UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ABORDAGEM RPN PARA A RECUPERAÇÃO DE PROCESSOS DE NEGÓCIO BASEADA NA ANÁLISE ESTÁTICA DO CÓDIGO FONTE

LUIZ ALEXANDRE PACINI RABELO

ORIENTADOR: PROF. DR. ANTONIO FRANCISCO DO PRADO

São Carlos – SP

2015

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ABORDAGEM RPN PARA A RECUPERAÇÃO DE PROCESSOS DE NEGÓCIO BASEADA NA ANÁLISE ESTÁTICA DO CÓDIGO FONTE

LUIZ ALEXANDRE PACINI RABELO

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. Antonio Francisco do Prado

São Carlos – SP

2015

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária UFSCar
Processamento Técnico
com os dados fornecidos pelo(a) autor(a)

Rabelo, Luiz Alexandre Pacini
R114ar Abordagem RPN para a recuperação de processos de negócio baseada na análise estática do código fonte / Luiz Alexandre Pacini Rabelo. -- São Carlos : UFSCar, 2016.
94 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2015.

1. Processo de negócio. 2. Recuperação de processos de negócio. 3. Reengenharia de software. 4. Engenharia reversa. 5. Análise estática. I. Título.

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a Defesa de Dissertação de Mestrado do candidato Luiz Alexandre Pacini Rabelo, realizada em 02/09/2015:

Prof. Dr. Antonio Francisco do Prado
UFSCar

Prof. Dr. Wanderley Lopes de Souza
UFSCar

Prof. Dr. Luís Ferreira Pires
University of Twente

Primeiramente dedico este trabalho a Deus, que é minha fonte de inspiração. De modo especial, dedico esse trabalho a minha esposa Karen, que completa minha vida, e aos meus pais Valdecir e Sueli, que sempre batalharam para dar condições para que eu pudesse vencer na vida.

AGRADECIMENTOS

Em primeiro lugar eu agradeço a Deus, por ser essencial em minha vida, autor do meu destino, meu guia e inspiração, e por me dar oportunidade, capacidade e vontade para realizar este trabalho. Apesar das dificuldades, em nenhum momento fiquei desamparado e, graças a Ele, pude concluir meus objetivos.

De modo especial, agradeço a minha esposa Karen, minha fiel companheira que sempre me apoiou, me deu força e me incentivou ao longo dessa jornada. Agradeço também aos meus pais, Valdecir e Sueli, que me ensinaram a importância dos estudos e sempre batalharam para me dar condições para que eu pudesse vencer na vida.

Agradeço ao meu orientador Dr. Antonio Francisco do Prado pela oportunidade de desenvolver meu mestrado, pela paciência e sobretudo pelos seus ensinamentos que me fizeram crescer ao longo desses anos. Gostaria de agradecer também aos professores Dr. Luís Ferreira Pires e Dr. Wanderley Lopes de Souza pelas inúmeras contribuições ao longo do desenvolvimento desse projeto de mestrado.

Obrigado aos meus amigos e companheiros de café que de certa forma participaram direta e indiretamente desta conquista, em especial ao Vitinho, Papotti, Guido, Jãum, Alessandro e Steve do LaBDES (The Best), Maranhão e Durelli do AdvanSE, Abade, Bento e Odair do LaPES, sem contar Lucas Venezian e Denis de outros laboratórios.

Aos professores do DC - UFSCar e FACOM - UFMS que participaram da minha formação.

A CAPES e ao PPGCC pelo apoio financeiro.

Por fim, a todos que ajudaram, torceram, ou de alguma forma, contribuíram para meu sucesso na realização deste trabalho.

O Importante é Lutar

"Não confunda derrotas com fracasso nem vitórias com sucesso. Na vida de um campeão sempre haverá algumas derrotas, assim como na vida de um perdedor sempre haverá vitórias. A diferença é que, enquanto os campeões crescem nas derrotas, os perdedores se acomodam nas vitórias"

Roberto Shinyashiki

RESUMO

Ao longo do tempo, processos de negócio se tornaram um artefato chave para organizações, visto que esses processos permitem gerenciar o que acontece dentro de seus ambientes. É possível automatizar algumas atividades de processos de negócio recorrendo ao uso de sistemas de informação e, dessa forma, diminuir o tempo de execução dessas atividades e aumentar a produção. Entretanto, ao longo do tempo, sistemas de informação sofrem diversas manutenções e tornam-se obsoletos em suas tecnologias e um processo de reengenharia torna-se necessário. Nesse caso, o conhecimento do negócio, localizado mais precisamente à realidade no código fonte do sistema de informação, deve ser mantido. Por este motivo, este trabalho propõe uma abordagem para apoiar a recuperação de processos de negócio a partir do código fonte. A abordagem, nomeada RPN, recorre à técnica de análise estática do código fonte, uma vez que essa técnica permite analisar o código fonte de um sistema sem a necessidade de modificá-lo e executá-lo. Além disso, a abordagem utiliza o padrão *Knowledge Discovery Metamodel* (KDM) com um conjunto de regras de heurísticas para recuperar elementos de código relevantes à camada de negócio. Como resultado, são gerados modelos de processos de negócio de acordo com a especificação padrão *Business Process Model and Notation* (BPMN). Esses modelos, em conjunto com outros artefatos de software, fornecem maiores subsídios para o processo de reengenharia de software. Para avaliar a abordagem proposta, foi realizado um estudo de caso no domínio acadêmico para mensurar a eficácia da abordagem comparado às outras abordagens e ao processo manual. Os resultados obtidos foram satisfatórios e a abordagem RPN mostrou-se muito eficaz e eficiente para executar seu propósito.

Palavras-chave: Processo de Negócio; Recuperação de Processos de Negócio; Reengenharia de Software; Engenharia Reversa; Análise Estática; BPMN.

ABSTRACT

Over time, Business Processes have become a key asset for organizations since it allows managing what happens within their environments. It is possible to automate some activities of business processes resorting to the use of Information Systems and accordingly decrease the execution time and increase the production. However, Information systems often suffer maintenance over time and become obsolete in their technologies and a reengineering process becomes necessary. In this case, the Business Knowledge, located more accurately the reality in information system source code, should be mantained. Thereof, in this work, we propose an Approach to support the Business Process Recovery from Source Code. The approach, entitled RPN, uses a static analysis technique of source code because it allows to analyze the source code without the need to modify and run the information system source code. Furthermore, the approach uses the Knowledge Discovery Metamodel (KDM) standard with a set of Heuristic rules to identify relevant code elements to the business layer. As result, Business Process Models are generated according to Business Process Model and Notation (BPMN) standard specification. This models, together with other software artifacts, provide more subsidies to the Software Reengineering process. To evaluate the proposed approach, a case study was performed in Academic Domain to measure the effectiveness of the approach compared to the other approaches and the manual process. The results exceeded expectations and prove that the approach is effective.

Keywords: Business Process; Business Process Recovery; Software Reengineering; Reverse Engineering; Static Analysis; BPMN.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS

SUMÁRIO

Capítulo 1

INTRODUÇÃO

Neste capítulo, apresenta-se a introdução ao tema desta dissertação de mestrado, suas motivações e objetivos. Na Seção 1.1, aborda-se o contexto desta pesquisa; na Seção 1.2, trata-se das motivações e objetivos; na Seção 1.3, apresenta-se as questões de pesquisa e a metodologia empregada; e na Seção 1.4, descreve-se a estrutura deste documento.

1.1 Contexto

Processos de Negócio representam um conjunto de atividades realizadas em uma organização com o propósito de alcançar um Objetivo de Negócio, isto é, um produto ou um serviço. Esses processos fornecem um meio de mapear os objetivos de negócio em relação à melhor forma de se realizar as operações em diferentes domínios de aplicação, e por este motivo tornaram-se um artefato chave (WESKE, 2012; RUMMLER; BRACHE, 2012).

Além disso, processos de negócio apoiam o Processo de Desenvolvimento de Software, constitui a etapa de Análise de Requisitos e define requisitos essenciais para alcançar os objetivos de negócio (JESTON; NELIS, 2008).

Atualmente, a maioria das organizações recorrem ao uso de sistemas de informação para automatizar determinadas atividades de seus processos de negócio e, consequentemente, reduzir o tempo de execução das mesmas (PÉREZ-CASTILLO, 2012). No entanto, sistemas de informação não estão imunes aos efeitos do tempo e, ao longo de diversas manutenções, se tornam obsoletos em suas tecnologias (VISAGGIO, 2001). Isso deve-se ao fato do Domínio de Negócio e do Domínio de Software estarem sujeitos a constantes mudanças e evoluções independentes. Enquanto no Domínio de Negócio os processos são adaptados para atender a exigências de clientes, no Domínio de Software novos recursos são adicionados aos sistemas de informação para

manter a competitividade.

Com isso, ao longo do tempo, o vínculo entre ambos os domínios se afasta do vínculo inicial documentado, tornando o sistema de informação cada vez mais defasado e sem uma documentação coerente e capaz de auxiliar as manutenções (ZOU, 2004). Logo, para manter a coerência entre os processos de negócio e o código fonte, torna-se necessário realizar um processo de Reengenharia de Software, isto é, atualizar ou substituir o sistema de informação (PRESSMAN, 2010).

Contudo, realizar um processo de Reengenharia para reconstruir um sistema de informação não é trivial, visto a necessidade de se preservar o Conhecimento de Negócio incorporado a ele. Comparado a outros artefatos, com por exemplo a Documentação do sistema, o conhecimento de negócio é encontrado mais precisamente à realidade no código fonte do sistema de informação (PARADAUSKAS; LAURIKAITIS, 2006).

Os processos de negócio podem ser utilizados pelas organizações de duas formas: (I) para fornecer aos especialistas de negócio uma melhor compreensão do real funcionamento atual das organizações; e (II), se necessário, para desenvolver um novo sistema melhorado, a fim de atenuar os efeitos do envelhecimento. O sistema melhorado mantêm os processos de negócio e melhora o Retorno do Investimento (do inglês, *Return of Investment (ROI)*), uma vez que se aumenta a vida útil do sistema (PÉREZ-CASTILLO, 2012).

1.2 Motivações e Objetivos

Embora grandes avanços tenham sido realizados na área de Reengenharia de Software, sobretudo na fase de Engenharia Reversa, ainda restam desafios para melhorar este cenário. A fase de Engenharia Reversa é capaz de obter representações abstratas de sistemas de informação, incluindo a representação do conhecimento de negócio, porém, Projetos de Reengenharia de Software raramente chegam ao nível de abstração de negócio, atingindo geralmente apenas o nível de projeto do sistema (KHUSIDMAN; ULRICH, 2007).

Diante desse cenário, ferramentas para a análise do código fonte tornam-se alternativas para apoiar a recuperação de processos de negócio incorporados ao código fonte e, consequentemente, a Reengenharia de Software.

Uma vez atendidas essas necessidades, têm-se uma redução do tempo e do esforço despendido para realizar esta tarefa e, com isso, é possível realocar recursos para outras tarefas do processo de reengenharia. Além disso, os artefatos obtidos, em conjunto com os demais arte-

fatos existentes, fornecem maiores subsídios para os profissionais envolvido compreenderem o que o sistema faz e, assim, potencializam o processo de reengenharia.

1.3 Questões de Pesquisa e Metodologia

Nesse sentido, o objetivo deste trabalho foi desenvolver uma abordagem para a recuperação de processos de negócio baseada na análise estática do código fonte. Para o desenvolvimento dessa abordagem foram consideradas as seguintes questões de pesquisa:

QP_1: Como realizar a recuperação de processos de negócio?

- **$QP_{1.1}$:** Quais são as soluções existentes?
- **$QP_{1.2}$:** Quais são os artefatos de software necessários?
- **$QP_{1.3}$:** Como extrair o conhecimento de negócio de artefatos de software?
- **$QP_{1.4}$:** Como representar os processos de negócio recuperados?

QP_2: Como validar a abordagem?

- **$QP_{2.1}$:** Quais são as métricas avaliadas?
- **$QP_{2.2}$:** Quais são os resultados esperados?

Para responder a essas questões de pesquisa, a metodologia empregada constituiu-se dos seguintes passos:

1. Efetuar o levantamento bibliográfico sobre abordagens de recuperação de processos de negócio;
2. Definir um modelo independente de plataforma para representar os artefatos de software;
3. Definir um conjunto de regras para recuperar elementos de processos de negócio;
4. Definir uma linguagem de especificação de processos de negócio;
5. Definir um mapeamento e as transformações entre o modelo independente de plataforma e a linguagem de especificação de processos de negócio;
6. Desenvolver um protótipo da abordagem; e
7. Realizar um estudo de caso no domínio acadêmico empregando o protótipo desenvolvido.

1.4 Estrutura da Dissertação

Conforme ilustrado na Figura 1.1, esta dissertação está organizada em seis capítulos, incluindo este capítulo introdutório. O conteúdo de cada capítulo é brevemente descrito a seguir:

Figura 1.1: Estrutura da Dissertação.

- No **Capítulo 2** é descrita a fundamentação teórica relacionada aos principais conceitos e técnicas abordados nesta pesquisa: Processo de Negócio, Engenharia Reversa e Descoberta de Conhecimento;
- No **Capítulo 3** é apresentada e detalhada a abordagem RPN desenvolvida neste trabalho para apoiar a recuperação de processos de negócio a partir do código fonte. Cada uma das etapas da abordagem são definidas, apresentando suas respectivas entradas e saídas e a relação entre essas etapas;
- No **Capítulo 4** é mostrado um estudo de caso onde a abordagem RPN é aplicada a um sistema real no domínio acadêmico e realizada a avaliação dos resultados obtidos para fins de comparação;

- No **Capítulo 5** é brevemente descrito todos os passos conduzidos na Revisão Sistemática a fim de se caracterizar o estado da arte. Em seguida, são discutidos e comparados os principais trabalhos encontrados na literatura que se relacionam com o trabalho desenvolvido; e

- No **Capítulo 6** são discutidas as conclusões deste trabalho, apresentando as contribuições e limitações e norteando pesquisadores interessados em trabalhos futuros.

Capítulo 2

FUNDAMENTAÇÃO TEÓRICA

Para o desenvolvimento deste trabalho, foi fundamental a aquisição de conhecimentos relativos à Processos de Negocio, Engenharia Reversa na Análise de Código Fonte e Descoberta de Conhecimento e o Meta-modelo KDM, os quais são descritos respectivamente na Seção 2.1, Seção 2.2 e Seção 2.3 deste capítulo.

2.1 Processo de Negócio

Embora a literatura ofereça inúmeras definições para processos de negócio, a maioria delas convergem para a mesma ontologia, isto é, a maioria dessas definições convergem para a mesma ideia geral, diferenciando-se apenas em alguns aspectos específicos. A maioria das definições modernas do termo 'processo de negócio' podem ser rastreadas até as definições dos proponentes da área de Reengenharia de Processos de Negócio do início da década de 1990 (KO, 2009). Hammer e Champy (HAMMER; CHAMPY, 2009) definem um processo de negócio, em seu livro sobre Reengenharia de Processos de Negócio, como sendo "[...] um conjunto de atividades que assume um ou mais tipos de entrada e gera uma saída de valor para o cliente. Um processo de negócio tem um objetivo e é afetado por eventos que ocorrem no mundo externo ou em outros processos". Apesar de sua forma genérica, esta é uma das definições mais conhecidas devido sua abrangência.

No entanto, ao invés de visualizar processos de negócio como somente um conjunto de atividades, há a necessidade de visualizá-los como uma sistemática ordenação de atividades de trabalho específicas através do tempo e lugar (KO, 2009). Davenport propôs uma definição em seu livro sobre o uso de tecnologias da informação para a inovação de processos (DAVENPORT, 1993), que "[...] um processo é simplesmente um conjunto estruturado de atividades destinadas a produzir uma saída específica para um dado cliente ou mercado. Isso implica numa forte

ênfase em como o trabalho é feito dentro de uma organização, em contraste com a ênfase focada no produto. O processo é, portanto, uma ordenação específica de atividades de trabalho através do tempo e lugar, com um começo, um fim, e entradas e saídas claramente identificadas".

Em resumo, ambas as definições definem os objetivos e a estrutura (fluxo, tempo e lugar) de um processo de negócio. No entanto, existem outros elementos ausentes nessas definições, como os atores das atividades e a colaboração entre eles, mas que não serão aprofundados neste trabalho uma vez que este trabalho foca principalmente nos objetivos e estrutura definidos nos parágrafos anteriores.

Processos de negócio podem ser representado por modelos gráficos ou textuais e tem como objetivo representar o conjunto de operações realizadas para se alcançar o objetivo de negócio pretendido. Esses modelos são definidos por Linguagens de Processos de Negócio que podem ser gráficas ou textuais. As Linguagens de Processos de Negócio Gráficas permitem aos usuários expressar processos de negócio e seus possíveis fluxos e transições de uma forma diagramática, enquanto as Linguagens de Processos de Negócio Textuais visam a execução desses processos. Duas das principais Linguagens de Processos de Negócio são *Business Process Model and Notation* (BPMN) e *Web Service Business Process Execution Language* (WS-BPEL), sendo a primeira uma linguagem gráfica orientada a grafos e a segunda textual e estruturada em blocos (KO; LEE; Wah Lee, 2009; JORDAN, 2007; MAZANEK; HANUS, 2011).

O BPMN é uma especificação para a modelagem gráfica de fluxos de processos de negócio e serviços web atualmente padronizada pela Object Management Group (OMG). Sua primeira versão, o BPMN 1.0, foi criado pela *Business Process Management Initiative* (BPMI) e disponibilizado ao público em Maio de 2004 e seu principal objetivo é oferecer uma notação facilmente compreensível por todos os envolvidos, desde os analistas de negócio que criam os rascunhos iniciais dos processos, os desenvolvedores responsáveis pela implementação da tecnologia que irá executar esses processos, até os gerentes de negócio que monitoram esses processos (WHITE, 2004a).

Além disso, o BPMN também conta com um mecanismo de transformação de modelos gráficos descritos em BPMN para modelos executáveis descritos em WS-BPEL e vice-versa. Por esse motivo, foi criado um modelo interno que relaciona a modelagem e a implementação de processos de negócio permitindo a transformação entre ambos os modelos (OWEN; RAJ, 2003; WHITE, 2004a).

A modelagem BPMN é realizada com diagramas baseados na técnica de fluxograma que possuem um conjunto de elementos gráficos responsáveis por representar as atividades e os fluxos de controle. Apesar de simples e intuitivo, o BPMN é capaz de lidar com a complexidade

inerente aos processos de negócio. Para isso, o BPMN define cinco tipos básicos de elementos gráficos, sendo que para cada elemento básico existem variações e informações adicionais que podem ser adicionadas para suportar os requisitos de complexidade sem mudar radicalmente sua aparência (OWEN; RAJ, 2003; WHITE, 2004a; OMG, 2011b). Mais detalhes sobre a especificação podem ser encontradas em `http://www.bpmn.org/`.

Como este trabalho tem como objetivo a representação de modelos de processos de negócio para fins de documentação, torna-se necessário representar tais processos de maneira intuitiva para facilitar sua compreensão. O BPMN antede esses requisitos e, por este motivo, foi adotado no contexto deste trabalho. Para auxiliar na construção dos modelos BPMN, foi utilizado o JBoss jBPM[1], um conjunto de ferramentas de gerenciamento de processos de negócio *open source* que permite modelar, executar e monitorar processos de negócio. A modelagem pode ser feita de três maneiras: I) a partir de um editor gráfico disponibilizado; II) a partir de um arquivo XML em conformidade com a especificação do BPMN; e III) a partir da programação Java utilizando uma API também disponibilizada (The JBoss jBPM Team, 2014).

Tendo em vista a necessidade de se automatizar a construção dos modelos, foi utilizada a API fornecida pelo jBPM. Ela permite descrever processos de negócio de forma programática na linguagem Java usando *factories*, gerar sua representação de acordo com a especificação definida pelo BPMN e validá-los. Os principais elementos da API são definidos nos pacotes `org.jbpm.workflow.core` e `org.jbpm.workflow.core.node`. O Código 2.1 a seguir ilustra um exemplo de um processo de negócio em jBPM.

```
1  RuleFlowProcessFactory factory = RuleFlowProcessFactory
       .createProcess("br.ufscar.sin.prograd.negocio.Matricula");
2
3  factory
4      // Cabecalho
5    .name("Matricula")
6    .version("1.0")
7    .packageName("br.ufscar.sin.prograd.negocio")
8
9    // Nos
10   .startNode(1).done()
11   .actionNode(2).name("obterTodasMatriculasAtivasParaCursosPresenciais()").done()
12   .splitNode(3).type(TYPE_XOR)
13     .constraint(4, "true", "code", "Java", "Matricula matricula : matriculas")
14     .constraint(14, "false", "code", "Java", "Matricula matricula :
           matriculas").done()
15   .actionNode(4).name("obterInscricoes(matricula)").done()
16   .splitNode(5).type(TYPE_XOR)
17     .constraint(6, "true", "code", "Java", "Inscricao inscricao : inscricoes")
18     .constraint(3, "false", "code", "Java", "Inscricao inscricao :
           inscricoes").done()
```

[1] Disponível em `http://www.jbpm.org/`

```
19    ...
20    .endNode(14).done()
21
22    // Conexoes
23    .connection(1,2)
24    .connection(2,3)
25    .connection(3,4)
26    .connection(3,14)
27    .connection(4,5)
28    .connection(5,6)
29    .connection(5,3)
30    ...;
31
32  RuleFlowProcess process = factory.validate().getProcess();
```

Código 2.1: Exemplo de processo de negócio em jBPM.

O método `createProcess` da classe `RuleFlowProcessFactory` é responsável por criar um novo processo. Um processo típico é composto por três partes: o Cabeçalho, que compreende elementos globais como o nome do processo, importações, variáveis, etc; os Nós, que contém os diferentes nós que fazem parte do processo; e as Conexões, que interligam os nós uns aos outros para criar um fluxograma.

No Código 2.1, o cabeçalho possui apenas as informações relacionadas ao nome do processo e nome do pacote. O processo é composto por seis nós, iniciando-se no nó `startNode`, que dá início ao processo, e finalizando no nó `endNode`, que marca o final do processo. Além disso, o processo possui duas atividade do tipo `actionNode`, que realizam operações de consulta a outros métodos, e dois *gateways*, responsáveis por dividir o fluxo em vários caminhos conforme condições (do inglês *constraint*) estabelecidas e/ou unir diversos fluxos em um único. Cada tipo de nó possui propriedades específicas que podem ser definidas. Ao final de cada uma das definições é necessário invocar o método `done` para marcar o final de uma definição e iniciar a próxima. Definidos os nós, é necessário interligá-los a fim de formar o fluxograma. Isso pode ser feito chamando o método `connection` que interliga um par de nós de acordo com seu identificador. Finalmente, é chamado o método `validate` para validar o processo construído e recuperar o objeto `RuleFlowProcess` criado. A Figura 2.1 abaixo ilustra o modelo BPMN construído a partir desse código.

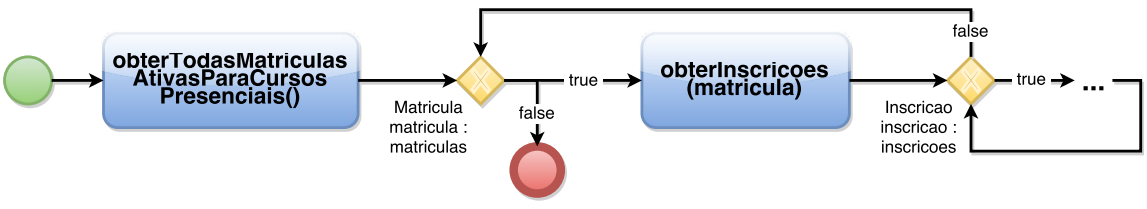

Figura 2.1: Modelo BPMN construído a partir do exemplo apresentado no Código 2.1.

2.2 Engenharia Reversa na Análise de Código Fonte

A Engenharia Reversa é um ramo da Engenharia de Software responsável por possibilitar a recuperação de informações perdidas ao londo do desenvolvimento do software. De acordo com (CHIKOFSKY; CROSS et al., 1990), a engenharia reversa pode ser definida como o processo de análise e compreensão do software existente, para identificar seus componentes e seus inter-relacionamentos e apresentá-los em um grau ou nível mais alto de abstração, de forma a não envolver mudanças no software ou criação de software (MAGALHÃES, 2008; BRAGA, 1998). Em outras palavras, quer dizer "andar para trás ao longo do ciclo de desenvolvimento" (HALL, 1992).

