o interesse que estabelece que se a classe é abstrata não deve ser processada.

```
context Table inv SourceClass:
self.helperNode.class->exists(c | (c.name = self.name))
```

A restrição *SourceClass* representa o interesse que estabelece que a tabela gerada tem que ter o mesmo nome que a classe fonte. Com essas duas restrições, o algoritmo atribui a cada um dos interesses uma cor diferente. Porém, se a restrição abrange mais de um interesse, então a coloração será composta:

```
context Class inv NonAbstractAndProcessed:
not self.abstract and not self.isProcessed
```

A restrição *NonAbstractAndProcessed* incorpora dois interesses: *(i)* as classes não devem ser abstratas e *(ii)* as classes não devem ser processadas. Neste caso, duas cores são atribuídos pelo algoritmo. As cores são atribuídas por meio de expressões meta OCL.

O passo 4 extrai a restrições e cria unidades modularizadas. Se a restrição tem exatamente um cor a modularização é imediata mas se tem mais de uma cor então aplicam-se relocalizações e decomposições a fim de eliminar o acoplamento dos interesses com as restrições.

As principal diferença da abordagem apresentada no Capítulo 4 com a abordagem apresentada por Lengyel *et al.* (2009) é que eles procuram por restrições transversais definidas com a linguagem OCL. Essas restrições têm uma estrutura específica então a identificação delas não é uma tarefa complexa. A complexidade reside em determinar se elas entrecortam mais de um elemento do modelo.

Finalmente, não foram encontrados trabalhos relacionados com mineração de interesses em modelo KDM até o momento em que esta dissertação estava sendo escrita.

7.2 Mineração de Interesses em Código Fonte

Zhang *et al.* (2008a) apresentam uma proposta de mineração de interesses chamada *Clustering Fan-in Based Analysis "CFBA"* para programas em Java e C. Essa proposta está baseada em duas técnicas combinadas de mineração, método de *clustering* e método de *Fan-in*. *Clustering* é utilizado para agrupar métodos que possam ter nomes semelhantes, ou seja mediante a similaridade de nomes dos métodos são agrupados em um *cluster* e *Fan-in* é utilizado para conhecer a quantidade de vezes que um método é referenciado no código fonte. A Figura 7.3 apresenta o processo completo de mineração e é detalhado em seguida.

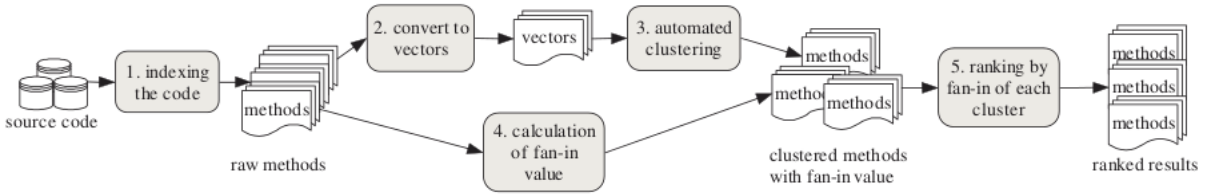


Figura 7.3: Técnica de mineração *"CFBA"* (ZHANG *et al.*, 2008a).

O primeiro passo da técnica *"CFBA"* é identificar todos os métodos e as chamadas a métodos envolvidos no código fonte. Cada elemento identificado é indexado por meio de um identificador único. Na segunda parte, os métodos analisados são convertidos em vetores baseados nos nomes dos métodos. Por exemplo, se um método tem o nome *fireSelectionChanged()*, então esse nome é separado em três tokens; *fire*, *Selection* e *Changed* os quais serão os atributos desse método. Assim, com um universo composto por todos os atributos de todos os métodos analisados (nomes de métodos separados quando tem uma maiúscula), o vetor correspondente para cada método conterá um 1, se o nome do método contém parte de um nome de método e um 0 em caso contrario. O terceiro passo é de agrupar os métodos similares mediante a técnica de *clustering*. Isso é feito mediante a métrica de *Jaccard Coeficient*, que é utilizado para calcular a similaridade de atributos binários. Define-se a métrica *Jaccard Coeficient* em seguida:

$$J(O_i, O_j) = \frac{N_{ij}}{N_i + N_j + N_{ij}}$$

Onde N_{ij} representa o total de número de atributos dos vetores O_i e O_j, onde ambos tenham o valor 1. N_i representa o total de atributos onde o atributo de O_i é 1 e o atributo de O_j é 0. N_J representa o total de atributos onde o atributo de O_i é 0 e o atributo de O_j é 1. O algoritmo cria vários *clusters* onde os métodos serão agrupados dependendo das similaridades dos nomes dos métodos. Se a similaridade dos nomes supera um valor limite de 0.3, esses métodos serão agrupados em um *cluster*. Paralelamente é calculado o valor do *Fan-in* de cada método. Na

quinta etapa, são calculados os *Fan-in* dos *Clusters* mediante a seguinte formula:

$$F(C) = \sigma f(m_i), m_i \in C$$

Onde f retorna o valor *Fan-in* de um determinado método. Os resultados são classificados e apresentados em ordem decrescente para recomendar os *clusters*. Os autores fizeram experimentos empíricos em dois sistemas *Linux* e *JHotDraw*, onde utilizaram duas métricas para avaliar os resultados de mineração, *concern coverage* e *true positives* baseados em métricas da precisão e cobertura. Os resultados foram superiores em relação a outras técnicas desenvolvidas como *Fan-in*, *identifier analysis* e *dynamic analysis*.

O trabalho apresenta uma técnica que combina duas técnicas de mineração com bons resultados, pelo qual, é um indicativo para continuar combinando técnicas para melhorar a cobertura dos aspectos à serem recomendados.

Nguyen *et al.* (2011) apresentam uma proposta para identificar unidades de código que compartilham interesses transversais. As unidades de código identificadas são recomendadas para a criação ou atualização de aspectos. Essas unidades de código são chamadas *concern peers* e são detectadas por meio das interações similares que possam ter. Essas interações podem ser internas, ou seja dentro do corpo do método ou externas que são chamadas para métodos de outras classes que estejam relacionadas em contextos similares. A proposta está sustentada em definições matemáticas baseadas na teoria de conjuntos. A ferramenta apresentada é *XScan*, a qual utiliza outras ferramentas para apoiar a tarefa de mineração tais como *Clever*, *OAT*, *AJDT* e *Pointcutdoctor*.

A solução algorítmica para a detecção dos *concern peers* é apresentada na Figura 7.4 e explicada em seguida:

1 Busca dos candidatos.

- 1.1 Na linha 2, XScan procura e adiciona todos os pares de métodos com porções de código similares na lista de candidatos *C*.
- 1.2 Na linha 3, XScan adiciona os métodos que tenham o nomes similares na lista de candidatos *C*.
- 1.3 Na linha 4, XScan adiciona métodos que sobrecarregam ou implementam um mesmo método pai na lista de candidatos *C*.

2 Busca de *peers*.

```
1  function DetectPeerMethod(P)
2    C.add(MethodsHaveSimilarCode(P)) //add peer candidates with
3    C.add(MethodsHaveSimilarName(P)) //similar code, name,
4    C.add(RelativeMethods(P)) //and inheritance
5    repeat
6     (x, y) = C.next() //repeatedly process candidates
7     if sim(x, y) ≥ σ //similar enough
8       C.remove((x, y)) //remove from candidates
9       L.add((x, y)) //add as peers
10      X=ClassOf(x), Y=ClassOf(y)) //check enclosing classes
11      C.add(MethodsHaveSimilarName(X, Y)) //more candidates
12      foreach (u, v) ∈ C: recalculate sim(u, v)//update similarity
13   until no new peer pairs is detected
14   G = RankGroup(L)
15 return G
```

Figura 7.4: Detecção dos *concern peers*.

2.1 Na linha 6, os pares de métodos candidatos na lista *C* estão ordenados descendentemente por o valor de suas interações similares.

2.2 Na linha 7, se o cálculo da função *sim* para dois métodos candidatos é maior que um valor limite customizável, então XScan elimina o par de métodos da lista *C*, linha 8 e é adicionado na lista *L*, linha 9.

2.3 Nas linhas 10 e 11, com os *peer pairs* da lista *L*, procura-se novos candidatos de métodos com nomes similares e que estejam dentro das classes dos métodos selecionados. Na linha 12 são atualizados os valores da função *sim*.

2.4 Na linha 13, se repete o processo até que não sejam identificados mais *peers*.

3 Detectar e classificar por rango os *peer groups*, linha 14.

3.1 Todos os peer pairs da lista *L* são usados para formar um grafo onde um nodo representa um método peer, e cada aresta representa um peer pair em *L*.

3.2 XScan percorre o grafo e reporta a cada conjunto de nodos conectados como um grupo de *peers*. O ranking é feito por a maior quantidade de interações externas que possa ter esse grupo de *peers*.

