UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

GLEISON SAMUEL DO NASCIMENTO

Um Método para Descoberta Semi-Automática de Processos de Negócio Codificados em Sistemas Legados

Tese apresentada como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação

Prof. Dr. Cirano Iochpe
Orientador

Prof. Dr. Álvaro Freitas Moreira
Co-orientador

Porto Alegre, janeiro de 2014.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Nascimento, Gleison Samuel do

Um Método para Descoberta Semi-Automática de Processos de Negócio Codificados em Sistemas Legados / Gleison Samuel do Nascimento – Porto Alegre: Programa de Pós-Graduação em Computação, 2013.

109 f.:il.

Orientador: Cirano Iochpe; Co-orientador: Álvaro Freitas Moreira.

Tese (doutorado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2013.

1.BPM. 2.Processos de Negócio. 3.Sistemas Legados. I. Iochpe, Cirano. II. Moreira, Álvaro Freitas. III. Um Método para Descoberta Semi-Automática de Processos de Negócio Codificados em Sistemas Legados.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Carlos Alexandre Netto
Vice-Reitor: Prof. Rui Vicente Oppermann
Pró-Reitor de Pós-Graduação: Prof. Vladimir Pinheiro do Nascimento
Diretor do Instituto de Informática: Prof. Luís da Cunha Lamb
Coordenador do PPGC: Prof. Luigi Carro
Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

AGRADECIMENTOS

Agradeço a minha esposa Janaina Figueiredo da Silva pelo apoio, conversas, compreensão e paciência durante todo tempo de estudo. Aos meus pais, Ivete Maria do Nascimento e Joaquim Samuel do Nascimento, que sempre incentivaram meus estudos e me apoiaram em toda trajetória estudantil. Aos familiares Greicemara Samuel do Nascimento, Marlene Alves da Silva, Patrik Zick e demais familiares pelo estímulo adicional dado durante este período de doutorado.

Agradeço aos meus orientadores, Prof. Dr. Cirano Iochpe e Prof. Dr. Alvaro Freitas Moreira, pelas lições que me ensinaram durante o desenvolvimento deste trabalho e que levarei para toda a vida acadêmica.

Aos meus colegas do grupo de pesquisa, Lucinéia Heloísa Thom e André Cristiano Kalsing, pela amizade e pelos valorosos auxílios que contribuíram para a conclusão desta pesquisa.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

BPEL	*Business Process Execution Language*
BPM	*Business Process Management*
BPMN	*Business Process Management Notation*
BPMS	*Business Process Management System*
BR	Brasil
BRMS	*Business Rules Management System*
CRM	*Customer Relationship Management*
ECA	*Event-Control-Action*
ERP	*Enterprise Resource Plain*
KDM	*Knowledge Discovery Metamodel*
OMG	*Object Management Group*
SOA	*Service Oriented Architecture*
UFRGS	Universidade Federal do Rio Grande do Sul
WfMC	*Workflow Management Coalition*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Há mais de uma década, BPM vem sendo introduzida nas organizações devido suas vantagens tais como documentação, gerenciamento, monitoração e melhoria contínua de seus processos de negócio.

Na abordagem BPM, normalmente, os processos de negócio da organização são executados sob o controle de um Sistema Gerenciador de Processos de Negócio. Estes sistemas executam os processos de negócio, coletando informações úteis para organização. Por exemplo, através destes sistemas é possível identificar as atividades que mais demoram ou consomem mais recursos humanos. Desta forma, é possível redesenhar os processos de maneira ágil, garantido a evolução contínua do negócio.

Entretanto, para se beneficiar da tecnologia BPM, a organização deve mapear seus processos de negócio e modelá-los no Sistema Gerenciador de Processos de Negócio. Normalmente, esse trabalho é realizado por especialistas humanos, que observam e identificam o funcionamento da organização, definindo, em detalhes, os fluxos de trabalho realizados para cumprir determinadas metas de negócio.

Contudo, na maior parte das organizações os processos de negócio encontram-se implementados em sistemas de informações legados. Tais sistemas possuem pouca documentação, foram desenvolvidos com uso de tecnologias obsoletas e os processos de negócio, neles contidos, foram programados implicitamente no seu código fonte.

Deste modo, além das entrevistas com usuários-chave da organização, os analistas precisam também entender o funcionamento dos sistemas legados a fim de identificar os processos de negócio da organização.

Geralmente, os analistas de negócio fazem este trabalho manualmente, interpretando os algoritmos escritos no código fonte legado e identificando os fluxos de trabalho nele escritos. Esse trabalho é complexo, demorado e suscetível a erros, pois depende do nível de conhecimento que os analistas de negócio têm sobre o código fonte legado.

Pensando neste problema, essa tese apresenta um método que automatiza a descoberta de processos de negócio implementados implicitamente no código fonte de sistemas legados. O método propõe uma técnica híbrida, que usa análise estática do código fonte e análise dinâmica (mineração de processos) para descobrir os processos de negócio codificados em sistemas legados.

A tese apresenta os passos para aplicação do método, definindo para cada passo, um conjunto de ferramentas capazes de automatizar a descoberta de informações no código fonte legado. Este trabalho também mostra três estudos de caso, onde o método foi aplicado com sucesso e comparado a outras técnicas existentes na literatura.

Palavras-Chave: BPM, processos de negócio, regras de negócio, mineração de processos, sistemas legados.

A Semi-Automatic Method to Discovery Business Processes Encoded in Legacy Systems

ABSTRACT

For over a decade, BPM is being introduced in organizations due to its advantages such as documentation, management, monitoring and continuous improvement of its business processes.

In BPM approach, business processes of the organization are executed under the control of a Business Processes Management System. These systems monitor the execution of the processes and measuring the operational efficiency of the organization through, for example, of the identification of activities those are slower or consume more resources. Thus, the organization can redesign their business processes in an agile and fast mode, thereby ensuring the continued evolution of your business.

However, to take advantage of BPM technology, the organization must map their business processes and model them in the Business Processes Management System. Typically, organizations execute the business process mapping through manual techniques, such as interviews, meetings with users, questionnaires, document analysis and observations of the organizational environment.

However, in most organizations business processes are executed in legacy systems information. Such systems have not documentation, have been developed with obsolete technologies and the business processes are programmed implicitly in its source code.

Thus, in addition to interviews with expert users of the organization, analysts must also understand the working of legacy systems in order to identify the business processes of the organization.

Generally, business analysts do this work manually, interpreting algorithms written in legacy source code and identifying workflows written in the source code. This work is complex, time consuming and error prone, since it depends on the knowledge level that business analysts have about the legacy source code.

Thinking about this problem, this thesis presents a method that automates the discovery of business processes implemented implicitly in the source code of legacy systems. The method proposes a hybrid technique that uses static analysis of the source code and dynamic analysis (mining process) to discover business processes encoded in legacy systems.

The thesis presents the steps for applying the method, defining for each step, a set of tools that automate the discovery of information in the legacy source code. This work also shows three case studies where the method was successfully applied and compared to other existing techniques in the literature.

Keywords: BPM, business processes, business rules, mining processes, legacy systems.

1 INTRODUÇÃO

Para manterem-se competitivas em ambientes de negócio dinâmicos, para melhor interagir com clientes e para garantir a qualidade de seus serviços, as organizações buscam maior padronização e eficiência na execução de seus processos de negócio. Neste contexto, o gerenciamento de processos de negócio ou **BPM** (*Business Process Management*), através da documentação e automação dos processos de negócio executados na organização, proporciona a redução de custos, de tempo e de erros na execução de procedimentos essenciais para a organização. BPM também proporciona maior controle sobre o funcionamento do negócio e capacidade de adequação a situações inesperadas, levando ao incremento da qualidade dos processos, de seus resultados e da organização como um todo (WESKE, 2012).

Um processo de negócio é um conjunto de atividades inter-relacionadas, executadas em uma determinada ordem parcial, para cumprir um determinado objetivo de negócio da organização (WESKE, 2012). Por exemplo, a realização de uma venda na organização é considerada um processo de negócio, onde, vender um produto é o objetivo de negócio e para cumpri-lo é necessário executar um conjunto de atividades, como verificar o estoque, definir as condições de pagamento, verificar o crédito do cliente, entre outras.

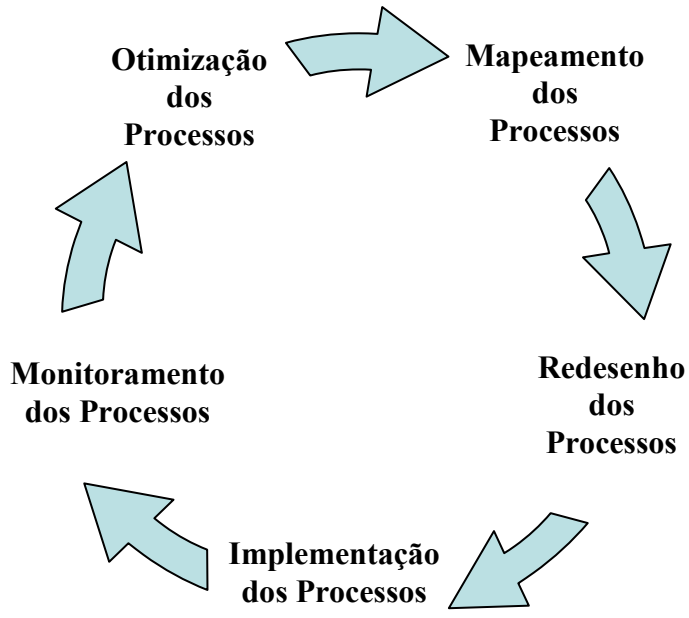
Os processos de negócio refletem como uma organização funciona e são responsáveis por produzirem os serviços ou produtos que serão entregues aos clientes da organização. Através dos processos de negócio as organizações criam seus diferenciais competitivos.

BPM define um ciclo de vida para os processos de negócio, como mostra a Figura 1.1. O ciclo de vida consiste em cinco grandes etapas (WESKE, 2012):

1. **Mapeamento dos Processos de Negócio:** nesta etapa são gerados modelos de processos chamados *"AS IS"* (como é). O objetivo aqui é obter quais são e como são executados os fluxos de trabalho na organização, ou seja, quais atividades são realizadas em um processo de negócio, qual sequencia destas atividades, quem são as pessoas ou departamentos envolvidos e quais objetivos de negócio são cumpridos pelos processos. Estes modelos são utilizados como documentação dos processos atuais da organização e servem de base para próxima etapa do ciclo de vida BPM.
2. **Redesenho dos Processos de Negócio:** aqui são gerados modelos de processos chamados *"TO BE"* (para ser). Estes modelos são evoluções dos modelos identificados na etapa de mapeamento, nos quais são reavaliadas questões de negócio buscando-se, através de melhorias culturais e organizacionais, maior eficiência na execução dos processos de negócio.

3. **Implementação dos Processos de Negócio:** nesta etapa os modelos de processos de negócio são implementados em um Sistema Gerenciador de Processos de Negócio ou BPMS (*Business Process Management System*). Estes sistemas interpretam os modelos e automatizam a execução das atividades dos processos, encaminhando as tarefas para os devidos responsáveis, controlando o tempo de execução por atividade e do processo como um todo.
4. **Monitoramento dos Processos de Negócio:** após a automação dos processos de negócio em um BPMS, a execução dos processos pode ser monitorada por analistas de negócio da organização. Aqui os analistas podem verificar o andamento de execução de um processo, como, quanto tempo é utilizando para executar o processo, se está parado em alguma atividade, onde está parado, com quem está parado, quantas pessoas utilizam os processos, entre outros pontos de controle. Alguns BPMS apresentam gráficos e relatórios mostrando indicadores extremamente úteis para a gestão do negócio.
5. **Otimização dos Processos de Negócio:** nesta etapa os indicadores coletados na etapa de monitoramento são utilizados para melhorar a eficiência dos processos de negócio da organização. Com base nestas informações os analistas sugerem modificações nos processos de negócio, reiniciando ciclo para modelar e implementar um processo de negócio mais otimizado.

Figura 1.1: Ciclo de vida BPM.



Fonte: Weske (2012).

1.1 Problema e Motivação da Tese

O mapeamento dos processos de negócio é a etapa com custo financeiro mais elevado na implantação do gerenciamento de processos de negócio nas organizações

(KETTINGER; TENG; GUHA, 1997). Isto porque, os processos são mapeados através de técnicas manuais executadas por analistas de negócio[1]. Os analistas de negócio modelam os processos a partir de entrevistas ou reuniões com usuários da organização, aplicação de questionários, análise de documentos sobre negócio da organização e observações do ambiente de trabalho (KETTINGER; TENG; GUHA, 1997).

Desta forma, o mapeamento inicial dos processos de negócio tende a ser um trabalho lento e com custo financeiro elevado, pois são necessárias muitas horas de trabalho de profissionais especialistas em BPM, além de interferir nas atividades diárias da organização, já que seus usuários (ex.: funcionários, clientes, fornecedores) estarão envolvidos em reuniões, entrevistas, entre outras atividades (KETTINGER; TENG; GUHA, 1997).

Entretanto, a maioria das organizações possuem sistemas de informação que suportam a execução de seus processos de negócio. Estes sistemas foram desenvolvidos ao longo da história da organização e são conhecidos como sistemas legados.

Sistemas legados são sistemas de informação que realizam tarefas úteis para a organização, mas foram desenvolvidos com tecnologias atualmente consideradas obsoletas (WARD; BENNETT, 1995). Estes sistemas têm informações e procedimentos essenciais para o funcionamento da organização e normalmente não estão devidamente documentados.

Deste modo, a análise das informações e procedimentos implementados nos sistemas legados pode acelerar o trabalho de mapeamento de processos de negócios. A análise do comportamento de sistemas legados pode apoiar os analistas de negócio na identificação dos processos de negócios de uma organização (NASCIMENTO et. al., 2012; NASCIMENTO et. al., 2013).

Recentemente, algumas pesquisas propõem técnicas para descobrir fragmentos de processos de negócio implementados em sistemas legados. Estas técnicas são baseadas em análise estática do código fonte (ZOU; HUNG, 2006; PEREZ-CASTILLO et. al., 2009) ou análise dinâmica da execução do sistema legado (GUNTHER; AALST, 2006; AALST; REIJERS; WEIJTERS, 2007; DI FRANCESCOMARINO; MARCHETTO; TONELLA, 2009).

Técnicas de análise estática extraem informações de arquivos de código fonte, analisando as instruções de uma linguagem de programação, em busca de sentenças que representam informações sobre o negócio da organização (ZOU; HUNG, 2006; PEREZ-CASTILLO et. al., 2009). Por exemplo, uma declaração *"if-then-else"* tem uma expressão booleana que pode representar uma decisão no negócio da organização.

Técnicas de análise estática são usadas para determinar fragmentos de processos de negócios, tais como atividades, gateways ou pequenas sequências de atividades. As técnicas de análise estática não conseguem determinar a ordem parcial das atividades dos processos de negócio. Um analista de negócio precisa conectar os fragmentos identificados na análise estática, formando um modelo de processo de negócio completo.

[1] Neste contexto, os analistas de negócio são profissionais de tecnologia da informação, especialistas em BPM e com conhecimento prévio do negócio da organização.

Por sua vez, técnicas baseadas em análise dinâmica permitem extrair fluxos de trabalho a partir de arquivos de log do sistema de informação. Os arquivos de log capturam os eventos de negócios gerados durante a execução do sistema. Através da mineração dos arquivos de log é possível determinar a ordem parcial de execução dos eventos de negócio. Normalmente, técnicas que utilizam análise dinâmica identificam processos de negócio de sistemas de informação orientados a processos, isto é, em sistemas que geram logs de eventos, onde cada evento representa a execução de uma atividade de negócio. Assim, a mineração deste arquivo de log determina a ordem parcial das atividades de um processo de negócio. Estas técnicas são também chamadas de mineração de processos de negócios (GUNTHER; AALST, 2006; AALST; REIJERS; WEIJTERS, 2007).

Entretanto, grande parte dos sistemas legados não é beneficiada pelas técnicas de mineração de processos, pois não foram codificados para gerar logs de eventos. Sendo assim, a análise dinâmica não pode ser usada para extrair processos de negócio de sistemas legados.

Portanto, atualmente, as informações sobre processos de negócio codificados em sistemas legados não são utilizadas pelos analistas de negócio ao implantar o gerenciamento de processos de negócio em uma organização. Normalmente, os analistas descartam essas informações, por falta de documentação adequada no sistema legado e por falta de uma técnica eficaz para descobrir os modelos de processos em sistemas legados.