Além disso, de acordo com (SOMMERVILLE, 2001), se o código fonte estiver disponível, ele é a entrada para o processo de engenharia reversa. Caso contrário, o processo de engenharia reversa se inicia pelo código executável. Em resumo, o objetivo da engenharia reversa é derivar o projeto ou a especificação de um software a partir de seu código fonte. A engenharia reversa é usada durante o processo de reengenharia a fim de recuperar o projeto do Software usados pelos engenheiros para sua compreensão antes de reorganizar sua estrutura. Contudo, a engenharia reversa não precisa ser sempre seguida da reengenharia (MAGALHÃES, 2008).

Mediante o nível de entendimento do software e o escopo das informações utilizadas, (OMAN; COOK, 1990) e (CHIKOFSKY; CROSS et al., 1990) definem duas categorias de engenharia reversa:

- **Visualização de código**. Tem como objetivo melhorar o entendimento do software criando representações a partir de dados coletados no código fonte. As formas das representações são consideradas visões alternativas, cujo objetivo é melhorar a compreensão do sistema global. Esta é a forma mais simples e antiga de engenharia reversa;
- **Entendimento do programa**. Visa a recuperações mais complexas, como função, propósitos ou essência do software, a partir da análise de uma combinação de código fonte, documentos existentes, experiências pessoais e conhecimentos do domínio, a fim de se recriar abstrações de projeto para compreender o software.

Este trabalho foca na categoria de visualização de código, uma vez que o objetivo deste trabalho é oferecer modelos de processos de negócio a fim de se melhorar a compreensão dos processos de negócio inerentes ao software. A criação dessas visões adicionais do código fonte permitem recriar a documentação do software, que existiu ou deveria ter existido, por meio da análise do código fonte (FELTRIM, 1999; MAGALHÃES, 2008).

A análise do código fonte é o processo de extração automática de informações de um programa a partir do seu código fonte. Ela pode ser realizada estaticamente, sem executar o programa, dinamicamente, em tempo de execução, ou numa combinação de ambas. As informações extraídas ajudam os desenvolvedores a terem uma visão quanto ao significado do código fonte e é muitas vezes descrito em modelos (BINKLEY, 2007).

A análise estática pode ser definida como a análise do código fonte para a extração de informações básicas do software, como acesso a variáveis e chamadas de funções, sendo normalmente realizada com o uso de um analisador sintático (*parser*[2]). A análise dinâmica, por outro lado, consiste na execução de um programa e monitoramento dos valores das variáveis, funções chamadas, etc (VASCONCELOS, 2007).

A análise estática refere-se ao processo de análise de um pedaço de código sem executá-lo (CHESS; WEST, 2007; BARDAS, 2010). Ela é feita com base em regras pré-definidas e é tipicamente automatizado com o uso de ferramentas (SVOBODA, 2014). Existem vários tipos de abordagens de análise estática, cada qual com seu propósito (CHESS; WEST, 2007). Um dos tipos é a Compreensão de Programa, cujo objetivo é ajudar os desenvolvedores a entenderem a lógica inerente ao código fonte, especialmente de grandes programas que contém dezenas de milhares de linhas de código (SVOBODA, 2014).

Uma das principais diferenças entre as análises estática e dinâmica é que, enquanto a análise dinâmica exercita um cenário de uso específico a cada execução, com um determinado conjunto de valores de entrada e gerando um caminho específico de execução da aplicação, a análise estática varre completamente o código fonte a fim de identificar todos os caminhos possíveis a se percorrer sem considerar um conjunto de dados de entrada. Logo, torna-se difícil generalizar os modelos extraídos através da análise dinâmica a fim de se cobrir totalmente o código fonte.

A análise estática, ao contrário da análise dinâmica, não requer o uso de instrumentação do código fonte para obter o fluxo de execução do programa. Qualquer programa pode ser analisado estaticamente, mas apenas programas com determinada instrumentação do código fonte podem ser analisados dinamicamente. A atividade de instrumentar programas para adequá-los ao uso da análise dinâmica pode demandar uma quantidade significativa de tempo e torná-la mais custosa em comparação com a análise estática. Por outro lado, a análise estática não é capaz de obter determinadas informações como informações temporais, possíveis somente em tempo de execução com a análise dinâmica (KLEIDERMACHER; KLEIDERMACHER, 2012).

[2] *Parsing* é o processo de analisar uma sequência de entrada para determinar sua estrutura gramatical segundo uma determinada gramática formal. Esse processo é formalmente chamado de análise sintática. Um *parser* é um programa de computador que realiza essa tarefa (VASCONCELOS, 2007).

Tendo em vista a necessidade deste trabalho em se obter todos os fluxos de chamadas de função possíveis do software, a análise estática torna-se uma opção, uma vez que é suficiente para se obter quais funções uma outra função pode chamar. Por este motivo, este trabalho utiliza a análise estática do código fonte para se obter o encadeamento e relacionamento das funções.

2.3 Descoberta de Conhecimento e o Meta-modelo KDM

Existem diversas definições para Descoberta de Conhecimento. Segundo Fayyad *et al.* (FAYYAD; SHAPIRO; SMYTH, 1996), este termo pode ser definido como o processo de extração de conhecimentos válidos, novos, potencialmente úteis e compreensíveis, visando melhorar o entendimento de um problema. A descoberta de conhecimento pode recorrer ao uso de diversas áreas para ser realizada, como métodos estatísticos, reconhecimento de padrões, visualização, aprendizado de máquina, dentre outras (SASSI, 2006).

Para descobrir conhecimento que seja relevante, é importante estabelecer metas bem definidas. Fayyad *et al.* (FAYYAD, 1996b) define também que no processo de descoberta de conhecimento, as metas são definidas em função dos objetivos, podendo ser de dois tipos básicos: Verificação ou Descoberta (ROMÃO, 2002).

Quando a meta é do tipo Verificação, o sistema está limitado a verificar hipóteses definidas pelo usuário, enquanto que na Descoberta o sistema encontra novos padrões de forma autônoma. A meta do tipo descoberta pode ser subdividida em Previsão e Descrição. A Descrição procura encontrar padrões interpretáveis pelos usuários que descrevem os dados. Já a previsão parte de diversas variáveis para prever outras variáveis ou valores desconhecidos (FAYYAD, 1996a; ROMÃO, 2002).

O processo de descoberta de conhecimento é composto por uma série de etapas que buscam transformar os dados de baixo nível em conhecimento de alto nível (GOEBEL; GRUENWALD, 1999). Basicamente, as etapas são: (I) pré-processamento, (II) mineração de dados e (III) pós-processamento. A etapa de mineração de dados é a principal atividade do processo de descoberta de conhecimento e tem como finalidade descobrir as informações úteis (FELDENS, 1998).

A descoberta de conhecimento de software legado tem como objetivo descobrir e extrair o conhecimento do negócio de fontes legada, ou seja, gerar uma descrição detalhada das fontes legada, incluindo entidades, relacionamentos, regras de negócio, etc. Coletivamente, essas informações podem ser denominadas de Conhecimento do Negócio (PARADAUSKAS; LAURIKAITIS, 2006).

O Knowledge Discovery Meta-model (KDM)[3] é uma especificação que define um meta-modelo para a modelagem de diferentes artefatos de software envolvidos em um sistema de informação legado (por exemplo, código fonte, banco de dados, etc.) a fim de representar o conhecimento implícito nesses artefatos. O KDM é uma especificação padrão definida pela OMG e uma norma internacional ISO/IEC 19506, cujo meta-modelo fornece uma visão geral do comportamento, estrutura e dados de sistemas legados. O meta-modelo é representado em XML (*eXtensible Markup Language*) usando padrões OMG, como o MOF (*Meta-Object Facility*) e XMI (*XML Meta-data Interchange*) (ISO/IEC, 2012; OMG, 2011a; PÉREZ-CASTILLO; De Guzman; PIATTINI, 2011).

O KDM se diferencia da UML (*Unified Modeling Language*) pois não é um modelo que representa restrições, comumente usados na fase de projeto, mas um modelo que captura precisamente o conhecimento de uma aplicação. Além disso, enquanto a UML pode ser usada para gerar um novo código fonte de forma *top-down*, o KDM parte de diferentes artefatos de software legado para construir modelos de nível de abstração maior de forma *bottom-up* através de técnicas de Engenharia Reversa (MOYER, 2009; PÉREZ-CASTILLO; De Guzman; PIATTINI, 2011).

Para minimizar a complexidade do KDM, a estrutura do meta-modelo KDM é dividido em várias camadas que representam os artefatos de software legado em diferentes níveis de abstração. A Figura 2.2 mostra a organização do meta-modelo KDM.

Figura 2.2: Arquitetura do KDM (ISO/IEC, 2012).

O meta-modelo é composto por quatro camadas: Infraestrutura, Elementos de Programa, Recursos em Tempo de Execução e Abstração. Cada camada de abstração é baseada na camada anterior e organizada em pacotes que definem um conjunto de elementos cujo objetivo é representá-los de forma específica e independente. Cada pacote depende de um ou mais paco-

[3]Disponível em `http://www.omg.org/spec/KDM/`

tes das camadas inferiores, sendo que todos os pacotes dependem do pacote Core responsável por definir todos os elementos do meta-modelo, e do pacote KDM, responsável por fornecer a infraestrutura para todos os modelos KDM. Cada elemento acima da camada KDM define um modelo KDM que corresponde a um determinado aspecto do conhecimento de um software (ISO/IEC, 2012).

O pacote `Source` e o pacote `Code` representam o fundamental conhecimento primitivo de softwares existentes. A maior parte deste conhecimento é explicitamente representado pelo código fonte original do software existente. Espera-se que implementações do KDM extraiam esse tipo de informação automaticamente através de um mapeamento de uma linguagem de programação para uma representação KDM independente de linguagem (ISO/IEC, 2012).

Os pacotes da camada Recursos em Tempo de Execução representam um nível maior de conhecimento de softwares existentes. A maior parte do conhecimento é implicitamente representado pelo código fonte original e as descrições de configuração e recursos correspondentes. Esse tipo de conhecimento não é determinado pela sintaxe e semântica da linguagem de programação, mas pela plataforma de execução corresponde. A análise incremental da representação da camada inferior pode ser necessária para extrair e representar parte desse conhecimento. Implementações do KDM para esses pacotes devem definir um mapeamento de artefatos de plataforma específica para uma representação KDM independente de linguagem e plataforma (ISO/IEC, 2012).

Finalmente, os pacotes da camada Abstração representam até mesmo o nível de abstração mais alto de softwares existentes, como o conhecimento específico de domínio, regras de negócio, conhecimento da arquitetura do software, etc. Este conhecimento está implícito, e muitas vezes não há uma representação formal desse conhecimento em artefatos de software (e, muitas vezes, nem mesmo na documentação). Para se extrair este tipo de conhecimento pode ser necessário o envolvimento de especialistas e analistas (ISO/IEC, 2012).

Em geral, o KDM permite: (I) armazenar os fatos de um sistema de informação em uma estrutura compatível; (II) analisar e discutir sobre esses fatos; (III) transformar esses fatos em documentos XML; e (IV) criar ferramentas de análise de código fonte em uma determinada linguagem de programação para gerar fatos independente de linguagem (PÉREZ-CASTILLO; De Guzman; PIATTINI, 2011). Maiores detalhes sobre a especificação KDM podem ser encontrados em `http://www.omg.org/spec/KDM/`.

O MoDisco[4] é um projeto de código aberto oficialmente parte da *Eclipse Fundation* e integrado ao projeto de Modelagem para a promoção de técnicas de Engenharia dirigida por

[4]Disponível em `http://www.eclipse.org/MoDisco/`

modelos (do inglês, *Model-Driven Engineering (MDE)*) e seu desenvolvimento dentro da comunidade Eclipse. Ele oferece um *Framework* extensível para o desenvolvimento de ferramentas baseada em modelos para apoiar o processo de Modernização de Software. Atualmente, é reconhecido e citado pela OMG por fornecer implementações e ferramentas de vários padrões, como o KDM (FAVRE, 2005; BRUNELIERE, 2014).

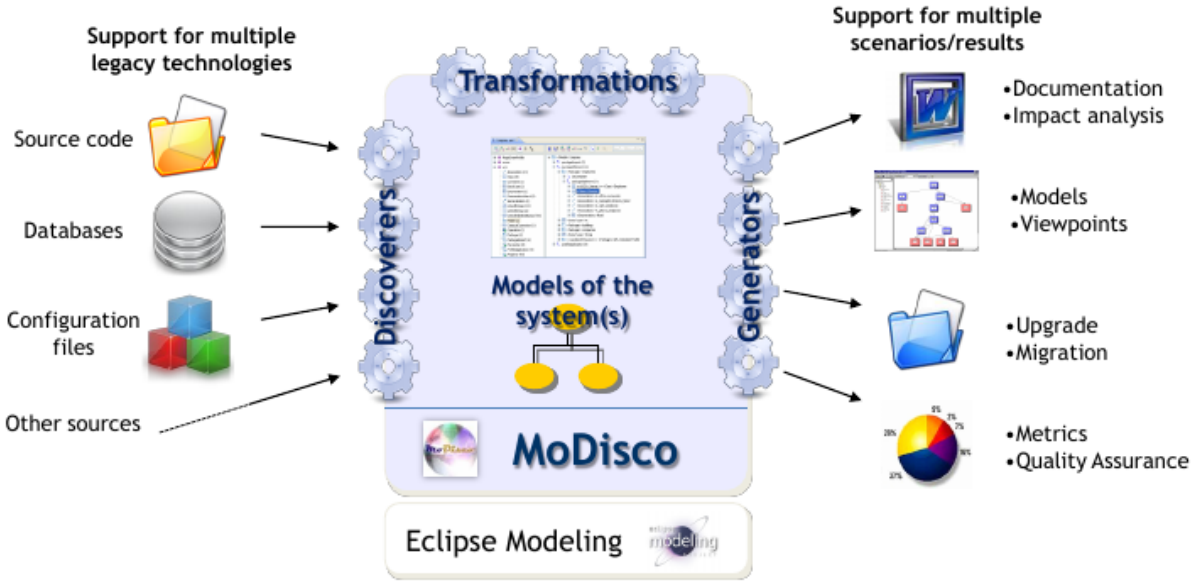

Figura 2.3: Visão geral do MoDisco (BRUNELIERE, 2014).

A Figura 2.3 apresenta uma visão geral do MoDisco. A partir de artefatos legados (por exemplo, código fonte, banco de dados, documentação, etc) o MoDisco visa fornecer os recursos necessários para a criação de modelos de representação e permitir sua manipulação, análise e computação. Para isso, ele oferece vários componentes para a Descoberta, Transformação e Geração de artefatos. Como saída, o *Framework* visa a produção de diferentes artefatos de software, dependendo do objetivo da Engenharia Reversa (por exemplo, Modernização de Software, Refatoração, Re-documentação, etc.) (BRUNELIERE, 2010, 2014).

Um componente adicional do MoDisco permite ainda realizar a descoberta do modelo KDM a partir de Modelos Java e a rastreabilidade de forma automática integrando ambos os modelos KDM e Java (BRUNELIERE, 2014).

Capítulo 3

ABORDAGEM RPN

Neste capítulo é descrita a abordagem RPN desenvolvida neste trabalho. Na Seção 3.1 é fornecida uma visão geral da abordagem RPN; na Seção 3.2 é tratada a atividade de Descoberta do Modelo KDM; na Seção 3.3 é apresentada a atividade de Recuperação dos Elementos de Processos de Negócio; e na Seção 3.4 é abordada a atividade de Representação dos Processos de Negócio.

3.1 Visão Geral

Com o objetivo de apoiar o processo de Reengenharia de Software, neste trabalho propõe-se uma abordagem de apoio à recuperação de processos de negócio baseada na análise estática do código fonte de sistemas de informação. A abordagem, intitulada RPN (Recuperação de Processos de Negócio), visa proporcionar uma forma prática e automatizada de se recuperar processos de negócio incorporados ao código fonte fazendo uso da técnica de análise estática, e, consequentemente, reduzir o tempo e o esforço despendido para realizar esta tarefa. Além disso, a abordagem permite também representar os processos de negócio recuperados em modelos BPMN, uma notação gráfica intuitiva e de fácil compreensão.

A Figura 3.1 ilustra uma visão geral da abordagem RPN. Como artefato de entrada, é necessário fornecer o código fonte de sistemas de informação. A partir do código fonte, é realizada a **Descoberta do Modelo KDM**, que transforma o código fonte em um modelo independente de plataforma (KDM PIM) utilizando o *Framework* MoDisco Discoverer. O modelo KDM PIM tem como objetivo representar o código fonte de sistemas de informação independente de plataforma e de linguagem de programação. Em seguida, o mesmo é submetido à **Recuperação dos Elementos de Processos de Negócio**, que aplica um conjunto de regras de heurística pré-definidas para selecionar apenas os elementos que são relevantes à camada de negócio. Como

resultado, é obtido um modelo KDM PIM modificado que contêm somente elementos de processos de negócio. Por fim, a partir deste modelo, é feita a **Representação dos Processos de Negócio**, que mapeia os elementos de processos de negócio para modelos descritos em BPMN com o auxílio da API jBPM. Os modelos BPMN abstraem a complexidade dos processos de negócio e facilitam seu entendimento para os diversos *stakeholders* envolvidos.

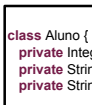

Figura 3.1: Visão Geral da Abordagem RPN.

Conforme ilustrado na Figura 3.1, a abordagem RPN é composta por 3 sub-processos (ou macro-atividades) sequenciais, sendo eles respectivamente, "Descoberta do Modelo KDM", "Recuperação dos Elementos de Processos de Negócio" e "Representação dos Processos de Negócio". Dentre os sub-processos, o segundo, Recuperação dos Elementos de Processos de Negócio, é o sub-processo chave da abordagem, uma vez que contém um conjunto de regras de heurística responsável por identificar quais são os elementos relevantes à camada de negócio. **Note que é importante destacar que o conjunto de regras de heurística é uma das principais contribuições deste trabalho**.

Além da representação apresentada na Figura 3.1, a abordagem RPN poderia ser representada também em forma de um processo descrito em BPMN, como ilustrado a seguir na Figura 3.2.

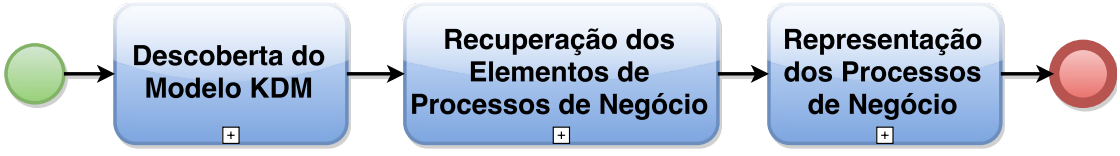

Figura 3.2: Processo da Abordagem RPN em BPMN.

As subseções subsequentes apresentam cada um dos sub-processos da abordagem RPN de forma detalhada, definindo suas respectivas atividades, seus artefatos de entrada e saída, os mecanismos e técnicas utilizadas.

3.2 Descoberta do Modelo KDM

O sub-processo de Descoberta do Modelo KDM tem como objetivo derivar um ou mais arquivos de código fonte de um determinado sistema de informação e representá-los de forma homogênea e uniforme por meio de modelos. Cada modelo está em conformidade com um determinado meta-modelo e tem uma correspondência de um-para-um com o código fonte. O MoDisco é utilizado como suporte nesta atividade para obter o modelo KDM PIM a partir de um código fonte descrito em uma dada linguagem de programação. O modelo obtido contém a AST (*Abstract Syntax Tree*) completa do código fonte da aplicação, a resolução dos vínculos entre identificadores no código fonte e o relacionamento entre os elementos do modelo (COSENTINO, 2013).

Para obter o modelo KDM PIM, são necessárias duas atividades: (I) primeiro, as informações obtidas dos artefatos de software (isto é, o código fonte) são usadas para construir o modelo PSM de acordo com o metamodelo específico da linguagem de programação alvo (por exemplo, um modelo em conformidade com o metamodelo Java); e, em seguida, (II) o KDM PIM é obtido a partir do modelo PSM anterior por meio de transformações de modelos implementadas em ATL (*ATLAS Transformation Language*) (PÉREZ-CASTILLO; De Guzman; PIATTINI, 2011). Portanto, para realizar a Descoberta do Modelo KDM são necessárias duas atividades, uma para se obter o modelo PSM e outra para transformá-lo no modelo KDM PIM.

A Figura 3.3 ilustra as duas atividades do sub-processo Descoberta do Modelo KDM. A primeira atividade é a **Construção do Modelo PSM**, que recebe como artefato de entrada o código fonte e constrói o modelo PSM. Em seguida, a segunda atividade é a **Transformação do Modelo PSM no Modelo KDM PIM**, que realiza a transformação entre o modelo PSM e o modelo KDM PIM.

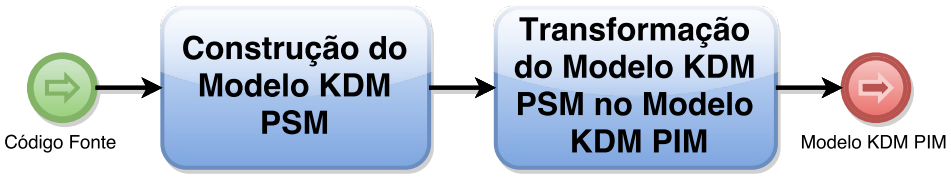

Figura 3.3: Atividades da Descoberta do Modelo KDM.

No decorrer desta Seção, é ilustrado o passo a passo da execução desta macro-atividade para um trecho de código de um sistema real no domínio acadêmico. O Código 3.1 apresenta um trecho da classe `Matricula` em Java, responsável pela matrícula de alunos. Nos exemplos seguintes, será utilizada esta classe para ilustrar os artefatos gerados nas próximas atividades e sub-processos da abordagem RPN. Para maiores detalhes, uma versão estendida da classe Matricula por ser encontrada no Anexo A.

```
1  package br.ufscar.sin.prograd.negocio; // 1
2
3  public class Matricula { // 2
4
5    public void calcularIra() throws DAOException, SQLException { // 3
6
7      List<Matricula> matriculas =
           MatriculaDAO.obterTodasMatriculasAtivasParaCursosPresenciais(); // 4
8
9      for (Matricula matricula : matriculas) { // 5
10
11       List<Inscricao> inscricoes = inscricaoDao.obterInscricoes(matricula); // 6
12
13       for (Inscricao inscricao : inscricoes) { // 7
14
15         if (Integer.parseInt(inscricao.getResultado()) < 6 &&
               inscricao.getStatus().matches("[3,6-9,17]")) { // 8
16           somatoriaNotasXCredCursados = somatoriaNotasXCredCursados +
                 this.somarNotaXCreditosCursado(inscricao); // 9
17         }
18
19         somatoriaCredInscritos = somatoriaCredInscritos +
               this.somarCreditosInscritos(inscricao); // 10
20
21         if (Integer.parseInt(inscricao.getResultado()) == 7) { // 11
22           somatoriaCredCancelados = somatoriaCredCancelados +
                 this.somarCreditosCancelados(inscricao); // 12
23         }
24
25         if (Integer.parseInt(inscricao.getResultado()) == 6) { // 13
26           somatorioCredDesistentes = somatorioCredDesistentes +
                 this.somarCreditosDesistentes(inscricao); // 14
27         }
28       }
29
30       Double ira = (somatoriaNotasXCredCursados / somatoriaCredInscritos) * (2 -
             ((2 * somatorioCredDesistentes + somatoriaCredCancelados) /
             somatoriaCredInscritos)); // 15
31
32       matricula.atualizarIra(ira); // 16
33     }
34   }
35 }
```

Código 3.1: Classe Matricula em Java.

Para facilitar o entendimento, foram adicionados comentários ao código fonte contendo numerações. A numeração é única no escopo do conteúdo do artefato, mas, para tornar evidente as transformações realizadas, a numeração se repete nos trechos correspondentes dos demais artefatos, ou seja, trechos de código de artefatos que possuem a mesma numeração são correspondentes entre si.

O trecho de código apresentado enfoca em um métodos específico implementados na classe Matricula, denominado `calcularIra`. Este método é responsável por calcular o rendimento dos alunos com base em suas notas e créditos. Cada um dos trechos foi enumerado de 1 à 16 e, no decorrer deste capítulo, poder-se-à acompanhar a evolução dos artefatos decorrente das transformações que a abordagem realiza, tendo em vista a numeração associada aos trechos de código.