A solução algorítmica para a recomendação de uma criação ou atualização de um aspecto é apresentada na Figura 7.5 e explicada em seguida:

1 Criação de aspectos em programas OO.

1.1 Uso de Eclipse para obter as interações internas e externas.

```
1  function RecommendUpdate(P1, A1, P2, A2)
2    M = DetectChange(P1, P2) //map and find changed methods
3    N = MatchAspect(A1, A2, M) //map and find changed aspects
4    G2 = DetectPeerMethod(P2);
5    for each shadow group S in A2 //match peer groups to
6      X = Match(S, G2) // shadow groups and recommend
7      if X\S ≠ ∅ Recommend(X\S, N, M) //relevant methods
8    for each unmatched peer group Y in G2 //recommend for
9      recommend Y for new aspects //creating new aspects
```

Figura 7.5: Algoritmo de recomendação de atualização dos aspectos.

1.2 Eliminar as bibliotecas padrões de Java e os métodos *get*, *set* e *util* para distinguir interesses transversais.

1.3 Utilizar o algoritmo para uma classificação dos *peers*, Figura 7.4.

2 Criação ou atualização de aspectos em programas AO.

2.1 Na linha 2, se detectam as mudanças nos métodos dos programas *P1* com *P2*.

2.2 Na linha 3, se detectam novos aspectos eliminados ou adicionados *A1* com *A2*.

2.3 Na linha 4, se detectam *peer groups* em *P2*.

2.4 Nas linhas 5 e 6, para cada *shadow group S* nos aspectos de *P2*, ou seja em *A2*, fazer uma correspondência com os *peer groups* detectados. Se *S* corresponde com um novo aspecto adicionado, todos os métodos que fazem a correspondência são recomendados. De outra maneira, só os métodos modificado ou adicionados são levados em conta.

2.5 Nas linhas 8 e 9, todos os métodos que não se correspondem, XScan recomenda a criação de novos aspectos.

Os autores realizaram uma avaliação empírica utilizando sistemas de softwares que têm sido modularizados em aspectos por meio de outros abordagens para avaliar a eficácia do XScan e comparar os resultados. Levando em conta características como a escalabilidade, a precisão e cobertura, se conclui que a ferramenta desenvolvida apresenta um alto grau de escalabilidade e exatidão na mineração de aspectos. A exatidão foi definida em termos de precisão e cobertura onde os resultados obtidos comparados com os resultados de outras ferramentas de mineração de aspectos são, na maioria dos casos, melhores. A técnica poderia ser estendida para minerar de outros tipos de interesses combinada com outras técnicas de mineração.

Capítulo 8

CONCLUSÕES E TRABALHOS FUTUROS

Esse capítulo apresenta de maneira sintética as conclusões da abordagem proposta. Depois, sugerem-se alguns tópicos para trabalhos futuros, baseado nas limitações da abordagem proposta. Por fim, apresentam-se considerações finais.

8.1 Conclusões

O grupo AdvanSE (*Advanced Research on Software Engineering*) da Universidade Federal de São Carlos visa, dentre outras coisas, pesquisar tópicos relacionados com a manutenção e modernização de sistemas legados. As contribuições desse grupo concentram-se em todo o processo de modernização de sistemas, porém este trabalho se encaixa na fase de engenharia reversa, especificamente na mineração de interesses.

A mineração de interesses em código fonte tem sido uma área de pesquisa bem investigada (KELLENS *et al.*, 2007; DURELLI *et al.*, 2013). Porém, a identificação de interesses em outros artefatos tais como documentos de requisitos e modelos ainda apresenta grandes desafios de pesquisa porque surgem perguntas interessantes a serem contestadas como por exemplo, o que é um interesse nesses artefatos?. Qual é a melhor forma para identificá-los?. Como se relacionam entre eles?, entre outras que podem ajudar aos engenheiros a ter uma melhor compreensão dos sistemas analisados.

Uma primeira conclusão deste trabalho é que técnicas de mineração de interesses em código fonte podem ser adaptadas para realizar mineração em modelos. Em específico, mostrou-se que a mineração de interesses em modelos KDM é factível para auxilar aos engenheiros de manutenção no processo de modernização. Além disso, com as marcações realizadas no KDM, outras técnicas/ferramentas poderão modularizar os interesses identificados independente da

linguagem de programação.

Uma segunda conclusão é que combinando técnicas de mineração a efetividade na identificação dos interesses aumenta. Embora a técnica desenvolvida é baseada na identificação por tokens, demonstrou-se que a biblioteca de interesses combinada com um algoritmo de *clustering* de strings atinge bons resultados de precisão e cobertura em um experimento controlado para a procura do interesse de Persistência.

Por último, a ferramenta desenvolvida (CCKDM) poderia evoluir em duas frentes; *(i)* Adicionar novas técnicas de mineração no modulo *miner* com o intuito de transformar a ferramenta em um framework de mineração de interesses onde múltiplas técnicas possam ser combinadas. *(ii)* Atualizar o modulo de geração de logs para apresentar saídas mais amigáveis para os usuários.

8.2 Limitações e Trabalhos Futuros

Essa dissertação trata de mineração de interesses transversais em modelos KDM, mas a abordagem proposta conta com três limitações; *(i)* No momento da escrita desta dissertação a melhor alternativa para gerar instâncias de modelos KDM a partir do código java era o framework Modisco^{TM}. Porém, ele só consegue gerar estruturas de baixo nível (Camada de Infraestrutura e Camada Elementos de Programa) do metamodelo KDM, então a técnica desenvolvida não é capaz de identificar interesses em estruturas de mais alto nível. *(ii)* O CCKDM só pode marcar um determinado elemento do modelo KDM com um só interesse. *(iii)* A técnica apresentada faz uma análise estática de strings e condicionada a padrões de codificação, então pode haver interesses que não sejam identificados pela técnica.

Assim, podemos listar algumas perspectivas de trabalho futuras:

- O framework Modisco^{TM} esta em constante desenvolvimento pelo qual em um futuro próximo poderia dar suporte para estruturas do metamodelo KDM de mais alto nível. Dessa forma, CCKDM poderia ser capaz de identificar outros tipos interesses tais como padrões de projeto e padrões arquiteturais. Além disso, utilizar outras formas de consulta ao modelo KDM para que seja independente do JMQ.

- Estender o modelo de base de dados para suportar estruturas de programação relacionadas com AspectJ. Pretende-se minerar modelos estendidos com suporte para AOSD.

- Melhorar a técnica de mineração com uma combinação de técnicas estáticas e dinâmicas para melhorar os valores de precisão e cobertura e determinar se um elemento pode

implementar mais de um interesse. A ideia é que o CCKDM seja transformado para um framework de mineração de interesses onde distintas técnicas de mineração sejam adicionadas mediante plugins.

- Identificar interesses em sistemas de maior tamanho ($\geq$ 15K) e habilitar a opção de marcar com mais de um interesse as estruturas do modelo.

- Comprovar que as combinações das técnicas apresentadas na revisão sistemática do Capítulo 3 são bons candidatos para melhorar a precisão e cobertura.

8.3 Considerações Finais

Conforme apresentado no Capítulo 4, foi proposta uma técnica combinada de mineração de interesses transversais; uma biblioteca de interesses e um algoritmo de *clustering* de strings para a identificação de interesses em modelos KDM.

Além disso, a técnica foi implementada em um plugin Eclipse™ apresentado no Capítulo 5. Com isso, futuras técnicas que modularizem interesses em modelos KDM poderão contar com uma ferramenta para pode-los identificar previamente.

Outra característica positiva é que a técnica apresentada foi avaliada no Capítulo 6 atingindo bons resultados em termos de precisão e cobertura.

Finalmente, outros experimentos são interessantes a fim de avaliar e aprimorar a abordagem proposta, pois este é o primeiro trabalho apresentado para a mineração de interesses em modelos KDM.

REFERÊNCIAS

BENNETT, K. H.; RAJLICH, V. T. Software maintenance and evolution: a roadmap. In: *Proceedings of the Conference on The Future of Software Engineering*. New York, NY, USA: ACM, 2000. (ICSE '00), p. 73–87. ISBN 1-58113-253-0. Disponível em: <http://doi.acm.org/10.1145/336512.336534>.

BERKHIN, P. A survey of clustering data mining techniques. In: *Grouping Multidimensional Data*. [S.l.]: Springer Berlin Heidelberg, 2006. p. 25–71.

BERNARDI, M. L.; Di Lucca, G. A. The ConAn Tool to Identify Crosscutting Concerns in Object Oriented Systems. In: *Proceedings of the 2010 IEEE 18th International Conference on Program Comprehension*. Washington, DC, USA: IEEE Computer Society, 2010. p. 48–49.