1.2 Hipótese e Objetivos da Tese

Pensando na reutilização das informações de sistemas legados para apoiar e acelerar o mapeamento de processos de negócio nas organizações, esta pesquisa propõe como hipótese que é possível definir um método para, pelo menos, semi-automatizar a descoberta de processos de negócio implementados implicitamente no código fonte de sistemas legados.

Para atingir a hipótese desta pesquisa são definidos os seguintes objetivos específicos:

1. Investigar diferentes técnicas de extração de conhecimento aplicadas a sistemas legados, a fim de determinar as técnicas mais adequadas para definição do método proposto nesta tese.
2. Estudar como as atividades de negócio são caracterizadas no código fonte de sistemas legados. Através destas características pode-se identificar os trechos de código fonte responsáveis pela implementação das atividades dos processos de negócio.
3. Projetar algoritmos e ferramentas capazes de automatizar a aplicação do método de descoberta de processos de negócios implementados em sistemas legados.
4. Definir um método capaz de ser utilizado em diferentes categorias de sistemas legados, independente do domínio de aplicação e da tecnologia utilizada na codificação do sistema legado.
5. Demonstrar a aplicação do método em diferentes categorias de sistemas legados.

1.3 Metodologia

O desenvolvimento deste trabalho iniciou-se com o estudo das técnicas de extração de conhecimento de sistemas legados existente na literatura. Neste estudo foram pesquisados os pontos fortes e fracos de cada técnica para identificação automática de processos de negócio implementados em sistemas legados.

Através deste estudo observou-se que a combinação de técnicas de análise estática e técnicas de análise dinâmica poderiam levar a um método híbrido, capaz de identificar modelos de processos de negócio implementados implicitamente no código fonte de sistemas legados.

Deste modo, definiu-se um método híbrido baseado em análise estática e dinâmica composto de três grandes passos (ambos automáticos):

- **Etapa 01:** instrumentalizar o código fonte legado, anotando o inicio e o fim da codificação das atividades de negócio, conforme mostra a Figura 1.2(a). As anotações são sentenças adicionais inseridas no código fonte para escrever eventos em um arquivo de log.
- **Etapa 02:** executar o sistema legado com cenários das principais tarefas executadas no sistema. Esta etapa gera um arquivo de log que inclui os eventos instrumentalizados na etapa 01. A Figura 1.2(b) mostra um exemplo de arquivo de log gerado após a execução do sistema.
- **Etapa 03:** minerar o arquivo de log obtendo a ordem parcial de execução dos eventos instrumentalizados no sistema legado. A Figura 1.2(c) mostra a ordem parcial que pode ser expressa em um modelo de processo.

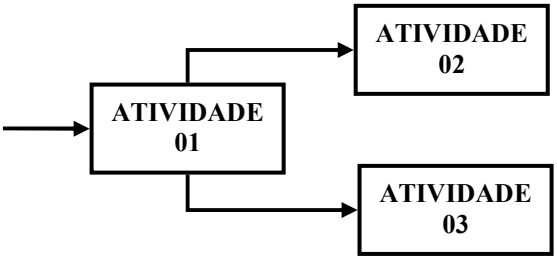
Figura 1.2: Método proposto para descoberta de processos de negócio em legados.

```
fprint(logfile, "%d – ATIVIDADE 01", order_number);
if (x > 100) {
   fprint(logfile, "%d – ATIVIDADE 02", order_number);
   send_order(order_number);
   // FIM ATIVIDADE 02
} else {
   fprint(logfile, "%d - ATIVIDADE 03", order_number);
   cancel_order(order_number);
   // FIM ATIVIDADE 03
}
// FIM ATIVIDADE 01
```

(a) Código Fonte Anotado (análise estática).

```
100301 - ATIVIDADE 01
100320 - ATIVIDADE 01
100301 - ATIVIDADE 02
100320 - ATIVIDADE 03
100331 - ATIVIDADE 01
100331 - ATIVIDADE 03
100340 - ATIVIDADE 01
100340 - ATIVIDADE 02
100355 - ATIVIDADE 01
100355 - ATIVIDADE 03
...
```

(b) Log de Eventos.



(c) Ordem Parcial de Atividades extraída do Log (análise dinâmica).

Fonte: elaborado pelo autor.

A partir da definição do método, o próximo passo da pesquisa verificou como identificar as atividades de negócio implementadas no código fonte de sistemas legados. Ou seja, como identificar os fragmentos de código fonte que devem ser anotados como possíveis implementações de atividades de negócio em um sistema legado.

Para isso, foi utilizado o conceito de regras de negócio definido por Karakostas (1990). Segundo Karakostas (1990), as regras de negócio são declarações textuais utilizadas para especificar o comportamento de um sistema de informação. Durante a codificação do sistema, os desenvolvedores usam estas declarações para escrever o código fonte.

Da mesma forma, um processo de negócio pode ser especificado, modelado e implementado através do conceito de regras de negócio, como demonstram os trabalhos de Knolmayer, Endl e Pfahrer (2000), Charfi e Mezini (2004) e Lee et. al. (2007). Deste modo, este trabalho usa as regras de negócio como invariantes para mapear as atividades de negócio implementadas no código fonte para um modelo de processo de negócio.

Na sequencia do trabalho elaborou-se uma técnica para identificar e instrumentalizar as regras de negócio codificadas em sistemas legados. A técnica é baseada em uma ontologia de regras de negócio, construída nesta tese, para definir as principais características de codificação das regras de negócio em um sistema de informação. A ontologia é utilizada como base de conhecimento (formal) para identificar as regras de negócio em código fonte e mapeá-las para componentes de uma notação de processos (ex.: BPMN).

Por último, o método é avaliado através de três estudos de caso realizados em sistemas legados executados em ambientes reais. Isto é, sistemas sendo utilizados por suas respectivas organizações. Nos estudos de caso procura-se demonstrar que o método pode ser aplicado em sistemas legados de diferentes domínios de aplicação (saúde, gestão de obras e gestão de pessoas), desenvolvidos em diferentes plataformas (web e desktop) e diferentes linguagens de programação (php e c).

Além disso, o método proposto nesta tese é comparado com duas técnicas empregadas atualmente na identificação de processos de negócio em sistemas legados (DI FRANCESCOMARINO; MARCHETTO; TONELLA, 2009; PEREZ-CASTILLO et. al. 2011). Desta forma, demonstra-se a eficácia do método proposto na tese, ou seja, demonstra-se que o método obtém processos de negócio mais completos que as técnicas de Di Francescomarino, Marchetto e Tonella (2009) e Pérez-Castillo et. al. (2011).

1.4 Contribuições

As principais contribuições desta tese são descritas a seguir e detalhadas no Capítulo de Conclusões, onde são relacionadas às publicações obtidas com o trabalho desenvolvido:

- Estudo teórico e definição de conceitos relacionados ao tema de extração de conhecimento de sistemas legados, principalmente no que se refere a identificação de processos de negócio codificados implicitamente em sistemas legados;
- Definição de uma técnica para identificação de regras de negócio codificadas em sistemas legados, permitindo a descoberta de informações essenciais para

o negócio de uma organização e que ao longo de sua história ficaram esquecidas no código fonte de seus sistemas legados;

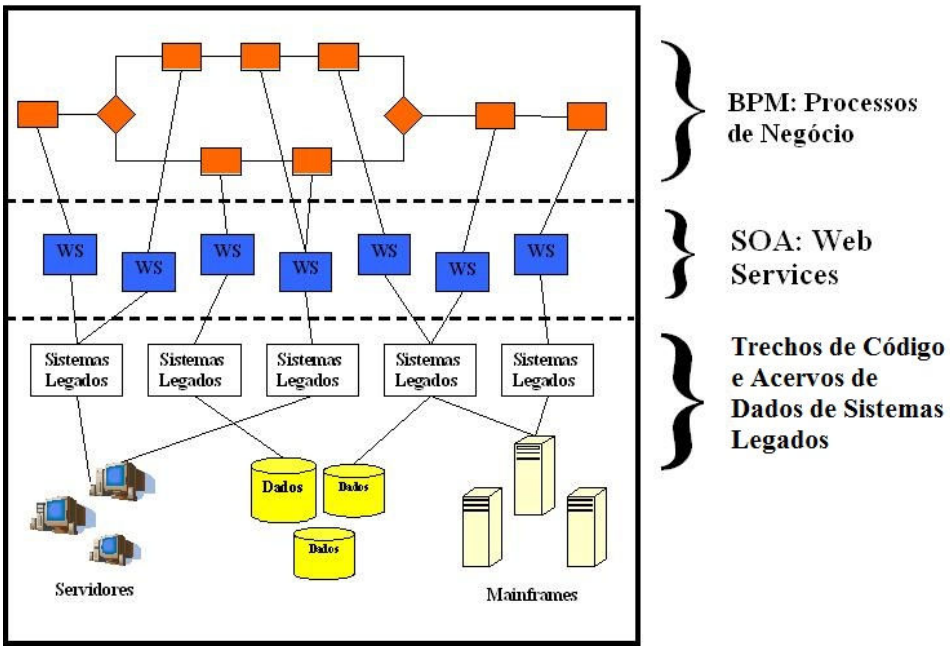
- Definição de uma ontologia de regras de negócio que pode ser usada como modelo formal para especificar sistemas de informação e processos de negócio em uma organização, além de ser usada como base de conhecimento para identificação de processos de negócio no método proposto na tese;
- Definição de um método híbrido, baseado em técnicas de análise estática e dinâmica, para identificação semi-automática de processos de negócio implementados implicitamente em sistemas legados.

O método proposto neste trabalho apoia os analistas de negócio no mapeamento inicial dos processos de negócio de uma organização, etapa mais demorada e com alto custo financeiro, do ciclo de vida BPM (KETTINGER; TENG; GUHA, 1997). O método permite acelerar o trabalho de mapeamento de processos de negócio, utilizando todo o conhecimento incorporado durante anos nos sistemas de informação da organização. Desta forma, o método se configura na principal contribuição desta tese, pois permite acelerar a implantação do gerenciamento de processos de negócio em uma organização.

Além disso, o método pode auxiliar na modernização de sistemas legados. A implantação do gerenciamento de processos de negócio, aliada a arquitetura orientada a serviços (SOA), cria as condições necessárias para uma rápida modernização do sistema legado, sem a necessidade de reescrita total do código fonte (BORGES et. al., 2005; JASMINE, 2005; WOODLEY; GAGNON, 2005; KAMOUN, 2007; NASCIMENTO et. al., 2009).

Enquanto BPM fornece ferramentas para construção e execução de processos de negócio, SOA prove uma interface padronizada que interliga os processos de negócio e os sistemas legados. Ao final do procedimento de modernização do sistema legado, o novo sistema terá uma arquitetura semelhante à apresentada na Figura 1.3.

Figura 1.3: Arquitetura dos sistemas após a modernização usando BPM e SOA.



Fonte: Kamoun (2007).

Note na Figura 1.3, que a arquitetura apresenta três camadas (KAMOUN, 2007). A primeira camada (de cima para baixo) contém o modelo do processo de negócio implementado em um BPMS. O processo de negócio, por sua vez, invoca serviços na segunda camada (SOA), que são gerados a partir da reutilização de fragmentos selecionados no código fonte do sistema legado. Os fragmentos de código legado reutilizados formam a terceira camada da arquitetura.

1.5 Organização do Trabalho

O conteúdo que segue está organizado nos capítulos:

- **Capítulo 2 –** apresenta a fundamentação teórica da tese. Aqui são detalhadas as principais técnicas encontradas na literatura para extração de conhecimento em sistemas legados. O capítulo analisa criticamente as técnicas empregadas atualmente para identificação de processos de negócio em sistemas legados. A partir desta análise, é apresentada uma nova abordagem para semi-automatizar a descoberta de processos de negócio implementados implicitamente em sistemas legados. Este capítulo também apresenta conceitos importantes que serão utilizados no restante da tese.
- **Capítulo 3** – detalha o conceito de regras de negócio, usado para identificar as atividades de negócio implementadas no código fonte de um sistema legado. O capítulo apresenta as classificações de regras de negócio existentes na literatura e define uma ontologia de regras que será usada no restante da tese.
- **Capítulo 4** – este capítulo é destinado a mostrar como as regras de negócio são implementadas no sistema legado. Aqui é demonstrado como a ontologia de regras de negócio é usada para criar uma ferramenta capaz de identificar e anotar os trechos de código fonte que implementam atividades de um processo.
- **Capítulo 5** – apresenta uma ferramenta para identificação e anotação das regras de negócio no código fonte. A ferramenta proposta neste capítulo usa a ontologia de regras de negócio apresentada nos Capítulos 3 e 4.
- **Capítulo 6** – define o método semi-automático para descoberta de processos de negócio implementados em sistemas legados. Com base na ferramenta de identificação de regras de negócio apresentada no Capítulo 5, este capítulo apresenta as etapas que formam o método proposto nesta tese de doutorado.
- **Capítulo 7** – apresenta três estudos de caso realizados com o método definido no Capítulo 6. Os estudos de caso foram executados em sistemas reais, executados em uma instituição da área de saúde e uma instituição da área de educação. O capítulo descreve todas as etapas de aplicação do método, além de comparar os resultados de nosso método em relação aos demais métodos ou técnicas propostas na literatura.
- **Capítulo 8** – apresenta as considerações finais da tese, descrevendo as principais contribuições e resultados obtidos na pesquisa, além de apontar possíveis trabalhos futuros para a sequencia da pesquisa.

2 FUNDAMENTAÇÃO TEÓRICA

O objetivo geral da tese é a definição de um método semi-automático para descoberta de processos de negócio implementados implicitamente no código fonte de sistemas legados. Deste modo, os principais fundamentos teóricos relacionados a este objetivo são descritos neste capítulo. São apresentados:

- Conceitos e técnicas atuais empregadas na extração de conhecimento em sistemas legados;
- Análise crítica das principais técnicas empregadas na identificação de processos de negócio codificados em sistemas legados;
- Proposta de solução dos problemas encontrados nas técnicas atuais empregadas na identificação de processos de negócio codificados em sistemas legados;
- Conceitos e definições de uma linguagem de programação imperativa, necessária para uso nos demais capítulos da tese;

2.1 Extração de Conhecimento em Sistemas Legados

Esta seção apresenta as principais técnicas usadas na extração de conhecimento de sistemas legados. São apresentados trabalhos encontrados na literatura e como estes trabalhos podem contribuir na identificação de processos de negócio codificados em sistemas legados. A partir deste estudo procura-se responder o primeiro objetivo desta tese.

A extração de conhecimento de sistemas legados é um assunto que tem sido abordado por várias pesquisas nos últimos anos (SNEED; ERDOS, 1996; LIU; ALDERSON; QURESHI, 1999; WANG et. al., 2004; AALST; REIJERS; WEIJTERS, 2007; MILLHAM, 2010). Estas pesquisas mostram que o conhecimento de um sistema legado pode ser extraído de várias fontes de informações. As principais fontes são: a "*documentação*" do sistema; as bases de dados manipuladas pelo sistema; o código fonte legado; e os "*traces*" de execução do sistema.

2.1.1 Documentação

Entende-se por documentação, todos os artefatos que não são usados pelo computador para executar o sistema, mas destinam-se aos usuários e engenheiros de software para compreensão do funcionamento deste sistema. Relatórios técnicos, arquivos de ajuda do sistema, comentários no código fonte, diagramas do projeto, entre outros documentos são exemplos de artefatos que fazem parte da documentação de um sistema legado.

Como a documentação se destina aos seres humanos, normalmente são artefatos descritos em linguagem natural, como português ou inglês. Deste modo, é difícil analisar automaticamente estes artefatos.

As técnicas mais encontradas na literatura baseiam-se na análise morfológica das palavras e na frequência em que elas aparecem na documentação (GHOSE; KOLIADIS; CHEUNG, 2007; HASEGAWA et. al., 2009).

Através da análise morfológica as palavras são classificadas em categorias gramaticais (substantivos, verbos, preposições, entre outras). Posteriormente, as categorias gramaticais e a frequência das palavras são usadas para classificar a importância das palavras dentro da especificação do sistema. Por exemplo, quanto maior a frequência de uma palavra classificada como substantivo (nome próprio), maior a probabilidade de ser um conceito de negócio empregado na construção do sistema legado.