A transformação da classe Matricula em Java em um modelo PSM (Modelo Java) pode ser facilmente realizada de forma programática utilizando o *Framework* MoDisco. O Código 3.2 abaixo ilustra o trecho de código responsável por invocar a transformação. Primeiro, é necessário criar um objeto do tipo `DiscoverJavaModelFromJavaProject`, conforme a linha 2. Em seguida, um objeto `Map` é criado para adicionar os parâmetros relacionados à descoberta, conforme ilustrado nas linhas 3 a 5. O objeto `discoverer` herda o método `discoverElement` da classe `AbstractModelDiscoverer`, que recebe como parâmetros o tipo do artefato de origem (no caso, um Projeto Java) e um objeto do tipo `Map` contendo os parâmetros da descoberta, e realiza a descoberta do modelo resultante, conforme a linha 6. Por fim, na linha 7, executa o método `get`, que captura o modelo resultante e armazena em um objeto do tipo `Resource`.

```
1 ...
2 DiscoverJavaModelFromJavaProject discoverer = new
      DiscoverJavaModelFromJavaProject();
3 Map javaDiscoveryParameters = new HashMap();
4 javaDiscoveryParameters.put(DefaultDiscoverer.PARAMETER_SILENT_MODE,true);
5 javaDiscoveryParameters.put(DefaultDiscoverer.PARAMETER_BROWSE_RESULT,false);
6 discoverer.discoverElement(javaProject, javaDiscoveryParameters);
7 Resource javaModel= (Resource)
      javaDiscoveryParameters.get(DefaultDiscoverer.PARAMETER_TARGET_RESOURCE)
8 ...
```

Código 3.2: Descoberta do modelo PSM usando a *Framework* MoDisco Discoverer.

No caso da linguagem Java, a descoberta do modelo PSM é realizada por um *Parser* Java, conhecido como ANTLR[1], que constrói a AST do código Java segundo a BNF (*Backus-Naur Form*) que define a linguagem Java. Em seguida, um componente do MoDisco mapeia os elementos da AST para uma estrutura em XMI, resultando no modelo PSM. O mapeamento é

[1] http://www.antlr.org/

realizado com base no Metamodelo da linguagem Java, um metamodelo obtido a partir da BNF da linguagem Java que define o conjunto de elementos existentes na linguagem. A Figura 3.4 abaixo ilustra o metamodelo da linguagem Java.

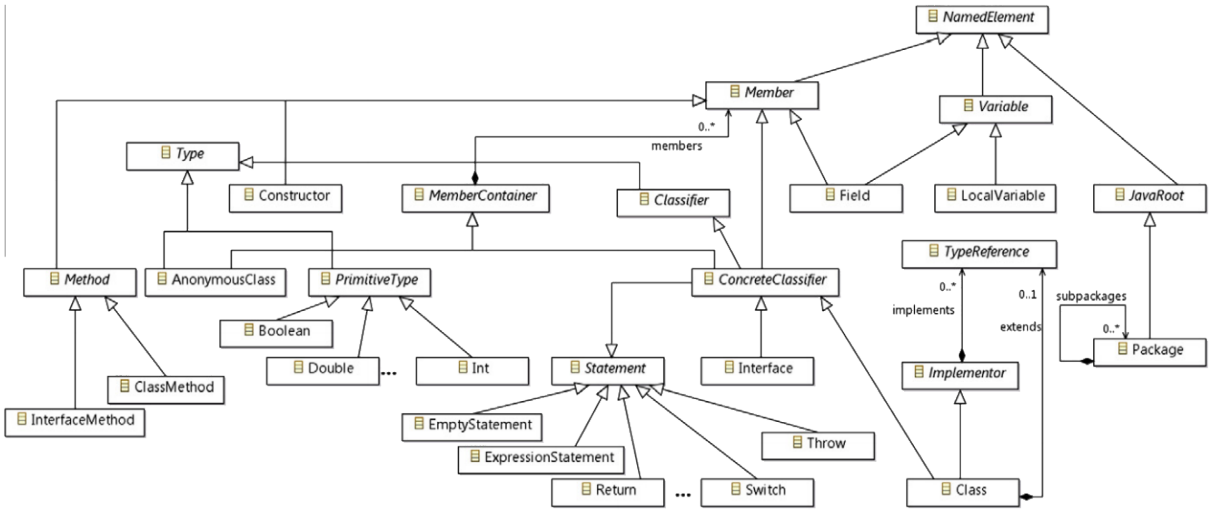

Figura 3.4: Metamodelo da linguagem Java (PÉREZ-CASTILLO; GUZMÁN; PIATTINI, 2011).

Neste metamodelo é possível identificar elementos básicos existentes na grande maioria dos programas Java, como Pacotes, Classes, Métodos e Variáveis, representados respectivamente por `Package`, `Class`, `Method` e `Variable`. Ao identificar os elementos presentes na AST construída, uma instância do metamodelo Java é criada. Como resultado, se obtém o modelo PSM, também conhecido como modelo Java. Isto se dá devido o modelo Java (PSM) se tratar de uma instância do metamodelo Java.

O Código 3.3 ilustra um trecho do modelo PSM obtido a partir da execução da atividade Construção do Modelo PSM para a classe Matricula ilustrada anteriormente no Código 3.1. Este código ilustra apenas um trecho considerado relevante para a compreensão da abordagem e, assim como a classe Matricula, uma versão estendida pode ser encontrada no Anexo B. Vale ressaltar que, assim como a classe Matricula, este modelo atribui numerações a trechos de código de forma a associar os trechos do modelo com os trechos da classe Matricula, bem como com os demais modelos que virão. Assim, é possível compreender de forma mais fácil a transformação sofrida pelo código fonte para se gerar o modelo PSM.

Este modelo tem como finalidade representar o código fonte do ponto de vista de diversos aspectos presentes no código fonte. Por exemplo, a definição da classe Matricula produz uma tag `ownedElements` com os parâmetros `type` igual a `ClassDeclaration`, que descreve a ação de uma declaração de classe, `name` igual Matricula, que associa o nome da classe, além de outras informações. Analogamente, a declaração de um método, como `calcularIra`, produz uma tag `bodyDeclaration` também com os parâmetros de nome e tipo, além de outras propri-

edades inerentes, como exceções, tipo de retorno e argumentos. Existem ainda instruções como declarações de variável, laço `for`, condicional `if`, entre outras, definidas por tags `statements`. Maiores detalhes sobre a estrutura e os elementos do metamodelo Java podem ser encontrados na documentação do pacote `org.eclipse.jdt.core.dom`[2].

```
1  <?xml version="1.0" encoding="ASCII"?>
2  <java:Model>
3
4   <!-- 1 -->
5   <ownedPackages name="negocio">
6
7    <!-- 2 -->
8    <ownedElements xsi:type="java:ClassDeclaration"
         originalCompilationUnit="Matricula.java" name="Matricula"
         usagesInTypeAccess="Class Declaration Matricula">
9
10    <!-- 3 -->
11    <bodyDeclarations xsi:type="java:MethodDeclaration" name="calcularIra">
12     <thrownExceptions type="DAOException"/>
13     <thrownExceptions type="SQLException"/>
14     <returnType type="void"/>
15     <body originalCompilationUnit="Matricula.java">
16
17      <!-- 4 -->
18      <statements xsi:type="java:VariableDeclarationStatement"
            originalCompilationUnit="Matricula.java">
19       <type type="java.util.List<Matricula>"/>
20       <fragments originalCompilationUnit="Matricula.java" name="matriculas"
             usageInVariableAccess="matriculas">
21        <initializer xsi:type="java:MethodInvocation"
              originalCompilationUnit="Matricula.java"
              method="obterTodasMatriculasAtivasParaCursosPresenciais"/>
22       </fragments>
23      </statements>
24
25      <!-- 5 -->
26      <statements xsi:type="java:EnhancedForStatement"
            originalCompilationUnit="Matricula.java">
27       <body xsi:type="java:Block" originalCompilationUnit="Matricula.java">
28
29        <!-- 8 -->
30        <statements xsi:type="java:IfStatement"
              originalCompilationUnit="Matricula.java">
31         <expression xsi:type="java:InfixExpression"
               originalCompilationUnit="Matricula.java"
               operator="&amp;amp;amp;">...</expression>
32         <thenStatement xsi:type="java:Block" originalCompilationUnit="Matricula.java">
33
34          <!-- 9 -->
35          <statements xsi:type="java:ExpressionStatement"
                originalCompilationUnit="Matricula.java">
36           <expression xsi:type="java:Assignment"
                 originalCompilationUnit="Matricula.java">
37     ...
38 </java:Model>
```

Código 3.3: Modelo PSM gerado a partir da classe Matricula em Java.

[2]Disponível em `http://help.eclipse.org/juno/topic/org.eclipse.jdt.doc.isv/reference/api/org/eclipse/jdt/core/dom/package-summary.html`

Assim como para a atividade anterior, o MoDisco oferece um meio simples para a transformação do modelo Java, recém apresentado, para o modelo KDM PIM. O Código 3.4 ilustra um trecho de código que utiliza a classe `TranslateJavaModelToKdm` do MoDisco responsável por realizar a transformação. Essa classe implementa o método getKDMModelFromJavaModelWithCustomTransformation, que recebe como parâmetros o modelo Java e seu endereço, bem como o endereço do modelo KDM, e executa a transformação entre o modelo Java e o modelo KDM PIM.

```
...
TranslateJavaModelToKdm kdmTranslater = new TranslateJavaModelToKdm();
Resource kdmModel =
    kdmTranslater.getKDMModelFromJavaModelWithCustomTransformation(
    javaModel.getURI(), javaModel, kdmModelURI);
...
```

Código 3.4: Transformação do modelo PSM em KDM PIM usando a *Framework* MoDisco Discoverer.

Na sequência, o Código 3.5 apresenta um trecho de código que implementa as transformações na linguagem ATL.

```
...

-- Transfoms a class declaration into a class unit
rule ClassDeclarationToClassUnit {
  from cd : java!ClassDeclaration
  to cu : kdm!ClassUnit (
    name <- cd.name,
    codeElements <- cd.methods
  )
}

-- Transforms a method into a method unit
rule MethodDeclarationToMethodUnit {
  from md : java!MethodDeclaration
  to mu : kdm!MethodUnit(
    type <- signature
  ),
  signature : kdm!Signature(
    name <- md.name
    parameters <- returnParameterUnit
  ),
  returnParameterUnit : kdm!ParameterUnit (
    kind <- return
    type <- md.returnType
  )
}
```

Código 3.5: Transformação do modelo PSM para KDM PIM descrito em ATL (Adaptado de (CUADRADO; ARACIL, 2014)).

A regra `ClassDeclarationToClassUnit` transforma todas as declarações de classes Java (`ClassDeclaration`) em elementos `ClassUnit` do KDM (linhas 4 a 10), enquanto a regra `MethodDeclarationToMethodUnit` transforma cada método Java (`MethodDeclaration`) em um elemento KDM `MethodUnit`, mas também cria um elemento de assinatura do método (`Signature`) e um (`ParameterUnit`) para representar o elemento retornado.

```
1  <?xml version="1.0" encoding="ASCII"?>
2  <kdm:Segment>
3  <model xsi:type="code:CodeModel" name="ProGradWeb">
4
5   <!-- 1 -->
6   <codeElement xsi:type="code:Package" name="negocio">
7
8   <!-- 2 -->
9   <codeElement xsi:type="code:ClassUnit" name="Matricula" isAbstract="false">
10   <attribute tag="export" value="public"/>
11
12   <!-- 3 -->
13   <codeElement xsi:type="code:MethodUnit" name="calcularIra">
14    <codeElement xsi:type="code:Signature" name="calcularIra">
15     <parameterUnit type="void" kind="return"/>
16     <parameterUnit type="DAOException" kind="throws"/>
17     <parameterUnit type="SQLException" kind="throws"/>
18    </codeElement>
19   <codeElement xsi:type="action:BlockUnit">
20
21    <!-- 4 -->
22    <codeElement xsi:type="action:ActionElement" name="variable declaration"
          kind="variable declaration">
23     <codeElement xsi:type="code:StorableUnit" name="matriculas"
           type="java.util.List<Matricula>" kind="local">
24      <attribute tag="export" value="none"/>
25      <codeRelation xsi:type="code:HasValue"
            to="obterTodasMatriculasAtivasParaCursosPresenciais" from="matriculas"/>
26     </codeElement>
27    </codeElement>
28
29    <!-- 5 -->
30    <codeElement xsi:type="action:ActionElement" name="foreach" kind="foreach">
31     <codeElement xsi:type="action:BlockUnit">
32
33      <!-- 8 -->
34      <codeElement xsi:type="action:ActionElement" name="if" kind="if">
35       <codeElement xsi:type="action:ActionElement" name="CONDITIONAL_AND" kind="infix
             expression">
36        ...
37       </codeElement>
38      <codeElement xsi:type="action:BlockUnit">
39
40       <!-- 9 -->
41       <codeElement xsi:type="action:ActionElement" name="expression statement"
            kind="expression statement">
42        ...
43       </codeElement>
44   ...
45  </codeElement>
46 </model>
47 </kdm:Segment>
```

Código 3.6: Modelo PIM gerado a partir do Modelo PSM Java

Para concluir o sub-processo de Descoberta do Modelo KDM, a segunda atividade, denominada Transformação do Modelo PSM no Modelo KDM PIM, é executada e o modelo KDM PIM é obtido pela transformação do Modelo PSM. O Código 3.6 ilustra um trecho do código obtido pela transformação do modelo PSM da classe Matricula. Assim como nos exemplos anteriores, somente um trecho relevante é apresentado para facilitar a compreensão da atividade executada, e uma versão estendida pode ser encontrada no Anexo C. Novamente, nota-se a existência dos comentários enumerados que relacionam os trechos de código deste artefato com os trechos dos demais.

Este modelo tem como objetivo representar o código fonte de forma independente de plataforma e linguagem de programação. Para descrever o código fonte neste modelo, alguns elementos básicos são utilizados, como, por exemplo, o elemento `codeElement`, responsável por mapear elementos de código como pacotes, classes, métodos, declarações de variáveis, laços, condicionais, blocos, etc. Tais elementos são compostos de uma série de características que, em conjunto, descrevem a sintaxe e a semântica presente no código fonte. Por exemplo, uma declaração de métodos é definida através do elemento `codeElement` e propriedades específicas, como atributos, tipo de retorno e exceções. Além disso, para cada elemento, pode ser definido um conjunto de parâmetros a fim de tornar implícito determinados aspectos do código fonte. Em geral, os elementos possuem ao menos os parâmetros que definem o nome (`name`) e o tipo (`xsi:type`) desse elemento. Existem também outros parâmetros de propósito específico como `to` e `from`, usados em atribuições, onde o conteúdo de `to` é atribuído à `from`; `kind`, usados em exceções, retornos, laços e condicionais, para determinar a instrução usada (`throws`, `return`, `for`, `if`, etc); entre outros.

O Código 3.6 recém apresentado, pode ainda ser visualizado graficamente em uma ferramenta fornecida pelo MoDisco. Essa ferramenta oferece uma visão alto-nível de diversos modelos textuais como o PSM e KDM PIM, facilitando assim a sua compreensão. A Figura 3.5 ilustra o código do modelo KDM PIM visto nesta ferramenta. Os elementos são organizados em uma estrutura em árvore e permite expandir ou retrair blocos de código

O modelo KDM PIM gerado nesta atividade será usado pela atividade seguinte da próxima Seção. Esse modelo permite que a abordagem seja aplicável a qualquer sistema de informação, independente da linguagem de programação utilizada no seu desenvolvimento. Isto permitiu generalizar o uso da abordagem RPN, sendo uma das principais contribuições deste trabalho e um dos diferenciais em comparação com outras abordagem existentes.

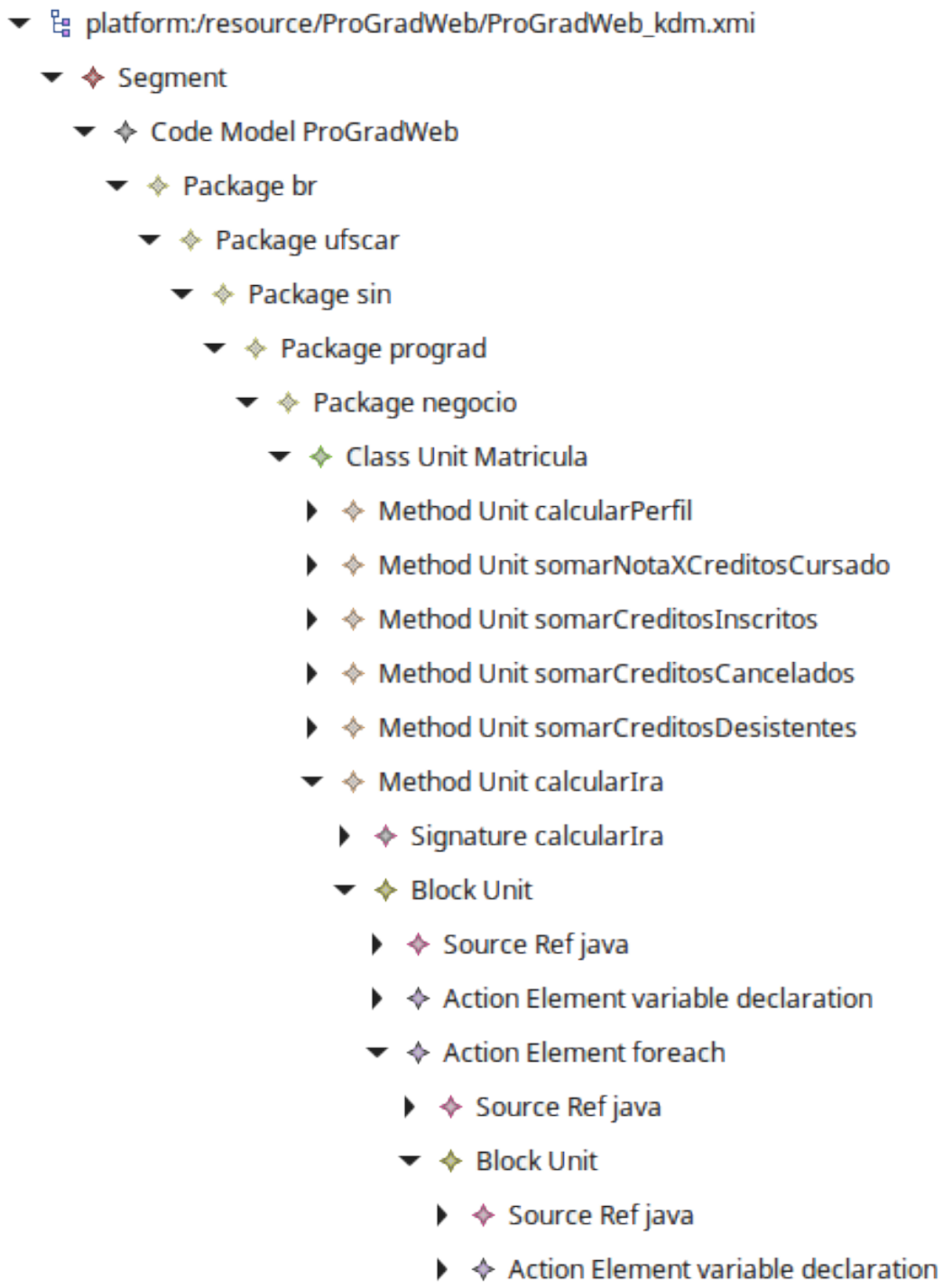

Figura 3.5: Visualização gráfica do modelo KDM PIM.

3.3 Recuperação dos Elementos de Processos de Negócio

O sub-processo Recuperação dos Elementos de Processos de Negócio tem como objetivo recuperar os elementos relevantes à camada de negócio. Para isso, este sub-processo aplica um conjunto de regras de heurística pré-definidas que varrem o modelo KDM PIM gerado pela

etapa anterior e seleciona os trechos relevantes. A Figura 3.6 ilustra a sequência de atividades deste sub-processo. Inicialmente é realizada a **Identificação dos Elementos de Processos de Negócio** com o auxílio de um conjunto de regras de heurística e, em seguida, é realizada a **Representação dos Elementos de Processos de Negócio** que modifica o modelo KDM PIM para conter somente os elementos de processos de negócio identificados pelas regras de heurística.

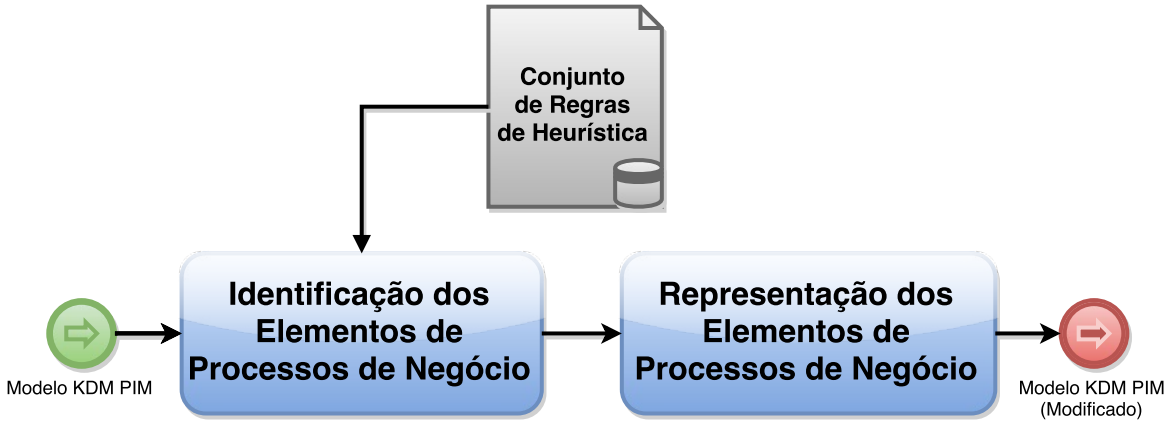

Figura 3.6: Atividades da Recuperação dos Elementos de Processos de Negócio.

O conjunto de regras de heurísticas utilizado na primeira atividade foi definido com base nos Padrões de Processos de Negócio (*Business Process Patterns*) (PÉREZ-CASTILLO; GUZMÁN; PIATTINI, 2010) e nos Padrões de Workflow (*Workflow Patterns*) (WHITE, 2004b). Ambos os padrões auxiliaram na construção das regras de heurística, uma vez que permitiram identificar padrões estruturais presentes na modelagem e na implementação de processos de negócio. Dessa forma, foi possível definir como os elementos de processos de negócio podem ser representados nos modelos KDM PIM para, posteriormente, procurar trechos de código similares.

As heurísticas podem ser classificadas em Regras de Inclusão (RI) e Regras de Exclusão (RE). As regras de inclusão tem como objetivo definir a estrutura básica de um trecho de código do modelo KDM PIM que pode ser relevante. Em outras palavras, as regras de inclusão selecionam todos os elementos de código que satisfazem uma dessas regras. Por outro lado, as regras de exclusão tem como objetivo delimitar o conjunto de elementos selecionados pelas regras de inclusão. Essas regras excluem trechos que possuem um comportamento específico não relevante para a camada de negócio. Portanto, para que um trecho de código seja considerado relevante, ele deve satisfazer pelo menos uma regra de inclusão e ao mesmo tempo não satisfazer nenhum regra de exclusão. Essa divisão permitiu selecionar os elementos de processos de negócio de maneira mais eficaz.

No total, foram definidas 10 Regras de Inclusão e 3 Regras de Exclusão. A seguir, essas regras são apresentadas e ilustradas por exemplos.

RI-1) **Esqueleto**: Identifica os pacotes definidos no modelo KDM PIM. Para cada pacote encontrado, é criado um novo diagrama contendo a estrutura básica de um modelo BPMN. Isso significa que a granularidade dos modelos gerados estão em nível de pacote. Quando um elemento `codeElement` com `xsi:type="code:Package"` é encontrado no modelo KDM PIM, um novo modelo BPMN é criado. Os modelos obtidos são organizados em diretórios mantendo-se a hierarquia de pacotes original. A Figura 3.7 ilustra a execução desta regra para um trecho de código.