BINKLEY, D.; DAVIS, M.; LAWRIE, D.; MALETIC, J.; MORRELL, C.; SHARIF, B. The impact of identifier style on effort and comprehension. *Empirical Software Engineering*, v. 18, n. 2, p. 219–276, 2013. Cited By (since 1996)0.

BRANT, J. *HotDraw*. Tese (Masters thesis), 1995.

BREU, S.; KRINKE, J. Aspect mining using event traces. In: *Proceedings of the 19th IEEE international conference on Automated software engineering*. Washington, DC, USA: IEEE Computer Society, 2004.

BREU, S.; ZIMMERMANN, T.; LINDIG, C. Aspect mining for large systems. In: *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. New York, NY, USA: ACM, 2006. p. 641–642.

BRITO, I. S.; MOREIRA, A. *Advanced Separation of Concerns for Requirements Engineering*. 2004.

BRUNELIERE, H.; CABOT, J.; JOUAULT, F.; MADIOT, F. Modisco: a generic and extensible framework for model driven reverse engineering. In: *Proceedings of the IEEE/ACM international conference on Automated software engineering*. New York, NY, USA: ACM, 2010. (ASE '10), p. 173–174. ISBN 978-1-4503-0116-9. Disponível em: <http://doi.acm.org/10.1145/1858996.1859032>.

BRUNTINK, M.; DEURSEN, A. van; ENGELEN, R. van; TOURWE, T. On the use of clone detection for identifying crosscutting concern code. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 31, p. 804–818, October 2005. ISSN 0098-5589. Disponível em: <http://dl.acm.org/citation.cfm?id=1100866.1100978>.

CAMARGO, V. V.; MASIERO, P. C. Frameworks Orientados a Aspectos. *Simpósio Brasileiro de Engenharia de Software (SBES 2005)*, p. 200–215, 2005.

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

CECCATO, M.; TONELLA, P. Dynamic aspect mining. *Software, IET*, v. 3, n. 4, p. 321 –336, august 2009. ISSN 1751-8806.

CHIKOFSKY, E. J.; CROSS, J. H. Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Software*, v. 7, n. 1, p. 13–17, 1990.

HO CHOI, D. K. K. A research on the transmission and variation of tales to build korean yadam computer aided digital archive. p. 502–505, 2013.

COJOCAR, G.; CZIBULA, G. On clustering based aspect mining. In: *Intelligent Computer Communication and Processing, 2008. ICCP 2008. 4th International Conference on*. [S.l.: s.n.], 2008. p. 129 –136.

CUADRADO, J.; JOUAULT, F.; GARCíA MOLINA, J.; BéZIVIN, J. Experiments with a high-level navigation language. In: PAIGE, R. (Ed.). *Theory and Practice of Model Transformations*. [S.l.]: Springer Berlin Heidelberg, 2009, (Lecture Notes in Computer Science, v. 5563). p. 229–238. ISBN 978-3-642-02407-8.

DELTOMBE, G.; GOAER, O. L.; BARBIER, F. Bridging kdm and astm for model-driven software modernization. In: *SEKE*. [S.l.]: Knowledge Systems Institute Graduate School, 2012. p. 517–524. ISBN 1-891706-31-4.

DEMEYER, S.; DUCASSE, S.; NIERSTRASZ, O. *Object Oriented Reengineering Patterns*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002. ISBN 1558606394.

DIJKSTRA, E. W. EWD 447: On the role of scientific thought. *Selected Writings on Computing: A Personal Perspective*, Springer-Verlag, p. 60–66, 1982. Disponível em: <http://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD447.html>.

DURELLI, R. S.; SANTIBANEZ, D. S. M.; ANQUETIL, N.; DELAMARO, M. E.; CAMARGO, V. V. de. A systematic review on mining techniques for crosscutting concerns. In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2013. (SAC '13), p. 1080–1087. ISBN 978-1-4503-1656-9. Disponível em: <http://doi.acm.org/10.1145/2480486.2480567>.

EADDY, M.; AHO, A. V.; ANTONIOL, G.; GUÉHÉNEUC, Y.-G. 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*. Washington, DC, USA: IEEE Computer Society, 2008. p. 53–62.

EIBLER, T.; HODGES, C. P.; MEIER, H. Ptpan overcoming memory limitations in oligonucleotide string matching for primer/probe design. *Bioinformatics*, v. 27, n. 20, p. 2797–2805, 2011. Disponível em: <http://bioinformatics.oxfordjournals.org/content/27/20-/2797.abstract>.

EVERITT, B. S.; LANDAU, S.; LEESE, M. *Cluster Analysis*. 4th. ed. [S.l.]: Wiley Publishing, 2009. ISBN 0340761199, 9780340761199.

GRAY, J.; BAPTY, T.; NEEMA, S.; TUCK, J. Handling crosscutting constraints in domain-specific modeling. *Commun. ACM*, ACM, New York, NY, USA, v. 44, n. 10, p. 87–93, out. 2001. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/383845.383864>.

GRISWOLD, W. G.; Y, Y. K.; YUAN, J. J. Aspect browser: Tool support for managing dispersed aspects. In: *In First Workshop on Multi-Dimensional Separation of Concerns in Object-oriented Systems - OOPSLA 99*. [S.l.: s.n.], 1999.

HANNEMANN, J.; KICZALES, G. Overcoming the prevalent decomposition of legacy code. In: *In Workshop on Advanced Separation of Concerns*. [S.l.: s.n.], 2001.

HASSAN, A. E.; XIE, T. Software intelligence: the future of mining software engineering data. In: *Proceedings of the FSE/SDP workshop on Future of software engineering research*. New York, NY, USA: ACM, 2010. (FoSER '10), p. 161–166. ISBN 978-1-4503-0427-6. Disponível em: <http://doi.acm.org/10.1145/1882362.1882397>.

HOTELLING, H. Relations Between Two Sets of Variates. *Biometrika*, Biometrika Trust, v. 28, n. 3/4, p. 321–377, dez. 1936. ISSN 00063444. Disponível em: <http://dx.doi.org/10-.2307/2333955>.

HUANG, J.; LU, Y.; YANG, J. Aspect mining using link analysis. In: *Proceedings of the 2010 Fifth International Conference on Frontier of Computer Science and Technology*. Washington, DC, USA: IEEE Computer Society, 2010. p. 312–317.

ISHIO, T.; DATE, H.; MIYAKE, T.; INOUE, K. Mining coding patterns to detect crosscutting concerns in java programs. In: *Proceedings of the 2008 15th Working Conference on Reverse Engineering*. Washington, DC, USA: IEEE Computer Society, 2008. p. 123–132.

ISO. *Information technology – Object Management Group Architecture-Driven Modernization (ADM) – Knowledge Discovery Meta-Model (KDM)*. 2012. `http://www.iso.org/iso/catalogue_detail.1128htm?csnumber=32625`.

IZQUIERDO, J.; MOLINA, J. An architecture-driven modernization tool for calculating metrics. *Software, IEEE*, v. 27, n. 4, p. 37–43, 2010. ISSN 0740-7459.

PARREIRA JUNIOR, P.; MENDES, W.; CAMARGO, V. de; PENTEADO, R.; COSTA, H. Mining crosscutting concerns with comscid: A rule-based customizable mining tool. In: *Informatica (CLEI), 2012 XXXVIII Conferencia Latinoamericana En*. [S.l.: s.n.], 2012. p. 1–9.

KARSAI, G.; MAROTI, M.; LEDECZI, A.; GRAY, J.; SZTIPANOVITS, J. Composition and cloning in modeling and meta-modeling. *Control Systems Technology, IEEE Transactions on*, v. 12, n. 2, p. 263–278, 2004. ISSN 1063-6536.

KELLENS, A.; MENS, K.; TONELLA, P. A survey of automated code level aspect mining techniques. In: . Berlin, Heidelberg: Springer-Verlag, 2007. p. 143–162.

KHATOON, S.; MAHMOOD, A.; LI, G. An evaluation of source code mining techniques. In: *Fuzzy Systems and Knowledge Discovery (FSKD), 2011 Eighth International Conference on*. [S.l.: s.n.], 2011. v. 3, p. 1929 –1933.

KICZALES, G.; HILSDALE, E.; HUGUNIN, J.; KERSTEN, M.; PALM, J.; GRISWOLD, W. G. An overview of aspectj. In: . [S.l.]: Springer-Verlag, 2001. p. 327–353.

KICZALES, G.; LAMPING, J.; MENDHEKAR, A.; MAEDA, C.; LOPES, C. V.; LOINGTIER, J.-M.; IRWIN, J. Aspect-oriented programming. In: *ECOOP*. [S.l.: s.n.], 1997. p. 220–242.

KICZALES, G.; LOPES, C. V.; TEKINERDOGAN, B.; MENS, K. Aspect-oriented programming workshop report. In: *ECOOP Workshops*. [S.l.: s.n.], 1997. p. 483–496.