Os trabalhos de Ghose, Koliadis e Cheung (2007) e Hasegawa et. al. (2009) são exemplos de técnicas que usam a análise da documentação para extrair conhecimento de sistemas legados.

Ghose, Koliadis e Cheung (2007) mostram uma técnica para extrair fragmentos de processos a partir da análise da documentação de um sistema legado. A técnica consiste em buscar padrões textuais de negócio, isto é, sentenças textuais que mais ocorrem dentro da documentação e que foram construídas a partir de um determinado padrão morfológico. Essas sentenças são classificadas e transformadas em fragmentos de processos modelados com BPMN (*Business Process Modeling Notation*). Através da conexão destes fragmentos é possível determinar os processos de negócio implementados no sistema legado.

Hasegawa et. al. (2009) usa a documentação para extrair grafos conceituais que representam requisitos do sistema de informação. Da mesma forma que Ghose, Koliadis e Cheung (2007), aqui são encontrados padrões textuais de acordo com modelos morfológicos pré-definidos. Entretanto, Hasegawa et. al. (2009) transforma estes padrões em grafos conceituais, que posteriormente podem ser usados para modelar diversos artefatos de um sistema legado (por exemplo, modelos de objetos, ontologia do domínio, modelos de processos de negocio, entre outros).

2.1.2 Bases de Dados

As bases de dados podem ser usadas como fonte de informação para auxiliar na reescrita de sistemas legados. Entretanto, a engenharia reversa de dados é também um trabalho específico que pode ser feito independentemente de qualquer sistema que possa manipular esses dados. Por exemplo, é possível converter um velho banco de dados sobre um "*mainframe*", para um banco de dados relacional e distribuído em várias máquinas.

Agrawal, Imielinski e Swami (1993), Júnior (2002) e Paradauskas e Laurikaitis (2006) são exemplos de autores cujos trabalhos usam base de dados para descoberta de conhecimento sobre sistemas legados.

Agrawal, Imielinski e Swami (1993) propõem uma técnica para extração de regras de associação, que determinam relações entre campos de um banco de dados. Ou seja, uma regra de associação indica a probabilidade de uma determinada estrutura de dados ser encontrada no banco de dados, acompanhada de uma segunda estrutura de dados. As

regras de associação representam combinações de estruturas de dados que ocorrem com frequência em um determinado banco. Através das regras de associação, um especialista do sistema pode inferir possíveis fluxo de informações ou tomadas de decisão implementadas no sistema legado.

Júnior (2002) propõe um método que utiliza técnicas de armazéns de dados e de mineração de dados, na obtenção parcial de regras de negócios e fluxos de trabalhos. Aqui, o autor utiliza os logs de transações de banco de dados para identificar pequenos fragmentos de processos. O método consiste em minerar os logs de transações, buscando a ordem parcial das transações ocorridas em uma determinada tabela do banco de dados. Através da ordem parcial de transações um usuário especialista pode inferir regras de negócio e/ou pequenos fluxos de trabalho executados em um sistema legado.

Paradauskas e Laurikaitis (2006) buscam padrões de negócio através da análise de metadados de um banco de dados (dicionário de dados). Os autores mostram que através da análise do dicionário de dados é possível encontrar recursos usados em um processo (por exemplo, perfis de usuários ou departamentos da organização), possíveis nomes de atividades de negócio, assim como a relação existente entre os recursos e as atividades de um processo.

2.1.3 Código Fonte Legado

A principal fonte de informações sobre um sistema legado é o código fonte. A análise automática do código fonte é chamada de análise estática (ERNST, 2003), em contraste com a análise da execução do sistema que é chamada de análise dinâmica. A análise do código fonte permite extrair as informações mais básicas do sistema, como, rotinas, estruturas de dados, estruturas de controle de fluxo e as relações existentes entre componentes do sistema legado.

As ferramentas que executam a análise estática do código fonte são chamadas de *"parsers"* (ERNST, 2003). Tais ferramentas varrem os arquivos de código fonte, analisando sintaticamente as instruções de uma determinada linguagem de programação, procurando estruturas de negócio que representam alguma informação relevante para o funcionamento do sistema (SNEED; ERDOS, 1996).

Os *"parsers"* são construídos para uma determinada linguagem de programação, e normalmente, buscam um tipo particular de informação sobre o negócio da organização. Por exemplo, pode-se construir um "*parser*" para buscar instruções *"if-then-else",* que usam sentenças condicionais ("*booleanas*") e expressam uma tomada de decisão no negócio da organização.

As técnicas de análise estática são as mais estudadas e empregadas na extração de conhecimento de sistemas legados. Isto porque, muitas vezes os sistemas legados não possuem documentação atualizada e foram desenvolvidos em tecnologias obsoletas, que não possuem registros de execução (logs de execução).

Os trabalhos de Biggerstaff (1989), Corbi (1989), Rich e Wills (1990) e Hartman (1991) foram pioneiros na utilização das técnicas de análise estática para extração de informações em sistemas legados. Estes trabalhos apresentam soluções para recuperar informações no nível de programação, ou seja, informações referentes à codificação do sistema, como, estruturas de dados, parametrização de rotinas, árvores de chamadas de rotinas, entre outras informações importantes para o entendimento do código fonte. O

objetivo destes trabalhos é obter informações relevantes para a reescrita do sistema legado em outras tecnologias.

Karakostas (1990) introduziu uma teoria para ligar as estruturas do código fonte com o modelo conceitual da organização. Karakostas (1990) mostra que as estruturas de código fonte representam informações sobre o negócio da organização, ou seja, as estruturas representam tanto, conhecimentos sobre a codificação do sistema, quanto, conhecimentos sobre o comportamento do negócio da organização. São informações relacionadas ao nível conceitual do sistema de informação, mais próximas do conhecimento humano e não relacionadas ao nível de programação do código fonte.

Karakostas (1990) chama estas estruturas de regras de negócio. Uma regra de negócio é uma declaração que define um comportamento do sistema de informação e uma política organizacional.

Por exemplo, a declaração – *"Se o valor da compra for maior que 3.000 reais, então o sistema deve calcular um desconto de 10%"* – é uma regra de negócio que define a política de descontos utilizada por uma organização. Segundo Karakostas (1990), durante a codificação do sistema de informação, os desenvolvedores usam estas declarações para escrever o código fonte deste sistema. A Figura 2.1 ilustra um possível trecho de código fonte para implementar a declaração mencionada neste exemplo.

Figura 2.1: Trecho de código fonte que implementa uma regra de negócio.

```
if (total_compra > 3000) {
   desconto = total_compra * 0.1;
}
```

Fonte: elaborado pelo autor.

A partir do trabalho de Karakostas (1990), outros autores passaram a definir ferramentas para identificar regras de negócio implementadas no código fonte de sistemas legados (HAUSLER et. al., 1990; BREUER; LANO, 1991; KARAKOSTAS, 1992). Basicamente, as ferramentas buscam identificar expressões aritméticas e comandos condicionais implementados no código fonte. Todos os trabalhos realizam a identificação automática das regras de negócio, usando técnicas de análise estática.

Em Sneed e Erdos (1996), os autores demonstram que a extração de regras de negócio também deve levar em consideração o conhecimento dos usuários que usam o sistema de informação. A identificação automática das estruturas nem sempre é eficaz, pois nem todos os comandos condicionais ou expressões aritméticas representam informações relevantes para o negócio da organização.

Sneed e Erdos (1996) propõem uma ferramenta para identificação de regras de negócio conhecidas pelos usuários do sistema de informação. Os usuários devem informar à ferramenta quais as saídas do sistema, ou seja, quais são os resultados finais produzidos pelo sistema de informação após um determinado processamento. Através destas informações, a ferramenta busca expressões aritméticas e comandos condicionais que de alguma forma foram utilizados pelo sistema para obter as saídas especificadas pelo usuário.

Huang et. al. (1996) e Wang et. al. (2004) também apresentam propostas para extração de regras de negócio que usam o conhecimento dos usuários do sistema para

aprimorar os resultados da extração. Entretanto, nestes trabalhos, as ferramentas identificam automaticamente as informações de entrada e saída do sistema. Os usuários não precisam indicar as entradas e saídas do sistema, como proposto em Sneed e Erdos (1996). Em Huang et. al. (1960) e Wang et. al. (2004) os usuários do sistema validam os resultados obtidos a partir da identificação automática das regras de negócio. Isto é, os usuários validam se as entradas e saídas foram identificadas corretamente, e se as regras extraídas são confiáveis. A partir das respostas do usuário, a ferramenta se adapta e melhora os resultados apresentados em uma próxima execução da análise estática.

Além dos trabalhos para extração de regras de negócio, pesquisas mais recentes, como as de Zou e Hung (2006) e Pérez-Castillo et. al. (2009), apresentam técnicas de análise estática para identificar trechos de código fonte que representam partes de um processo de negócio executado por um sistema legado. Nestes trabalhos, o foco da análise estática passa para a identificação de modelos de processos de negócio. Geralmente, processos de negócio implementam as regras de negócio (CHARFI; MEZINI, 2004; LEE et. al., 2007). Desta forma, ao identificar trechos de processos de negócio, também são encontradas as regras de negócio codificadas no sistema legado.

Zou e Hung (2006) propõem um framework para recuperar fluxos de trabalho implementados no código fonte de sistemas legados. O framework analisa estaticamente o código, procurando por estruturas de workflow pré-especificadas por um usuário especialista do sistema legado. Um usuário especialista apresenta um desenho do processo que deve ser identificado no código fonte e a ferramenta procura trechos de código fonte que podem implementar a estrutura de processo pré-definida pelo usuário. Desta forma, o trabalho procura por fragmentos de processos de negócio que já são conhecidos na organização. Ou seja, um usuário especialista especifica as estruturas que devem ser encontradas no código fonte, pois são processos de negócio conhecidos pela organização.

Já Pérez-Castillo et. al. (2009) propõem uma ferramenta de análise estática, que usa padrões de negócio para descobrir fragmentos de processos codificados em sistemas legados. Nesta ferramenta, o código fonte é mapeado para modelos KDM (*Knowledge Discovery Metamodel*), que posteriormente são analisados para descobrir padrões de negócio que podem ser modelados com BPMN (*Business Process Modeling Notation*). Por exemplo, ao analisar um modelo KDM a ferramenta pode transformar comandos condicionais (ex.: *if-the-else*) em gateways (ex.: *XOR-Split*) da notação BPMN.

A ferramenta proposta Pérez-Castillo et. al. (2009) descobre até 10 padrões de negócio, permitindo encontrar pequenos fragmentos de processos. Entretanto, estes fragmentos não têm relação entre si. A modelagem final do processo de negócio é feita por um usuário especialista do sistema legado, que deve conectar os fragmentos a fim de definir os processos de negócio implementados no sistema legado.

2.1.4 Traces de Execução do Sistema Legado

As técnicas de análise estática podem extrair muitas informações sobre o funcionamento de um sistema legado, mas não conseguem extrair todas as informações necessárias para a reescrita do sistema legado. Por exemplo, qual bloco de uma instrução *if-then-else* realmente é percorrido. Isso depende dos dados empregados na execução do sistema. Para descobrir esse tipo de informação, precisamos de uma análise sobre a execução do sistema do legado.

Esse tipo de análise consiste em executar o sistema legado, monitorando os valores atribuídos as variáveis, instruções condicionais executadas, quais funções são chamadas, entre outros aspectos que podem ser monitorados durante a execução do sistema legado.

O monitoramento gera *"traces"* (traços) de execução do sistema legado, que normalmente são registrados em arquivos de *log* criados durante a execução do sistema legado. Isto é, durante a execução do sistema é gerado um arquivo de eventos, em formato de texto, contendo as instruções executadas pelo sistema, assim como, os valores de variáveis usados na execução das instruções. Essa análise é conhecida como análise dinâmica, pois é obtida em tempo de execução do sistema legado.

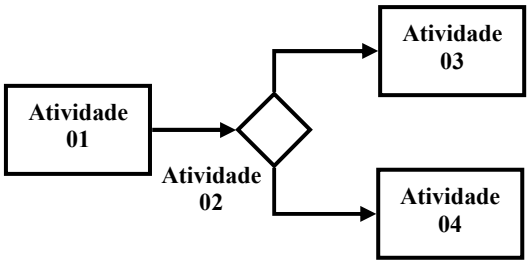
As principais técnicas pautadas na análise dinâmica visam extrair o fluxo de informações executados na organização. Através da análise do arquivo de *log* é possível identificar fluxos de trabalho executados através do sistema legado. A análise do arquivo de *log* para descobrir fluxos de trabalho é conhecida como **mineração de processos** (AALST; REIJERS; WEIJTERS, 2007).

Em Aalst et. al. (2007), Gunter et. al. (2007) e Ingvaldsen et. al. (2008) são apresentadas pesquisas que usam análise dinâmica para extrair fluxos de trabalho de sistemas de informações mais recentes, como por exemplo, Sistemas Gerenciadores de Processos, ERP's ou CRM's. Tais sistemas são capazes de gerar *logs* de eventos, onde cada evento é entendido como uma atividade de negócio. Deste modo, é possível analisar (minerar) o arquivo de *log* e determinar a ordem parcial de execução dos eventos. A ordem parcial dos eventos é entendida como um processo de negócio executado pelo sistema de informação, pois os eventos são vistos como atividades de negócio. A Figura 2.2 apresenta um exemplo de arquivo de *log* e o respectivo resultado da mineração do mesmo.

Figura 2.2: Arquivo de log e resultado da mineração de processos.

```
100301 INICIO ATIVIDADE 01
100301 FIM ATIVIDADE 01
100320 INICIO ATIVIDADE 01
100301 INICIO ATIVIDADE 02
100320 FIM ATIVIDADE 01
100301 INICIO ATIVIDADE 03
100301 FIM ATIVIDADE 03
100320 INICIO ATIVIDADE 02
100301 FIM ATIVIDADE 02
100320 INICIO ATIVIDADE 04
100320 FIM ATIVIDADE 04
100320 FIM ATIVIDADE 02
```

(a) Trecho de *log* com registros dos eventos de inicio e fim das atividades de negócio.



(b) Resultado da mineração do trecho de *log* apresentado em (a).

Fonte: elaborado pelo autor.

A mineração de processos também pode ser usada para identificar processos de negócio implementados em sistemas legados. Cai, Yang e Wang (2009), Di Francescomarino, Marchetto e Tonella (2009) e Pérez-Castillo et. al. (2011) são exemplos de aplicações das técnicas de mineração de processos em sistemas legados.

Cai, Yang e Wang (2009) propõem uma abordagem que combina análise de requisitos com mineração de *logs* para extrair processos de negócio que são disparados por atores do sistema legado. Primeiramente, são coletados casos de uso do sistema através de entrevistas com usuários do sistema. Posteriormente, os casos de uso são executados no sistema legado, gerando eventos em um arquivo de *log*. Os eventos representam execuções dos casos de uso definidos nas entrevistas com usuários. O próximo passo é minerar o arquivo de *log*, determinando a ordem parcial de execução dos casos de uso. Essa ordem é entendida como um processo de negócio executado no sistema legado.

É importante ressaltar que a técnica proposta por Cai, Yang e Wang (2009), considera que o sistema legado é capaz de gerar arquivos de *log* com os eventos criados pelos casos de uso. A técnica não pode ser aplicada para sistemas legados que não foram preparados para gerar arquivos de *log*.

Outra técnica de análise dinâmica para recuperação de processos de negócio foi apresentada por Di Francescomarino, Marchetto e Tonella (2009). Este trabalho permite recuperar modelos de processos de negócio de aplicações web. Para isso, os autores mineram arquivos de *logs*, onde cada evento representa o acesso a um formulário eletrônico de uma aplicação web.

Geralmente, servidores de aplicações web registram arquivos de *logs* com eventos de acessos as páginas web de uma aplicação. Desta forma, os autores usam estes arquivos de *logs* (como *logs* de eventos) para minerar a ordem parcial de execução dos formulários de uma aplicação web. Para Di Francescomarino, Marchetto e Tonella (2009) cada formulário é uma atividade humana no processo. Portanto, os autores extraem fluxos de trabalho de uma aplicação web.

Recentemente, Pérez-Castillo et. al. (2010) propõem uma técnica que usa análise estática e dinâmica para descobrir processos de negócio em um sistema legado. A técnica consiste em instrumentalizar o código fonte com linhas de código que serão responsáveis por gerar eventos em um arquivo de *log*.