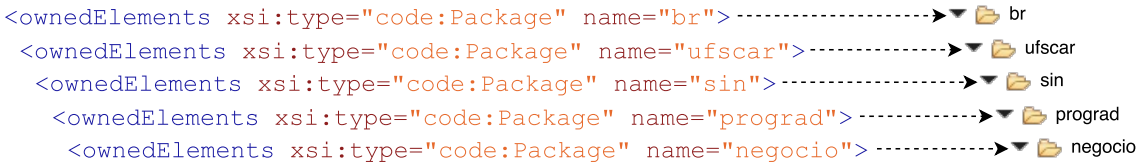

Figura 3.7: Regra de Inclusão Esqueleto.

RI-2) **Sequência**: Para cada declaração de método, representada por um elemento `codeElement` com `xsi:type="code:MethodUnit"`, é criada uma atividade no modelo BPMN correspondente. Em seguida, para cada método, é analisada a sequência de invocações, representadas por elementos `codeElement` e `kind="method invocation"`. Se um método realiza uma invocação à outro método, então é criado um elemento de fluxo de sequência para interligar ambas as atividades. Além disso, fluxos de sequência são criados para definir a sequência de operações dentro dos métodos. A Figura 3.8 ilustra a execução desta regra para um trecho de código.

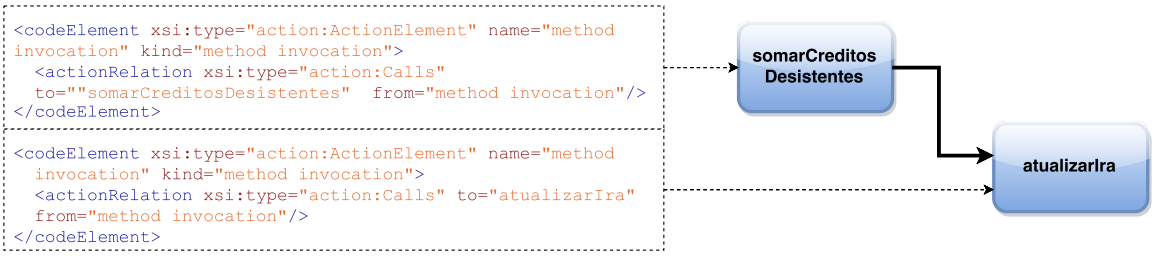

Figura 3.8: Regra de Inclusão Sequência.

RI-3) **Início**: Identifica a primeira atividade do processo, isto é, a atividade que não é invocada por nenhum outro método. Um elemento de evento inicial do BPMN é criado e ligado à essa atividade por meio de um elemento de fluxo de sequência. Cada fluxo, iniciado por um evento inicial, é adicionado a um elemento de Pool. Dessa forma, é possível representar processos que possuem dois ou mais fluxos que são executados de forma independente. A Figura 3.9 ilustra a execução desta regra para um trecho de código.

```
<codeElement xsi:type="code:MethodUnit" name="calcularIra">
  <codeElement xsi:type="action:BlockUnit">
    <codeElement xsi:type="action:ActionElement"
    kind="method invocation">
      <actionRelation xsi:type="action:Calls"
      to="obterTodasMatriculasAtivasParaCursosPresenciais"
      from="matriculas">
```

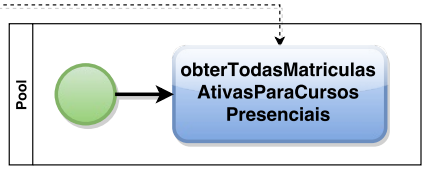

Figura 3.9: Regra de Inclusão Início.

RI-4) **Final**: Identifica as atividades que determinam o final de um determinado fluxo do processo, isto é, atividades que não possuem fluxos de sequência saindo. Um elemento de evento final é criado no modelo BPMN e as atividades finais são conectadas ao evento final por meio de um fluxos de sequência. A Figura 3.10 ilustra a execução desta regra para um trecho de código.

```
<codeElement xsi:type="code:MethodUnit"
name="somaNotaXCreditosCursado">
  <codeElement xsi:type="action:BlockUnit">
    ...
    <codeElement xsi:type="action:ActionElement"
    kind="return">
      <actionRelation xsi:type="action:Calls"
      to="somarCreditos" from="method invocation">
      ...
</codeElement>
```

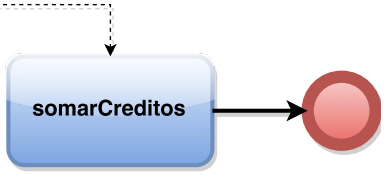

Figura 3.10: Regra de Inclusão Final.

RI-5) **Condicional**: Para cada desvio condicional encontrado no modelo KDM PIM, representado por elementos `codeElement` e `kind="if"` ou `kind="switch"`, é criado um elemento gateway exclusivo no modelo BPMN. Esses desvios geralmente estão relacionados com cláusulas `if then else` e `switch case`. A Figura 3.11 ilustra a execução desta regra para um trecho de código.

```
<codeElement xsi:type="action:ActionElement"
name="if" kind="if">
  <codeElement xsi:type="action:ActionElement"
  name="EQUALS" kind="infix expression">
    ...
    <codeElement xsi:type="action:BlockUnit">
      ...
</codeElement>
```

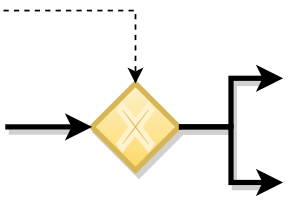

Figura 3.11: Regra de Inclusão Condicional.

RI-6) **Paralelismo**: Para cada desvio paralelo encontrado no modelo KDM PIM é criado um elemento gateway paralelo no model BPMN. Esses desvios geralmente estão relacionados com o uso de *Threads*. Logo, trechos que utilizam a classe `Thread` e a interface `Runnable` ou `Callable` são executados de forma paralela. A Figura 3.12 ilustra a execução desta regra para um trecho de código.

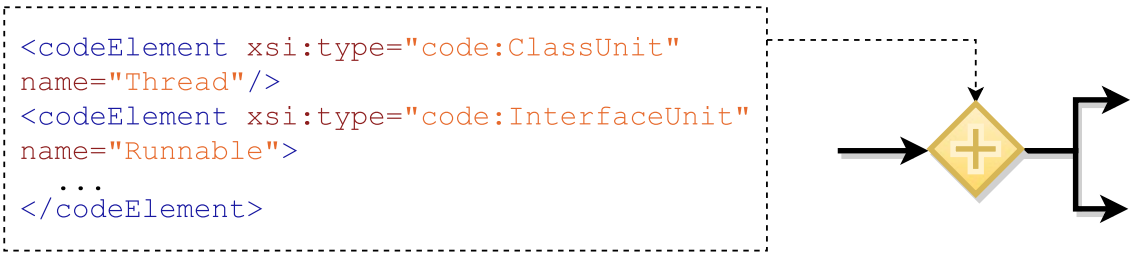

Figura 3.12: Regra de Inclusão Paralelismo.

RI-7) **Colaboração**: Identifica as invocações à métodos de classes externas pertencentes a outros pacotes, componentes ou bibliotecas (API). Esses métodos são transformados em tarefas auxiliares do BPMN e dois fluxos de sequência são criados, um em direção o método externo e outro na direção inversa com o resultado da operação. Para identificar se um método é externo, é utilizado a rastreabilidade fornecida pelo KDM, que permite identificar a origem da classe que implementa o método invocado. A Figura 3.13 ilustra a execução desta regra para um trecho de código.

Figura 3.13: Regra de Inclusão Colaboração.

RI-8) **Entrada de Dados**: Identifica a entrada de dados externos a serem lidos por uma determinada atividade. A leitura de dados externos pode ser representada no KDM como um `actionRelation` e `xsi:type="action:Reads"`. A Figura 3.14 ilustra a execução desta regra para um trecho de código.

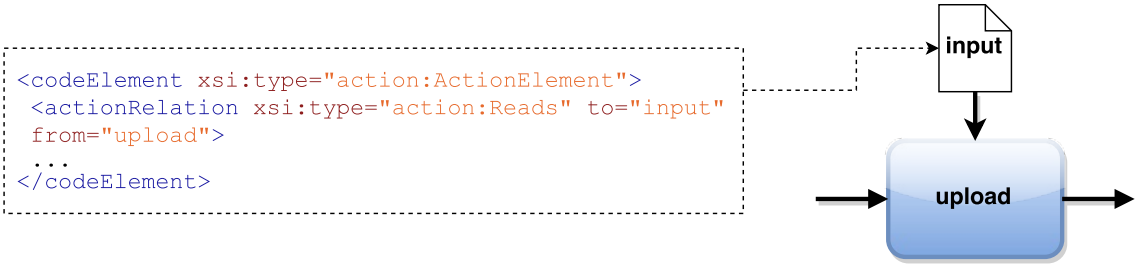

Figura 3.14: Regra de Inclusão Entrada de Dados.

RI-9) **Saída de Dados**: Identifica a saída de dados escritos por uma determinada atividade. De forma análoga à leitura de dados da regra anterior, a escrita de dados pode ser representada no KDM como um `actionRelation` e `xsi:type="action:Writes"`. A Figura 3.15 ilustra a execução desta regra para um trecho de código.

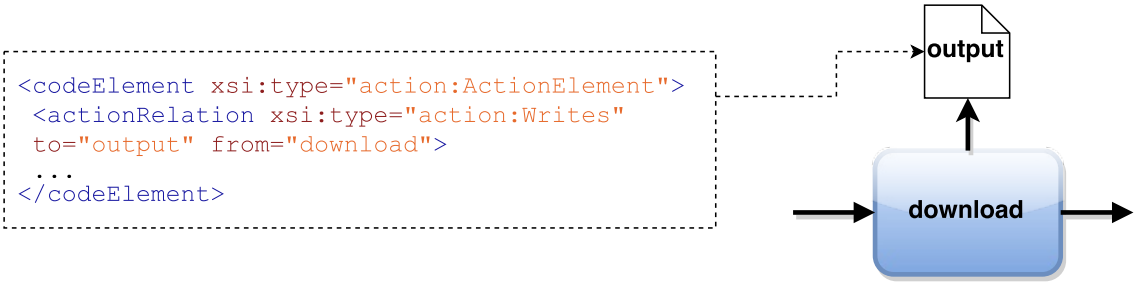

Figura 3.15: Regra de Inclusão Saída de Dados.

RI-10) **Exceção**: Identifica o lançamento e captura de exceções de uma determinada atividade, podendo ser representado no modelo KDM por um `parameterUnit` e `kind="throws"` nos casos de lançamento de exceção, ou por um `codeElement`, `xsi:type="action:CatchUnit"` e `kind="catch"`, nos casos de captura de exceção. Para cada exceção, cria um evento intermediário de erro no modelo BPMN. Um fluxo de sequência com uma condição implícita interliga a atividade que lança a exceção com o evento intermediário de erro. A condição implícita no fluxo de sequência é a própria exceção. A Figura 3.16 ilustra a execução desta regra para um trecho de código.

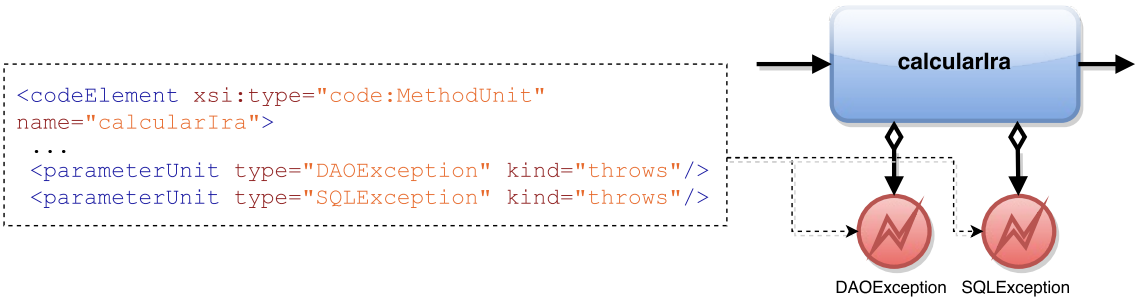

Figura 3.16: Regra de Inclusão Exceção.

As regras de inclusão apresentadas são genéricas e, a partir delas, é possível identificar trechos de código do modelo KDM que satisfaçam ao menos uma regra. No entanto, existem determinados elementos irrelevantes à camada de negócio que possuem estrutura similar à dos elementos relevantes. Para isso, foram definidas regras de exclusão que atuam somente nos elementos classificados como relevantes, a fim de se remover os elementos não-relevantes aceitos pelas regras de inclusão.

A primeira regra de exclusão (RE-1), denominada **Métodos Assessores**, tem como objetivo remover atividades que representam métodos assessores, como por exemplo, os métodos *getters*, *setters* e contrutores. Essa regra permite ainda identificar atividades isoladas, isto é, métodos que não foram invocadas e nem invocam outros métodos não assessores. Esse tipo de atividades são de propósito específico e possuem grau de abstração baixo, e, por este motivo, não são relevantes à camada de negócio.

Outra regra de exclusão é a regra de **Importações Gráficas** (RE-2). Essa regra exclui toda classe que utiliza importações de pacotes gráficos, como por exemplo, os pacotes `java.awt.*` e `javax.swing.*` do Java. Normalmente, essas classes tem como objetivo somente a representação gráfica do conteúdo e não contém atividades que impactam na construção do produto final desejado e, portanto, não são relevantes à camada de negócio.

Existe também uma terceira regra de exclusão (RE-3), denominada **Acesso ao Bando de Dados**. Esta regra possui uma peculiaridade particular de poder ser ao mesmo tempo de inclusão e exclusão, e tem como objetivo representar o acesso ao banco de dados. Ela exclui toda classe que utiliza importações de pacotes de bando de dados, como por exemplo, os pacotes `com.mysql.jdbc.*` e `net.sf.hibernate.*` do Java. No entanto, essa regra também cria um elemento Pool referente ao Bando de Dados e, para cada invocação de métodos pertencentes às classes excluídas, se cria um fluxo de comunicação entre o método que realizou a invocação e o Pool do Banco de Dados com a respectiva operação SQL a ser realizada, conforme ilustrado na Figura 3.17.

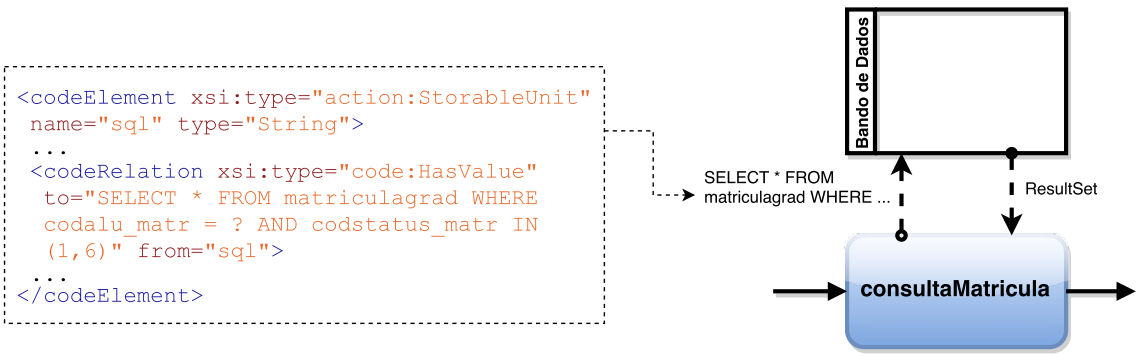

Figura 3.17: Regra de Exclusão Acesso ao Banco de Dados.

Após recuperar os elementos de processos de negócio, é necessário representar os elementos classificados como relevantes. A representação é feita no próprio modelo KDM PIM, utilizando a técnica de recorte do código fonte para retirar os trechos desnecessários e manter apenas o essencial. O modelo KDM PIM modificado resultante da execução desta atividade é similar ao modelo KDM PIM apresentado no Código 3.6 da Seção 3.2. No processo de recorte do código fonte, são realizadas anotação no código fonte para destacar qual regra cada trecho satisfez. Isso permite que a próxima atividade da abordagem realize o mapeamento do modelo KDM PIM modificado diretamente para o modelo BPMN.

3.4 Representação dos Processos de Negócio

O última sub-processo da abordagem RPN é Recuperação dos Processos de Negócio. Este sub-processo tem como objetivo representar os elementos de processos de negócio, identificados anteriormente, em formato de processos de negócio. Os processos são descritos na notação gráfica BPMN, uma especificação padrão para a modelagem de processos de negócio e altamente intuitiva. Para a obtenção dos processos descritos em BPMN, é necessária a execução de duas atividades. A Figura 3.18 a seguir apresenta a sequência de atividades necessárias para a obtenção dos modelos BPMN almejados pela proposta. Esses modelos representam o conhecimento do negócio inerente ao código fonte de sistemas de informação.

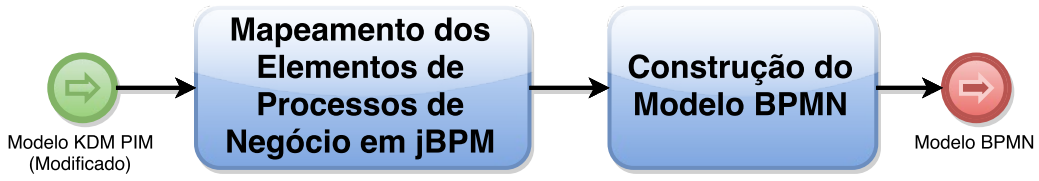

Figura 3.18: Atividades da Representação dos Processos de Negócio.

A atividade **Mapeamento dos Elementos de Processos de Negócio em jBPM**, recebe como artefato de entrada o modelo PSM modificado resultante da atividade anterior. A partir deste modelo, é realizado o mapeamento dos elementos de processos de negócio para elementos de código Java da API jBPM, que por sua vez, realiza a **Construção do Modelo BPMN** de forma automática. Ambas as atividades são apoiadas pela API jBPM, que permite a construção de modelos BPMN por meio da programação de *factories* Java.

Primeiramente, o mapeamento dos elementos de processos de negócio para elementos de código jBPM é realizado de acordo com as anotações existentes no código. Essas anotações,

realizadas pela atividade anterior, descrevem qual regra de heurística foi satisfeita por cada trecho de código. Assim, é possível relacionar os elementos do modelo KDM PIM e os elementos do modelo BPMN. A Tabela 3.1 apresenta o relacionamento entre as Regras de Heurística e a Representação no BPMN utilizado no mapeamento. Um determinado trecho do KDM PIM é representado no BPMN pelo(s) elemento(s) relacionado(s) à regra satisfeita por este trecho.

Tabela 3.1: Relacionamento entre as Regras de Heurística e a Representação no BPMN.

Regra de Heurística	Representação no BPMN
RI-1) Esqueleto	Diagrama de Processo de Negócio
RI-2) Sequência	Atividade e Fluxo de Sequência
RI-3) Início	Evento Inicial, Processo de Negócio e Pool
RI-4) Final	Evento Final
RI-5) Condicional	Gateway Exclusivo
RI-6) Paralelismo	Gateway Paralelo
RI-7) Colaboração	Atividade, Sub-processo e Fluxo de Sequência
RI-8) Entrada de Dados	Objetos de Dados e Associação
RI-9) Saída de Dados	Objetos de Dados e Associação
RI-10) Exceção	Evento de Erro
RE-1) Métodos Assessores	–
RE-2) Importações Gráficas	–
RE-3) Acesso ao Banco de Dados	Fluxo de Comunicação e Pool

A partir da relação apresentada acima, é feito o mapeamento dos elementos do modelo KDM PIM para o jBPM com o auxílio da API jBPM. O evento inicial por ser mapeado para um elemento `startNode` do jBPM, assim como o evento final pode ser mapeado para um elemento `endNode`. As atividades do processo podem ser definidas nos `actionsNodes` e as condicionais e laços nos `splitNodes`. Cada elemento possui uma série de propriedades que podem ser definidas. Por exemplo, condicionais `splitNodes` possuem condições (`constraints`) que devem ser satisfeitas para definir o fluxo a ser tomado. O trecho de Código 3.7 a seguir ilustra o mapeamento realizado a partir do modelo KDM PIM apresentado na seção anterior.

Novamente, nota-se a numeração dos comentários correspondente à numeração presente nos artefatos anteriores. Concluído o mapeamento dos elementos de processos de negócio para elementos do jBPM, é possível construir os modelos BPMN automaticamente com o auxílio da API jBPM e concluir o processo da abordagem RPN. O modelo BPMN resultante pode ser visualizado na Figura 3.19.

```
1  RuleFlowProcessFactory factory = RuleFlowProcessFactory
       .createProcess("ProGradWeb");
2
3  factory
4    // Header
5    .name("Matricula::calcularIra")
6    .version("1.0")
7    .packageName("br.ufscar.sin.prograd.negocio")
8
9    // Nodes
10   .startNode(1).done()
11   .actionNode(2).name("obterTodasMatriculasAtivasParaCursosPresenciais()").done()
12   .splitNode(3).type(TYPE_XOR)
13     .constraint(4, "true", "code", "Java", "Matricula matricula : matriculas")
14     .constraint(14, "false", "code", "Java", "Matricula matricula :
           matriculas").done()
15   .actionNode(4).name("obterInscricoes(matricula)").done()
16   ...
17   .endNode(14).done()
18
19   // Connections
20   .connection(1,2)
21   .connection(2,3)
22   ...
23   .connection(13,4);
24
25 RuleFlowProcess process = factory.validate().getProcess();
```

Código 3.7: Código jBPM referente à classe Matricula.

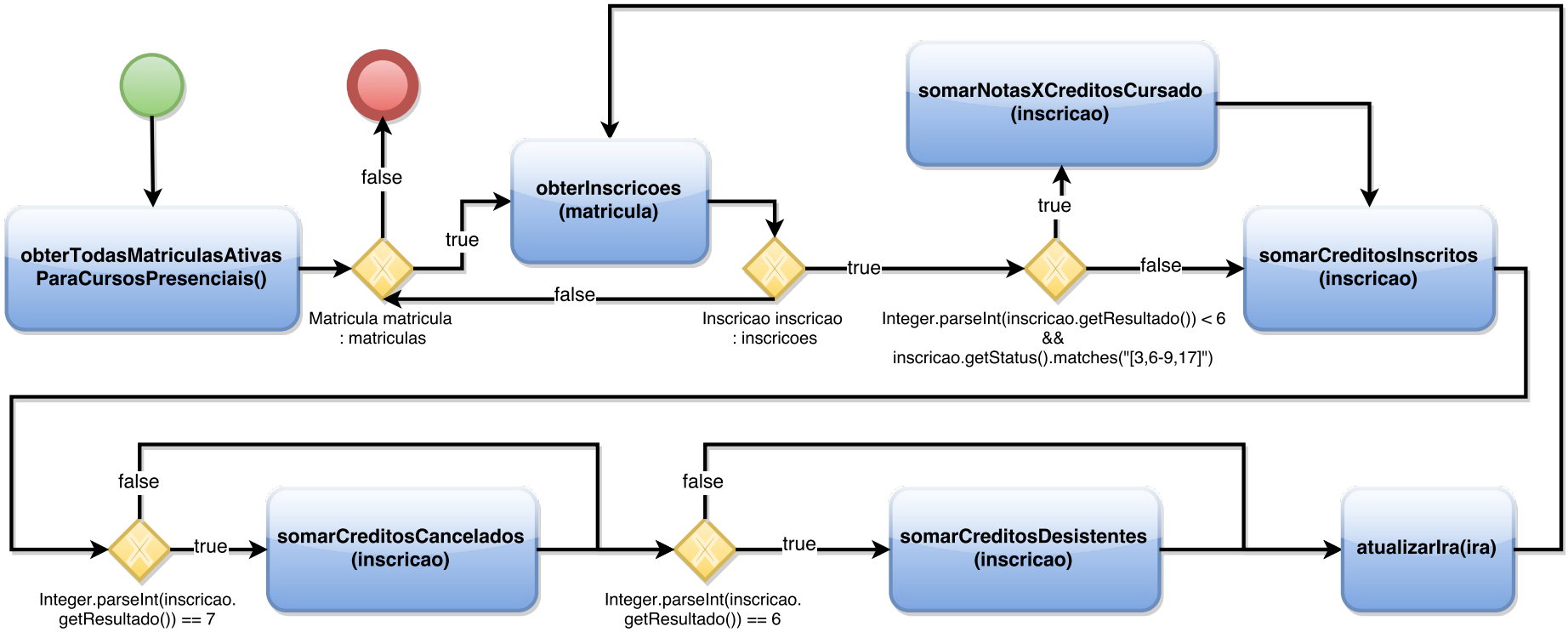

Figura 3.19: Matricula BPMN.