KICZALES, G.; MEZINI, M. Aspect-oriented programming and modular reasoning. In: *Proceedings of the 27th international conference on Software engineering*. New York, NY, USA: ACM, 2005. (ICSE '05), p. 49–58. ISBN 1-58113-963-2. Disponível em: <http://doi.acm.org/10.1145/1062455.1062482>.

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*, Butterworth-Heinemann, Newton, MA, USA, v. 51, p. 7–15, January 2009.

LEHMAN, M. M.; RAMIL, J. F.; WERNICK, P. D.; PERRY, D. E.; TURSKI, W. M. Metrics and laws of software evolution - the nineties view. In: *Proceedings of the 4th International Symposium on Software Metrics*. Washington, DC, USA: IEEE Computer Society, 1997.

PRADO LEITE, J. C. S. do; SANT'ANNA, M.; FREITAS, F. G. D. Draco-puc: a technology assembly for domain oriented software development. In: *PROCEEDINGS OF THE THIRD INTERNATIONAL CONFERENCE ON SOFTWARE REUSE*. [S.l.]: IEEE Computer Society Press, 1994. p. 94–100.

LENGYEL, L.; LEVENDOVSZKY, T.; ANGYAL, L. Identification of crosscutting constraints in metamodel-based model transformations. In: *EUROCON 2009, EUROCON '09. IEEE*. [S.l.: s.n.], 2009. p. 359–364.

LEVENSHTEIN, V. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, v. 10, p. 707, 1966.

MAINETTI, L.; PAIANO, R.; PANDURINO, A. Migros: A model-driven transformation approach of the user experience of legacy applications. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 7387 LNCS, p. 490–493, 2012. Cited By (since 1996) 0.

MAISIKELI, S. G. *Aspect mining using self-organizing maps with method level dynamic software metrics as input vectors*. Tese (Doutorado), 2010.

MALHEIROS, V.; HOHN, E.; PINHO, R.; MENDONCA, M.; MALDONADO, J. C. A visual text mining approach for systematic reviews. In: *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*. Washington, DC, USA: IEEE Computer Society, 2007. (ESEM '07), p. 245–254. ISBN 0-7695-2886-4. Disponível em: <http://dx.doi.org/10.1109/ESEM.2007.13>.

MANNING, C. D.; RAGHAVAN, P.; SCHüTZE, H. *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008. ISBN 0521865719, 9780521865715.

MARIN, M.; DEURSEN, A. V.; MOONEN, L. Identifying crosscutting concerns using fan-in analysis. *ACM Trans. Softw. Eng. Methodol.*, ACM, New York, NY, USA, v. 17, p. 3:1–3:37, December 2007. ISSN 1049-331X. Disponível em: <http://doi.acm.org/10.1145/1314493-.1314496>.

MARIN, M.; MOONEN, L.; DEURSEN, A. v. An integrated crosscutting concern migration strategy and its application to jhotdraw. In: *Proceedings of the Seventh IEEE International Working Conference on Source Code Analysis and Manipulation*. Washington, DC, USA: IEEE Computer Society, 2007. p. 101–110.

MARIN, M.; MOONEN, L.; DEURSEN, A. van. A common framework for aspect mining based on crosscutting concern sorts. In: *Proceedings of the 13th Working Conference on Reverse Engineering*. Washington, DC, USA: IEEE Computer Society, 2006. p. 29–38. ISBN 0-7695-2719-1. Disponível em: <http://dl.acm.org/citation.cfm?id=1174510.1174715>.

MCFADDEN, R.; MITROPOULOS, F. Aspect mining using model-based clustering. In: *Southeastcon, 2012 Proceedings of IEEE*. [S.l.: s.n.], 2012. p. 1–8. ISSN 1091-0050.

MENS, K.; KELLENS, A.; KRINKE, J. Pitfalls in aspect mining. In: *Proceedings of the 2008 15th Working Conference on Reverse Engineering*. Washington, DC, USA: IEEE Computer Society, 2008. p. 113–122. ISBN 978-0-7695-3429-9. Disponível em: <http://dl.acm.org/citation.cfm?id=1447565.1448039>.

MILLI, H. M. H. Understanding separation of concerns. *In Workshop on Early Aspects - Aspect Oriented Software Development (AOSD'04)*, 2004.

MORRISON, D. *Multivariate statistical methods*. 2. ed., internat student ed. ed. New York [u.a.]: McGraw-Hill, 1976. (McGraw-Hill series in probability and statistics). ISBN 0070431868.

MULDER, F.; ZAIDMAN, A. Identifying cross-cutting concerns using software repository mining. In: *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*. New York, NY, USA: ACM, 2010. p. 23–32.

NGUYEN, T. T.; NGUYEN, H. V.; NGUYEN, H. A.; NGUYEN, T. N. Aspect recommendation for evolving software. In: *Proceedings of the 33rd International Conference on Software Engineering*. New York, NY, USA: ACM, 2011. (ICSE '11), p. 361–370.

NORA, B.; GHOUL, S. A model-driven approach to aspect mining. In: *Proceedings of the 27th International Conference on Software Engineering*. New York, NY, USA: ACM, 2006. p. 361–370.

NORMANTAS, K.; SOSUNOVAS, S.; VASILECAS, O. An overview of the knowledge discovery meta-model. In: *Proceedings of the 13th International Conference on Computer Systems and Technologies*. New York, NY, USA: ACM, 2012. (CompSysTech '12), p. 52–57. ISBN 978-1-4503-1193-9. Disponível em: <http://doi.acm.org/10.1145/2383276.2383286>.

OMG. *ADM Task Force@ONLINE*. 2013. `http://adm.omg.org/`.

PÉREZ-CASTILLO, R.; GUZMáN, I. G.-R. de; PIATTINI, M. Knowledge discovery metamodel-iso/iec 19506: A standard to modernize legacy systems. *Comput. Stand. Interfaces*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 33, n. 6, p. 519–532, nov. 2011. ISSN 0920-5489. Disponível em: <http://dx.doi.org/10.1016/j.csi-.2011.02.007>.

PéREZ-CASTILLO, R.; DE GUZMáN, I.; CAIVANO, D.; PIATTINI, M. Database schema elicitation to modernize relational databases. In: . [S.l.: s.n.], 2012. v. 1 DISI, n. AIDSS/-, p. 126–132. Cited By (since 1996) 0.

QU, L.; LIU, D. Aspect mining using method call tree. In: *Proceedings of the 2007 International Conference on Multimedia and Ubiquitous Engineering*. Washington, DC, USA: IEEE Computer Society, 2007. p. 407–412.

READE, C. *Elements of functional programming*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN 0-201-12915-9.

ROBILLARD, M. P.; MURPHY, G. C. Feat: a tool for locating, describing, and analyzing concerns in source code. In: *Proceedings of the 25th International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2003. (ICSE '03), p. 822–823. ISBN 0-7695-1877-X. Disponível em: <http://dl.acm.org/citation.cfm?id=776816.776969>.

ROBILLARD, M. P.; MURPHY, G. C. Representing concerns in source code. *ACM Trans. Softw. Eng. Methodol.*, ACM, New York, NY, USA, v. 16, n. 1, fev. 2007. ISSN 1049-331X. Disponível em: <http://doi.acm.org/10.1145/1189748.1189751>.

SADOVYKH, A.; VIGIER, L.; HOFFMANN, A.; GROSSMANN, J.; RITTER, T.; GOMEZ, E.; ESTEKHIN, O. Architecture driven modernization in practice 150; study results. In: *Engineering of Complex Computer Systems, 2009 14th IEEE International Conference on*. [S.l.: s.n.], 2009. p. 50–57.

SERBAN, G.; MOLDOVAN, G. A new k-means based clustering algorithm in aspect mining. In: *Symbolic and Numeric Algorithms for Scientific Computing, 2006. SYNASC '06. Eighth International Symposium on*. [S.l.: s.n.], 2006. p. 69–74.

SHEPHERD, D.; PALM, J.; POLLOCK, L.; CHU-CARROLL, M. Timna: a framework for automatically combining aspect mining analyses. In: *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*. New York, NY, USA: ACM, 2005. p. 184–193.

SHEPHERD, D.; POLLOCK, L.; TOURWÉ, T. Using language clues to discover crosscutting concerns. In: *Proceedings of the 2005 workshop on Modeling and analysis of concerns in software*. New York, NY, USA: ACM, 2005. p. 1–6.

SILVA, L. Fernandes da; PRADO LEITE, J. C. Sampaio do. Integração de características transversais durante a modelagem de requisitos. In: *Simpósio Brasileiro de Engenharia de Software (SBES-2005)*. Uberlândia-MG: ACM, 2008. p. 3–7.

SOLOWAY, E.; LAMPERT, R.; LETOVSKY, S.; LITTMAN, D.; PINTO, J. Designing documentation to compensate for delocalized plans. *Commun. ACM*, ACM, New York, NY, USA, v. 31, n. 11, p. 1259–1267, nov. 1988. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/50087.50088>.