Primeiramente, a técnica de Pérez-Castillo et. al. (2010) analisa estaticamente o código fonte legado, anotando as chamadas de função e procedimentos encontradas neste código. Estas chamadas são anotadas como possíveis atividades de negócio de um processo. A anotação é uma linha de código responsável por gerar um evento no arquivo de *log* do sistema legado.

Posteriormente, os autores executam o sistema legado anotado, gerando um arquivo de *log* que contém os eventos relativos às chamadas de procedimentos executadas no código legado. Este arquivo é minerado, obtendo a ordem parcial das chamadas de procedimentos e funções. A ordem parcial é considerada um processo de negócio executado pelo sistema legado.

2.2 Técnicas de Identificação de Processos de Negócio em Legados

A seção anterior mostra que ao longo do tempo algumas propostas foram empregadas para extrair processos de negócio de sistemas legados. Nesta seção analisamos criticamente as fragilidades e qualidades das propostas encontradas na literatura. A partir desta análise, propomos uma abordagem própria para identificar processos de negócio implementados em sistemas legados, procurando solucionar as lacunas encontradas nas técnicas propostas na literatura.

As técnicas apresentadas por Ghose, Koliadis e Cheung (2007) e Hasegawa et. al. (2009) baseadas na documentação do sistema legado permitem encontrar fragmentos de processos mais próximos do conhecimento humano. Os artefatos de documentação são escritos com linguagem natural, deste modo, os fragmentos de processos identificados por estas técnicas usam termos relacionados ao negócio da organização. São fragmentos mais próximos do nível de modelagem dos processos de negócio.

Apesar desta vantagem, os artefatos de documentação não são uma boa fonte de informações para identificar processos de negócio em sistemas legados. Isso porque, sistemas legados não possuem artefatos de documentação ou estes não foram atualizados ao longo do tempo. Geralmente, as manutenções em sistemas legados são realizadas diretamente no código fonte, sem repassar as atualizações para os artefatos de documentação. Desta forma, a documentação não reflete o estado atual dos processos de negócio que são executados pelos sistemas legados.

Da mesma forma que a documentação, as técnicas baseadas em banco de dados relacionais, como de Agrawal, Imielinski e Swami (1993), Júnior (2002) e Paradauskas e Laurikaitis (2006), também permitem extrair informações mais próximas do conhecimento humano (nível de modelagem). Entretanto, o conhecimento extraído se refere a pequenos fragmentos de um processo de negócio. Por exemplo, são identificados nomes de atividades de negócio, nome de perfis de usuários, departamentos, estruturas organizacionais e pequenas sequencias de atividades.

Os processos de negócio não podem ser determinados completamente através da análise das bases de dados, pois as atividades de processamento das informações, como expressões aritméticas e condicionais, não são registradas nas tabelas ou *logs* de transações do banco de dados. Normalmente, essas informações são implementadas no código fonte do sistema. Portanto, essas técnicas permitem identificar fluxos de criação das informações na organização. Tais fluxos podem ser utilizados como suporte para a modelagem de processos de negócio, mas não são processos completos, pois necessitam de complementação humana para determinar a forma como as informações são manipuladas (processadas) até a gravação no banco de dados.

Além disso, sistemas legados escritos em tecnologias mais antigas, como COBOL, não usam bases de dados relacionais. Deste modo, as técnicas de Agrawal, Imielinski e Swami (1993), Júnior (2002) e Paradauskas e Laurikaitis (2006) teriam que ser adaptadas para suportar os sistemas de arquivos empregados nestes sistemas. Isto é, as técnicas não poderiam ser aplicadas diretamente, pois foram construídas para bases de dados relacionais.

As técnicas de análise estática, baseadas no código fonte, são as técnicas mais usadas e pesquisadas na literatura. Isto porque, em muitos sistemas legados o código fonte é o artefato de software mais atualizado, com as informações mais atuais e completas dos processos de negócio executados nestes sistemas.

Biggerstaff (1989), Corbi (1989), Rich e Wills (1990) e Hartman (1991) mostram que as técnicas de análise estática podem ser usadas para identificar pequenos componentes de um processo de negócio, como por exemplo, perfis de usuários, itens de trabalhos manipulados pelas atividades de negócio (estruturas de dados manipuladas pelo sistema legado), tomadas de decisões (*if-then-else* podem ser *gateways* em processos de negócio), entre outros componentes.

Já os trabalhos de Karakostas (1990), Hausler et. al. (1990), Breuer e Lano (1991), Sneed e Erdos (1996), Huang et. al. (1996) e Wang et. al. (2004) propõem a identificação de trechos de código fonte que representam a implementação de regras de negócios usadas em sistemas legados. As regras de negócio são trechos de código que representam a execução de uma tarefa ou tomada de decisão no negócio da organização. Tais trechos de código, podem ser entendidos como atividades de negócio executadas pelos processos implementados no sistema legado. No decorrer deste trabalho a relação entre regras de negócio e atividades de negócio ficará mais evidente.

As técnicas de análise estática do código fonte podem identificar grande parte dos componentes usados para modelar e implementar um processo de negócio. Os mesmos componentes identificados através das técnicas baseadas em documentação e bases de dados, também podem ser identificados nas técnicas de análise estática do código fonte.

A desvantagem da análise estática é que os componentes identificados são mais próximos do nível de implementação dos processos de negócio. Ou seja, são componentes descritos a partir de uma linguagem de programação empregada na codificação do sistema legado, ao contrário do que ocorre nas técnicas baseadas na documentação e bases de dados, onde os componentes são descritos a partir da linguagem natural (ex.: português, inglês), mais próxima do nível de modelagem dos processos de negócio.

Além das técnicas de análise estática para identificação de componentes dos processos, os trabalhos de Zou e Hung (2006) e Pérez-Castillo et. al. (2009) propõem a identificação de fragmentos de processos a partir do código fonte. Estes fragmentos são estruturas que definem um pequeno fluxo de trabalho utilizado na organização. Entretanto, é importante ressaltar que estas pesquisas identificam fragmentos de processos que não tem relação entre si. Isto é, a modelagem final dos processos de negócio é realizada por um usuário especialista, que conecta tais fragmentos formando um modelo de processo mais completo.

Portanto, da mesma forma que as técnicas baseadas em banco de dados, a análise estática do código fonte não permite identificar a ordem parcial das atividades que compõem os processos de negócio implementados no sistema legado. Estas técnicas também são usadas como suporte na modelagem de processos, onde os fluxos de trabalho são complementados através da intervenção humana.

A identificação da ordem parcial das atividades é a principal vantagem das técnicas de análise dinâmica. Através da análise dos *"traces"* de execução do sistema legado, como apresentado na Seção 2.1.4, é possível identificar a ordem que as tarefas foram executadas, determinando assim os modelos de processos de negócio implementados nos sistemas legados.

Aalst, Reijers e Weijters (2007) e Cai, Yang e Wang (2009) propõem técnicas de descoberta de processos a partir da mineração de arquivos de *logs* gerados pelos sistemas de informação. Cada evento registrado no *log* representa a execução de uma atividade de negócio dos processos que executam através do sistema de informação. Deste modo, as técnicas percorrem o arquivo de *log* montando a ordem parcial que os eventos ocorrem no arquivo.

Entretanto, estes trabalhos consideram que os sistemas geram automaticamente um arquivo de *log* de eventos. Assim, sistemas legados que não foram preparados para gerar este tipo de arquivo, não são beneficiados pelas técnicas proposta por Aalst,

Reijers e Weijters (2007) e Cai, Yang e Wang (2009). Isto é, a maior parte dos sistemas legados não são beneficiados por estas técnicas, pois não foram codificados para gerar arquivos de *log* de eventos.

No trabalho de Di Francescomarino, Marchetto e Tonella (2009) este problema é minimizado, pois os autores propõe a mineração de arquivos de *logs* gerados por servidores web. Estes servidores registram em arquivos de *log*, as páginas que são acessadas pelos usuários durante a navegação de uma aplicação web. Desta forma, é possível usar este arquivo como um registro de eventos. A desvantagem está no fato da técnica ser usada apenas em aplicações web. Os sistemas legados que não foram implementados no paradigma web não são beneficiados pelo trabalho de Di Francescomarino, Marchetto e Tonella (2009). Além disso, o trabalho identifica apenas as atividades humanas de um processo de negócios (páginas acessadas). As atividades automáticas são descartadas do processo de negócio (persistência em banco de dados, cálculos matemáticos, entre outras).

Pérez-Castillo et. al. (2010) propõem uma técnica hibrida, que usa análise estática e dinâmica para descobrir processos de negócio em sistemas legados. Através da análise estática os autores instrumentalizam o código fonte para gerar um arquivo de *log*. Após a instrumentação, o sistema legado passa a gerar o *log* de eventos, que pode então ser minerado através das técnicas de análise dinâmica.

Neste trabalho os autores resolvem parcialmente o problema dos sistemas legados não gerarem os arquivos de *logs*. A fragilidade do trabalho de Pérez-Castillo et. al. (2010) encontra-se na instrumentação do código fonte para geração do *log* de eventos. Os autores propõem a instrumentação das chamadas de procedimentos[2] como atividades de negócio. Cada chamada de procedimento identificada no código fonte é instrumentalizada para gerar um evento de negócio no arquivo de *log*. Sendo assim, cada chamada de procedimento será considerada uma atividade de negócio dos processos codificados no sistema legado.

Entretanto, uma chamada de procedimento não representa uma atividade de negócio. Não há garantias que um procedimento codifica uma atividade de negócio. Um procedimento pode implementar uma ou mais atividades de negócio, ou ainda não implementar nenhuma atividade de negócio. Além disso, alguns sistemas legados não são estruturados em subrotinas e não poderiam ser instrumentalizados pela técnica de Pérez-Castillo et. al. (2010).

Outra desvantagem deste trabalho é a não identificação da semântica das atividades de negócio. Ou seja, a instrumentação não identifica o tipo de processamento que é executado no procedimento. Assim, não é possível determinar o tipo de atividade de negócio que é executado no procedimento. Por exemplo, não é possível determinar se a atividade é humana ou automática (cálculo aritmético), se possui tratamento de exceção, qual o item de trabalho manipulado na atividade, entre outras informações relevantes para modelar um processo de negócio.

Desta forma, a ordem parcial encontrada por Pérez-Castillo et. al. (2010) não representa necessariamente um processo de negócio, mas sim, a ordem das chamadas de

[2] Chamadas de procedimentos: invocações a subrotinas do código fonte, como funções, métodos e *procedures*.

procedimentos existentes no sistema legado. Novamente o modelo extraído através desta técnica terá que ser complementado por um usuário especialista no sistema.

A Tabela 2.1 sumariza as vantagens e desvantagens de cada técnica encontrada na literatura para identificação de componentes ou fragmentos de processos de negócio executados em sistemas legados.

Tabela 2.1: Técnicas empregadas na identificação de processos de negócio.

Técnicas	Vantagem	Desvantagem
Baseadas na Documentação	Componentes mais próximos do conhecimento humano, ou seja, mais próximos do nível de modelagem dos processos.	Sistemas legados não possuem documentação ou a documentação encontra-se desatualizada.
Baseadas em Bases de Dados	Componentes mais próximos do conhecimento humano, ou seja, mais próximos do nível de modelagem dos processos.	Não é possível determinar atividades de processamento das informações (ex.: cálculos). Aplicadas apenas para sistemas legados que usam bases de dados relacionais.
Baseadas no Código Fonte (Análise Estática)	Código fonte é o artefato mais atualizado do sistema legado. Identifica componentes e pequenos fragmentos de processos de negócio.	Componentes mais próximos do nível de programação, ou seja, do nível de implementação dos processos. Não identifica a ordem parcial das atividades de um processo de negócio (apenas pequenos fragmentos).
Baseadas no *Trace* de Execução (Análise Dinâmica)	Identifica a ordem parcial das atividades de um processo de negócio.	Sistemas legados normalmente não registram *logs* de eventos.

Fonte: elaborado pelo autor.

2.3 Instrumentação do Código Fonte

A análise critica da literatura mostra que a identificação de processos de negócios implementados em sistemas legados ainda é um desafio na engenharia de software. Atualmente, não existe uma solução ideal para identificar processos de negócio codificados em sistemas legados. Existem diversas abordagens, com vantagens e desvantagens, que podem ser utilizadas para suportar o trabalho de identificação e modelagem de processos de negócio codificados em sistemas legados.

A análise sumarizada na Tabela 2.1, mostra que técnicas baseadas na análise da documentação e de bases de dados não são empregadas para todos os sistemas legados. Os artefatos de documentação, quando existentes, normalmente não estão atualizados e não refletem o estado atual do sistema legado. Já as bases de dados de sistema mais antigos não são relacionais. Isso impede o uso das técnicas baseadas em bases de dados

nestes sistemas. Além disso, as técnicas baseadas em banco de dados não permitem a descoberta das atividades de processamento das informações.

Desta forma, assim como Pérez-Castillo et. al. (2010), esta tese também defende que é possível construir uma solução próxima do ideal a partir da combinação das técnicas de análise estática e dinâmica. A análise estática pode ser usada para determinar os trechos de código fonte que implementam atividades de negócio (inclusive atividades de gravação de informações em um banco de dados). Enquanto a análise dinâmica pode ser usada para determinar a ordem parcial de execução destas atividades de negócio.

Entretanto, é necessário definir uma solução de instrumentação do código fonte mais eficiente que a abordagem proposta por Pérez-Castillo et. al. (2010). Ou seja, uma solução capaz de instrumentalizar trechos de código fonte que realmente representem implementações de atividades de negócio, com semântica para o negócio da organização.

Portanto, para iniciar a definição da solução é necessário verificar como se dá a implementação de uma atividade de negócio no código fonte legado. Quais as estruturas sintáticas de uma linguagem de programação são usadas para codificar atividades de negócio? Qual a melhor estratégia para identificar o conceito de atividade de negócio dentro do código fonte legado?

2.3.1 Conceito de Atividade de Negócio

Para responder estes questionamentos, revisamos o conceito de atividade de negócio definido na notação 2.0 da Linguagem de Modelagem de Processos de Negócio – BPMN (OMG, 2011).

Uma atividade de negócio (ou tarefa) é uma unidade de processamento, um passo lógico dentro do processo de negócio (OMG, 2011). A atividade de negócio deve executar um processamento atômico, ou seja, uma unidade de trabalho indivisível, um passo lógico indivisível, que executa um processamento que não pode ser interrompido (OMG, 2011).

Existem diferentes tipos de atividades de negócio: atividade humana (*human task*); atividade de serviço (*service task*); atividade manual (*manual task*); atividade de envio de informações (*send task*); entre outras (OMG, 2011). Cada tipo representa um comportamento diferente dentro do negócio da organização. Por exemplo, um cálculo aritmético pode ser representado no processo por uma atividade de serviço (*service task*), enquanto uma aprovação orçamentária pode ser representada por uma atividade humana (*human task*).

Normalmente, as atividades manipulam itens de trabalho (WfMC, 1999). Um item de trabalho é a estrutura de dados que é manipulada durante o processamento da atividade, a fim de gerar um determinado resultado para o negócio (WfMC, 1999).

Por exemplo, o cálculo do valor total de uma nota fiscal pode ser representado como uma atividade de negócio em um processo de vendas. Neste caso, a atividade deve possuir um item de trabalho (estrutura de dados) formado pelas informações de quantidades e preços dos produtos contidos na nota fiscal. Além disso, o item de trabalho também deve ter o valor total que é calculado durante a execução da atividade.

Uma atividade de negócio também usa um recurso organizacional, isto é, um recurso responsável pela execução da atividade no processo de negócio. Este recurso pode ser humano ou uma máquina (para atividades automáticas). No exemplo do cálculo do

valor total de uma nota fiscal, a atividade é executada automaticamente por uma máquina, que neste exemplo, é o recurso responsável pela execução das operações aritméticas para encontrar o valor total da nota fiscal.

Portanto, analisando o conceito de atividade de negócio definido pelo OMG (2011), nós podemos afirmar que uma técnica de análise estática para nossa solução deve identificar trechos de código fonte com, pelo menos, as seguintes características:

- O código deve representar uma unidade de processamento indivisível (atômica);
- Deve manipular estruturas de dados (variáveis) que representam itens de trabalho relacionados ao negócio da organização;
- Deve executar um processamento relacionado a um dos tipos de atividades de negócio existentes na modelagem de processos (deve ter um significado, uma semântica para o negócio da organização);
- O processamento deve ser executado por um recurso organizacional, ou seja, por um usuário do sistema ou automaticamente pela máquina.