Capítulo 4

ESTUDO DE CASO

Este capítulo apresenta um estudo de caso da abordagem RPN realizado em um sistema legado real no domínio acadêmico. A Seção 4.1 descreve o sistema legado ProGradWeb utilizado neste estudo de caso; e as Seções 4.2 e 4.3 apresentam a avaliação dos resultados obtidos neste estudo de caso sob a perspectiva da Eficiência e da Eficácia respectivamente.

4.1 Sistema ProGradWeb

Com o objetivo de analisar a aplicabilidade da abordagem proposta no apoio à Reengenharia de Software, foi realizado um estudo de caso aplicando a abordagem RPN para a recuperação dos processos de negócio do sistema ProGradWeb[1]. O ProGradWeb é um sistema real proprietário da Universidade Federal de São Carlos (UFSCar) que tem sido utilizado desde meados de 1998. Sua utilização está relacionada à gestão acadêmica de informações a respeito dos cursos de graduação da UFSCar, incluindo dados sobre professores, alunos, cursos e emissões de documentos oficiais.

Ao longo dos anos, o ProGradWeb tem sofrido sucessivas manutenções visando melhorar o uso do sistema e adicionar novas funcionalidades para se adequar às mudanças que a Universidade tem passado. Atualmente, sua manutenção tornou-se onerosa em virtude de algumas tecnologias utilizadas estarem obsoletas, e o sistema carece urgentemente de grandes modificações estruturais. Para isso, é crucial que o conhecimento adquirido ao longo de todos esses anos seja mantido

A arquitetura do sistema consiste em uma única camada, contendo interfaces, controles e regras de negócio, e não faz uso de padrões estruturais. O sistema foi desenvolvido utilizando

[1] ProGradWeb - https://prograd web.ufscar.br/prograd web/

a linguagem Java e utiliza a API JDBC (Java Database Connectivity) para comunicar-se com um banco de dados PostgreSQL[2]. Além disso, o sistema possui cerca de 70.000 SLOC (*Source Lines Of Code*) ou 70 KLOC (*Kilo Lines Of Code*) organizados em 7 pacotes e 167 classes.

Este estudo de caso consiste de duas avaliações, sendo uma quantitativa e outra qualitativa. A avaliação quantitativa visa analisar a eficiência da abordagem desenvolvida com relação ao processo manual, enquanto a avaliação qualitativa visa analisar a eficácia dos artefatos gerados. Nas seções subsequentes, ambas as avaliações são descritas.

4.2 Avaliação Quantitativa

Para avaliar quantitativamente a abordagem RPN, optou-se por fazer uma estimativa quanto a eficiência da abordagem. Para isto, foi estimado o tempo necessário para realizar o processo de recuperação de processos de negócio de forma manual, realizado por um Engenheiro de Software, e comparado com o tempo necessário para realizar o processo automatizado proposto nesta abordagem. Logo, a eficiência pode ser expressa pela seguinte fórmula 4.1.

$$\text{Eficiência} = \frac{\text{Tempo gasto pelo processo manual}}{\text{Tempo gasto pelo processo automatizado}} \tag{4.1}$$

Para a execução do processo automatizado, tendo como entrada o sistema ProGradWeb apresentado na Seção anterior, a abordagem RPN levou cerca de 22 minutos para concluir o processo. Por outro lado, para a execução do processo manual, foi feito uma estimativa em função do tamanho do sistema ProGradWeb, equivalente a aproximadamente 70 KLOC, e da Produtividade do especialista. Assim, o tempo gasto pelo processo manual pode ser obtido pela seguinte fórmula 4.2.

$$\text{Tempo Processo Manual} = \frac{\text{Quantidade de SLOC}}{\text{Produtividade}} \tag{4.2}$$

Por fim, a produtividade por ser calculada através da razão entre a quantidade de linhas de código fonte (SLOC) e o tempo gasto para analisá-las, e pode ser expressa pela seguinte fórmula 4.3:

[2]PostgreSQL - http://www.postgresql.org.br/

$$\text{Produtividade} = \frac{\text{Quantidade de SLOC}}{\text{Tempo de Análise}} \tag{4.3}$$

Para se calcular a Produtividade, foram selecionadas 10 classes aleatórias do sistema ProGradWeb e medido os respectivos tempos gastos para realizar a análise manual dos elementos de negócio presentes em cada uma dessas classes. A Tabela 4.1 sumariza os resultados obtidos.

Tabela 4.1: Resultados obtidos da análise manual.

Classe	SLOC	Tempo em horas
Afastamento	23	0.17
Aluno	703	2.02
Curso	96	0.41
Disciplina	78	0.38
Inscricao	160	0.64
InscricaoAutomatica	158	0.62
Instituicao	174	0.75
Matricula	114	0.47
Trancamento	23	0.15
Turma	132	0.50
Total	**1661**	**6.11**

Conforme apresentado acima, foram analisados um total de 1661 SLOC em aproximadamente 6.11 horas, o que significa que foram analisados cerca de 271.85 SLOC/hora (ou 0.27 KLOC/hora), que representa a Produtividade Média para este conjunto de classes.

$$\text{Produtividade Média} = \frac{1661 \text{ SLOC}}{6.11 \text{ horas}} \approx 271.85 \; \textit{SLOC}/\textit{hora} \tag{4.4}$$

Considerando que a Produtividade Média se mantêm para as demais classes do sistema ProGradWeb, é possível determinar o tempo necessário para se executar o processo manual para todo o sistema ProGradWeb, ou seja, 70 KLOC. Como resultado, têm-se que o tempo estimado para realizar o processo manual para o sistema ProGradWeb é de aproximadamente 259.26 horas.

$$\text{Tempo Estimado Processo Manual} = \frac{70 \text{ KLOC}}{0.27 \text{ KLOC/hora}} \approx 259.26 \; \textit{horas} \tag{4.5}$$

Comparado ao processo automatizado, o processo manual leva cerca de 700 vezes mais tempo, pois, enquanto o processo automatizado leva cerca de 22 minutos (ou 0.37 horas), o processo automatizado leva cerca de 259.26 horas. Portanto, a eficiência da abordagem RPN automatizada com relação ao processo manual é de cerca de 70000%.

$$\text{Eficiência} = \frac{259.26 \text{ horas}}{0.37 \text{ horas}} \approx 700.70 = 70000\% \tag{4.6}$$

4.3 Avaliação Qualitativa

A maioria das propostas de recuperação de processos de negócio encontradas na literatura tem sido validadas qualitativamente através de estudos empíricos em função da eficácia, ou seja, o grau de qualidade dos processos obtidos. Várias métricas têm sido utilizadas para medir a eficácia dessas propostas através da comparação dos processos de negócio recuperados com processos de negócio de referência. Das métricas de eficácia existentes, as mais notáveis são **Precisão** e **Sensibilidade**, duas métricas que tem sido amplamente utilizadas em grande parte dos experimentos e estudos empíricos (PEREZ-CASTILLO, 2011).

A precisão, do inglês *precision*, tem como objetivo mensurar a exatidão e a fidelidade dos processos de negócio recuperados e pode ser definida como o número de atividades relevantes recuperadas dividido pelo total de atividades recuperadas, conforme apresentado na fórmula 4.7 a seguir.

$$\text{Precisão} = \frac{|\{\text{atividades relevantes}\} \bigcap \{\text{atividades recuperadas}\}|}{|\{\text{atividades recuperadas}\}|} \tag{4.7}$$

Por outro lado, a sensibilidade, do inglês *recall*, tem como objetivo mensurar a completude e a especificidade dos processos de negócio recuperados e pode ser definida como o número de atividades relevantes recuperadas dividido pelo número total de atividades relevantes, conforme apresentado na fórmula 4.8 a seguir.

$$\text{Sensibilidade} = \frac{|\{\text{atividades relevantes}\} \bigcap \{\text{atividades recuperadas}\}|}{|\{\text{atividades relevantes}\}|} \tag{4.8}$$

Uma atividade é considerada relevante se representa fielmente uma operação de negócio da organização, isto é, uma atividade que está presente no processo de negócio tomado como

referência. Logo, a avaliação dessas métricas consiste em medir a diferença entre os resultados obtidos e resultados desejados, e permite não apenas avaliar os processos de negócio obtidos, mas também avaliar a eficácia da técnica empregada.

Ambas as métricas apresentadas possuem valores no intervalo [0,1] e a tendência é que o valor da sensibilidade seja maior que o valor da precisão. Porém, além dessas métricas, torna-se necessário definir valores de referência para determinar o quão bom ou ruim é um resultado obtido e se esse valor é ou não aceitável. Em vista disso, (PEREZ-CASTILLO, 2011) desenvolveu uma pesquisa para definir limites de referência para a interpretação dos valores de precisão e sensibilidade. Esses limiares foram obtidos a partir da aplicação do Método de Bender (BENDER, 1999) e associados a um rótulo, usado para classificar amostras de precisão e sensibilidade em muito baixa, baixa, média, alta e muito alta, conforme apresentado na Tabela 4.2 a seguir.

Tabela 4.2: Intervalos de referência para métricas de Precisão e Sensibilidade.

Rótulo	Precisão	Sensibilidade
Muito Baixa	[0.00, 0.47]	[0.00, 0.70]
Baixa	]0.47, 0.56]	]0.70, 0.76]
Média	]0.56, 0.63]	]0.76, 0.81]
Alta	]0.63, 0.72]	]0.81, 0.88]
Muito Alta	]0.72, 1.00]	]0.88, 1.00]

Além dessas duas métricas, existe ainda uma terceira métrica conhecida como **F-*measure***. Essa métrica combina as métricas de Precisão e Sensibilidade em uma única métrica e consiste na média harmônica entre ambas, conforme apresentado na fórmula 4.9. Essa métrica é frequentemente utilizada devido a dificuldade em se avaliar as métricas anteriores (Precisão e Sensibilidade) dada a relação inversa entre elas.

$$\text{F-}measure = \frac{(1+\alpha)\cdot \text{Precisão}\cdot \text{Sensibilidade}}{\alpha\cdot \text{Precisão} + \text{Sensibilidade}} \tag{4.9}$$

O cálculo de F-*measure* permite determinar o grau de importância de precisão e sensibilidade por meio do valor de α. Geralmente na literatura, assim como neste trabalho, adota-se o valor 1 para α, que implica na igualdade de importância para precisão e sensibilidade no cálculo de F-*measure*.

Quanto maior o valor de F-*measure*, maior é a eficácia. Como F-*measure* depende da combinação da precisão e da sensibilidade, então, para se obter um valor alto de F-*measure*, é

necessário um valor alto tanto da precisão quanto da sensibilidade. A Tabela 4.3 sumariza os intervalos de F-*measure* obtidos da combinação dos pares de rótulos de precisão e sensibilidade. Cada célula da tabela representa os valores mínimo e máximo de F-*measure* que podem ser obtidos a partir dos respectivos valores associados aos rótulos de precisão e sensibilidade.

Tabela 4.3: Sumarização dos intervalos de F-*measure*.

		Sensibilidade				
		Muito Baixa	Baixa	Média	Alta	Muito Alta
Precisão	Muito Baixa	[0.00, 0.56]	[0.00, 0.58]	[0.00, 0.59]	[0.00, 0.61]	[0.00, 0.64]
	Baixa	[0.00, 0.62]	[0.56, 0.64]	[0.58, 0.66]	[0.59, 0.68]	[0.61, 0.72]
	Média	[0.00, 0.66]	[0.62, 0.69]	[0.64, 0.71]	[0.66, 0.73]	[0.68, 0.77]
	Alta	[0.00, 0.71]	[0.66, 0.74]	[0.69, 0.76]	[0.71, 0.79]	[0.73, 0.84]
	Muito Alta	[0.00, 0.82]	[0.71, 0.84]	[0.74, 0.90]	[0.76, 0.94]	[0.79, 1.00]

De acordo com (PEREZ-CASTILLO, 2011), uma técnica de recuperação de processos de negócio pode ser considerada como aceitável se (I) precisão for Média (maior do que 0.56) e sensibilidade for Média (maior do que 0.81); ou se (II) sensibilidade for Baixa (maior do que 0.70) e precisão for Alta (maior do que 0.63). Porém, para uma técnica ser considerada eficaz, é necessário que sensibilidade seja pelo menos Alta e precisão Muito Alta. Isso significa que os valores de precisão e sensibilidade devem ser acima de 0.72 e 0.81 respectivamente, ou F-*measure* acima de 0.75.

Para avaliar a eficácia da abordagem RPN desenvolvida nesta dissertação, foram calculadas as métricas de precisão, sensibilidade e F-*measure*. Para isto, foram comparados dois modelos: (I) modelo obtido pela abordagem RPN; e (II) modelo de referência, obtido manualmente por um especialista em Engenharia de Software. Os resultados obtidos são apresentados na Tabela 4.4 a seguir.

Tabela 4.4: Resultados obtidos na avaliação qualitativa.

Atividades	**Quantidade**
Recuperadas	257
Recuperadas e Relevantes	204
Recuperadas e Não-relevantes	53
Não-recuperadas e Relevantes	24
Métrica	**Valor**
Precisão	0.79
Sensibilidade	0.89
F-*measure*	0.84

Contabilizando o total de atividades recuperadas pela abordagem RPN e o total de atividades identificadas no modelo de referência, temos que a abordagem RPN recuperou um total de 257 atividades, dentre as quais 204 são consideradas relevantes e 53 não-relevantes. Além disso, a abordagem não foi capaz de recuperar 24 atividades presentes no modelo de referência e que são consideradas como relevantes.

Com base nos limiares apresentados anteriormente, podemos interpretar o resultados obtidos e concluir que a abordagem RPN é eficaz, visto que as métricas de precisão e sensibilidade obtidas, cujos valores são respectivamente 0.79 e 0.89, correspondem ambas à classificação Muito Alta.

A Tabela 4.5 sumariza os resultados de precisão e sensibilidade obtidos por outros trabalhos relacionados encontrados na literatura. Vale salientar que os resultados apresentados foram obtidos a partir de conjuntos de dados diferentes, isto é, os valores de precisão e sensibilidade correspondem aos valores obtidos pelos próprios autores em seus trabalhos. Isso se deve à dificuldade em se utilizar as técnicas existentes dado que a grande maioria não disponibiliza o código fonte ou um arquivo executável para o uso.

Tabela 4.5: Resultados de precisão e sensibilidade obtidos por trabalhos relacionados encontrados na literatura.

Trabalhos	**Precisão**		**Sensibilidade**		**F-*measure***
	Valor	Classificação	Valor	Classificação	
(LO; KHOO, 2006)	0.10	Muito Baixa	0.90	Muito Alta	0.18
(PHILIPPOW, 2005)	0.37	Muito Baixa	1.00	Muito Alta	0.54
(LOU, 2010)	0.50	Baixa	0.99	Muito Alta	0.66
(ASENCIO, 2002)	0.68	Alta	0.71	Baixa	0.69
(POVZNER, 2009)	0.65	Alta	0.85	Alta	0.73
(PÉREZ-CASTILLO, 2011)	0.70	Alta	0.77	Média	0.73
(YEH, 2005)	0.74	Muito Alta	0.81	Média	0.77
(VAKULENKO,)	0.76	Muito Alta	0.84	Alta	0.80
(ZOU, 2009)	0.99	Muito Alta	0.99	Muito Alta	0.99

Apesar dos valores de precisão e sensibilidade, apresentados na Tabela 4.5, terem sido obtidos a partir de conjuntos de dados diferentes, é possível comparar superficialmente os trabalhos entre si e com a abordagem RPN. Ressalta-se que os valores podem variar conforme o conjunto de dados, podendo ter um maior impacto quando a complexidade e o tamanho do conjunto de dados variam.

Capítulo 5

TRABALHOS RELACIONADOS

Diversos trabalhos de pesquisa têm sido propostos pela comunidade científica e estão relacionados com os temas abordados nesta pesquisa. Este capítulo contextualiza esses trabalhos, descrevendo brevemente as características daqueles que possuem maior relação com a abordagem RPN desenvolvida. A Seção 5.1 apresenta a metologia de revisão sistemática da literatura utilizada para identificar o estado da arte; a Seção 5.2 discute brevemente sobre os principais trabalhos relacionados identificados; e por fim, a Seção 5.3 compara esses trabalhos relacionados com a abordagem RPN.

5.1 Estado da Arte

A fim de caracterizar o Estado da Arte do tema de pesquisa abordado nesta dissertação, foi adotado o processo de Revisão Sistemática (RS) da Literatura, que consiste em uma investigação científica com o objetivo de apresentar uma avaliação criteriosa a respeito de um tema de pesquisa (KITCHENHAM, 2004, 2012). Este processo conta com um conjunto de passos bem definidos e planejados de acordo com um protocolo previamente estabelecido (BIOLCHINI, 2005).

Dentre os diferentes processos de condução de RS, adotou-se o processo proposto por (KITCHENHAM, 2004) composto por três etapas. A primeira etapa é o Planejamento, e consiste em definir os objetivos e o protocolo da RS; A segunda etapa é a Execução, na qual são identificados, selecionados e avaliados os estudos primários conforme critérios de inclusão e exclusão previamente estabelecidos no protocolo; e, finalmente, a terceira etapa é a Análise dos Resultados, que realiza uma síntese das informações coletadas dos estudos selecionados.

A partir dos trabalhos identificados, foi possível analisar as soluções propostas, bem como as limitações, e, com base nisso, definir uma solução que contemple contribuições significativas

para esta área de pesquisa.

Para apoiar o processo de RS adotado, foi utilizada a ferramenta StArt[1], uma ferramenta que suporta a maioria das etapas do processo de RS. Em seguida, o seguinte protocolo foi definido:

- **Objetivo:** Investigar o estado da arte relativo à recuperação de processos de negócio inerentes a sistemas de informação.
- **Questão Principal:** Quais são as técnicas existentes na literatura que oferecem apoio computacional para a recuperação de processos de negócio embutidos no código fonte de sistemas de informação?
- **Fonte de Busca:** SCOPUS, devido o fato de indexar artigos de várias bases de busca.
- **Palavras-chave:** Recuperação; Extração; Mineração; Arqueologia; Processo de Negócio; Conhecimento de Negócio; Código fonte; Programa; Sistema de Informação; Software; bem como suas derivações e respectivos termos em inglês.
- **String de Busca:** (Recupera* OR Recover* OR Extra* OR Min* OR Arqueol* OR Archeolog*) AND ("Processo de Negócio"OR "Processos de Negócio"OR "Business Process*"OR "Conhecimento de Negócio"OR "Business Knowledge") AND ("Código Fonte"OR "Source Code"OR "Program*"OR "Software"OR "Sistema de Informação"OR "Sistemas de Informação"OR "Information System*")
- **Critérios de Inclusão:** 1. Estudos primários que propõem uma técnica para a recuperação de processos de negócio; 2. Estudos primários que utilizam uma técnica de recuperação de processos de negócio existente; 3. Estudos secundários que sumarizam técnicas de recuperação de processos de negócio encontradas na literatura.
- **Critérios de Excusão:** 1. Estudos não relacionados à recuperação de processos de negócio; 2. Estudos primários sobre técnicas de recuperação de processos de negócio em contextos diferentes do de Software; 3. Estudos primários incompletos, sem avaliação ou duplicados; 4. Estudos não escritos em português ou inglês.

Como resultado da execução da String de Busca nas Fontes de Busca, um total de 60 artigos foram obtidos. Dentre eles, somente 36 foram selecionados como relevantes. Para expandir a cobertura da RS, também foi adotada a técnica de *Snowballing*, uma técnica utilizada em RS que consiste em identificar estudos adicionais percorrendo as referências de trabalhos selecionados

[1]StArt – Ferramenta de apoio ao processo de Revisão Sistemática. Disponível em `http://lapes.dc.ufscar.br/tools/start_tool`

previamente (WOHLIN, 2014). Esta técnica permitiu identificar trabalhos relevantes não identificados pela string de busca ou indexados em outras bases de dados. 3 artigos foram selecionados com a técnica de Snowballing, totalizando 39 artigos selecionados como relevantes. A seguir, alguns dos principais trabalhos selecionados são apresentados como trabalhos relacionados.

5.2 Principais Trabalhos Relacionados

Nesta Seção são apresentados os principais trabalhos que inspiraram e estão relacionados de alguma forma à abordagem RPN desenvolvida nesta dissertação.

5.2.1 *A Model-Based Approach for Extracting Business Rules out of Legacy Information Systems*

A Model-Based Approach for Extracting Business Rules out of Legacy Information Systems (COSENTINO, 2013) é uma abordagem denominada BREX para a extração de regras de negócio de sistemas legados. A abordagem BREX baseia-se nas técnicas do MDE e é capaz de extrair regras de negócio de diferentes linguagens de programação, como Java e COBOL, além de Banco de Dados Relacionais.

A Figura 5.1 apresenta as etapas que compõem a abordagem BREX. Elas são respectivamente, *Model Discovery*, *Business Term Identification*, *Business Rule Identification* e *Business Rule Representation*.

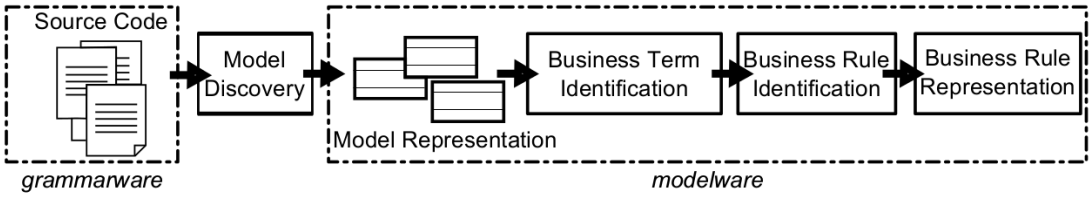

Figura 5.1: Visão geral da abordagem BREX (COSENTINO, 2013).

A etapa *Model Discovery* gera um modelo intermediário específico de plataforma a partir do código fonte. Para isto, a abordagem utiliza diversas ferramentas de descoberta, como o MoDisco para a linguagem Java, o IBM COBOL Application Model para COBOL e Xtext para Banco de Dados Relacionais. As etapas seguintes da abordagem manipulam o modelo gerado para realizar suas tarefas.

Na etapa *Business Term Identification*, os termos de negócio inerentes ao sistema são identificados por meio da análise das variáveis usadas no código fonte. Heurísticas implementadas

localizam as variáveis presentes em operações matemáticas, condicionais, declarações, entre outras.

Por sua vez, a etapa *Business Rule Identification* varre o modelo intermediário utilizando a técnica de análise estática a fim de se identificar o fluxo da aplicação e o uso dos termos (variáveis) identificados na etapa anterior.

Finalmente, as regras de negócio identificadas e seu encadeamento são descritos em artefatos gráficos na etapa *Business Rule Representation*.

Nota-se que a abordagem BREX destina-se exclusivamente à extração de regras de negócio. Além disso, ela não se preocupa em utilizar padrões para a representação dos modelos. Como isso, a abordagem limita-se quanto a utilização de um modelo intermediário próprio e uma notação gráfica pobre para representar as regras de negócio. Apesar dos objetivos serem diferentes, a abordagem RPN, comparada à abordagem BREX, busca utilizar padrões difundidos para representar os artefatos construídos.

5.2.2 *Model-Driven Business Process Recovery*

Model-Driven Business Process Recovery (ZOU, 2004) propõe uma *Framework* para a recuperação de processos de negócio que utiliza a técnica de rastreabilidade para identificar o fluxo da aplicação e, em seguida, aplica um conjunto de regras de heurística para selecionar apenas atividades relevantes à camada de negócio.

A rastreabilidade é realizada através da análise estática do código fonte, partindo do método *main* da aplicação e percorrendo recursivamente as chamadas de função. Em seguida, é gerado a AST do código fonte e analisados os elementos como nós, onde cada nó representa uma operação do código fonte. Regras de heurística definidas com base em experiência prévia são executadas e os elementos identificados são mapeados para elementos de workflow.

Por fim, os elementos identificados são representados em um grafo utilizando o *IBM WebSphere Business Integration Workbench*.

Embora esta técnica recupere processos de negócio do código fonte, as regras de heurísticas se basearam somente na experiência em desenvolvimento de aplicações e-*commerce* desenvolvidos na linguagem Java e dependem exclusivamente das tecnologias empregadas nessas aplicações. Logo, esta técnica limita-se ao escopo deste tipo de aplicações e à linguagem Java, enquanto a abordagem RPN pode ser aplicada a uma escopo mais amplo de aplicações independente da linguagem de programação.