TEKINERDOGAN, B. *ASAAM: Aspectual Software Architecture Analysis Method*. 2003.

TOURWE, T.; MENS, K. Mining aspectual views using formal concept analysis. In: *Proceedings of the Source Code Analysis and Manipulation, Fourth IEEE International Workshop*. Washington, DC, USA: IEEE Computer Society, 2004. p. 97–106.

TRAVASSOS, G. *Introdução à engenharia de software experimental*. UFRJ, 2002. (RT-ES-590/02). Disponível em: <http://books.google.com.br/books?id=4SnKZwEACAAJ>.

TRIBBEY, W.; MITROPOULOS, F. Construction and analysis of vector space models for use in aspect mining. In: *Proceedings of the 50th Annual Southeast Regional Conference*. New York, NY, USA: ACM, 2012. (ACM-SE '12), p. 220–225. ISBN 978-1-4503-1203-5. Disponível em: <http://doi.acm.org/10.1145/2184512.2184564>.

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

ULRICH, W. M.; NEWCOMB, P. *Information Systems Transformation: Architecture-Driven Modernization Case Studies*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010. ISBN 0123749131, 9780123749130.

VAZIRI, M.; JACKSON, D. Some shortcomings of ocl, the object constraint language of uml. In: *Proceedings of the Technology of Object-Oriented Languages and Systems (TOOLS 34'00)*. Washington, DC, USA: IEEE Computer Society, 2000. (TOOLS '00), p. 555–. ISBN 0-7695-0774-3. Disponível em: <http://dl.acm.org/citation.cfm?id=832261.833293>.

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. ISBN 0-7923-8682-5.

ZHANG, C.; JACOBSEN, H.-A. Mining crosscutting concerns through random walks. *IEEE Transactions on Software Engineering*, IEEE Computer Society, n. PrePrints, 2011.

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: IEEE Computer Society, 2008. (ASE '08), p. 278–287. ISBN 978-1-4244-2187-9. Disponível em: <http://dx.doi.org/10.1109-/ASE.2008.38>.

ZHANG, J.; GRAY, J.; LIN, Y.; TAIRAS, R. Aspect mining from a modelling perspective. *Int. J. Comput. Appl. Technol.*, Inderscience Publishers, Inderscience Publishers, Geneva, SWITZERLAND, v. 31, n. 1/2, p. 74–82, mar. 2008. ISSN 0952-8091. Disponível em: <http://dx.doi.org/10.1504/IJCAT.2008.017720>.

GLOSSÁRIO

ADMTF – *Architecture-Driven Modernization Task Force*

ADM – *Architecture-Driven Modernization*

AdvanSE – *Advanced Research Group on Software Engineering*

CFBA – *Clustering Fan-in Based Analysis*

CIM – *Computation-Independent Model*

CSV – *Comma-Separated Values*

GME – *Generic Modeling Environment*

ISO – *International Organization for Standardization*

JMQ – *Java Model Query*

KDM – *Knowledge Discovery MetaModel*

MDA – *Model-Driven Architecture*

MOF – *Meta-Object Facility*

OCL – *Object Constraint Language*

OMG – *Object Management Group*

PIM – *Platform-Independent Model*

POA – *Programação Orientada a Aspectos*

POO – *Programação Orientada a Objetos*

PSM – *Platform-Specific Model*

SQL – *Structured Query Language*

VMTS – *Visual Modeling and Transformation System*

XMI – *XML Metadata Interchange*

XML – *eXtensible Markup Language*

Apendice A

METODOLOGIA

A pesquisa em Engenharia de Software pode ser conduzida através de quatro métodos: o científico, o de engenharia, o experimental e o analítico (WOHLIN *et al.*, 2000). O método científico foca na observação de um ambiente e na tentativa de extrair dele algum modelo ou teoria que possa explicar o fenômeno estudado. O método de engenharia, por sua vez, é baseado na observação de soluções existentes, na identificação de problemas nessas soluções e na sugestão de abordagens para melhorar as soluções analisadas. Já o método experimental pretende fornecer um modelo novo para solução de um problema e tenta estudar o impacto desse modelo. Por fim, o método analítico oferece uma base analítica (ou matemática) para o desenvolvimento de modelos. Este trabalho de pesquisa seguiu o método de engenharia (baseado na Figura A.1).

1. Inicialmente foi feita uma revisão bibliográfica sobre os conceitos de AOSD e as técnicas de mineração de interesses no código fonte;

2. Nesta fase foi realizada uma análise comparativa das abordagens para identificar os conceitos, vantagens e desvantagens de cada uma. Após isso, foi elaborado um artigo no qual estendeu-se a taxonomia proposta por Kellens (KELLENS *et al.*, 2007). O artigo foi submetido e aceito na ACM SAC (DURELLI *et al.*, 2013);

3. Foi apresentada a proposta de qualificação para o mestrado;

4. O grupo AdvanSE realizou uma revisão bibliográfica relacionada com a ADM e os conceitos envolvidos nesta metodologia de modernização de sistemas. Para este trabalho o foco de estudo foi o metamodelo KDM e as possíveis relações entre as estruturas fornecidas pelo metamodelo e a identificação de interesses;

5. Nesta fase foi criada a abordagem e a ferramenta para a mineração de interesses no processo ADM. Além disso foi feita uma avaliação estatística com alguns sistemas. Foram elaborados dois artigos os quais foram submetidos para o CBSOFT; e