2.3.2 Análise Estática e as Regras de Negócio

Conforme apresentado na Seção 2.1.3, alguns autores como, Karakostas (1990), Sneed e Erdos (1996) e Wang et. al. (2004) propõem técnicas de análise estática para identificar trechos de código fonte com características semelhantes às descritas para o conceito de atividades de negócio. Nestes trabalhos são identificadas sentenças que representam **regras de negócio** executadas no sistema legado.

Segundo Karakostas (1990) as regras de negócio são declarações utilizadas para especificar o comportamento de um sistema de informação. Durante a codificação do sistema, os desenvolvedores usam estas declarações para escrever o código fonte (cf. Seção 2.1.3). Portanto, pode-se dizer que o comportamento de um sistema de informação é determinado pelas regras de negócio que este implementa e estão programadas em seu código fonte (KARAKOSTAS, 1990; SNEED; ERDOS, 1996; WANG et. al., 2004).

Nos últimos anos o conceito de regras de negócio vem sendo aperfeiçoado e utilizado com maior frequência por engenheiros de software. Recentemente, alguns autores como Kolber et. al. (2000), Wang et. al. (2004) e Bajec e Krisper (2005) definem as regras de negócio através de um conjunto de características semelhantes ao conceito de atividades de negócio. Para estes autores as regras de negócio codificadas em um sistema de informação possuem as seguintes características:

- São declarações atômicas, que ao serem decompostas perdem seu significado para o negócio (perdem semântica). Por exemplo, o cálculo de um imposto pode ser descrito com várias expressões aritméticas, entretanto, tais expressões isoladas não fazem sentido para o negócio. É a união das expressões que formam a regra de negócio;
- As regras de negócio são categorizadas, isto é, são divididas em classes para padronizar a codificação do sistema de informação. Cada classe representa um determinado comportamento do sistema. Por exemplo, uma tomada de decisão, um cálculo aritmético, uma interação com usuário, entre outros comportamentos presentes em um sistema de informação;

- As regras de negócio trabalham com conceitos de negócio (termos do negócio), que são traduzidos para estruturas de dados (variáveis) no momento da codificação do sistema de informação;
- Uma regra de negócio está relacionada a um ator e/ou um recurso organizacional. Este recurso determina quem executa a regra de negócio na organização.
- Uma regra de negócio está relacionada a um componente físico do sistema de informação. A regra deve ser implementada em um dos componentes do sistema de informação (ex.: banco de dados, código fonte, entre outros).

Note que as características das regras de negócio definidas por Kolber et. al. (2000), Wang et. al. (2004) e Bajec e Krisper (2005) são semelhantes às características do conceito de atividade de negócio definido pelo OMG (2011) e WfMC (1999).

Tanto as regras de negócio em sistemas de informação, quanto às atividades de negócio na modelagem de processos, são implementadas com atomicidade de processamento, manipulam estruturas de dados, usam recursos organizacionais e podem ser categorizadas de acordo com comportamentos pré-definidos para o negócio da organização.

2.3.3 Relação entre Atividades de Negócio e Regras de Negócio

A relação entre atividades e regras de negócio pode ser encontrada em outras pesquisas científicas.

Knolmayer, Endl e Pfahrer (2000) defendem o uso de regras de negócio para especificar processos de negócio. Para estes autores os padrões de workflow que devem ser empregados em uma organização, podem ser identificados e especificados a partir das regras de negócio definidas para esta organização.

Mais tarde, Charfi e Mezini (2004) e Lee et. al. (2007) também demonstram que processos de negócio podem ser modelados e implementados através de regras de negócio. Em Charfi e Mezini (2004) as regras de negócio são usadas para especificar o controle de fluxo (ex.: tomadas de decisões) em linguagens de composição de *web services* (BPEL). Já em Lee et. al. (2007) as atividades de negócio são especificadas através de regras de negócio construídas com regras ECA (*Event-Control-Action*).

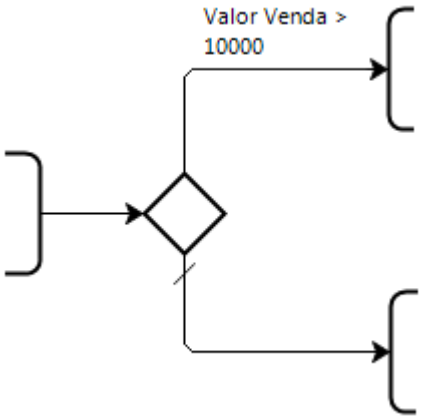
Processos de negócio modelados com regras de negócio são mais flexíveis, pois permitem a utilização de Sistemas Gerenciadores de Regras de Negócio (BRMS). Atualmente, muitos Sistemas Gerenciadores de Processos de Negócio suportam e são associados a Sistemas Gerenciadores de Regras de Negócio (STINEMAN, 2010).

Sistemas Gerenciadores de Regras de Negócio são repositórios que armazenam todas as regras de negócio da organização, permitindo que políticas organizacionais e as decisões operacionais sejam definidas, implantadas, monitoradas e mantidas separadamente do modelo de processo de negócio (STINEMAN, 2010). Ao exteriorizar regras de negócio e fornecer ferramentas para gerenciá-las, um BRMS permite que especialistas de negócio definam e mantenham as decisões que guiam o comportamento dos processos, reduzindo tempo e esforços necessários para atualizar processos de negócio que se encontram em execução, ampliando também a capacidade da organização de responder as mudanças no ambiente de negócios (STINEMAN, 2010).

Por exemplo, no processo de negócio apresentado na Figura 2.3, a decisão *"valor venda > 10000"* é uma regra de negócio que pode ser gerenciada em um BRMS. Neste

caso, o teste condicional é retirado do modelo de processo e especificado no BRMS. Desta forma, um analista de negócio pode alterar o teste para *"valor venda > 12000"*, sem a necessidade de parar a execução das instâncias de processo que estão em plena execução no momento da alteração. Isto se deve ao fato de que o teste condicional está especificado no repositório do BRMS e não no modelo de processo executado pelo gerenciador de processos de negócio. Verifica-se ainda que, essa regra poderá ser reutilizada em outros processos de negócio da organização (STINEMAN, 2010).

Figura 2.3: Exemplo de uma regra de negócio definida no modelo do processo.



Fonte: Stineman (2010).

Portanto, as regras de negócio vêm sendo empregadas com sucesso na modelagem de processos de negócio. A Figura 2.3 mostra que o *gateway* modelado no processo de negócio é uma regra de negócio da organização e pode ser implementado através de um BRMS. Atualmente, muitas soluções comerciais de BPM incluem BPMS e BRMS no mesmo pacote comercial, ou seja, são sistemas comercializados de forma integrada (STINEMAN, 2010).

Deste modo, usamos nesta tese o conceito de regras de negócio para identificar atividades de negócio codificadas em sistemas legados. Para o restante da tese, a codificação de uma regra de negócio representa a implementação de uma atividade de negócio dos processos codificados no sistema legado. Ou seja, estabelecemos uma relação de *um-para-um* entre regras de negócio e atividades de negócio.

2.3.4 Solução de Identificação de Processos de Negócio

A partir da relação estabelecida entre regras de negócio e atividades de negócio, esta tese propõe um método que usa tanto análise estática, quanto análise dinâmica para semi-automatizar a descoberta dos processos de negócio implementados implicitamente em sistemas legados.

Nós defendemos que um sistema legado executa seus processos de negócio através da execução das regras de negócio escritas em seu código fonte. Assim, o método deve identificar trechos de código fonte que implementam regras de negócio, usando estes trechos como implementações das atividades de negócio dos respectivos processos executados no sistema legado.

A análise estática é usada para identificar as regras de negócio implementadas no código fonte legado. Ao identificar uma regra de negócio, o trecho de código é delimitado por anotações, que apontam o início e o fim da codificação desta regra de

negócio. As anotações consistem de novas linhas de código que são responsáveis por gerar eventos em um arquivo de *log* do sistema legado. A Figura 2.4 mostra um exemplo de código instrumentalizado para gerar eventos um arquivo de *log*. Na figura os comandos *fprint* são as anotações adicionadas ao código fonte, permitindo o envio de mensagens para um arquivo de *log*. O exemplo mostra um código fonte implementado na linguagem de programação C.

Figura 2.4: Exemplo de instrumentação do código fonte.

```
fprint(logfile, "%d - INICIO ATIVIDADE 01", order_number);
x = find_stock(order_number);
fprint(logfile, "%d – FIM ATIVIDADE 01", order_number);
fprint(logfile, "%d – INICIO ATIVIDADE 02", order_number);
if (x > 100) {
  fprint(logfile, "%d – INICIO ATIVIDADE 03", order_number);
  send_order(order_number);
  fprint(logfile, "%d – FIM ATIVIDADE 03", order_number);
} else {
  fprint(logfile, "%d – INICIO ATIVIDADE 04", order_number);
  cancel_order(order_number);
  fprint(logfile, "%d – FIM ATIVIDADE 04", order_number);

}
fprint(logfile, "%d – FIM ATIVIDADE 02", order_number);
```

Fonte: Elaborado pelo autor.

Logo após a instrumentação do código fonte legado, o sistema é executado várias vezes, gerando um arquivo de *log* que será analisado através de técnicas de análise dinâmica.

Através da mineração do arquivo de *log* será identificada a ordem parcial de execução das regras de negócio, chegando assim, aos processos de negócio que são executados a partir das regras de negócio definidas no sistema legado.

Desta forma, este método pode oportunizar a construção de ferramentas, tanto para extração dos fluxos de trabalho executados na organização (processos de negócio), quanto para extração das políticas organizacionais que regem a empresa (regras de negócio + processos de negócio = políticas organizacionais).

O método apresentado nessa tese se diferencia do método proposto em Pérez-Castillo et. al. (2010) na forma de instrumentação do código fonte. Pérez-Castillo et. al. (2010) propõe a anotação das chamadas de subrotinas como ocorrências de atividades de negócio. Entretanto, como apresentado na Seção 2.2, uma chamada de subrotina não representa a execução de uma atividade de negócio. Não há garantias que uma subrotina implementa um trecho de código relacionado ao negócio da organização, pois não é possível determinar o tipo de processamento executado na subrotina, se os procedimentos possuem ou não uma operação de negócio atômica e quais as estruturas de dados são manipuladas nos procedimentos. Ou seja, não são identificados trechos de código com as características definidas para as atividades de negócio (OMG, 2011).

Em nossa abordagem são identificadas unidades de processamento que representam unidades lógicas de negócio, isto é, identificam-se regras de negócio com semântica bem definida, com significado para o negócio da organização (ex.: cálculo aritmético, tomada de decisão, entre outros comportamentos), que manipulam itens de trabalho e

usam os recursos organizacionais na execução. Nossa proposta permite encontrar processos de negócio mais completos, com detalhes semânticos da implementação de cada atividade de negócio.

Sem estas informações não é possível afirmar que uma ordem parcial extraída do sistema legado é um processo de negócio. Uma ordem parcial sem estas informações está representando somente o fluxograma do algoritmo executado pelo sistema. Os nós do grafo não possuem um significado para o negócio da organização.

Os próximos capítulos desta tese irão detalhar o conceito de regras de negócio e como este conceito foi utilizado para identificar as atividades de negócio codificadas em um sistema legado. São apresentadas as categorias de regras encontradas na literatura especializada, e como estas categorias podem ser usadas para criar uma técnica capaz de identificar e anotar as regras de negócio codificadas em um sistema legado.

É importante ressaltar que esta tese define um método de identificação de processos de negócio, pois se utiliza ordenadamente um conjunto técnicas para atingir um determinado objetivo. De acordo com Galliano (1979), método é um conjunto de etapas, ordenadamente dispostas, a serem vencidas para alcançar um determinado objetivo. Enquanto, técnica é um modo de fazer da forma mais hábil algum tipo de tarefa (GALLIANO, 1979).

Nossa proposta consiste em aplicar um conjunto de etapas para atingir a meta de identificação de processos de negócio. Em cada etapa propomos o uso de diferentes técnicas, como a técnica de análise estática para identificação das atividades de negócio e a técnica de análise dinâmica para identificação da ordem parcial das atividades de negócio. Deste modo, estamos definindo um método de identificação de processos de negócio, onde em cada etapa pode ser empregada uma determinada técnica para automatizar o trabalho.

2.4 Linguagem de Programação

Antes de prosseguir com a definição do método proposto nesta tese, é necessário definir uma linguagem de programação fictícia, que contém os principais comandos de uma linguagem de programação imperativa. A definição desta linguagem é importante para demonstrar o funcionamento da técnica de análise estática usada para instrumentalizar o código fonte legado. Como apresentado na Seção 2.1.4, a análise estática consiste em verificar as estruturas sintáticas da linguagem de programação utilizada na codificação do sistema, inferindo a existência de algum conhecimento importante para o negócio da organização.

A linguagem de programação fictícia definida nesta seção será usada nos demais capítulos deste trabalho. Nossa linguagem de programação fictícia reúne os principais comandos e estruturas sintáticas contidas em uma linguagem de programação imperativa. A adoção de uma linguagem de programação fictícia permite definir uma solução que pode ser usada para qualquer linguagem de programação imperativa.

2.4.1 Gramática

Plotkin (1977) define uma linguagem de programação fictícia que utiliza um subconjunto dos principais comandos encontrados em uma linguagem de programação imperativa. A Figura 2.5 mostra a gramática da linguagem de programação definida para este trabalho, que tem como base a gramática definida por Plotkin (1977).

Na gramática da Figura 2.5 usa-se *"e, e_1, e_2"*, para representar expressões aritméticas, assim como, *"b, b_1, b_2"* e *"c, c_1, c_2"*, para representar expressões booleanas e comandos, respectivamente. Ainda usa-se *"n"* para números naturais, *"s"* para strings, *"v"* para variáveis, *"proc"* e *"func"* para identificadores de procedimentos e funções, além de *"query"* para representar uma sentença SQL válida. Isto é, *"query"* pode ser uma sentença de inserção, atualização, exclusão ou seleção da linguagem estruturada de consulta, usada para acessar tabelas em banco de dados.

Figura 2.5: Gramática da linguagem de programação.

```
e ::= n | s | v | e1 + e2 | e1 - e2 | e1 * e2 | e1 / e2

b ::= true | false | e1 = e2 | e1 > e2 | e1 < e2 |
      b1 and b2 | b1 or b2

c ::= v := e | c1;c2 | call proc | v := call func |
      read(v) | print(e) | if b then c end |
      if b then c1 else c2 end | while b do c end |
      begin sql c end sql | exec sql query
```

Fonte: Elaborado pelo autor.

Em relação à linguagem de programação definida por Plotkin (1977), a gramática apresentada na Figura 2.5 adiciona os comandos, *"**begin sql** c **end sql**"*, *"***exec sql** *query"*, *"**read(v)**"* e *"**print(e)**"*. Estes comandos são acrescentados para permitir a caracterização de algumas classes de regras de negócio que serão apresentadas nos próximos capítulos da tese.

O bloco *"**begin sql** c **end sql**"* representa a execução de uma transação de banco de dados, ou seja, início e término de uma conexão com o banco de dados. Já o comando *"***exec sql** *query"* executa uma sentença SQL no banco de dados. Por fim, os comandos *"**read(v)**"* e *"**print(e)**"* representam comandos de entrada e saída de dados, respectivamente. São comandos necessários para representar a interação dos usuários com o sistema legado.

2.4.2 Semântica

O comportamento dos comandos da linguagem de programação apresentada na Figura 2.5 é definido através de grafos conceituais. Posteriormente, os grafos conceituais de cada comando são usados para representar o comportamento das classes de regras de negócio, além de serem utilizados na técnica de análise estática para identificar as regras de negócio codificadas no sistema legado.

Os grafos conceituais foram propostos por Sowa (2004) para representar um determinado significado em um formato logicamente preciso, humanamente legível e computacionalmente interpretável.

A Figura 2.6 apresenta um exemplo de grafo conceitual que representa a sentença: "*André estuda informática*". Os grafos conceituais são grafos formados por dois tipos de nós: **Conceitos** e **Relações**. Os conceitos são dispostos dentro dos retângulos e ligados através de outro tipo de nó, as relações. Na Figura 2.6, os conceitos são "*André*"

e "*Informática*". As relações são dispostas dentro de losangos e ligam os conceitos por todos os lados que eles podem ser ligados. Na Figura 2.6 a relação é o nó "*Estuda*". Em grafos conceituais só é permitida a ligação entre conceitos e relações, ou seja, não é possível ligar conceitos a conceitos.