5.2.3 *Workflow Mining: Discovering Process Models from Event Logs*

Workflow Mining: Discovering Process Models from Event Logs (AALST; WEIJTERS; MARUSTER, 2004) apresenta um algoritmo para extrair modelos de processos de logs de eventos e representá-los em redes de Petri. O algoritmo proposto, denominado α*-algorithm*, armazena em logs as informações das tarefas executadas e, em seguida, realiza a mineração desses dados.

A Figura 5.2 ilustra o funcionamento do algoritmo. À esquerda, observa-se uma tabela que ilustra os *logs* de evento capturados. Para cada elemento da tabela, são definidos um identificador para o caso (instância da execução) e a tarefa executada. O algoritmo analisa todas as sequências de tarefas executadas para construir o processo, isto é, todas as sequências possíveis dos elementos apresentados na tabela, podem se resumir nas seguintes sequências de execução de tarefas: {ABCD, ACBD, AED}. A partir disso, o algoritmo mapeia as combinações de sequências para o modelo de processos ilustrado à direita nesta figura.

case identifier	task identifier
case 1	task A
case 2	task A
case 3	task A
case 3	task B
case 1	task B
case 1	task C
case 2	task C
case 4	task A
case 2	task B
case 2	task D
case 5	task A
case 4	task C
case 1	task D
case 3	task C
case 3	task D
case 4	task B
case 5	task E
case 5	task D
case 4	task D

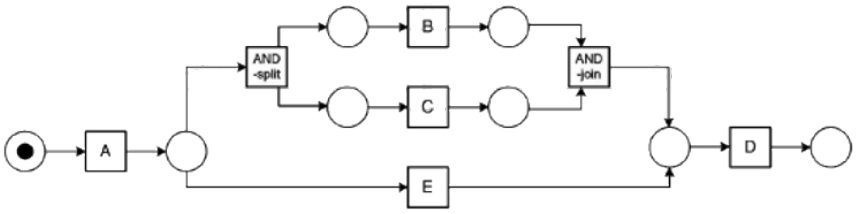

Figura 5.2: Funcionamento do algoritmo α (AALST; WEIJTERS; MARUSTER, 2004).

Adicionalmente, o algoritmo considera como tarefas relevantes para o processo, apenas as atividades com alto grau de incidência nos *logs* de eventos.

No entanto, apesar desta técnica ter sido formulada por meio de formalismo matemático, ela não apresenta, de fato, uma avaliação que demonstra o seu funcionamento. Além disso, esta técnica possui um série de requisitos e limitações quanto a identificação de laços e de paralelismo, uma vez que o número de sequências possíveis pode ser infinito. Em contrapartida, a abordagem RPN é capaz de identificar não apenas laços e paralelismo, mas também outros elementos explícitos no código fonte.

5.2.4 *Business Knowledge Extraction from Legacy Information Systems*

Business Knowledge Extraction from Legacy Information Systems (PARADAUSKAS; LAURIKAITIS, 2006) apresenta uma técnica para a recuperação do conhecimento de negócio implícito em bancos de dados relacionais. Esta técnica combina técnicas empregadas na Engenharia Reversa com Algoritmos de Mineração existentes (como os algoritmos de *Chiang* (CHIANG, 1995) e *Petit el al.* (PETIT, 1996)).

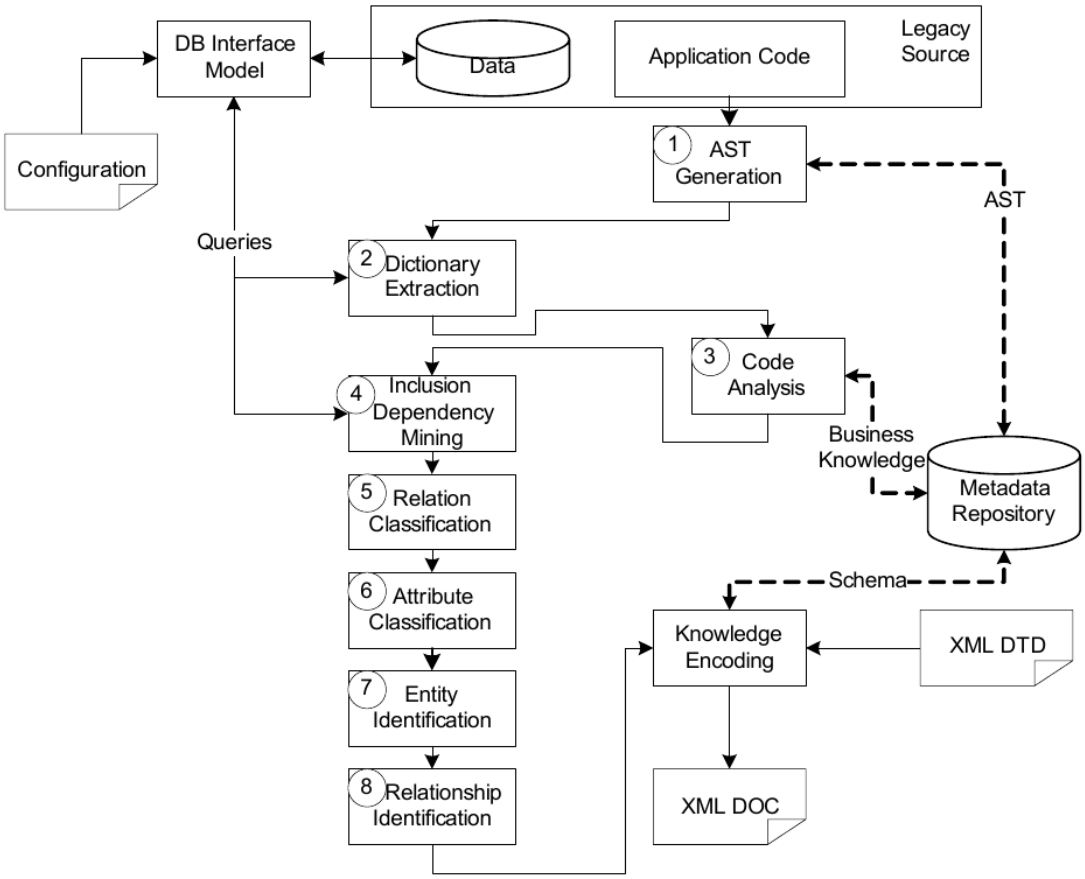

Figura 5.3: Visão geral da técnica (PARADAUSKAS; LAURIKAITIS, 2006).

Conforme ilustrado na Figura 5.3, esta técnica é composta por 8 passos. Inicialmente é gerada a AST do código da aplicação, isto é, AST do código SQL. Em seguida, são extraídos o dicionário de termos (nomes de atributos) e os relações. Uma análise do código é então realizada para aumentar o número de entidades extraídas na etapa anterior e identificar regras de negócio e restrições implícitas no banco de dados. Na sequência, é feita a mineração de dependências de inclusão, que consiste em identificar restrições inter-relacionais em relacionamentos entre classes/subclasses. Posteriormente, as relações do banco de dados são classificados, seguido pela classificação dos atributos. Logo após, são identificadas as entidades e, por fim, o seu relacionamento. Como resultado, o conhecimento extraído do decorrer dos 8 passos são representados por diagramas de entidade-relacionamento.

Embora esta técnica se assemelhe com a abordagem RPN, elas possuem objetivos diferentes. Enquanto a técnica proposta por Paradauskas visa extrair o conhecimento de bancos de dados relacionais e representá-los apenas em diagramas de entidade-relacionamento, a abordagem RPN destina-se a recuperação do conhecimento de negócio implícitos no código fonte de aplicações. Além disso, a técnica apresentada nesta seção foi ilustrada com pequenos exemplos, não sendo submetida a um processo de validação.

5.2.5 *MARBLE. Modernization Approach for Recovering Business Process from Legacy Information Systems*

MARBLE. Modernization Approach for Recovering Business Process from Legacy Information Systems (PEREZ-CASTILLO, 2012) é um *Framework* para a recuperação de processos de negócio baseado na Modernização Orientada pela Arquitetura (do inglês, *Architecture-Driven Modernization (ADM)*) que usa o KDM para representar informações recuperadas de uma forma comum e padronizada.

Em geral, o *Framework* MARBLE parte do código fonte e realiza uma sequência de três transformações de modelos até obter os modelo que representam processos de negócio, conforme ilustrado na Figura 5.4. A primeira transformação aplica técnicas de Engenharia Reversa à diferentes artefatos de software. Em seguida, o modelo específico de plataforma (PSM) obtido

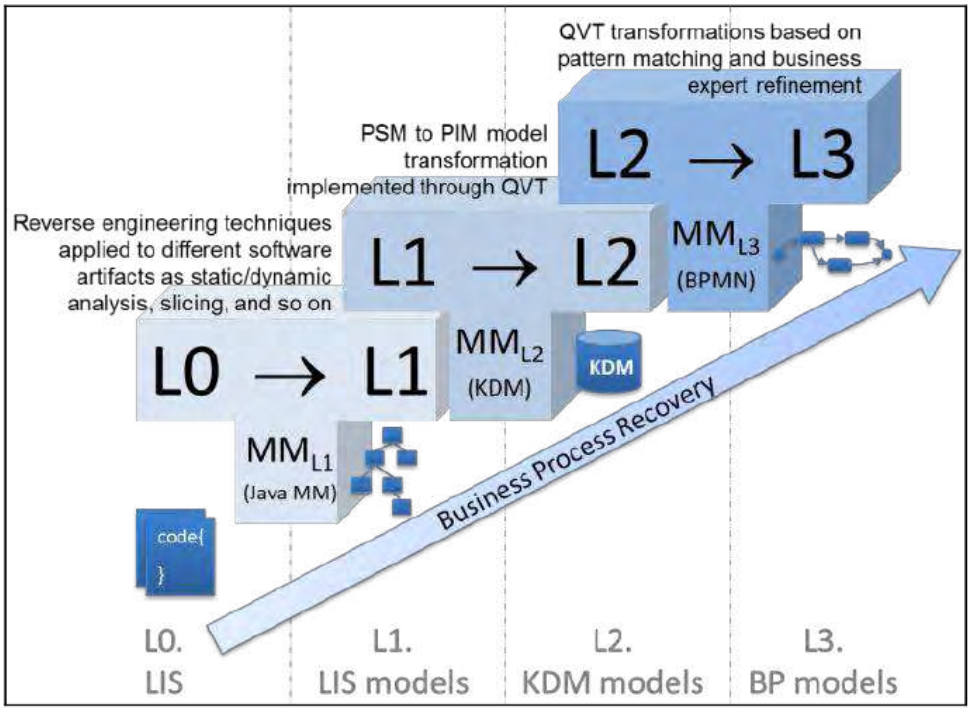

Figura 5.4: Visão geral do *Framework* MARBLE (PEREZ-CASTILLO, 2012).

pela primeira transformação é transformado em um modelo independente de plataforma (PIM) através de transformações QVT. Por fim, o modelo PIM é aplicado a outra transformação QVT baseada na correspondência de padrões.

MARBLE oferece duas soluções, uma Estática e uma Dinâmica. A primeira baseia-se na análise estática do código fonte para obter o modelo PSM que representa o código fonte. Deste modelo, são realizadas duas transformações QVT (explicadas anteriormente) e obtêm o modelo de processos de negócio desejado. Por outro lado, a segunda solução baseia-se na análise dinâmica do código fonte, no qual são capturados logs de eventos da execução do sistema de informação. Em seguida, são aplicados algoritmos de mineração de processos em conjunto com as transformações QVT e os modelos BPMN são obtidos. A Figura 5.5 ilustra ambas as soluções presentes no *Framework* MARBLE.

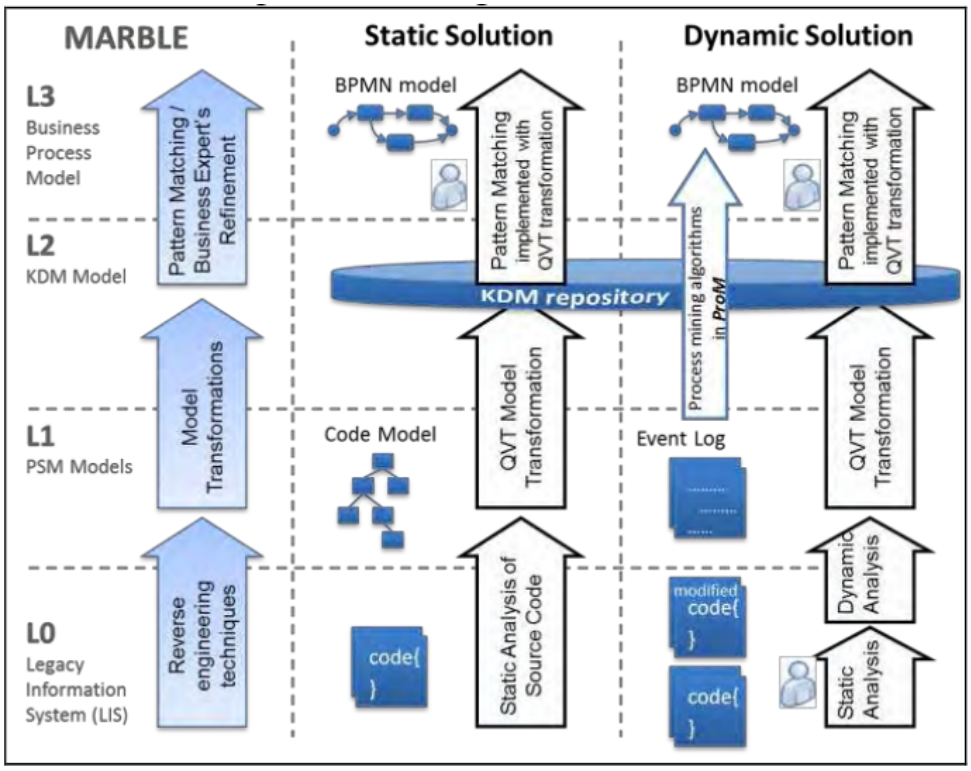

Figura 5.5: Soluções Estática e Dinâmica presente no *Framework* MARBLE (PEREZ-CASTILLO, 2012).

De todos os trabalhos relacionados encontrados na literatura, o MARBLE é o mais difundido, sendo a referência no domínio de recuperação de processos de negócio devido a quantidade de trabalhos publicados (cerca de 40). Comparado com a abordagem RPN, o MARBLE oferece mais recursos, embora a abordagem RPN tenha obtido resultados de eficácia superior ao MARBLE.

5.3 Análise Comparativa

Em geral, os trabalhos relacionados visam objetivos distintos e, portanto, diferem em alguns aspectos. A seguir, na Tabela 5.1, os trabalhos são sumarizados e comparados com a abordagem RPN segundo algumas características.

Tabela 5.1: Análise comparativa entre os trabalhos relacionados e a abordagem RPN.

	Características							
Trabalhos	**Recupera Processos de Negócio**	**Análise Estática**	**Análise Dinâmica**	**Utiliza Padrões**	**Independente de Plataforma**	**Utiliza Código Fonte**	**Utiliza Outros Artefatos**	**Validação da Técnica**
(COSENTINO, 2013)	✗	✓	✗	✗	✓	✓	✓	✓
(ZOU, 2004)	✓	✓	✗	✗	✗	✓	✗	✓
(AALST; WEIJTERS; MARUSTER, 2004)	✓	✗	✗	✗	✓	✗	✓	✗
(PARADAUSKAS; LAURIKAITIS, 2006)	✓	✓	✗	✗	✗	✓	✓	✗
(PEREZ-CASTILLO, 2012)	✓	✓	✓	✓	✓	✓	✓	✓
Abordagem RPN	✓	✓	✗	✓	✓	✓	✗	✓

Observa-se que a abordagem RPN não atende apenas às características de análise dinâmica e uso de outros artefatos de Software, uma vez que os objetivos da abordagem é oferecer um meio de recuperar processos de negócio de sistemas de informação sem restrições, geralmente presentes em soluções dinâmicas.

Dentre as técnicas apresentadas na seção anterior, somente a técnica proposta por (COSENTINO, 2013) não objetiva a recuperação de processos de negócio, tendo como objetivo recuperar regras de negócio. Com relação a técnica de estática ou dinâmica de análise de artefatos, a maioria realiza estaticamente, sendo que apenas MARBLE (PEREZ-CASTILLO, 2012) oferece ambas as soluções.

Além disso, é possível observar que várias técnicas não se preocupam em generalizar a solução para múltiplas plataformas e, tampouco, em utilizar padrões. A maioria dos trabalhos optaram por definir seus próprios modelos e notações, agravando as limitações de uso das técnicas.

Com relação aos artefatos analisados, varia-se conforme a técnica empregada. Em geral, soluções dinâmicas requerem o uso de artefatos adicionais ao código fonte, enquanto soluções estática analisam geralmente código fonte, seja da aplicação ou do banco de dados.

Por fim, apesar de bem definidas, duas dessas técnicas não apresentam uma validação concreta aplicada a sistemas reais com dados quantificados sobre a eficiência e/ou eficácia da técnica. Assim, têm-se uma incógnita com relação ao seu real funcionamento.

Pode-se dizer, que a abordagem RPN se sobressai às demais, com exceção à MARBLE, que, além da semelhança da solução estática, também oferece uma solução dinâmica.

Capítulo 6

CONCLUSÃO

Neste capítulo discorre-se as conclusões da pesquisa desta dissertação de mestrado. Na Seção 6.1 são pontuadas as conclusões, contribuições e publicações resultantes desta dissertação; na Seção 6.2 são apresentadas as respostas das questões de pesquisa definidas no início desta dissertação; na Seção 6.3 são apontadas as principais limitações referentes ao trabalho desenvolvido; e na Seção 6.4 são apresentadas algumas possibilidades de trabalhos futuros.

6.1 Conclusões

Um dos principais desafios da Engenharia de Software é como minimizar os efeitos do tempo em Software. Esses efeitos fazem com que os softwares tornem-se obsoletos e defasados em suas tecnologias, além da tendência de torná-los incoerentes com suas documentações. Nesse sentido, uma alternativa amplamente recorrida é a Reengenharia de Software, que consiste na atualização ou substituição do software. No entanto, é altamente importante preservar o conhecimento de negócio incorporado a esses softwares, o que torna esta tarefa complexa, uma vez que, na grande maioria dos casos, há incoerências entre o software e a documentação ou casos em que a documentação não é totalmente confiável.

Por este motivo, focou-se na investigação de técnicas que apoiem a extração do conhecimento de negócio inerente ao código fonte. Para isso, foi proposta uma abordagem de recuperação de processos de negócio baseada na análise estática do código fonte de sistemas de informação. Optou-se pela análise estática ao invés da análise dinâmica, pois a análise estática não requer a execução do software e nenhum tipo de instrumentação do código fonte.

Com o objetivo de criar uma abordagem de propósito geral, isto é, uma abordagem genérica com relação à linguagem de programação alvo, optou-se por utilizar uma representação inter-

mediária independente de plataforma. Para isso, optou-se pelo uso do KDM PIM, uma vez que é um metamodelo padronizado pela OMG e existem ferramentas disponibilizadas que auxiliam na conversão do código fonte para o KDM PIM.

Inicialmente a abordagem transforma o código fonte no modelo KDM PIM e posteriormente é submetido a um conjunto de regras de heurística pré-definido com o objetivo de recuperar os elementos de processos de negócio que compõem o conhecimento de negócio. Esse conhecimento é então representado por modelos gráficos especificados em BPMN.

Para avaliar a aplicabilidade da abordagem e quantificar os resultados obtidos por meio da utilização da abordagem proposta, dois estudos de casos foram conduzidos, ambos aplicados ao sistema ProGradWeb da UFSCar. O primeiro estudo de caso consistiu em estimar o tempo e o esforço possivelmente poupados com a utilização da abordagem proposta. O segundo estudo de caso consistiu em avaliar a eficácia da abordagem proposta, ou seja, o quão preciso e específico foi a recuperação dos modelos BPMN.

Em suma, as contribuições desta dissertação de mestrado são:

- Abordagem de propósito geral (independente de linguagem e plataforma) eficaz para a recuperação de processos de negócio que pode ser aplicável a qualquer sistema de informação sem a necessidade de adaptações (como por exemplo, a instrumentação do código fonte);
- Uso de especificações padrões da OMG, como o KDM para a representação do código fonte e o BPMN para a representação dos processos de negócio; e
- Conjunto de Regras de Heurística para a Recuperação de Elementos de Processos de Negócio em Modelos KDM.

Além disso, esta dissertação de mestrado resultou na seguinte publicação abaixo. Além disso, obteve-se o aceite para publicação como *SHORT PAPER* nas conferências ICEIS 2015[1] e EDOC 2015[2]. Em ambos os casos preferiu-se não publicá-los como *short paper*, sendo este re-submetido para o AICCSA 2015[3].

- Luiz Alexandre Pacini Rabelo, Antonio Francisco do Prado, Wanderley Lopes de Souza, Luís Ferreira Pires. *"An Approach to Business Process Recovery from Source Code"*.

[1]ICEIS 2015 – `http://www.iceis.org/?y=2015`
[2]EDOC 2015 – `https://edoc2015.unisa.edu.au`
[3]AICCSA 2015 – `http://aiccsa.net/website/home`

In: **2015 12th International Conference on Information Technology-New Generations (ITNG)**. IEEE, 2015. p. 361-366. Las Vegas, NV. DOI 10.1109/ITNG.2015.64.[4] (PACINI RABELO, 2015).

6.2 Respostas às Questões de Pesquisa

As questões de pesquisa que foram definidas no início desta pesquisa foram respondidas ao longo desta dissertação. A seguir, são sintetizadas as respostas às questões de pesquisa.

QP_1: Como realizar a recuperação de processos de negócio?

- **$QP_{1.1}$:** Quais são as soluções existentes?
- **$QP_{1.2}$:** Quais são os artefatos de software necessários?
- **$QP_{1.3}$:** Como extrair o conhecimento de negócio de artefatos de software?
- **$QP_{1.4}$:** Como representar os processos de negócio recuperados?

No capítulo 5 foi realizada uma Revisão Sistemática da Literatura com a finalidade de se estudar as técnicas existentes de recuperação de processos de negócio, além de outros trabalhos que inspiraram soluções alternativas. A partir da análise desses trabalhos, optou-se por adotar uma solução estática (detalhado na Seção 2.2), devido esta técnica analisar o código fonte (artefato mais confiável de Software) e não requerer modificações e/ou sua execução. O conhecimento extraído através da análise estática do código fonte foi então representado em BPMN, uma notação padrão para a modelagem de processos de negócio (apresentada na Seção 2.1).

QP_2: Como validar a abordagem proposta?

- **$QP_{2.1}$:** Quais são as métricas avaliadas?
- **$QP_{2.2}$:** Quais são os resultados esperados?

Para validar a abordagem, foi novamente recorrido aos trabalhos identificados na Revisão Sistemática apresentada no Capítulo 5, no qual estudou-se os tipos de avaliações realizadas, assim como as métricas. Por fim, optou-se por realizar uma avaliação qualitativa (Seção 4.3), mensurando-se a eficácia da abordagem, e uma avaliação quantitativa (Seção 4.2), estimando-se a eficiência em utilizar esta abordagem em comparação ao processo manual.

[4]Disponível em: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=7113499`

6.3 Limitações

Uma vez tendo concluído o desenvolvimento do trabalho proposto, algumas limitações foram identificadas por meio de uma análise crítica.

Embora a técnica de análise estática empregada pela abordagem proposta tenha apresentado alta eficácia no contexto deste trabalho, essa técnica não permite identificar elementos comportamentais. Esses elementos comportamentais são identificáveis somente em tempo de execução, isto é, são possíveis de se identificar somente ao executar os sistemas de informação e analisar seu comportamento durante o uso. Além disso, essa técnica não permite a identificação dos atores do sistema, o que implica em não ser possível identificar a relação entre as atividades do processo de negócio e suas respectivas *swimlanes* nos modelos BPMN.

Outra limitação encontrada na abordagem refere-se à baixa precisão na recuperação de elementos de processos de negócio que não estão explicitamente representados em níveis de funções no código fonte, ou seja, atividades mais abstratas do processo que estão presentes em diversos níveis de funções.

Além disso, embora os resultados obtidos do estudo tenham evidenciado ganhos no tempo e no esforço despendido na realização da recuperação de processos de negócio além de sua eficácia utilizando-se a abordagem proposta, é necessário considerar que esses resultados estão limitados ao escopo de ambientes acadêmicos e ao sistema avaliado (ProGradWeb). Considerando-se questões de validade, para generalizar e estender os resultados obtidos para um contexto mais amplo, torna-se necessária a replicação do estudo de caso em ambientes industriais com sistemas de informação maiores e, de preferência, desenvolvidos em linguagens de programação diferentes.