6 Apresentação do trabalho.

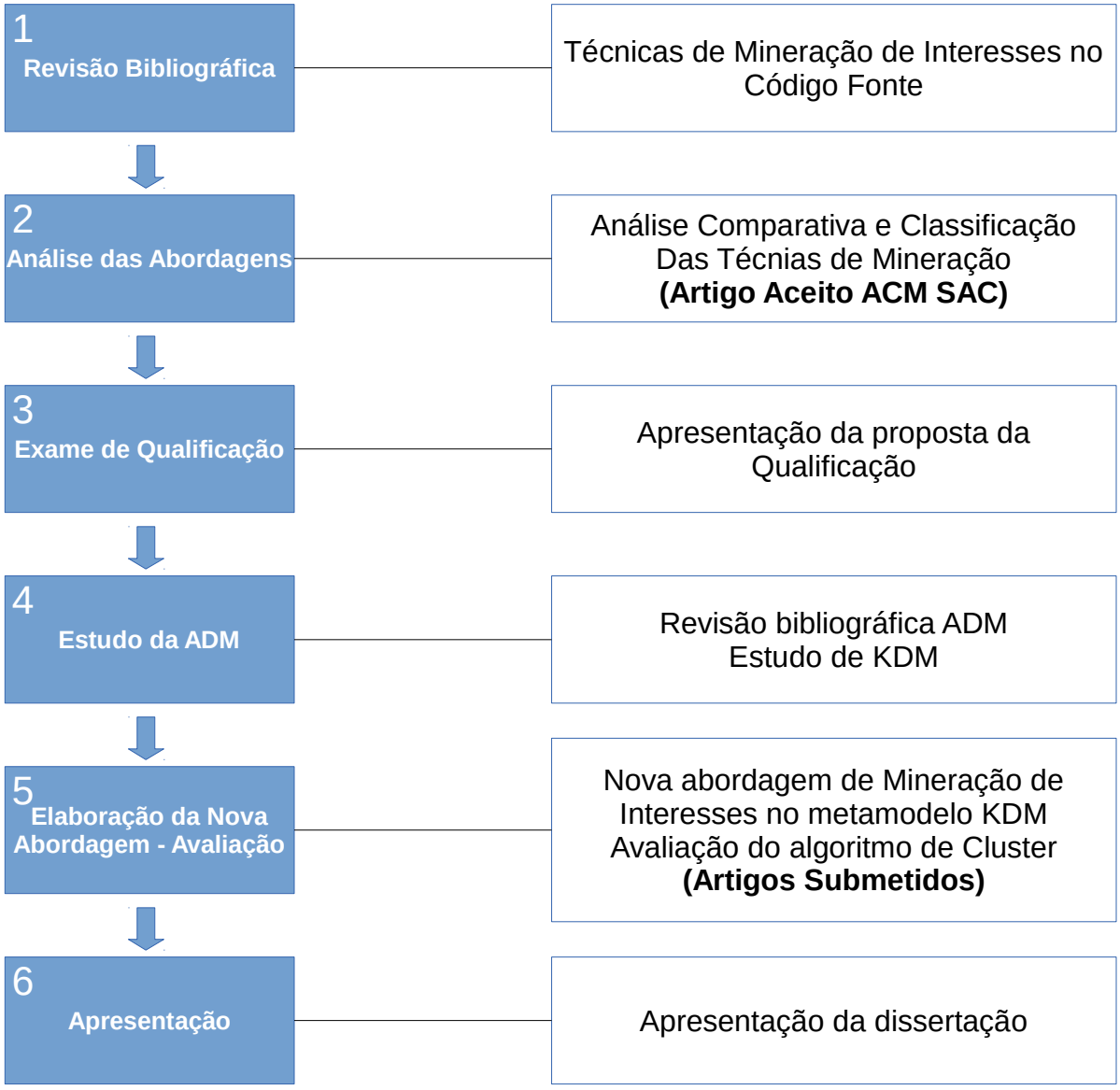


Figura A.1: Estrutura da pesquisa conduzida

Anexo A

TABELAS DE AVALIAÇÃO: HEALTHWATCHER

	0.3		0.4		0.5	
Centroids	Precision	Recall	Precision	Recall	Precision	Recall
con	100%	100%	100%	100,00%	100%	100,00%
getHealthUnitList	100%	100%	100%	78,02%	100%	70,64%
stmt2	100%	100%	100%	67,19%	100%	74,47%
SpecialityRepositoryRDB	100%	100%	100%	81,35%	100%	88,69%
accessComplaint	100%	100%	100%	80,46%	100%	48,98%
DiseaseTypeRepositoryRDB	100%	100%	100%	79,51%	100%	78,42%
searchTimestamp	100%	100%	100%	81,59%	100%	73,53%
search	100%	100%	100%	74,06%	100%	74,62%
AddressRepositoryRDB	100%	100%	100%	90,37%	100%	81,38%
resultSet	100%	100%	100%	82,53%	100%	71,79%
accessAnimal	100%	100%	100%	79,52%	100%	10,36%
getHealthUnitListBySpeciality	100%	100%	100%	69,82%	100%	69,75%
deepInsertCommon	100%	100%	100%	73,50%	100%	76,32%
getComplaintList	100%	100%	100%	82,77%	100%	63,35%
updateTimestamp	100%	100%	100%	77,81%	100%	76,95%
insertFood	100%	100%	100%	85,86%	100%	80,93%
getInstance	100%	100%	100%	76,58%	100%	79,17%
resultSet2	100%	100%	100%	71,36%	100%	74,96%
releaseCommunicationChannel	100%	100%	100%	71,45%	100%	83,51%
insertSpecial	100%	100%	100%	70,56%	100%	63,41%
incCurrentMirror	100%	100%	100%	86,44%	100%	79,78%
SymptomRepositoryRDB	100%	100%	100%	77,07%	100%	83,60%
deepInsertAnimal	100%	100%	100%	84,03%	100%	71,57%
ComplaintRepositoryRDB	100%	100%	100%	70,72%	100%	80,82%
conexoesCriadas	100%	100%	100%	70,34%	100%	68,93%
rs	100%	100%	100%	93,11%	100%	84,32%
getCommunicationChannel	100%	100%	100%	79,25%	100%	83,54%
getPreparedStatement	100%	100%	100%	76,59%	100%	81,55%
filter	100%	100%	100%	83,02%	100%	90,34%
update	100%	100%	100%	77,22%	100%	77,70%
remove	100%	100%	100%	87,60%	100%	78,12%
connect	100%	100%	100%	79,60%	100%	72,61%
HealthUnitRepositoryRDB	100%	100%	100%	88,20%	100%	75,69%
deepInsertSpecial	100%	100%	100%	71,10%	100%	55,30%
insertAnimal	100%	100%	100%	92,62%	100%	74,65%
beginTransaction	100%	100%	100%	63,22%	100%	70,09%
sqlException	100%	100%	100%	93,29%	100%	91,50%
getSpecialityList	100%	100%	100%	73,01%	100%	69,93%
conexoesLivres	100%	100%	100%	82,94%	100%	84,72%
EmployeeRepositoryRDB	100%	100%	100%	90,00%	100%	77,74%
getPartialHealthUnitList	100%	100%	100%	78,00%	100%	76,86%
getDiseaseTypeList	100%	100%	100%	84,00%	100%	78,06%
getSymptomList	100%	100%	100%	90,74%	100%	65,70%
ex	100%	100%	100%	61,50%	100%	94,49%
resposta	100%	100%	100%	67,21%	100%	98,56%
rollbackTransaction	100%	100%	100%	74,99%	100%	88,93%
accessSpecial	100%	100%	100%	92,61%	100%	83,78%
stmt	100%	100%	100%	100,00%	100%	100,00%
PersistenceMechanism	100%	100%	100%	86,21%	100%	76,23%
deepInsertFood	100%	100%	100%	86,71%	100%	71,80%
accessFood	100%	100%	100%	74,51%	100%	72,46%
partialSearch	100%	100%	100%	86,01%	100%	75,04%
commitTransaction	100%	100%	100%	66,19%	100%	69,05%
exists	100%	100%	100%	78,78%	100%	74,83%
disconnect	100%	100%	100%	68,00%	100%	73,19%
insert	100%	100%	100%	90,73%	100%	69,31%
	100%	100%	100%	80,00%	100%	76,11%

Figura A.1: Precisão e cobertura com os centróides para a técnica combinada

	0.3			0.4			0.5		
Centroids	Precision	Recall	Q	Precision	Recall	Q	Precision	Recall	Q
con	43%	100%	44	47%	89%	36	57%	42%	14
getHealthUnitList	50%	100%	10	63%	100%	8	63%	100%	8
stmt2	53%	100%	15	58%	88%	12	0%	0%	5
SpecialityRepositoryRDB	83%	100%	6	100%	100%	5	100%	100%	5
accessComplaint	58%	100%	40	67%	96%	33	69%	87%	29
DiseaseTypeRepositoryRDB	100%	100%	3	100%	100%	3	100%	33%	1
searchTimestamp	10%	100%	21	0%	0%	18	0%	0%	13
search	11%	100%	18	100%	100%	2	100%	100%	2
AddressRepositoryRDB	50%	100%	4	100%	50%	1	0%	0%	0
resultSet	100%	100%	9	100%	100%	9	100%	100%	9
accessAnimal	71%	100%	7	71%	100%	7	80%	80%	5
getHealthUnitListBySpeciality	0%	100%	19	0%	0%	16	0%	0%	9
deepInsertCommon	0%	100%	7	0%	0%	1	0%	0%	0
getComplaintList	0%	100%	30	0%	0%	26	0%	0%	14
updateTimestamp	10%	100%	40	9%	75%	34	9%	50%	22
insertFood	22%	100%	18	27%	75%	11	50%	25%	2
getInstance	69%	100%	16	0%	0%	2	0%	0%	2
resultSet2	100%	100%	1	100%	100%	1	0%	0%	0
releaseCommunicationChannel	100%	100%	12	0%	0%	0	0%	0%	0
insertSpecial	43%	100%	7	33%	33%	3	33%	33%	3
incCurrentMirror	86%	100%	7	100%	67%	4	100%	50%	3
SymptomRepositoryRDB	67%	100%	6	67%	100%	6	67%	100%	6
deepInsertAnimal	67%	100%	6	0%	0%	1	0%	0%	0
ComplaintRepositoryRDB	8%	100%	12	8%	100%	12	17%	100%	6
conexoesCriadas	50%	100%	8	100%	25%	1	100%	25%	1
rs	80%	100%	10	75%	75%	8	75%	75%	8
getCommunicationChannel	75%	100%	8	0%	0%	1	0%	0%	0
getPreparedStatement	33%	100%	3	0%	0%	0	0%	0%	0
filter	0%	100%	33	0%	0%	4	0%	0%	2
update	26%	100%	38	33%	40%	12	33%	40%	12
remove	23%	100%	30	0%	0%	11	0%	0%	0
connect	54%	100%	13	44%	57%	9	75%	43%	4
HealthUnitRepositoryRDB	17%	100%	6	17%	100%	6	17%	100%	6
deepInsertSpecial	67%	100%	6	0%	0%	2	0%	0%	1
insertAnimal	80%	100%	5	100%	75%	3	100%	25%	1
beginTransaction	44%	100%	9	100%	25%	1	100%	25%	1
sqlException	32%	100%	37	35%	100%	34	50%	58%	14
getSpecialityList	25%	100%	28	28%	100%	25	70%	100%	10
conexoesLivres	25%	100%	4	0%	0%	1	0%	0%	0
EmployeeRepositoryRDB	22%	100%	9	25%	100%	8	14%	50%	7
getPartialHealthUnitList	0%	100%	15	0%	0%	13	0%	0%	11
getDiseaseTypeList	22%	100%	9	29%	100%	7	25%	50%	4
getSymptomList	18%	100%	11	22%	100%	9	0%	0%	4
ex	22%	100%	23	6%	20%	16	7%	20%	14
resposta	75%	100%	40	80%	53%	20	80%	53%	20
rollbackTransaction	50%	100%	4	0%	0%	1	0%	0%	0
accessSpecial	33%	100%	3	0%	0%	0	0%	0%	0
stmt	29%	100%	7	67%	100%	3	50%	50%	2
PersistenceMechanism	100%	100%	3	100%	100%	3	100%	100%	3
deepInsertFood	0%	100%	1	0%	0%	0	0%	0%	0
accessFood	67%	100%	6	75%	75%	4	0%	0%	0
partialSearch	11%	100%	9	100%	100%	1	0%	0%	0
commitTransaction	33%	100%	6	0%	0%	3	0%	0%	0
exists	69%	100%	49	71%	94%	45	10%	3%	10
disconnect	75%	100%	8	0%	0%	2	0%	0%	0
insert	9%	100%	23	13%	100%	16	33%	100%	6
	37%	100%	822	38%	68%	520	41%	51%	299