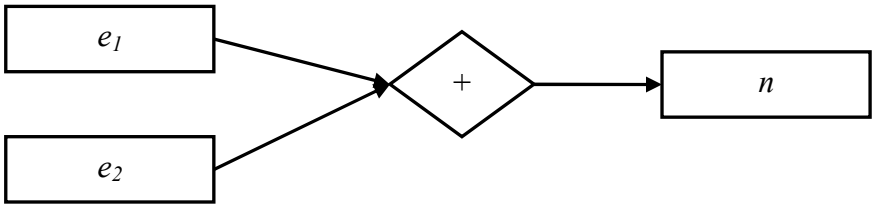
Figura 2.6: Exemplo de um grafo conceitual.

Fonte: Elaborado pelo autor.

A definição do comportamento para a linguagem de programação inicia com a definição dos conceitos que serão usados para construção dos grafos conceituais. Os elementos sintáticos que representam comandos (*"c"*), expressões numéricas (*"e"*), expressões booleanas (*"b"*), valores verdadeiro e falso (***"true"*** *e* ***"false"***), números (*"n"*), strings (*"s"*), variáveis (*"v"*), identificadores de procedimentos e funções (*"proc"* e *"func"*) e sentenças SQL (*"query"*) são conceitos da linguagem de programação. Assim, estes elementos sintáticos da linguagem de programação serão expressos por retângulos nos grafos conceituais que irão determinar o comportamento de nossa linguagem de programação fictícia.

Nas expressões aritméticas *"e_1 + e_2"*, *"e_1 - e_2"*, *"e_1 * e_2"*, *"e_1 / e_2"*, as expressões *"e_1"* e *"e_2"* são conceitos, enquanto os operadores aritméticos (*"+"*, *"-"*, *"*"* e *"/"*) são as relações entre estes conceitos, que irá levar a um novo conceito, um número (*"n"*), como mostra a Figura 2.7.

Figura 2.7: Grafo conceitual para a expressão *"e_1 + e_2"*.



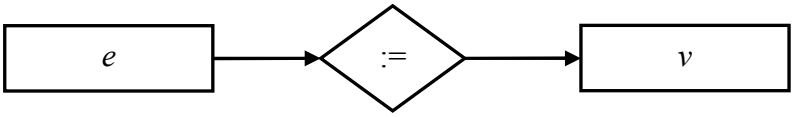
Fonte: Elaborado pelo autor.

Note que as expressões *"e_1"* e *"e_2"* são conceitos que podem ser expandidos para outros subgrafos, desde que, *"e_1"* ou *"e_2"* representem outras expressões aritméticas. A expansão destes conceitos deve ser realizada até que as expressões alcancem elementos sintáticos finais, ou seja, variáveis *"v"*, números *"n"* ou strings (*"s"*). Observe também que o operador aritmético de adição (*"+"*) é uma relação entre os conceitos, que gera um novo número, isto é, um novo conceito (*"n"*).

As demais expressões aritméticas possuem grafos semelhantes, trocando apenas o operador aritmético. Além disso, as expressões booleanas (*"$b_1 = b_2$"*, *"$b_1 > b_2$"*, *"$b_1 < b_2$"*, *"b_1 and b_2"*, *"b_1 or b_2"*) seguem a mesma lógica para definição dos grafos conceituais. Isto é, *"b_1"* e *"b_2"* são conceitos relacionados através de um operador condicional (*"="*, *">"*, *"<"*, *"and"* e *"or"*) que irá levar a um novo valor booleano (***"true"*** ou ***"false"***).

Para o comando de atribuição foi definido o grafo conceitual apresentado na Figura 2.8. Da mesma forma que as expressões *"e_1"* e *"e_2"*, na Figura 2.7, a expressão *"e"* também será expandida caso não seja uma variável (*"v"*), número (*"n"*) ou string (*"s"*).

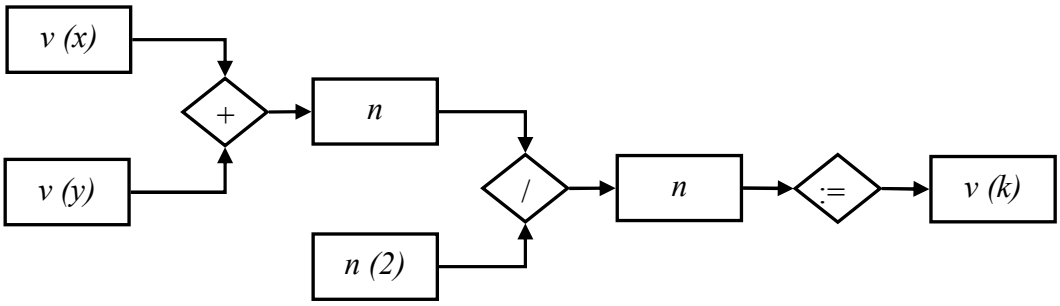
Figura 2.8: Grafo conceitual para o comando de atribuição.



Fonte: Elaborado pelo autor.

Por exemplo, para o comando de atribuição *"k := (x + y)/2"*, onde *"k"*, *"x"* e *"y"* são variáveis, o grafo conceitual resultante é ilustrado na Figura 2.9. Note na figura, que o grafo conceitual é gerado a partir dos conceitos encontrados na expressão, ou seja, variáveis e números (*"v"* e *"n"*). Entretanto, para relacionar a expressão aritmética ao grafo conceitual, adicionamos aos conceitos o identificador das variáveis e o número encontrado na respectiva expressão.

Figura 2.9: Grafo conceitual para o comando de atribuição *"k := (x + y)/2"*.



Fonte: Elaborado pelo autor.

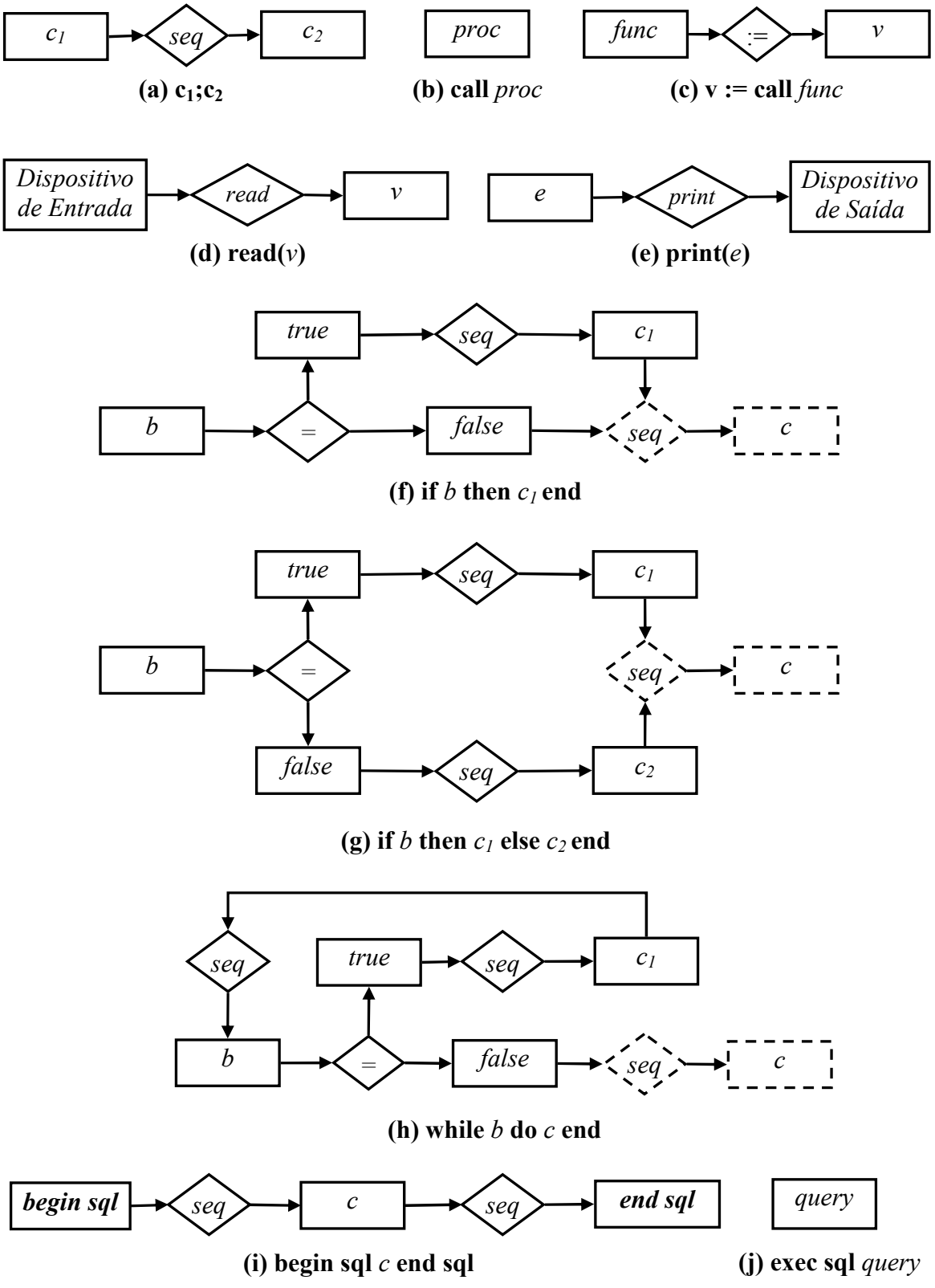
Os grafos conceituais para os demais comandos da linguagem de programação são apresentados na Figura 2.10. Da mesma forma que o comando de atribuição, onde *"e"* pode ser substituído pelo seu respectivo subgrafo, na Figura 2.10, as ocorrências de *"c"*, *"c_1"* ou *"c_2"* podem ser substituídas pelos subgrafos dos seus respectivos comandos. Além disso, observe que a relação *"seq"* representa sequência de comandos ou fluxo de informações.

Na Figura 2.10 (b) e (c) os identificadores de procedimentos e funções são representados por conceitos *"proc"* e *"func"*, respectivamente. A chamada de um procedimento (Figura 2.10 (b)) é definida como um grafo conceitual singular, ou seja, é um conceito individual dentro do sistema, representado pelo identificador do procedimento. Já a chamada de função (Figura 2.10(c)) é uma combinação da chamada de procedimento e do comando de atribuição, pois uma função retorna resultado, ou seja, um novo conceito a ser atribuído a uma variável.

Os grafos conceituais para os comandos de entrada e saída de dados (Figura 2.10(d) e 2.10(e)) usam dois conceitos *"Dispositivo de Entrada"* e *"Dispositivo de Saída"*. Estes conceitos representam dispositivos de I/O manipulados pela linguagem de programação, isto é, teclado, monitor, entre outros dispositivos. Para o comando de entrada de dados, o dispositivo de entrada envia um valor para uma variável, através da

relação de leitura (*"read"*). No comando de saída de dados, uma expressão é enviada para o dispositivo de saída, através da relação de impressão (*"print"*).

Figura 2.10: Grafos conceituais para os comandos da linguagem de programação.



Fonte: Elaborado pelo autor.

Nos comandos condicionais (Figura 2.10(f) e 2.10(g)) e de repetição (Figura 2.10(h)) o grafo conceitual inicia com uma expressão booleana (*"b"*). A avaliação da expressão leva a dois conceitos (*"true"* ou *"false"*), que estão relacionados ao conceito *"b"* através da relação *"="*. De acordo com a avaliação da expressão a sequência de processamento segue (*"seq"*) para o próximo comando (*"c_1"* ou *"c_2"*). Note que a saída dos blocos condicionais e de repetição é uma relação (*"seq"*) e um conceito (*"c"*) tracejados. Isso representa o próximo comando fora do bloco condicional e de repetição.

Normalmente, as linguagens de programação possuem blocos para delimitar uma ação de persistência. Este blocos foram representados em nossa linguagem de programação fictícia pelo comando ***"begin sql-end sql"***. ***"begin sql"*** representa a abertura de uma conexão com o banco de dados ou arquivo, enquanto ***"end sql"*** o fechamento desta conexão. Representa-se o início do bloco e o término do bloco como dois conceitos, obtendo assim, o grafo conceitual apresentado na Figura 2.10(i).

Por último, o comando ***"exec sql"*** representa a execução de uma instrução de persistência. Este comando é usado dentro de um bloco ***"begin sql"*** e é semelhante a chamada de um procedimento (***"call** proc"*). Também é tratado como um conceito individual, ou seja, um grafo conceitual singular representando a sentença SQL executada na ação de persistência (Figura 2.10(j)).

3 REGRAS DE NEGÓCIO

Para desenvolver a solução proposta na Seção 2.3.4 é necessário conhecer o conceito de regras de negócio e como esse conceito é utilizado em sistemas legados. Este capítulo irá detalhar o conceito de regras de negócio, apresentando os tipos de regras existentes na literatura e quais os tipos empregados na codificação de sistemas legados. Através deste estudo foi definida uma ontologia de regras de negócio, que posteriormente é usada como base de conhecimento para uma ferramenta de análise estática identificar as regras de negócio contidas no código fonte legado.

3.1 Regras de Negócio em Sistemas Legados

Como apresentado anteriormente, sistemas de informação abrangem, com frequência, um grande volume de conhecimento organizacional (SOMMERVILLE, 2007). Este conhecimento é especificado e codificado através de regras de negócio (BELL et. al., 1990; KARAKOSTAS, 1990; SNEED; ERDOS, 1996; KOLBER et. al., 2000; ROSS, 2003; WANG et. al., 2004; PUTRYCZ; KARK, 2007).

As regras de negócio foram definidas por Bell et. al. (1990), como sendo, declarações abstratas que descrevem como o negócio está sendo executado ou descrevem as restrições aplicadas a este negócio. Em outras palavras, as regras de negócio definem como um sistema de informação funciona. Elas descrevem as operações, definições e restrições que se aplicam a uma organização. Deste modo, as regras de negócio são declarações que devem ser respeitadas na construção de um sistema de informação (KARAKOSTAS, 1990; ROSS, 1997; KOLBER et. al., 2000).

Normalmente, as regras de negócio são usadas para especificar um sistema de informação. São declarações que guiam os desenvolvedores na escrita do código fonte. Portanto, pode-se dizer que o comportamento de um sistema de informação é determinado pelas regras de negócio que este implementa em seu código fonte (KARAKOSTAS, 1990; SNEED; ERDOS, 1996; WANG et. al., 2004).

O exemplo da Figura 3.1 mostra a declaração de uma regra de negócio que determina o comportamento de um sistema de informação. Durante a codificação do sistema, os desenvolvedores reescrevem a regra de negócio no código fonte usando instruções de uma linguagem de programação.

As regras de negócio podem ser classificadas de diferentes formas (ROSS, 1997; KOLBER et. al., 2000). Cada tipo de regra de negócio é caracterizado por um conjunto de propriedades sintáticas, isto é, palavras ou expressões que são usadas para escrever a regra de negócio em linguagem natural. Estas palavras ou expressões, posteriormente, são traduzidas para comandos de uma linguagem de programação no momento da codificação do sistema de informação (ROSS, 1997).

Figura 3.1: Exemplo de especificação e codificação de uma regra de negócio.

Se o valor da compra ultrapassar mil reais, então, o valor da compra terá desconto de 10%.

(a) Descrição textual de uma regra de negócio.

```
if (valor > 1000) {
  valor -= (valor * 0.1);
}
```

(b) Codificação da regra de negócio na linguagem de programação C.

Fonte: Elaborado pelo autor.

Portanto, para identificar as regras de negócio no código fonte é necessário conhecer os tipos de regras existentes na literatura, quais tipos são empregados na codificação de sistemas legados, quais as propriedades que caracterizam os tipos de regras e como estas propriedades são codificadas em uma linguagem de programação imperativa.

Para descobrir a relação entre regras de negócio e sistemas legados, foram estudados os diferentes esquemas de classificações de regras de negócio encontrados na literatura (ROSS, 1997; KOLBER et. al., 2000; HALLE, 2002; MORGAN, 2002; WEIDEN et. al., 2002; ROSS, 2003; BAJEC; KRISPER, 2005) e como estes esquemas foram usados para codificar cinco sistemas legados selecionados no Portal do Software Público Brasileiro (MPOG, 2011).