Os resultados de eficácia obtidos foram comparados com outras abordagens existentes na literatura. No entanto, os resultados de cada uma delas, refere-se à avaliação feita pelos próprios autores da abordagem. Portanto, foram comparadas as abordagem utilizando conjuntos de dados diferentes. Uma avaliação mais precisa e justa, que não foi considerada neste trabalho, refere-se à realização de um estudo comparativo com as outras abordagens de recuperação de processos de negócio utilizando-se o mesmo conjunto de dados.

6.4 Trabalhos Futuros

Embora diversas questões tenham sido exploradas durante o desenvolvimento deste trabalho, existem diversas possibilidades de melhorias relacionadas a este estudo. Dentre os tra-

balhos futuros que podem contribuir para a evolução desta pesquisa, destaca-se a avaliação da abordagem proposta para outros sistemas de informação desenvolvidos em outras linguagens de programação, a fim de se avaliar a eficácia, a portabilidade e a escalabilidade para esses casos.

Além disso, uma Linguagem de Domínio Específico, do inglês *Domain Specific Language (DSL)*, pode ser definida para descrever as regras de heurísticas da abordagem. Dessa forma, a abordagem poderá realizar a busca por elementos de processos de negócio com base nas regras definidas pela DSL, permitindo adicionar e/ou remover novas regras. Assim, a abordagem se tornará refinável e facilmente mantida.

Também pode-se adaptar a abordagem para flexibilizar o grau de granularidade dos modelos recuperados, a fim de se poder escolher qual o grau de granularidade do modelo a cada execução da abordagem. Assim, será possível obter o modelo com o grau de granularidade que mais se adéqua à cada caso.

E por fim, mas não menos importante, a abordagem pode ser adaptada para contemplar a análise dinâmica em conjunto com a análise estática. Dessa forma, ambas as análises poderão se complementar a fim de se obter modelos com maior eficácia.

REFERÊNCIAS

AALST, W. van der; WEIJTERS, T.; MARUSTER, L. Workflow mining: Discovering process models from event logs. *IEEE Trans. on Knowl. and Data Eng.*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 16, n. 9, p. 1128–1142, set. 2004. ISSN 1041-4347. Disponível em: <http://dx.doi.org/10.1109/TKDE.2004.47>.

ASENCIO, A. et al. Relating Expectations to Automatically Recovered Design Patterns. *2013 20th Working Conference on Reverse Engineering (WCRE)*, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 0087, 2002.

BARDAS, A. G. Static Code Analysis. *Journal of Information Systems & Operations Management*, Romanian-American University, v. 4, n. 2, p. 99–107, 2010.

BENDER, R. Quantitative risk assessment in epidemiological studies investigating threshold effects. *Biometrical Journal*, Berlin, Akademie-Verlag, 1977-, v. 41, n. 3, p. 305–319, 1999.

BINKLEY, D. Source Code Analysis: A Road Map. In: *2007 Future of Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007. (FOSE '07), p. 104–119. ISBN 0-7695-2829-5. Disponível em: <http://dx.doi.org/10.1109/FOSE.2007.27>.

BIOLCHINI, J. et al. Systematic review in software engineering. *System Engineering and Computer Science Department COPPE/UFRJ, Technical Report ES*, v. 679, n. 05, p. 45, 2005.

BRAGA, R. T. V. *Padrões de Software a partir da Engenharia Reversa de Sistemas Legados*. Tese (Doutorado) — Instituto de Ciências Matemáticas e de Computação, 1998.

BRUNELIERE, H. et al. Modisco: A model driven reverse engineering framework. *Information and Software Technology*, Elsevier, v. 56, n. 8, p. 1012–1032, 2014.

BRUNELIERE, H. et al. MoDisco: A Generic and Extensible Framework for Model Driven Reverse Engineering. In: ACM. *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>.

CHESS, B.; WEST, J. *Secure programming with static analysis*. First. Boston, MA, USA: Pearson Education, 2007. ISBN 9780321424778.

CHIANG, R. H. A knowledge-based system for performing reverse engineering of relational databases. *Decision Support Systems*, Elsevier, v. 13, n. 3, p. 295–312, 1995.

CHIKOFSKY, E. J.; CROSS, J. H. et al. Reverse engineering and design recovery: A taxonomy. *Software, IEEE*, IEEE, v. 7, n. 1, p. 13–17, 1990.

COSENTINO, V. *A model-based approach for extracting business rules out of legacy information systems*. Tese (Doutorado) — Ecole des Mines de Nantes, 2013.

CUADRADO, J. S.; ARACIL, J. P. Scheduling model-to-model transformations with continuations. *Software: Practice and Experience*, Wiley Online Library, v. 44, n. 11, p. 1351–1378, 2014.

DAVENPORT, T. H. *Process Innovation: Reengineering Work Through Information Technology*. Boston, MA, USA: Harvard Business School Press, 1993. ISBN 0-87584-366-2.

FAVRE, J.-M. Foundations of model (driven) (reverse) engineering: Models. In: BEZIVIN, J.; HECKEL, R. (Ed.). *Language Engineering for Model-Driven Software Development*. Dagstuhl, Germany: Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005. (Dagstuhl Seminar Proceedings, 04101). ISSN 1862-4405. Disponível em: <http://drops.dagstuhl.de/opus/volltexte/2005/13>.

FAYYAD, U.; SHAPIRO, P. G.; SMYTH, P. Data mining and knowledge discovery in databases: an overview. *Communications of the ACM*, v. 39, n. 11, 1996.

FAYYAD, U. M. et al. Advances in knowledge discovery and data mining. The MIT Press, 1996.

FAYYAD, U. M. et al. Knowledge Discovery and Data Mining: Towards a Unifying Framework. In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*. Portland, Oregon: AAAI Press, 1996. p. 82–88.

FELDENS, M. A. et al. Towards a methodology for the discovery of useful knowledge combining data mining, data warehousing and visualization. *CNPq/Protem-cc Fase III (SIDI Project), Universidade Federal do Rio Grande do Sul, Brazil*, 1998.

FELTRIM, V. D. *Apoio à documentação de engenharia reversa de software por meio de hipertextos*. Tese (Doutorado) — Instituto de Ciências Matemáticas e de Computação, 1999.

GOEBEL, M.; GRUENWALD, L. A survey of data mining and knowledge discovery software tools. *ACM SIGKDD Explorations Newsletter*, ACM, v. 1, n. 1, p. 20–33, 1999.

HALL, P. A. V. *Software Reuse and Reverse Engineering in Practice*. London, UK, UK: Chapman & Hall, Ltd., 1992. ISBN 0442314094.

HAMMER, M.; CHAMPY, J. *Reengineering the Corporation: A Manifesto for Business Revolution*. New York, NY: HarperCollins, 2009.

ISO/IEC. *ISO/IEC 19506:2012. Information Technology – Object Management Group Architecture-Driven Modernization (ADM) – Knowledge Discovery Meta-Model (KDM)*. ISO/IEC, 2012.

JESTON, J.; NELIS, J. *Business Process Management: Practical Guidelines to Successful Implementations*. 2 ed. Oxford: Butterworth-Heinemann (Elsevier Ltd.), 2008. 469 p. ISBN 978-0750686563.

JORDAN, D. et al. Web services business process execution language version 2.0. *OASIS standard*, v. 11, n. 120, p. 5, 2007.

KHUSIDMAN, V.; ULRICH, W. Architecture-driven modernization: Transforming the enterprise. *DRAFT V.5, OMG*, 2007.

KITCHENHAM, B. Procedures for performing systematic reviews. *Keele University and NICTA*, v. 33, n. 2004, p. 1–26, 2004.

KITCHENHAM, B. A. Systematic review in software engineering: Where we are and where we should be going. In: *Proceedings of the 2Nd International Workshop on Evidential Assessment of Software Technologies*. New York, NY, USA: ACM, 2012. (EAST '12), p. 1–2. ISBN 978-1-4503-1509-8. Disponível em: <http://doi.acm.org/10.1145/2372233.2372235>.

KLEIDERMACHER, D.; KLEIDERMACHER, M. *Embedded systems security: practical methods for safe and secure software and systems development*. 1 ed. Kidlington, Oxford: Elsevier, 2012. ISBN 978-0-12-386886-2.

KO, R. K. A computer scientist's introductory guide to business process management (BPM). *Crossroads*, ACM, v. 15, n. 4, p. 4, 2009.

KO, R. K.; LEE, S. S.; Wah Lee, E. Business process management (BPM) standards: a survey. *Business Process Management Journal*, Emerald Group Publishing Limited, v. 15, n. 5, p. 744–791, 2009.

LO, D.; KHOO, S.-C. QUARK: Empirical assessment of automaton-based specification miners. In: IEEE. *Reverse Engineering, 2006. WCRE'06. 13th Working Conference on*. Benevento, 2006. p. 51–60. ISSN 1095-1350.

LOU, J.-G. et al. Mining Program Workflow from Interleaved Traces. In: ACM. *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2010. (KDD '10), p. 613–622. ISBN 978-1-4503-0055-1. Disponível em: <http://doi.acm.org/10.1145/1835804.1835883>.

MAGALHÃES, L. P. A. *Estudo sobre engenharia reversa e avaliação da usabilidade de ferramentas case para engenharia reversa de software*. Tese (Doutorado) — Universidade Federal de Lavras, 2008.

MAZANEK, S.; HANUS, M. Constructing a bidirectional transformation between BPMN and BPEL with a functional logic programming language. *Journal of Visual Languages & Computing*, Elsevier, v. 22, n. 1, p. 66–89, 2011.

MOYER, B. Software archeology. *Modernizing old systems, Embedded Technology Journal*, v. 1, p. 1–4, 2009.

OMAN, P. W.; COOK, C. R. The book paradigm for improved maintenance. *Software, IEEE*, IEEE, v. 7, n. 1, p. 39–45, 1990.

OMG. *Architecture-Driven Modernization (ADM): Knowledge Discovery Metamodel (KDM), v1.3*. OMG, 2011.

OMG. Business Process Model and Notation (BPMN) Version 2.0. *Disponível em: http://www.omg.org/spec/BPMN/2.0*, 2011.

OWEN, M.; RAJ, J. BPMN and business process management: Introduction to the new business process modeling standard. Citeseer, 2003.

PACINI RABELO, L. et al. An approach to business process recovery from source code. In: *Information Technology - New Generations (ITNG), 2015 12th International Conference on*. Las Vegas, Nevada, USA: IEEE, 2015. p. 361–366.

PARADAUSKAS, B.; LAURIKAITIS, A. Business knowledge extraction from legacy information systems. *Information Technology and Control*, v. 35, n. 3, p. 214–221, 2006.

PEREZ-CASTILLO, R. Marble: Modernization approach for recovering business processes from legacy information systems. In: *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. Trento: IEEE, 2012. p. 671–676. ISSN 1063-6773.

PÉREZ-CASTILLO, R.; De Guzman, I. G.-R.; PIATTINI, M. Knowledge Discovery Metamodel-ISO/IEC 19506: A standard to modernize legacy systems. *Computer Standards & Interfaces*, Elsevier, v. 33, n. 6, p. 519–532, 2011.

PÉREZ-CASTILLO, R. et al. A Case Study on Business Process Recovery Using an e-Government System. *Softw. Pract. Exper.*, John Wiley & Sons, Inc., New York, NY, USA, v. 42, n. 2, p. 159–189, fev. 2012. ISSN 0038-0644. Disponível em: <http://dx.doi.org/10.1002/spe.1057>.

PÉREZ-CASTILLO, R.; GUZMÁN, I. G. R. de; PIATTINI, M. On the Use of Patterns to Recover Business Processes. In: *Proceedings of the 2010 ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2010. (SAC '10), p. 165–166. ISBN 978-1-60558-639-7. Disponível em: <http://doi.acm.org/10.1145/1774088.1774121>.

PÉREZ-CASTILLO, R.; GUZMÁN, I. G.-R. de; PIATTINI, M. Business process archeology using MARBLE. *Information and Software Technology*, Elsevier, v. 53, n. 10, p. 1023–1044, 2011.

PÉREZ-CASTILLO, R. et al. An Empirical Comparison of Static and Dynamic Business Process Mining. In: *Proceedings of the 2011 ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2011. (SAC '11), p. 272–279. ISBN 978-1-4503-0113-8. Disponível em: <http://doi.acm.org/10.1145/1982185.1982249>.

PEREZ-CASTILLO, R. et al. Obtaining Thresholds for the Effectiveness of Business Process Mining. In: *Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement*. Washington, DC, USA: IEEE Computer Society, 2011. (ESEM '11), p. 453–462. ISBN 978-0-7695-4604-9. Disponível em: <http://dx.doi.org/10.1109/ESEM.2011.64>.

PETIT, J.-M. et al. Towards the reverse engineering of renormalized relational databases. In: IEEE. *Data Engineering, 1996. Proceedings of the Twelfth International Conference on*. New Orleans, LA, 1996. p. 218–227.

PHILIPPOW, I. et al. An approach for reverse engineering of design patterns. *Software & Systems Modeling*, Springer, v. 4, n. 1, p. 55–70, 2005.

POVZNER, A. et al. Autograph: automatically extracting workflow file signatures. *ACM SIGOPS Operating Systems Review*, ACM, v. 43, n. 1, p. 76–83, 2009.

PRESSMAN, R. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Higher Education, 2010. (McGraw-Hill series in computer science). ISBN 9780071267823. Disponível em: <http://books.google.com.br/books?id=aeijPwAACAAJ>.

ROMÃO, W. *Descoberta de conhecimento relevante em banco de dados sobre ciência e tecnologia*. Tese (Doutorado) — Universidade Federal de Santa Catarina, 2002.

RUMMLER, G. A.; BRACHE, A. P. *Improving performance: How to manage the white space on the organization chart*. 3 ed. Hoboken, NJ: John Wiley & Sons, 2012.

SASSI, R. J. *Uma arquitetura híbrida para descoberta de conhecimento em bases de dados: teoria dos rough sets e redes neurais artificiais mapas auto-organizáveis*. Tese (Doutorado) — Universidade de São Paulo, 2006.

SOMMERVILLE, I. *Software Engineering (6th Ed.)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001. ISBN 0-201-39815-X.

SVOBODA, J. Effectively Combining Static Code Analysis and Manual Code Reviews. Masarykova univerzita, 2014.

The JBoss jBPM Team. *jBPM Documentation v6.2*. JBoss jBPM, 2014.

VAKULENKO, S. Extraction of Process Models from Business Process Descriptions. Citeseer.

VASCONCELOS, A. P. V. de. *Uma abordagem de apoio à criação de arquiteturas de referência de domínio baseada na análise de Sistemas Legados*. Tese (Doutorado) — UNIVERSIDADE FEDERAL DO RIO DE JANEIRO, 2007.

VISAGGIO, G. Ageing of a Data-intensive Legacy System: Symptoms and Remedies. *Journal of Software Maintenance*, John Wiley & Sons, Inc., New York, NY, USA, v. 13, n. 5, p. 281–308, out. 2001. ISSN 1040-550X. Disponível em: <http://dl.acm.org/citation.cfm?id=565153.565154>.

WESKE, M. *Business Process Management: Concepts, Languages, Architectures*. 2 ed. Berlin: Springer-Verlag Berlin Heidelberg, 2012. 404 p. ISBN 978-3-642-28615-5.

WHITE, S. A. Introduction to BPMN. *IBM Cooperation*, v. 2, n. 0, p. 0, 2004.

WHITE, S. A. Process modeling notations and workflow patterns. *Workflow handbook*, v. 2004, p. 265–294, 2004.

WOHLIN, C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. New York, NY, USA: ACM, 2014. (EASE '14), p. 38:1–38:10. ISBN 978-1-4503-2476-2. Disponível em: <http://doi.acm.org/10.1145/2601248.2601268>.

YEH, D. et al. An Empirical Study of a Reverse Engineering Method for Aggregation Relationship Based on Operation Propagation. In: *Proceedings of the 29th Annual International Computer Software and Applications Conference - Volume 01*. Washington, DC, USA: IEEE Computer Society, 2005. (COMPSAC '05), p. 95–100. ISBN 0-7695-2413-3. Disponível em: <http://dx.doi.org/10.1109/COMPSAC.2005.44>.

ZOU, Y. et al. Recovering business processes from business applications. *Journal of Software Maintenance and Evolution: Research and Practice*, Wiley Online Library, v. 21, n. 5, p. 315–348, 2009.

ZOU, Y. et al. Model-Driven Business Process Recovery. In: *Proceedings of the 11th Working Conference on Reverse Engineering*. Washington, DC, USA: IEEE Computer Society, 2004. (WCRE '04), p. 224–233. ISBN 0-7695-2243-2. Disponível em: <http://dl.acm.org/citation.cfm?id=1038267.1039054>.

LISTA DE ABREVIATURAS E SIGLAS

ACS – *Arab Computerg Society* – Sociedade Árabe de Computação

ADM – *Architecture – Driven Modernization* – Modernização Orientada a Arquitetura

AICCSA – *ACS/IEEE International Conference on Computer Systems and Applications* – Conferência Internacional sobre Sistemas e Aplicações Computacionais da ACS/IEEE

API – *Application Programming Interface* – Interface de Programação de Aplicações

AST – *Abstract Syntax Tree* – Árvore Sintática Abstrata

ATL – *ATLAS Transformation Language* – Linguagem de Transformação ATLAS

BNF – *Backus – Naur Form* – Formalismo de Backus-Naur

BPMI – *Business Process Management Initiative*

BPMN – *Business Process Model and Notation* – Modelo e Notação de Processos de Negócio

BPM – *Business Process Management* – Gerenciamento de Processos de Negócio

BREX – *Business Rules Extraction* – Extração de Regras de Negócio

CAPES – Coordenação de Aperfeiçoamento de Pessoal de Nível Superior

DC – Departamento de Computação

DOI – *Digital Object Identifier* – Identificador de Objeto Digital

DSL – *Domain Specific Language* – Linguagem de Domínio Específico

EDOC – *Enterprise Computing Conference* – Conferência de Computação Empresarial

FACOM – Faculdade de Computação

ICEIS – *International Conference on Enterprise Information Systems* – Conferência Internacional sobre Sistemas de Informação Empresariais

ISO – *International Organization for Standardization* – Organização Internacional de Padronização

ITNG – *Information Technology – New Generations* – Tecnologias de Informação-Novas Gerações

JDBC – *Java Database Connectivity*

KDM – *Knowledge Discovery Metamodel* – Metamodelo de Descoberta de Conhecimento

KLOC – *Kilo Lines Of Code* – Milhares de Linhas de Código

LIS – *Legacy Information System* – Sistema de Informação Legado

LaBDES – Laboratório de Banco de Dados e Engenharia de Software

LaPES – Laboratório de Pesquisa em Engenharia de Software

MARBLE – *Modernization Approach for Recovering Business Process from Legacy Information Systems* – Abordagem de Modernização para a Recuperação de Processos de Negócio de Sistemas de Informação Legados

MDE – *Model – Driven Engineering* – Engenharia Orientada por Modelo

MOF – *Meta – Object Facility*

OMG – *Object Management Group*

PIM – *Platform Independent Model* – Modelo Independente de Plataforma

PPGCC – Programa de Pós-Graduação em Ciência da Computação

PSM – *Platform Specific Model* – Modelo Específico de Plataforma

QP – Questão de Pesquisa

QVT – *Query/View/Transformation* – Consulta/Visão/Transformação

RE – Regra de Exclusão

RI – Regra de Inclusão

ROI – *Return of Investment* – Retorno do Investimento

RPN – Recuperação de Processos de Negócio

RS – Revisão Sistemática

SLOC – *Source Lines Of Code* – Linhas de Código Fonte

SQL – *Structured Query Language* – Linguagem de Consulta Estruturada

UFMS – Universidade Federal do Mato Grosso do Sul

UFSCar – Universidade Federal de São Carlos

UML – *Unified Modeling Language* – Linguagem de Modelagem Unificada

WS-BPEL – *Web Services Business Process Execution Language* – Linguagem de Execução de Processos de Negócio para Web Services

XML – *eXtensible Markup Language* – Linguagem de Marcação Estensível

Anexo A

Classe Matricula

```
1  package br.ufscar.sin.prograd.negocio; // 1
2
3  public class Matricula { // 2
4
5    public Integer calcularPerfil(Integer ano, Integer semestre, Integer
         numeroDeTrancamentos, Integer numeroDeAfastamentos, Matricula matricula) {
         // 3
6      return (ano - matricula.getAnoIngresso()) * 2 + (semestre -
           matricula.getSemestreIngresso()) + matricula.getDelta() -
           numeroDeTrancamentos - numeroDeAfastamentos;
7    }
8
9    private Double somarNotaXCreditosCursado(Inscricao inscricao) { // 4
10     return (inscricao.getNota() *
           disciplina.somarCreditos(inscricao.getTurma().getDisciplina()));
11   }
12
13   private Integer somarCreditosInscritos(Inscricao inscricao) { // 5
14     return disciplina.somarCreditos(inscricao.getTurma().getDisciplina());
15   }
16
17   private Integer somarCreditosCancelados(Inscricao inscricao) { // 6
18     return disciplina.somarCreditos(inscricao.getTurma().getDisciplina());
19   }
20
21   private Integer somarCreditosDesistentes(Inscricao inscricao) { // 7
22     return disciplina.somarCreditos(inscricao.getTurma().getDisciplina());
23   }
24
25   /**
26    * Calcula o Indice de Rendimento Academico (IRA) de estudantes com
27    * matriculas ativas em cursos presenciais.
28    */
29   public void calcularIra() throws DAOException, SQLException { // 8
30
31     List<Matricula> matriculas =
           MatriculaDAO.obterTodasMatriculasAtivasParaCursosPresenciais(); // 9
```

```
32
33      for (Matricula matricula : matriculas) { // 10
34
35        List<Inscricao> inscricoes = inscricaoDao.obterInscricoes(matricula); // 11
36
37        for (Inscricao inscricao : inscricoes) { // 12
38
39          if (Integer.parseInt(inscricao.getResultado()) < 6 &&
                inscricao.getStatus().matches("[3,6-9,17]")) { // 13
40            somatoriaNotasXCredCursados = somatoriaNotasXCredCursados +
                  this.somarNotaXCreditosCursado(inscricao); // 14
41          }
42
43          somatoriaCredInscritos = somatoriaCredInscritos +
                this.somarCreditosInscritos(inscricao); // 15
44
45          if (Integer.parseInt(inscricao.getResultado()) == 7) { // 16
46            somatoriaCredCancelados = somatoriaCredCancelados +
                  this.somarCreditosCancelados(inscricao); // 17
47          }
48
49          if (Integer.parseInt(inscricao.getResultado()) == 6) { // 18
50            somatorioCredDesistentes = somatorioCredDesistentes +
                  this.somarCreditosDesistentes(inscricao); // 19
51          }
52        }
53
54        Double ira = (somatoriaNotasXCredCursados / somatoriaCredInscritos) * (2 -
              ((2 * somatorioCredDesistentes + somatoriaCredCancelados) /
              somatoriaCredInscritos)); // 20
55
56        matricula.atualizarIra(ira); // 21
57      }
58    }
59  }
```

Código A.1: Classe Matricula em Java

Anexo B

Modelo Java PSM

```
<?xml version="1.0" encoding="ASCII"?>
<java:Model xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:java="http://www.eclipse.org/MoDisco/Java/0.2.incubation/java"
    name="ProGradWeb">

 <!-- 1 -->
 <ownedElements name="br">
  <ownedPackages name="ufscar">
   <ownedPackages name="sin">
    <ownedPackages name="prograd">
     <ownedPackages name="negocio">

      <!-- 2 -->
      <ownedElements xsi:type="java:ClassDeclaration"
          originalCompilationUnit="Matricula.java" name="Matricula"
          usagesInTypeAccess="Class Declaration Matricula">
       <modifier visibility="public"/>

       <!-- 3 -->
       <bodyDeclarations xsi:type="java:MethodDeclaration" name="calcularPerfil">
        <modifier visibility="public"/>
        <parameters name="ano">
         <type type="Integer"/>
        </parameters>
        ...
        <returnType type="Integer"/>
        <body originalCompilationUnit="Matricula.java">
         <statements xsi:type="java:ReturnStatement">
          ...
         </statements>
        </body>
       </bodyDeclarations>

       <!-- 4 -->
       <bodyDeclarations xsi:type="java:MethodDeclaration"
           name="somarNotaXCreditosCursado">...</bodyDeclarations>

       <!-- 5 -->
       <bodyDeclarations xsi:type="java:MethodDeclaration"
           name="somarCreditosInscritos">...</bodyDeclarations>