Figura A.2: Precisão e cobertura com os centróides para *clustering*

Anexo B

TABELAS DE AVALIAÇÃO: PETSTORE

	0.3		0.4		0.5	
Centroids	Precision	Recall	Precision	Recall	Precision	Recall
searchItems	93,40%	100,00%	95,30%	91,22%	97,19%	62,37%
check	98,48%	100,00%	94,99%	80,02%	98,88%	70,99%
printSQLStatement	98,71%	100,00%	95,66%	77,55%	99,99%	100,00%
exception	96,10%	100,00%	100,00%	86,11%	98,95%	75,67%
dropTables	94,40%	100,00%	94,62%	65,60%	100,00%	75,32%
createTables	95,43%	100,00%	95,74%	85,52%	97,40%	79,76%
se	93,60%	100,00%	95,90%	88,34%	100,00%	74,37%
statement	94,73%	100,00%	94,05%	86,20%	98,52%	73,50,%
CatalogPopulator	95,45%	100,00%	97,78%	87,17%	99,46%	65,56%
rs	98,78%	100,00%	97,10%	80,33%	98,80%	72,12%
makeSQLStatementKey	92,32%	100,00%	93,48%	80,83%	99,81%	73,12%
resultSet	93,05%	100,00%	95,79%	89,33%	98,50%	81,66%
PopulateUtils	91,00%	100,00%	94,91%	98,49%	97,61%	74,37%
executeSQLStatement	96,26%	100,00%	95,78%	98,93%	98,72%	75,10%
ps	93,25%	100,00%	94,84%	66,62%	98,04%	75,69%
	95,00%	100,00%	95,73%	84,15%	98,79%	75,44%

Figura B.1: Precisão e cobertura com os centróides para a técnica combinada

	0.3			0.4			0.5		
Centroids	Precision	Recall	Q	Precision	Recall	Q	Precision	Recall	Q
searchItems	37%	100%	79	37%	38%	30	50%	7%	4
check	48%	100%	44	50%	90%	38	0%	0%	2
printSQLStatement	37%	100%	19	42%	71%	12	100%	57%	4
exception	31%	100%	150	24%	46%	87	30%	30%	47
dropTables	14%	100%	88	12%	33%	34	20%	8%	5
createTables	22%	100%	206	28%	67%	108	20%	36%	80
se	34%	100%	253	30%	53%	153	30%	53%	151
statement	40%	100%	252	35%	35%	100	49%	20%	41
CatalogPopulator	36%	100%	107	49%	58%	45	49%	47%	37
rs	48%	100%	31	0%	0%	5	0%	0%	5
makeSQLStatementKey	50%	100%	8	0%	0%	0	0%	0%	0
resultSet	38%	100%	97	26%	35%	50	28%	19%	25
PopulateUtils	67%	100%	66	71%	23%	14	64%	16%	11
executeSQLStatement	17%	100%	12	0%	0%	0	0%	0%	0
ps	14%	100%	7	0%	0%	2	0%	0%	2
	49%	100%	1419	28%	47%	678	29%	37%	414

Figura B.2: Precisão e cobertura com os centróides para a *clustering*

Anexo C

TABELAS DE AVALIAÇÃO: PROGRADWEB

	0.3			0.4			0.5		
Centroids	Precision	Recall	Q	Precision	Recall	Q	Precision	Recall	Q
rsetDados	42%	100%	31	31%	31%	13	0%	0%	0
AbrirConexaoProex	0%	100%	0	0%	0%	0	0%	0%	0
fimOpcaoEnfase	0%	100%	1	0%	0%	0	0%	0%	0
stmtIngres3	100%	100%	4	100%	100%	4	100%	100%	4
stmt2	13%	100%	46	16%	50%	19	0%	0%	0
doGet	0%	100%	143	0%	0%	135	0%	0%	0
stmt3	0%	100%	6	0%	0%	6	0%	0%	0
stmtIngres2	0%	100%	0	0%	0%	0	0%	0%	0
stmtlog	100%	100%	21	100%	76%	16	0%	0%	0
connCargo	41%	100%	27	45%	82%	20	90%	82%	10
stmt1	0%	100%	21	0%	0%	21	0%	0%	0
rsetAux	75%	100%	4	75%	100%	4	0%	0%	0
inicioSimulacao	0%	100%	4	0%	0%	2	0%	0%	2
FecharContexto	85%	100%	132	95%	99%	117	100%	99%	111
retMedicina	9%	100%	11	0%	0%	6	0%	0%	0
connTurma	20%	100%	25	16%	60%	19	100%	40%	2
hourFormat	67%	100%	12	100%	63%	5	0%	0%	0
stmtHorario	70%	100%	37	80%	46%	15	75%	23%	8
rsetEnfase	20%	100%	105	15%	43%	60	11%	29%	57
preamble	35%	100%	17	100%	83%	5	100%	17%	1
AutenticarSessao	100%	100%	7	100%	29%	2	0%	0%	0
stmtControle3	0%	100%	0	0%	0%	0	0%	0%	0
stmtControle2	23%	100%	30	21%	71%	24	33%	43%	9
connUsuario	100%	100%	2	100%	100%	2	100%	100%	2
connEmail	37%	100%	19	64%	100%	11	86%	86%	7
fimCancelamento	13%	100%	55	17%	29%	12	0%	0%	10
dagora	0%	100%	0	0%	0%	0	0%	0%	0
FecharConexao	40%	100%	5	25%	50%	4	50%	50%	2
fimComplementacaoCurso	36%	100%	14	36%	100%	14	100%	80%	4
connAluno	100%	100%	20	100%	60%	12	100%	15%	3
sqlEx	67%	100%	6	100%	100%	4	100%	75%	3
stmtAluno2	80%	100%	102	82%	100%	100	27%	7%	22
stmtAluno	0%	100%	7	0%	0%	0	0%	0%	0
linearSearch	15%	100%	66	14%	50%	37	14%	50%	37
stmtTurma	50%	100%	16	0%	0%	5	0%	0%	4
rset	0%	100%	5	0%	0%	3	0%	0%	2
textToHtml	42%	100%	12	100%	100%	5	100%	20%	1
fimMapa	0%	100%	4	0%	0%	4	0%	0%	2
connUSE	12%	100%	17	67%	100%	3	100%	100%	2
rsetProex	9%	100%	11	9%	100%	11	0%	0%	4
fimSimulacao	100%	100%	7	100%	14%	1	0%	0%	0
stmtProex	0%	100%	4	0%	0%	1	0%	0%	1
connMatricula	100%	100%	2	100%	50%	1	0%	0%	0
inicioTrancamento	0%	100%	2	0%	0%	0	0%	0%	0
connProfessor	16%	100%	25	0%	0%	14	0%	0%	0
AbrirConexao	100%	100%	4	100%	25%	1	0%	0%	0
AlteracaoEmail	0%	100%	0	0%	0%	0	0%	0%	0
AbrirConexaoUSE	0%	100%	26	0%	0%	0	0%	0%	0
fimRetif	0%	100%	11	0%	0%	11	0%	0%	11