A estratégia do estudo consiste em procurar na documentação dos cinco sistemas legados, sentenças textuais com as características dos tipos de regras de negócio definidos nos esquemas de classificação disponíveis na literatura. Primeiramente foram procuradas as sentenças textuais que descrevem regras de negócio destes sistemas. Posteriormente, estas sentenças foram classificadas nos esquemas de regras de negócio definidos por Ross (1997), Kolber et. al. (2000), Halle (2002), Morgan (2002), Weiden et. al. (2002), Ross (2003) e Bajec e Krisper (2005).

Por exemplo, na documentação de um dos sistemas legados foi encontrada a seguinte sentença: *"Para solicitar um benefício o usuário* ***deve*** *ter* ***idade superior a 60 anos****"*. Esta sentença possui uma expressão de obrigação (***"deve"***) e uma expressão condicional (***"idade superior a 60 anos"***), que são características da regra de negócio de restrição, definida no esquema de classificação de Ross (2003). Deste modo, essa sentença foi classificada como uma regra de restrição do esquema de Ross (2003).

Através deste estudo foi possível descobrir os principais tipos de regras de negócio usados nos cinco sistemas legados selecionados no trabalho. Estes tipos de regras foram agrupados em uma ontologia, que posteriormente, pode ser usada como base de conhecimento para uma ferramenta identificar as regras de negócio implementadas no código fonte de um sistema legado.

As classes da ontologia representam os tipos de regras de negócio que foram encontrados nos cinco sistemas legados estudados. Os atributos de uma classe se referem a expressões ou palavras, em linguagem natural, que obrigatoriamente são usadas para especificar aquele determinado tipo de regra de negócio.

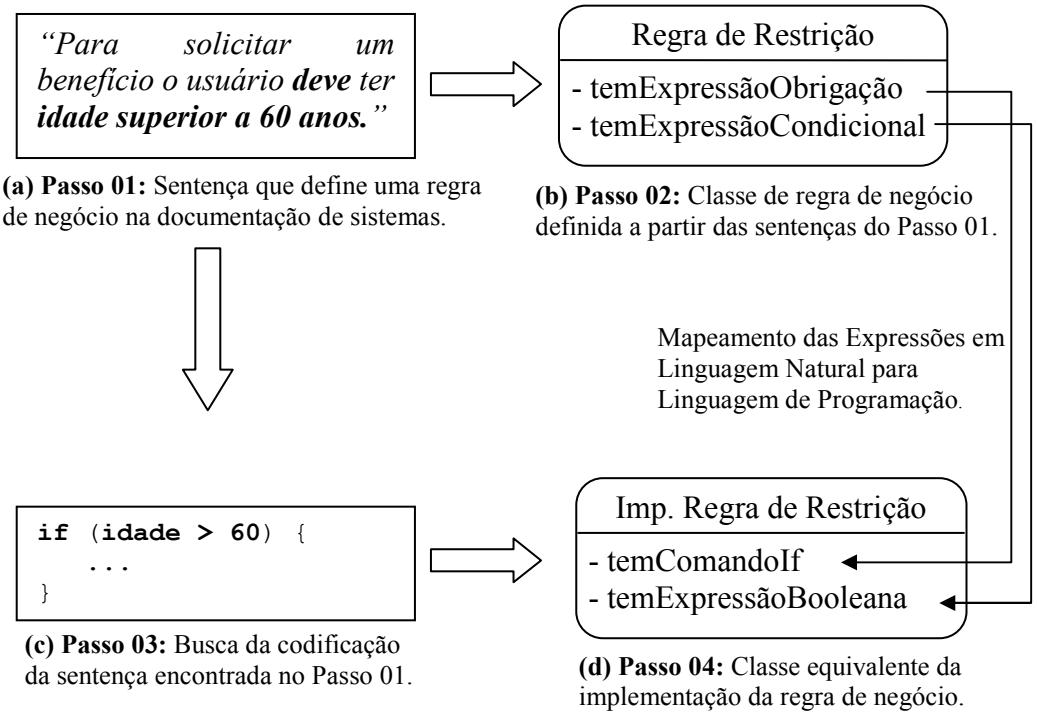
Após a definição da ontologia de regras de negócio, o próximo passo foi verificar como as regras de negócio encontradas nas documentações foram implementadas nos respectivos códigos fontes dos sistemas legados. Desta forma, foi possível observar

quais estruturas sintáticas das linguagens de programação foram empregadas para codificar os tipos de regras de negócio encontrados nestes sistemas legados.

Com base nesse estudo foi definido um conjunto de classes equivalentes para a ontologia de regras de negócio. Para cada classe da ontologia foi definida uma classe equivalente, onde os atributos da classe equivalente se referem a estruturas sintáticas de uma linguagem de programação, que normalmente são usados para codificar um determinado tipo de regra de negócio.

A Figura 3.2 apresenta graficamente os passos usados para a definição da ontologia de regras de negócio e seu conjunto de classes equivalentes. No primeiro passo (Figura 3.2(a)) foram pesquisadas sentenças textuais que especificam regras de negócio nos documentos de especificação dos sistemas legados selecionados para este trabalho. No segundo passo estas sentenças foram classificadas nos esquemas de regras de negócio existentes na literatura. A partir da classificação, para cada tipo de regra foi definida uma classe de regra de negócio na ontologia (Figura 3.2(b)). No terceiro passo (Figura 3.2(c)) foi pesquisado no código fonte dos sistemas legados a implementação das sentenças textuais encontradas no primeiro passo. Por fim, no último passo (Figura 3.2(d)) foram geradas classes equivalentes que representam o modo como as regras de negócio são codificadas nos sistemas legados estudados neste trabalho.

Figura 3.2: Passos para definição da ontologia de regras de negócio.



Fonte: Elaborado pelo autor.

As próximas seções destinam-se a detalhar os passos (a) e (b) da estratégia de estudo apresentada na Figura 3.2. No Capítulo 4 são detalhados os passos (c) e (d). Posteriormente, este estudo é usado para definir uma ferramenta de identificação das regras de negócio implementadas no código fonte legado.

3.2 Ontologia de Regras de Negócio para Sistemas Legados

Como mencionado anteriormente, para descobrir os tipos de regras de negócio que são usados na definição de um sistema legado, foram pesquisadas as documentações de cinco sistemas disponíveis no Portal do Software Público Brasileiro (MPOG, 2011):

- **Fila:** Sistema escrito em C/Java para gerenciamento de filas de atendimentos em agências bancárias;
- **GSAN:** Usado para gerenciar serviços de saneamento, como, ligações de água e esgoto, gerenciamento de pagamentos, gerenciamento de atendimentos aos clientes, entre outros serviços. Esse sistema foi escrito com tecnologia Java;
- **SGA Livre:** Outro sistema para gerenciamento de filas de atendimento. Porém, este sistema foi desenvolvido em plataforma web, usando a linguagem de programação PHP;
- **SGD:** Sistema desenvolvido em plataforma web e codificado com a linguagem de programação PHP. O sistema realiza a gestão de demandas para o Fundo Nacional de Desenvolvimento da Educação;
- **Sistema de Financiamentos:** Por último, foi analisado um sistema para gestão de financiamentos de um banco de desenvolvimento regional. Esse sistema foi escrito com linguagem de programação COBOL.

Os critérios usados para a seleção destes sistemas foram: possuir documentação textual das regras de negócio codificadas no sistema; e permitir acesso ao código fonte, para posterior busca das codificações das regras de negócio.

Como mostra a Figura 3.2(a), nas documentações foram encontradas sentenças textuais que declaram as regras de negócio destes sistemas. Ao todo foram encontradas 417 sentenças textuais que definem as regras de negócio destes sistemas. Estas sentenças foram classificadas nos tipos de regras de negócio encontrados nos esquemas de classificação definidos na literatura (ROSS, 1997; KOLBER et. al., 2000; HALLE, 2002; MORGAN, 2002; WEIDEN et. al., 2002; ROSS, 2003; BAJEC; KRISPER, 2005).

Ao executar a classificação percebe-se que o esquema definido por Ross (2003) é o mais completo. As 417 sentenças foram classificadas em algum tipo de regra de negócio do esquema definido por Ross (2003). Portanto, este esquema foi escolhido para a definição da ontologia de regras de negócio a ser empregada neste trabalho.

A partir da classificação de Ross (2003), foi criada uma ontologia para regras de negócio escritas em linguagem natural e mais usadas na definição de sistemas de informação. Para cada tipo de regra de negócio foi definida uma classe na ontologia, onde as propriedades da classe representam expressões textuais que geralmente são usadas para escrever uma regra de negócio na linguagem natural.

3.2.1 Classe Regra de Negócio

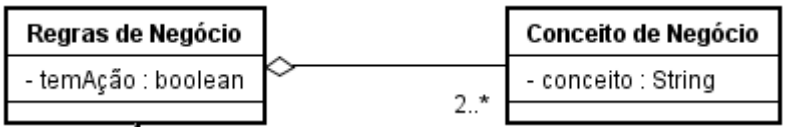
Uma regra de negócio especifica um relacionamento entre conceitos de negócio (ROSS, 2003). Um conceito de negócio é um termo que expressa uma ideia ou noção sobre o negócio da organização. Pode representar um objeto ou uma entidade presente

no negócio. É uma representação geral e abstrata da realidade de um determinado negócio (BELL et. al., 1990).

Os conceitos de negócio são relacionados através de expressões que indicam ações (normalmente verbos). No exemplo "*o cliente compra carros*", existem dois conceitos de negócio, "*cliente*" e "*carros*", que estão relacionados através da expressão de ação "*compra*". Desta forma, é possível afirmar que uma regra de negócio deve ter ao menos dois conceitos de negócio e uma expressão de ação para relacioná-los.

Portanto, a ontologia de regras de negócio inicia com a definição das classes "***Regra de Negócio***" e "***Conceito de Negócio***", como mostra a Figura 3.3. A classe "*Regra de Negócio*" define uma regra genérica, que deve obrigatoriamente possuir dois ou mais conceitos de negócio. O relacionamento com a classe "*Conceito de Negócio*" indica quais são os conceitos de negócio envolvidos na sentença textual, além da obrigatoriedade de existir ao menos dois conceitos de negócio. Já o atributo "*temAção*" indica a necessidade de existir uma expressão de ação relacionando os conceitos.

Figura 3.3: Classes *"Regra de Negócio"* e *"Conceito de Negócio"*.



Fonte: Elaborado pelo autor.

Segundo Ross (2003) as regras de negócio são divididas em seis classes: Regra de Restrição (*constraint*); Regra de Computação; Regra de Derivação; Regra de Habilitação; Regra de Cópia; e Regra de Execução.

A seguir são apresentadas estas classes como especializações da classe geral, "*Regra de Negócio*". Deste modo, define-se que todos os tipos de regras de negócio devem possuir conceitos de negócio e expressões de ação (definição dos atributos da superclasse "*Regra de Negócio*").

3.2.2 Classe Restrição

As regras de restrição especificam condições que protegem o negócio de informações incorretas. Por exemplo, a sentença "*para efetuar uma compra o cliente* ***deve*** *ter* ***crédito maior que o valor da compra****",* define uma restrição para um cliente efetuar uma compra.

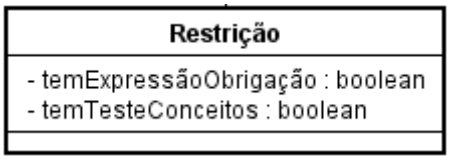
As regras de restrições são escritas usando expressões que indicam obrigação ou compromisso. As expressões, "**deve**", "**pode somente se**", "**não pode**", "**não deve**", entre outras, são expressões de obrigação usadas para escrever uma regra de restrição em linguagem natural (no exemplo anterior usa-se a palavra *"**deve**"*).

Uma regra de restrição também possui uma expressão condicional entre conceitos de negócio. Essa expressão é a condição para que a restrição ocorra (no exemplo anterior a condição é "***crédito maior que valor da compra***").

A Figura 3.4 mostra a classe definida para a ontologia de regras. A classe *"Restrição"* é uma especialização da classe *"Regra de Negócio"*. Além dos atributos herdados, a classe *"Restrição"* tem dois atributos próprios: *"temExpressãoObrigação"*, que indica a obrigatoriedade de existir uma expressão de obrigação (compromisso) na

sentença de uma regra de restrição; e *"temTesteConceitos"*, que indica a obrigatoriedade de existir uma expressão condicional entre conceitos do negócio.

Figura 3.4: Classe para as regras de negócio de restrição.



Fonte: Elaborado pelo autor.

3.2.3 Classe Computação

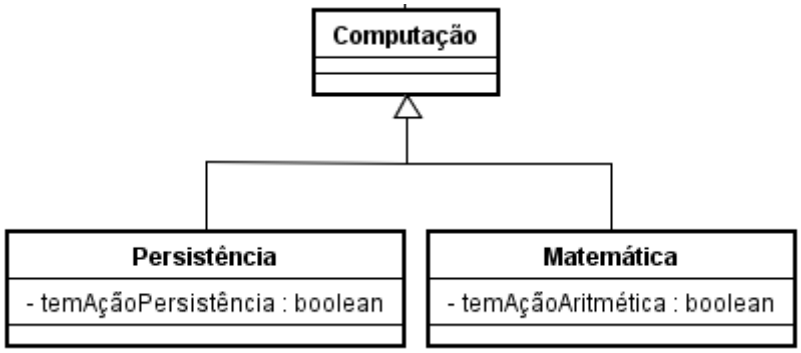
A regra de computação é usada para especificar o processamento de informações. Esta regra é usada para definir operações matemáticas (ex.: soma, multiplicação, média, entre outros cálculos) ou operações de persistência de informações (ex.: acesso/gravação em banco de dados e/ou sistema de arquivos).

A regra de computação permite especificar dois tipos de operações realizadas em sistemas de informação, portanto, a definição de Ross (2003) torna-se ambígua. Para tornar a definição não ambígua, nós especializamos a classe *"Computação"* em *"Persistência"* e *"Matemática"*, como mostra a Figura 3.5.

As regras matemáticas são especificadas usando ações matemáticas, ou seja, somar dois conceitos, calcular a média entre conceitos, realizar o produto de conceitos, calcular o percentual, entre outras operações aritméticas. A sentença "o ***valor dos impostos*** *é* ***5%*** *do* ***valor total da compra***" mostra um exemplo de regra de negócio matemática.

Por sua vez, as regras de persistência usam ações para persistir informações em um banco de dados ou arquivo, como, gravar, atualizar, inserir, entre outras. Por exemplo, a sentença "*o* ***usuário exclui*** *a* ***compra***", ilustra a declaração de uma regra de persistência, onde "*usuário*" e "*compra*" são conceitos relacionados pela ação "*excluir*".

Figura 3.5: Classes para as regras de negócio de computação.



Fonte: Elaborado pelo autor.

A Figura 3.5 mostra as classes definidas para as regras matemáticas e de persistência. Ambas são especializações da classe *"Computação"*, que por sua vez é uma especialização da classe *"Regra de Negócio"*. A classe *"Matemática"* tem o atributo *"temAçãoAritmética"* que indica a obrigatoriedade destas ações para caracterizar uma regra matemática. Já a classe *"Persistência"* possui o atributo

"temAcãoPersistência" indicando a obrigatoriedade destas ações na regra de persistência.

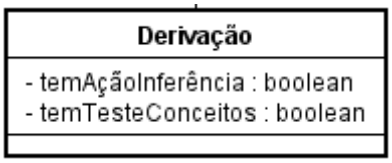
3.2.4 Classe Derivação

Esta regra é usada para definir a inferência de um novo termo (conceito de negócio), através de condições especificadas explicitamente no texto. A sentença *"o cliente* ***é considerado*** *de alto risco* ***se seu saldo devedor for superior a mil reais***" apresenta a especificação de uma regra de derivação.

A regra de derivação é caracterizada por testes condicionais entre conceitos e por ter uma ação de inferência de novos termos (ex: atribuir, considerar, inferir). No exemplo acima, a expressão "**é considerado**" determina a inferência de uma nova informação (cliente de alto risco), caso a condição "**saldo devedor for superior a mil reais**" seja satisfeita.

A Figura 3.6 mostra a definição da classe *"Derivação"*. Note na figura que os atributos *"temAçãoInferênica"* e *"temTesteConceitos"* indicam a obrigatoriedade de uma ação de inferência e uma expressão condicional.