```

```
36        <!-- 6 -->
37        <bodyDeclarations xsi:type="java:MethodDeclaration"
              name="somarCreditosCancelados">...</bodyDeclarations>
38
39        <!-- 7 -->
40        <bodyDeclarations xsi:type="java:MethodDeclaration"
              name="somarCreditosDesistentes">...</bodyDeclarations>
41
42        <!-- 8 -->
43        <bodyDeclarations xsi:type="java:MethodDeclaration" name="calcularIra">
44         <modifier visibility="public"/>
45         <thrownExceptions type="DAOException"/>
46         <thrownExceptions type="SQLException"/>
47         <returnType type="void"/>
48         <body originalCompilationUnit="Matricula.java">
49
50          <!-- 9 -->
51          <statements xsi:type="java:VariableDeclarationStatement"
                originalCompilationUnit="Matricula.java">
52           <type type="java.util.List<Matricula>"/>
53           <fragments originalCompilationUnit="Matricula.java" name="matriculas"
                usageInVariableAccess="matriculas">
54            <initializer xsi:type="java:MethodInvocation"
                  originalCompilationUnit="Matricula.java"
                  method="obterTodasMatriculasAtivasParaCursosPresenciais"/>
55           </fragments>
56          </statements>
57
58          <!-- 10 -->
59          <statements xsi:type="java:EnhancedForStatement"
                originalCompilationUnit="Matricula.java">
60           <body xsi:type="java:Block" originalCompilationUnit="Matricula.java">
61
62            <!-- 11 -->
63            <statements xsi:type="java:VariableDeclarationStatement"
                  originalCompilationUnit="Matricula.java">
64             <type type="java.util.List<Inscricao>"/>
65             <fragments originalCompilationUnit="Matricula.java" name="inscricoes"
                    usageInVariableAccess="inscricoes">
66              <initializer xsi:type="java:MethodInvocation"
                    originalCompilationUnit="Matricula.java" method="obterInscricoes">
67               <arguments xsi:type="java:SingleVariableAccess" variable="matricula"/>
68               <expression xsi:type="java:SingleVariableAccess" variable="InscricaoDAO"/>
69              </initializer>
70             </fragments>
71             <modifier/>
72            </statements>
73
74            <!-- 12 -->
75            <statements xsi:type="java:EnhancedForStatement"
                  originalCompilationUnit="Matricula.java">
76             <body xsi:type="java:Block" originalCompilationUnit="Matricula.java">
77
78              <!-- 13 -->
79              <statements xsi:type="java:IfStatement"
                    originalCompilationUnit="Matricula.java">
80               <expression xsi:type="java:InfixExpression"
                      originalCompilationUnit="Matricula.java" operator="&amp;amp;&amp;amp;">
81                ...
82               </expression>
83               <thenStatement xsi:type="java:Block"
                      originalCompilationUnit="Matricula.java">
84
85                <!-- 14 -->
86                <statements xsi:type="java:ExpressionStatement"
                      originalCompilationUnit="Matricula.java">
```

```
87                  <expression xsi:type="java:Assignment"
                        originalCompilationUnit="Matricula.java">
88                   <leftHandSide xsi:type="java:SingleVariableAccess"
                         variable="somatoriaNotasXCredCursados"/>
89                   <rightHandSide xsi:type="java:InfixExpression"
                         originalCompilationUnit="Matricula.java" operator="+">
90                    <rightOperand xsi:type="java:MethodInvocation"
                          originalCompilationUnit="Matricula.java"
                          method="somarNotaXCreditosCursado">
91                     <arguments xsi:type="java:SingleVariableAccess"
                           variable="inscricao"/>
92                     <expression xsi:type="java:ThisExpression"
                           originalCompilationUnit="Matricula.java"/>
93                    </rightOperand>
94                    <leftOperand xsi:type="java:SingleVariableAccess"
                          variable="somatoriaNotasXCredCursados"/>
95                   </rightHandSide>
96                  </expression>
97                 </statements>
98                </thenStatement>
99               </statements>
100
101              <!-- 15 -->
102              <statements xsi:type="java:ExpressionStatement"
                     originalCompilationUnit="Matricula.java">
103               <expression xsi:type="java:Assignment"
                      originalCompilationUnit="Matricula.java">
104                <leftHandSide xsi:type="java:SingleVariableAccess"
                       variable="somatoriaCredInscritos"/>
105                <rightHandSide xsi:type="java:InfixExpression"
                       originalCompilationUnit="Matricula.java" operator="+">
106                 <rightOperand xsi:type="java:MethodInvocation"
                        originalCompilationUnit="Matricula.java"
                        method="somarCreditosInscritos">
107                  <arguments xsi:type="java:SingleVariableAccess" variable="inscricao"/>
108                  <expression xsi:type="java:ThisExpression"
                         originalCompilationUnit="Matricula.java"/>
109                 </rightOperand>
110                 <leftOperand xsi:type="java:SingleVariableAccess"
                        variable="somatoriaCredInscritos"/>
111                </rightHandSide>
112               </expression>
113              </statements>
114
115              <!-- 16 -->
116              <statements xsi:type="java:IfStatement"
                     originalCompilationUnit="Matricula.java">
117               <expression xsi:type="java:InfixExpression"
                      originalCompilationUnit="Matricula.java" operator="==">
118                ...
119               </expression>
120               <thenStatement xsi:type="java:Block"
                      originalCompilationUnit="Matricula.java">
121
122                <!-- 17 -->
123                <statements xsi:type="java:ExpressionStatement"
                       originalCompilationUnit="Matricula.java">
124                 <expression xsi:type="java:Assignment"
                        originalCompilationUnit="Matricula.java">
125                  <leftHandSide xsi:type="java:SingleVariableAccess"
                         variable="somatoriaCredCancelados"/>
126                  <rightHandSide xsi:type="java:InfixExpression"
                         originalCompilationUnit="Matricula.java" operator="+">
127                   <rightOperand xsi:type="java:MethodInvocation"
                          originalCompilationUnit="Matricula.java"
                          method="somarCreditosCancelados">
```

```
128                 <arguments xsi:type="java:SingleVariableAccess"
                        variable="inscricao"/>
129                 <expression xsi:type="java:ThisExpression"
                        originalCompilationUnit="Matricula.java"/>
130                </rightOperand>
131                <leftOperand xsi:type="java:SingleVariableAccess"
                       variable="somatoriaCredCancelados"/>
132               </rightHandSide>
133              </expression>
134             </statements>
135            </thenStatement>
136           </statements>
137
138           <!-- 18 -->
139           <statements xsi:type="java:IfStatement"
                  originalCompilationUnit="Matricula.java">
140            <expression xsi:type="java:InfixExpression"
                   originalCompilationUnit="Matricula.java" operator="==">
141             ...
142            </expression>
143            <thenStatement xsi:type="java:Block"
                   originalCompilationUnit="Matricula.java">
144
145             <!-- 19 -->
146             <statements xsi:type="java:ExpressionStatement"
                    originalCompilationUnit="Matricula.java">
147              <expression xsi:type="java:Assignment"
                     originalCompilationUnit="Matricula.java">
148               <leftHandSide xsi:type="java:SingleVariableAccess"
                      variable="somatorioCredDesistentes"/>
149               <rightHandSide xsi:type="java:InfixExpression"
                      originalCompilationUnit="Matricula.java" operator="+">
150                <rightOperand xsi:type="java:MethodInvocation"
                       originalCompilationUnit="Matricula.java"
                       method="somarCreditosDesistentes">
151                 <arguments xsi:type="java:SingleVariableAccess"
                        variable="inscricao"/>
152                 <expression xsi:type="java:ThisExpression"
                        originalCompilationUnit="Matricula.java"/>
153                </rightOperand>
154                <leftOperand xsi:type="java:SingleVariableAccess"
                       variable="somatorioCredDesistentes"/>
155               </rightHandSide>
156              </expression>
157             </statements>
158            </thenStatement>
159           </statements>
160          </body>
161         </statements>
162
163         <!-- 20 -->
164         <statements xsi:type="java:VariableDeclarationStatement"
                originalCompilationUnit="Matricula.java">
165          <type type="Double"/>
166          <fragments originalCompilationUnit="Matricula.java" name="ira"
                usageInVariableAccess="ira">
167           <initializer xsi:type="java:InfixExpression"
                  originalCompilationUnit="Matricula.java">
168            <rightOperand xsi:type="java:ParenthesizedExpression"
                   originalCompilationUnit="Matricula.java">
169             <expression xsi:type="java:InfixExpression"
                    originalCompilationUnit="Matricula.java" operator="-">
170              <rightOperand xsi:type="java:ParenthesizedExpression"
                     originalCompilationUnit="Matricula.java">
171               <expression xsi:type="java:InfixExpression"
                      originalCompilationUnit="Matricula.java" operator="/">
```

```
172                 <rightOperand xsi:type="java:SingleVariableAccess"
                        variable="somatoriaCredInscritos"/>
173                 <leftOperand xsi:type="java:ParenthesizedExpression"
                        originalCompilationUnit="Matricula.java">
174                  <expression xsi:type="java:InfixExpression"
                         originalCompilationUnit="Matricula.java" operator="+">
175                   <rightOperand xsi:type="java:SingleVariableAccess"
                          variable="somatoriaCredCancelados"/>
176                   <leftOperand xsi:type="java:InfixExpression"
                          originalCompilationUnit="Matricula.java">
177                    <rightOperand xsi:type="java:SingleVariableAccess"
                           variable="somatorioCredDesistentes"/>
178                    <leftOperand xsi:type="java:NumberLiteral"
                           originalCompilationUnit="Matricula.java" tokenValue="2"/>
179                   </leftOperand>
180                  </expression>
181                 </leftOperand>
182                </expression>
183               </rightOperand>
184               <leftOperand xsi:type="java:NumberLiteral"
                      originalCompilationUnit="Matricula.java" tokenValue="2"/>
185              </expression>
186             </rightOperand>
187             <leftOperand xsi:type="java:ParenthesizedExpression"
                    originalCompilationUnit="Matricula.java">
188              <expression xsi:type="java:InfixExpression"
                     originalCompilationUnit="Matricula.java" operator="/">
189               <rightOperand xsi:type="java:SingleVariableAccess"
                      variable="somatoriaCredInscritos"/>
190               <leftOperand xsi:type="java:SingleVariableAccess"
                      variable="somatoriaNotasXCredCursados"/>
191              </expression>
192             </leftOperand>
193            </initializer>
194           </fragments>
195           <modifier/>
196          </statements>
197
198          <!-- 21 -->
199          <statements xsi:type="java:ExpressionStatement"
                 originalCompilationUnit="Matricula.java">
200           <expression xsi:type="java:MethodInvocation"
                  originalCompilationUnit="Matricula.java" method="atualizarIra">
201            <arguments xsi:type="java:SingleVariableAccess" variable="ira"/>
202            <expression xsi:type="java:SingleVariableAccess" variable="matricula"/>
203           </expression>
204          </statements>
205         </body>
206        </statements>
207       </body>
208      </bodyDeclarations>
209     </ownedElements>
210   ...
211  </ownedElements>
212 </java:Model>
```

Código B.1: Modelo Java PSM gerado a partir da classe Matricula em Java

Anexo C

Modelo KDM PIM

```
<?xml version="1.0" encoding="ASCII"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:code="http://www.eclipse.org/MoDisco/kdm/code"
    xmlns:kdm="http://www.eclipse.org/MoDisco/kdm/kdm"
    xmlns:source="http://www.eclipse.org/MoDisco/kdm/source">
 <kdm:Segment>
  <model xsi:type="code:CodeModel" name="ProGradWeb">

   <!-- 1 -->
   <codeElement xsi:type="code:Package" name="br">
    <codeElement xsi:type="code:Package" name="ufscar">
     <codeElement xsi:type="code:Package" name="sin">
      <codeElement xsi:type="code:Package" name="prograd">
       <codeElement xsi:type="code:Package" name="negocio">

        <!-- 2 -->
        <codeElement xsi:type="code:ClassUnit" name="Matricula" isAbstract="false">
         <attribute tag="export" value="public"/>

         <!-- 3 -->
         <codeElement xsi:type="code:MethodUnit" name="calcularPerfil">
          <attribute tag="export" value="public"/>
          <codeElement xsi:type="code:Signature" name="calcularPerfil">
           <parameterUnit type="Integer" kind="return"/>
           <parameterUnit name="ano" type="Integer" kind="unknown"/>
           <parameterUnit name="semestre" type="Integer" kind="unknown"/>
           <parameterUnit name="numeroDeTrancamentos" type="Integer" kind="unknown"/>
           <parameterUnit name="numeroDeAfastamentos" type="Integer" kind="unknown"/>
           <parameterUnit name="matricula" type="Matricula" kind="unknown"/>
          </codeElement>
          <codeElement xsi:type="action:BlockUnit">
           ...
          </codeElement>
         </codeElement>

         <!-- 4 -->
         <codeElement xsi:type="code:MethodUnit" name="somarNotaXCreditosCursado"
             export="private">
          <attribute tag="export" value="private"/>
          <codeElement xsi:type="code:Signature" name="somarNotaXCreditosCursado">
           <parameterUnit type="Double" kind="return"/>
```

```
38            <parameterUnit name="inscricao" type="Inscricao" kind="unknown"/>
39           </codeElement>
40           <codeElement xsi:type="action:BlockUnit">
41            <codeElement xsi:type="action:ActionElement" name="return" kind="return">
42             <codeElement xsi:type="action:ActionElement" name="parenthesized"
                   kind="parenthesized">
43              <codeElement xsi:type="action:ActionElement" name="TIMES" kind="infix
                    expression">
44               <codeElement xsi:type="action:ActionElement" name="method invocation"
                     kind="method invocation">
45                <actionRelation xsi:type="action:Calls" to="getNota" from="method
                     invocation"/>
46               </codeElement>
47               <codeElement xsi:type="action:ActionElement" name="method invocation"
                     kind="method invocation">
48                <codeElement xsi:type="action:ActionElement" name="method invocation"
                     kind="method invocation">
49                 <codeElement xsi:type="action:ActionElement" name="method invocation"
                      kind="method invocation">
50                  <actionRelation xsi:type="action:Calls" to="getTurma" from="method
                       invocation"/>
51                 </codeElement>
52                 <actionRelation xsi:type="action:Calls" to="getDisciplina" from="method
                     invocation"/>
53                </codeElement>
54                <actionRelation xsi:type="action:Calls" to="somarCreditos" from="method
                     invocation"/>
55               </codeElement>
56              </codeElement>
57             </codeElement>
58            </codeElement>
59           </codeElement>
60          </codeElement>
61
62          <!-- 5 -->
63          <codeElement xsi:type="code:MethodUnit" name="somarCreditosInscritos"
                export="private">
64           <attribute tag="export" value="private"/>
65           <codeElement xsi:type="code:Signature" name="somarCreditosInscritos">
66            <parameterUnit type="Integer" kind="return"/>
67            <parameterUnit name="inscricao" type="Inscricao" kind="unknown"/>
68           </codeElement>
69           <codeElement xsi:type="action:BlockUnit">
70            ...
71           </codeElement>
72          </codeElement>
73
74          <!-- 6 -->
75          <codeElement xsi:type="code:MethodUnit" name="somarCreditosCancelados"
                export="private">
76           <attribute tag="export" value="private"/>
77           <codeElement xsi:type="code:Signature" name="somarCreditosCancelados">
78            <parameterUnit type="Integer" kind="return"/>
79            <parameterUnit name="inscricao" type="Inscricao" kind="unknown"/>
80           </codeElement>
81           <codeElement xsi:type="action:BlockUnit">
82            ...
83           </codeElement>
84          </codeElement>
85
86          <!-- 7 -->
87          <codeElement xsi:type="code:MethodUnit" name="somarCreditosDesistentes"
                export="private">
88           <attribute tag="export" value="private"/>
89           <codeElement xsi:type="code:Signature" name="somarCreditosDesistentes">
90            <parameterUnit type="Integer" kind="return"/>
```

```
91            <parameterUnit name="inscricao" type="Inscricao" kind="unknown"/>
92           </codeElement>
93           <codeElement xsi:type="action:BlockUnit">
94            ...
95           </codeElement>
96          </codeElement>
97
98          <!-- 8 -->
99          <codeElement xsi:type="code:MethodUnit" name="calcularIra">
100          <attribute tag="export" value="public"/>
101          <codeElement xsi:type="code:Signature" name="calcularIra">
102           <parameterUnit type="void" kind="return"/>
103           <parameterUnit type="DAOException" kind="throws"/>
104           <parameterUnit type="SQLException" kind="throws"/>
105          </codeElement>
106          <codeElement xsi:type="action:BlockUnit">
107
108           <!-- 9 -->
109           <codeElement xsi:type="action:ActionElement" name="variable declaration"
                 kind="variable declaration">
110            <codeElement xsi:type="code:StorableUnit" name="matriculas"
                  type="java.util.List<Matricula>" kind="local">
111             <attribute tag="export" value="none"/>
112             <codeRelation xsi:type="code:HasValue"
                   to="obterTodasMatriculasAtivasParaCursosPresenciais"
                   from="matriculas"/>
113            </codeElement>
114           </codeElement>
115
116           <!-- 10 -->
117           <codeElement xsi:type="action:ActionElement" name="foreach" kind="foreach">
118            <codeElement xsi:type="action:BlockUnit">
119
120             <!-- 11 -->
121             <codeElement xsi:type="action:ActionElement" name="variable declaration"
                   kind="variable declaration">
122              <codeElement xsi:type="code:StorableUnit" name="inscricoes"
                    type="java.util.List<Inscricao>" kind="local">
123               <attribute tag="export" value="none"/>
124               <codeRelation xsi:type="code:HasValue" to="obterInscricoes"
                     from="inscricoes"/>
125              </codeElement>
126             </codeElement>
127
128             <!-- 12 -->
129             <codeElement xsi:type="action:ActionElement" name="foreach" kind="foreach">
130              <codeElement xsi:type="action:BlockUnit">
131
132               <!-- 13 -->
133               <codeElement xsi:type="action:ActionElement" name="if" kind="if">
134                <codeElement xsi:type="action:ActionElement" name="CONDITIONAL_AND"
                      kind="infix expression">
135                 ...
136                </codeElement>
137                <codeElement xsi:type="action:BlockUnit">
138
139                 <!-- 14 -->
140                 <codeElement xsi:type="action:ActionElement" name="expression
                       statement" kind="expression statement">
141                  ...
142                 </codeElement>
143                </codeElement>
144               </codeElement>
145
146               <!-- 15 -->
```

```
147                <codeElement xsi:type="action:ActionElement" name="expression statement"
                       kind="expression statement">
148                 <codeElement xsi:type="action:ActionElement" name="ASSIGN"
                        kind="assignment">
149                  ...
150                 </codeElement>
151                </codeElement>
152
153                <!-- 16 -->
154                <codeElement xsi:type="action:ActionElement" name="if" kind="if">
155                 <codeElement xsi:type="action:ActionElement" name="EQUALS" kind="infix
                        expression">
156                  ...
157                 </codeElement>
158                 <codeElement xsi:type="action:BlockUnit">
159
160                  <!-- 17 -->
161                  <codeElement xsi:type="action:ActionElement" name="expression
                         statement" kind="expression statement">
162                   <codeElement xsi:type="action:ActionElement" name="ASSIGN"
                          kind="assignment">
163                    ...
164                   </codeElement>
165                  </codeElement>
166                 </codeElement>
167                </codeElement>
168
169                <!-- 18 -->
170                <codeElement xsi:type="action:ActionElement" name="if" kind="if">
171                 <codeElement xsi:type="action:ActionElement" name="EQUALS" kind="infix
                        expression">
172                  ...
173                 </codeElement>
174                 <codeElement xsi:type="action:BlockUnit">
175
176                  <!-- 19 -->
177                  <codeElement xsi:type="action:ActionElement" name="expression
                         statement" kind="expression statement">
178                   <codeElement xsi:type="action:ActionElement" name="ASSIGN"
                          kind="assignment">
179                    ...
180                   </codeElement>
181                  </codeElement>
182                 </codeElement>
183                </codeElement>
184               </codeElement>
185              </codeElement>
186
187              <!-- 20 -->
188              <codeElement xsi:type="action:ActionElement" name="variable declaration"
                     kind="variable declaration">
189               <codeElement xsi:type="code:StorableUnit" name="ira" type="Double"
                      kind="local">
190                <attribute tag="export" value="none"/>
191                <codeRelation xsi:type="code:HasValue" to="expression statement"
                       from="ira"/>
192               </codeElement>
193              </codeElement>
194
195              <!-- 21 -->
196              <codeElement xsi:type="action:ActionElement" name="expression statement"
                     kind="expression statement">
197               <codeElement xsi:type="action:ActionElement" name="method invocation"
                      kind="method invocation">
198                <actionRelation xsi:type="action:Calls" to="atualizarIra" from="method
                       invocation"/>
```

```
199                </codeElement>
200               </codeElement>
201          ...
202     </codeElement>
203    </model>
204   </kdm:Segment>
205 </xmi:XMI>
```

Código C.1: Modelo KDM PIM transformado do Modelo Java PSM

Anexo D

Processo da classe Matricula em jBPM

```
RuleFlowProcessFactory factory = RuleFlowProcessFactory
    .createProcess("ProGradWeb");

factory
  // Header
  .name("Matricula::calcularIra")
  .version("1.0")
  .packageName("br.ufscar.sin.prograd.negocio")

  // Nodes
  .startNode(1).done()
  .actionNode(2).name("obterTodasMatriculasAtivasParaCursosPresenciais()").done()
  .splitNode(3).type(TYPE_XOR)
    .constraint(4, "true", "code", "Java", "Matricula matricula : matriculas")
    .constraint(14, "false", "code", "Java", "Matricula matricula :
        matriculas").done()
  .actionNode(4).name("obterInscricoes(matricula)").done()
  .splitNode(5).type(TYPE_XOR)
    .constraint(6, "true", "code", "Java", "Inscricao inscricao : inscricoes")
    .constraint(3, "false", "code", "Java", "Inscricao inscricao :
        inscricoes").done()
  .splitNode(6).type(TYPE_XOR)
    .constraint(7, "true", "code", "Java",
        "Integer.parseInt(inscricao.getResultado()) < 6 &&
        inscricao.getStatus().matches(\"[3,6-9,17]\")")
    .constraint(8, "false", "code", "Java",
        "Integer.parseInt(inscricao.getResultado()) < 6 &&
        inscricao.getStatus().matches(\"[3,6-9,17]\")")
  .actionNode(7).name("somarNotaXCreditosCursado(inscricao))").done()
  .actionNode(8).name("somarCreditosInscritos(inscricao)").done()
  .splitNode(9).type(TYPE_XOR)
    .constraint(10, "true", "code", "Java",
        "Integer.parseInt(inscricao.getResultado()) == 7")
    .constraint(11, "false", "code", "Java",
        "Integer.parseInt(inscricao.getResultado()) == 7")
  .actionNode(10).name("somarCreditosCancelados(inscricao)").done()
  .splitNode(11).type(TYPE_XOR)
```

```
29      .constraint(12, "true", "code", "Java",
            "Integer.parseInt(inscricao.getResultado()) == 6")
30      .constraint(13, "false", "code", "Java",
            "Integer.parseInt(inscricao.getResultado()) == 6")
31    .actionNode(12).name("somarCreditosDesistentes(inscricao)").done()
32    .actionNode(13).name("atualizarIra(ira)").done()
33    .endNode(14).done()
34
35    // Connections
36    .connection(1,2)
37    .connection(2,3)
38    .connection(3,4)
39    .connection(3,14)
40    .connection(4,5)
41    .connection(5,6)
42    .connection(5,3)
43    .connection(6,7)
44    .connection(6,8)
45    .connection(7,8)
46    .connection(8,9)
47    .connection(9,10)
48    .connection(9,11)
49    .connection(10,11)
50    .connection(11,12)
51    .connection(11,13)
52    .connection(12,13)
53    .connection(13,4);
54
55  RuleFlowProcess process = factory.validate().getProcess();
```

Código D.1: Representação do Processo de Negócio da classe Matricula em jBPM