Figura C.1: Precisão e cobertura com os centróides para a *clustering*, parte 1

FecharSessao	100%	100%	4	100%	100%	4	100%	100%	4
rsetSIAPE	88%	100%	17	88%	100%	17	88%	100%	17
conn	0%	100%	11	0%	0%	0	0%	0%	0
connIndef	0%	100%	0	0%	0%	0	0%	0%	0
rsetUSE	50%	100%	8	50%	75%	6	50%	50%	4
rsetTPSala	0%	100%	0	0%	0%	0	0%	0%	0
FecharConexaoUSE	5%	100%	42	8%	100%	26	20%	100%	10
stmtPredio	25%	100%	52	87%	100%	15	100%	46%	6
UserAlterado	100%	100%	1	0%	0%	0	0%	0%	0
stmtMatricula	0%	100%	46	0%	0%	43	0%	0%	21
stmtData	80%	100%	5	100%	100%	4	100%	100%	4
fimDesmembramento	0%	100%	0	0%	0%	0	0%	0%	0
sqlEx11	0%	100%	0	0%	0%	0	0%	0%	0
sqlEx10	35%	100%	31	46%	100%	24	25%	27%	12
rsetInscricao	16%	100%	38	12%	67%	33	13%	67%	32
fimDeferimento	44%	100%	59	0%	0%	0	0%	0%	0
stmtDiverso	0%	100%	0	0%	0%	0	0%	0%	0
stmtRequisito	0%	100%	2	0%	0%	0	0%	0%	0
retPoloFull	83%	100%	12	100%	100%	10	0%	0%	0
rset1	86%	100%	21	100%	61%	11	100%	61%	11
inicioTI	100%	100%	3	100%	100%	3	100%	100%	3
rsetTurma	0%	100%	45	0%	0%	5	0%	0%	5
Modulo11	0%	100%	0	0%	0%	0	0%	0%	0
rset3	0%	100%	0	0%	0%	0	0%	0%	0
rset2	14%	100%	22	0%	0%	16	0%	0%	6
rsetCargo	0%	100%	2	0%	0%	0	0%	0%	0
connHorario	7%	100%	15	0%	0%	12	0%	0%	3
stmtConjunto	0%	100%	53	0%	0%	51	0%	0%	50
stmtDisciplina	0%	100%	5	0%	0%	5	0%	0%	0
stmtControle	100%	100%	3	100%	33%	1	0%	0%	0
jbInit	0%	100%	4	0%	0%	2	0%	0%	1
inicioTE	70%	100%	37	61%	54%	23	57%	46%	21
rsetEmail	0%	100%	0	0%	0%	0	0%	0%	0
AbrirConexaoRH	100%	100%	15	100%	87%	13	100%	87%	13
rsetJuncao	0%	100%	0	0%	0%	0	0%	0%	0
rsetControle3	0%	100%	0	0%	0%	0	0%	0%	0
rsetControle2	20%	100%	15	20%	100%	15	20%	100%	15
stmtEnfase	0%	100%	19	0%	0%	1	0%	0%	0
rsetIndef	0%	100%	3	0%	0%	3	0%	0%	0
rsetDiverso	0%	100%	21	0%	0%	17	0%	0%	3
dData	0%	100%	1	0%	0%	0	0%	0%	0
rsetMatricula	0%	100%	0	0%	0%	0	0%	0%	0
inicioSR	0%	100%	2	0%	0%	2	0%	0%	0
rsetConjunto	0%	100%	1	0%	0%	0	0%	0%	0
stmtUSE	0%	100%	6	0%	0%	2	0%	0%	0
postamble	0%	100%	0	0%	0%	0	0%	0%	0
inicioRetif	0%	100%	0	0%	0%	0	0%	0%	0
rsetlog	0%	100%	0	0%	0%	0	0%	0%	0
rsetRequisito	52%	100%	31	7%	6%	15	9%	6%	11
FecharConexaoProex	54%	100%	26	0%	0%	10	0%	0%	10
stmtTipo	0%	100%	0	0%	0%	0	0%	0%	0

Figura C.2: Precisão e cobertura com os centróides para a *clustering*, parte 2

rsetSala	67%	100%	6	50%	50%	4	0%	0%	1
rsetPredio	100%	100%	2	100%	100%	2	100%	100%	2
pstmtUsuario	0%	100%	5	0%	0%	3	0%	0%	1
stmtTPSala	0%	100%	0	0%	0%	0	0%	0%	0
stmtDados	0%	100%	12	0%	0%	10	0%	0%	0
inicioOpcaoEnfase	100%	100%	2	100%	100%	2	0%	0%	0
rsetHorario	0%	100%	0	0%	0%	0	0%	0%	0
dataSol	100%	100%	1	100%	100%	1	100%	100%	1
rsetAluno	0%	100%	5	0%	0%	1	0%	0%	0
connConjunto	0%	100%	0	0%	0%	0	0%	0%	0
connRequisito	0%	100%	18	0%	0%	9	0%	0%	9
inicioCancelamento	22%	100%	9	100%	100%	2	100%	100%	2
stmtPolo	0%	100%	0	0%	0%	0	0%	0%	0
inicioDeferimento	0%	100%	0	0%	0%	0	0%	0%	0
rsetData	0%	100%	9	0%	0%	5	0%	0%	4
connIngres2	100%	100%	1	0%	0%	0	0%	0%	0
fimTrancamento	0%	100%	2	0%	0%	0	0%	0%	0
stmtIndef	100%	100%	1	100%	100%	1	0%	0%	0
connlog	16%	100%	31	16%	100%	31	0%	0%	26
rsetProfessor	100%	100%	3	100%	100%	3	100%	100%	3
stmtLog	0%	100%	2	0%	0%	0	0%	0%	0
connPolo	100%	100%	7	0%	0%	0	0%	0%	0
stmtEmail	0%	100%	5	0%	0%	5	0%	0%	0
dfim_validade	0%	100%	47	0%	0%	0	0%	0%	0
FiltraNull	82%	100%	17	82%	100%	17	0%	0%	3
stmtRH	0%	100%	0	0%	0%	0	0%	0%	0
agora	0%	100%	0	0%	0%	0	0%	0%	0
conn2	100%	100%	2	100%	100%	2	100%	100%	2
conn3	100%	100%	4	100%	100%	4	100%	100%	4
conn1	67%	100%	6	67%	100%	6	67%	100%	6
connInscricao	22%	100%	9	22%	100%	9	40%	100%	5
connDisciplina	0%	100%	5	0%	0%	0	0%	0%	0
stmtAux	0%	100%	40	0%	0%	18	0%	0%	0
connProex	31%	100%	16	50%	100%	10	44%	80%	9
stmtIngres	0%	100%	0	0%	0%	0	0%	0%	0
rsetDepartamentoCurso	0%	100%	0	0%	0%	0	0%	0%	0
connIngres	29%	100%	7	29%	100%	7	100%	100%	2
rsetDisciplina	0%	100%	0	0%	0%	0	0%	0%	0
retAC	0%	100%	21	0%	0%	17	0%	0%	13
connRH	100%	100%	5	100%	40%	2	100%	40%	2
dColacao	80%	100%	15	86%	100%	14	100%	100%	12
rsetPolo	0%	100%	0	0%	0%	0	0%	0%	0
fimColacao	100%	100%	2	0%	0%	0	0%	0%	0
inicioComplementacaoCurso	0%	100%	2	0%	0%	0	0%	0%	0
rsetRH	1%	100%	69	100%	100%	1	100%	100%	1
rsetControle	60%	100%	5	100%	100%	3	100%	33%	1
rsetTipo	0%	100%	2	0%	0%	0	0%	0%	0
stmtJuncao	0%	100%	0	0%	0%	0	0%	0%	0
dAgora	0%	100%	1	0%	0%	0	0%	0%	0
rsetLog	0%	100%	0	0%	0%	0	0%	0%	0
dateNow	67%	100%	3	0%	0%	0	0%	0%	0

Figura C.3: Precisão e cobertura com os centróides para a *clustering*, parte 3

stmtH	0%	100%	4	0%	0%	4	0%	0%	4
stmtProfessor	0%	100%	9	0%	0%	8	0%	0%	2
stmtSala	40%	100%	5	40%	100%	5	100%	100%	2
connPredio	0%	100%	0	0%	0%	0	0%	0%	0
fimSR	29%	100%	14	29%	100%	14	100%	100%	4
rsetIngres	0%	100%	1	0%	0%	0	0%	0%	0
stmt	0%	100%	0	0%	0%	0	0%	0%	0
inicioDesmembramento	0%	100%	4	0%	0%	4	0%	0%	0
stmtS	0%	100%	1	0%	0%	0	0%	0%	0
replaceText	86%	100%	14	78%	58%	9	78%	58%	9
floatToStr	17%	100%	6	100%	100%	1	100%	100%	1
FecharConexaoRH	19%	100%	16	100%	100%	3	0%	0%	0
stmtInscricao	58%	100%	69	91%	100%	44	93%	100%	43
inicioMapa	22%	100%	18	27%	100%	15	0%	0%	0
fimTI	0%	100%	0	0%	0%	0	0%	0%	0
rsetUsuario	38%	100%	13	38%	100%	13	100%	100%	5
dinicio_validade	0%	100%	10	0%	0%	1	0%	0%	0
fimTE	90%	100%	63	90%	100%	63	0%	0%	6
connDados	50%	100%	4	67%	100%	3	0%	0%	0
inicioColacao	67%	100%	12	0%	0%	0	0%	0%	0
stmtCargo	67%	100%	3	67%	100%	3	0%	0%	1
CampoBreak	0%	100%	0	0%	0%	0	0%	0%	0
doPost	0%	100%	0	0%	0%	0	0%	0%	0
stmtUsuario	0%	100%	0	0%	0%	0	0%	0%	0
	34%	100%	2469	42%	58%	1517	47%	52%	769

Figura C.4: Precisão e cobertura com os centróides para a *clustering*, parte 4