Figura 3.6: Classe para as regras de negócio de derivação.



Fonte: Elaborado pelo autor.

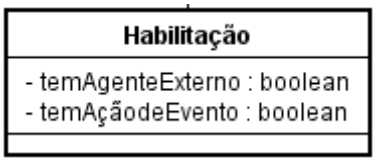
3.2.5 Classe Habilitação

Especifica a execução de uma determinada ação quando um evento externo ao sistema acontece. Geralmente, em sistemas da informação o evento externo é acionado por um usuário.

Essa regra é muito utilizada para especificar as entradas de dados (criação de novas informações), como ilustra a sentença "*o* ***usuário*** *deve* ***digitar*** *sua* ***senha***". Neste exemplo quando o "**usuário**" (conceito de negócio) digitar (evento) a "**senha**" (conceito de negócio), o sistema terá uma nova informação.

Esta classe é caracterizada por um conceito de negócio que é responsável por disparar o evento ("***usuário***"), uma ação que representa o evento ("***digitar***") e um novo conceito de negócio gerado pelo evento ("***senha***"). É importante observar que o evento, normalmente, é disparado por um agente externo ("***usuário***").

Figura 3.7: Classe para as regras de negócio de habilitação.



Fonte: Elaborado pelo autor.

A classe *"Habilitação"* é definida com o atributo *"temAgenteExterno"* indicando a obrigatoriedade de existir um conceito de negócio representando um agente externo ao

sistema. A classe também possui o atributo *"temAçãodeEvento"* que indica a existência de uma ação que representa o evento. Da mesma forma que as demais classes, a classe *"Habilitação"* também é uma especialização da classe *"Regra de Negócio"*. A Figura 3.7 mostra a classe definida para a ontologia de regras de negócio.

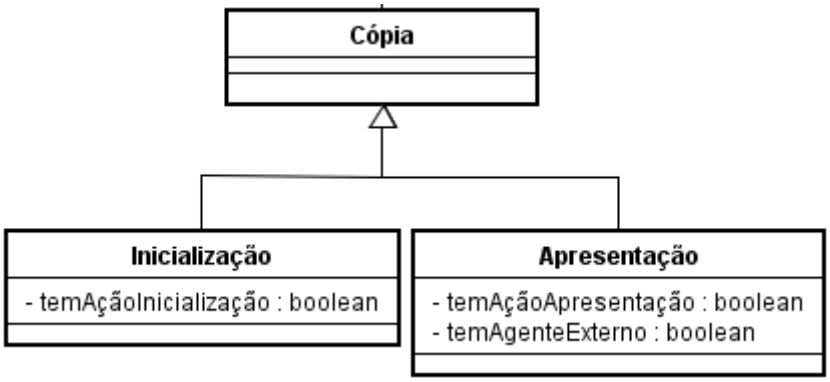
3.2.6 Classe Cópia

Essa categoria de regra de negócio é usada para especificar o estado inicial de um conceito de negócio ou mostrar o estado atual de um conceito de negócio para um agente externo do sistema.

Da mesma forma que as regras de computação, a definição de Ross (2003) para regras de cópia também é ambígua. Deste modo, especializamos a regra de cópia em regra de inicialização e regra de apresentação, como mostra a Figura 3.8.

Em contraste com a regra de habilitação, a regra de apresentação é usada para especificar a saída de dados de um sistema de informação, ou seja, mostrar o estado de um conceito para os usuários (agentes externos). A sentença "*o* ***valor do pagamento é apresentado*** *ao* ***usuário*** *em negrito*" é um exemplo de especificação de uma regra de negócio de apresentação.

Figura 3.8: Classes para as regras de negócio de cópia.



Fonte: Elaborado pelo autor.

A característica principal das regras de apresentação é possuir uma expressão que especifica uma ação de apresentação do estado de um conceito de negócio ("***apresentar***"). Além disso, também deve existir um conceito de negócio que representa o agente externo para o qual será apresentado o estado do conceito de negócio ("***usuário***"). Estas características são representadas pelos atributos *"temAçãoApresentação"* e *"temAgenteExterno"*, respectivamente.

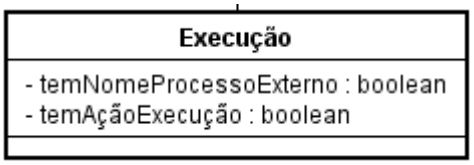
Por sua vez, as regras de inicialização possuem uma expressão que determina o estado inicial de um conceito de negócio. Por exemplo, a sentença "*a* ***data de entrega*** *da ordem de compra* ***é iniciada*** *com a* ***data corrente***" é um exemplo de regra de negócio de inicialização. O conceito de negócio "***data de entrega***" é inicializado ("***é iniciada***") com outro conceito de negócio chamado "***data corrente***". Esta característica é representada pelo atributo *"temAçãoInicialização"* na definição das classes apresentada na Figura 3.8.

3.2.7 Classe Execução

A regra de execução é usada para especificar a invocação de um procedimento ou processo externo ao sistema de informação.

Na sentença "***Notificar-Entrega*** *é* ***executado*** *quando a* ***compra for finalizada***", o conceito de negócio "***Notificar-Entrega***" se refere a um nome de procedimento ou processo que deve ser invocado (***executado***) ao finalizar uma compra (conceito de negócio). Deste modo, a classe *"Execução"* é definida com os atributos *"temNomeProcessoExterno"* e *"temAçãoExecução"*. Estes atributos indicam a obrigatoriedade de um conceito de negócio que representa o nome de um procedimento/processo e a obrigatoriedade de uma ação de execução. A Figura 3.9 mostra a classe que define uma regra de execução.

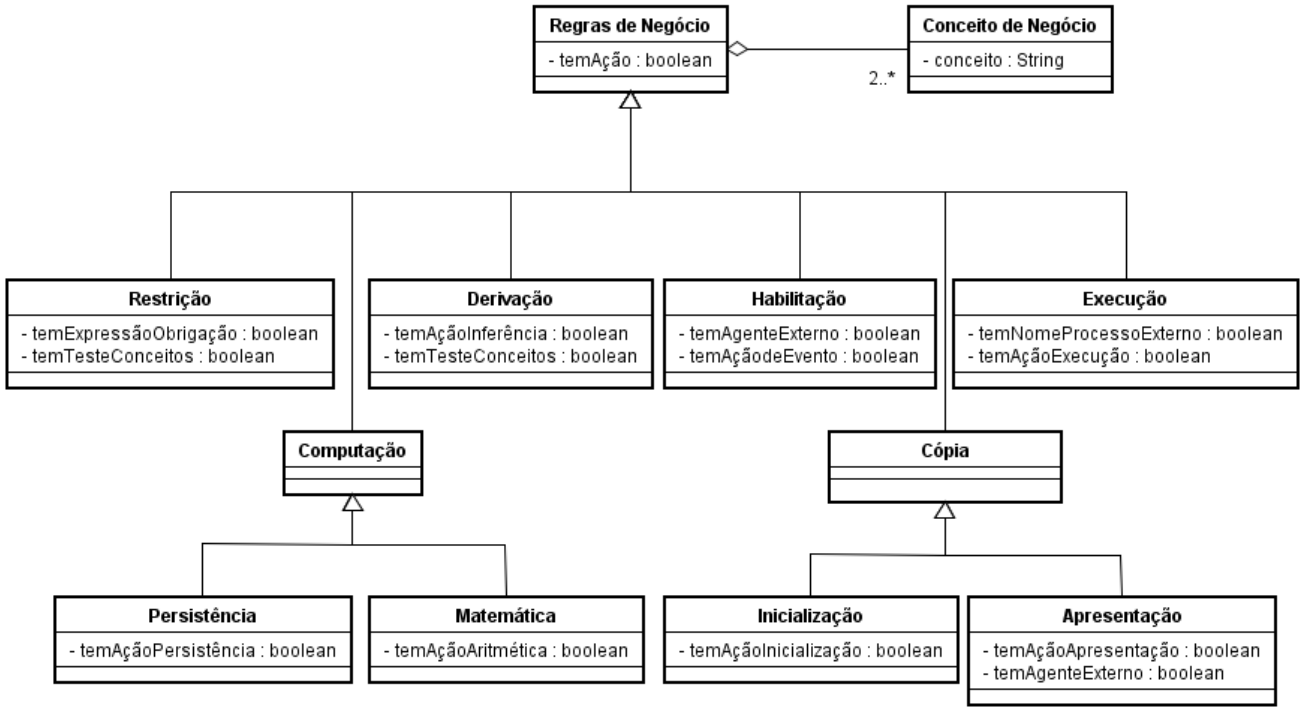
Figura 3.9: Classe para as regras de negócio de execução.



Fonte: Elaborado pelo autor.

Após a representação individual das classes de regras de negócio, a Figura 3.10 mostra a ontologia definida a partir da classificação das regras de negócio definida por Ross (2003).

Figura 3.10: Ontologia das regras de negócio em linguagem natural.



Fonte: Elaborado pelo autor.

Na ontologia foram acrescentadas quatro novos tipos de regras de negócio que não aparecem no esquema de classificação definido por Ross (2003). São as classes

"*Persistência*", "*Matemática*", "*Inicialização*" e "*Apresentação*". Ambas são especializações de classes definidas por Ross (2003) e foram acrescentadas para classificar com mais precisão as 417 sentenças de regras de negócio encontradas nos sistemas legados selecionados para este trabalho.

3.3 Suporte da Coleta de Regras de Negócio

Como mencionado no início da Seção 3.2, a definição da ontologia de regras de negócio apresentada na Figura 3.10, foi realizada através da classificação de 417 sentenças textuais que descrevem regras de negócio nas documentações de cinco sistemas legados selecionados do Portal de Software Público Brasileiro (MPOG, 2011). As 417 sentenças textuais foram classificadas nos esquemas de classificação descritos na literatura, permitindo assim, encontrar quais os tipos de regras de negócio usados nos sistemas legados estudados nesta pesquisa.

Para demonstrar a probabilidade de um tipo de regra de negócio aparecer na documentação dos sistemas legados foi coletada a medida de suporte das classes de regras de negócio.

O suporte é uma medida usada na mineração de regras de associação em registros em banco de dados (AGRAWAL; IMIELINSKI; SWAMI, 1993). Essa medida foi usada para verificar a probabilidade de associações de dados ocorrerem em uma determinada base de dados. Por exemplo, Agrawal, Imielinski e Swami (1993) usam a medida de suporte para determinar a probabilidade de clientes comprarem um tênis juntamente com um par de meias em uma loja de calçados.

Para essa tese, a medida de suporte foi adaptada para determinar a probabilidade de uma classe de regra de negócio aparecer em um dos sistemas legados selecionados no Portal do Software Público Brasileiro (MPOG, 2011).

Após a identificação e classificação das sentenças de regras de negócio contidas nas documentações dos sistemas analisados, foram contadas quantas ocorrências de cada classe de regra existem no conjunto total das regras de negócio. A divisão do número de ocorrências de uma determinada classe, pelo número total de regras existentes nos sistemas legados, determina o suporte desta classe de regra de negócio. A Figura 3.11 mostra a fórmula usada para calcular a medida de suporte.

Figura 3.11: Fórmula para calcular o suporte das classes de regras de negócio.

C = Classe de Regra de Negócio (Figura 3.10).

TS = Número Total de Regras de Negócio Identificadas nos Sistemas.

O(C) = Ocorrências de uma Classe de Regras de Negócio nos Sistemas.

Suporte (C) = O(C) / TS

Fonte: Elaborado pelo autor.

A Tabela 3.1 apresenta os resultados de suporte para cada classe de regras de negócio definida na ontologia da Figura 3.10. Os resultados mostram a totalização considerando os cinco sistemas legados analisados no contexto deste trabalho.

Tabela 3.1. Medida de suporte para as classes de regra de negócio.

Número Total de Regras de Negócio (TS): 417		
Classe de Regra de Negócio (C)	**O(C)**	**Suporte**
Regra de Restrição	129	129 / 417 = 0,30
Regra Matemática	9	0,02
Regra de Persistência	123	0,29
Regra de Derivação	32	0,08
Regra de Inicialização	8	0,02
Regra de Apresentação	38	0,09
Regra de Habilitação	57	0,14
Regra de Execução	21	0,05

Fonte: Elaborado pelo autor.

Todas as classes de regras de negócio aparecem nos cinco sistemas legados. As classes que mais aparecem são as regras de restrição e computação (especializadas em persistência e matemática), que somadas equivalem a 61% das regras de negócio encontradas nos sistemas legados avaliados neste trabalho.

Note que a regra matemática obteve suporte baixo. Entretanto, essa probabilidade pode não se repetir em outros sistemas legados, pois os cinco sistemas analisados neste trabalho caracterizam-se por serem sistemas de informações cadastrais (i.e. realizam grande interação com banco de dados e poucos cálculos matemáticos).

As regras de habilitação aparecem poucas vezes (14%), pois são entradas de dados do usuário, que geralmente são agrupadas em formulários eletrônicos. Isto é, cada formulário agrupa um conjunto de entrada de dados. Assim, as várias sentenças textuais que definem um único formulário eletrônico foram consideradas uma única regra de habilitação. Essa mesma lógica foi empregada para as regras de apresentação.

Todas as regras de negócio encontradas nos sistemas legados foram classificadas em uma classe de regra de negócio da ontologia definida na Figura 3.10. Portanto, infere-se que essa distribuição de regras de negócio deve ocorrer em outros sistemas legados. Assim, pretende-se usar a ontologia de regras de negócio da Figura 3.10, como base de conhecimento para definir uma ferramenta capaz identificar e anotar regras de negócio em um sistema legado.

4 CODIFICAÇÃO DAS REGRAS DE NEGÓCIO

A partir da ontologia de regras de negócio da Figura 3.10 é proposto um mapeamento das classes considerando as características de codificação dos tipos de regras de negócio em uma linguagem de programação imperativa. Esse mapeamento gera um conjunto de classes equivalentes, onde os atributos referem-se a comandos de uma linguagem de programação que normalmente são usados para codificar as regras de negócio em um sistema de informação.

Para realizar o mapeamento foram executados o terceiro e quarto passos da estratégia de estudo apresentada na Figura 3.2 (3.2(c) e 3.2(d)). Foram pesquisadas no código fonte dos cinco sistemas legados, as codificações das 417 regras de negócio encontradas na documentação destes sistemas. Assim, foram observadas quais estruturas sintáticas da linguagem de programação são mais empregadas para implementar cada classe de regra de negócio definida na ontologia da Figura 3.10.

É importante salientar que os sistemas legados estudados no contexto deste trabalho foram escritos nas linguagens C, COBOL e PHP. Entretanto, é possível generalizar a linguagem de programação, pois a semântica dos comandos usados na implementação das regras é a mesma nas três linguagens de programação estudadas nesta tese. Sendo assim, o restante do capítulo usará a linguagem de programação fictícia definida na Seção 2.4, que contém os principais comandos de uma linguagem de programação imperativa.

4.1 Classes de Implementação das Regras de Negócio

Como mencionado no início deste capítulo, o próximo passo da pesquisa consistiu em buscar as implementações das 417 regras de negócio encontradas na documentação dos sistemas legados, em seus respectivos códigos fontes (Figura 3.2(c)). Após a identificação e anotação dos trechos de código que implementam cada regra de negócio, observaram-se as estruturas sintáticas da linguagem de programação que mais foram empregadas para codificar as regras de negócio de uma determinada classe da ontologia (Figura 3.10). Assim, cada classe da ontologia é mapeada para uma classe equivalente, onde os atributos representam as estruturas sintáticas mais empregadas para codificar suas características em uma linguagem de programação imperativa.

Por exemplo, a classe *"Regra de Negócio"* (Figura 3.10) possui o atributo *"temConceitosNegócio"*. Este atributo é mapeado na classe equivalente para um atributo *"temVariáveis"*. Observou-se na codificação das 417 regras de negócio, que as variáveis são as estruturas da linguagem de programação mais empregadas para codificar os conceitos de negócio. Isto é, usualmente, os conceitos de negócio são

codificados em um sistema de informação usando o componente variável de uma linguagem de programação.

Além disso, para cada classe de regra de negócio é definido um grafo conceitual a partir dos grafos dos comandos da linguagem de programação fictícia (Figura 2.10). Com isso pretende-se definir o comportamento de cada classe de regra de negócio definida na ontologia.

4.1.1 Mapeamento para Classe Regra de Negócio

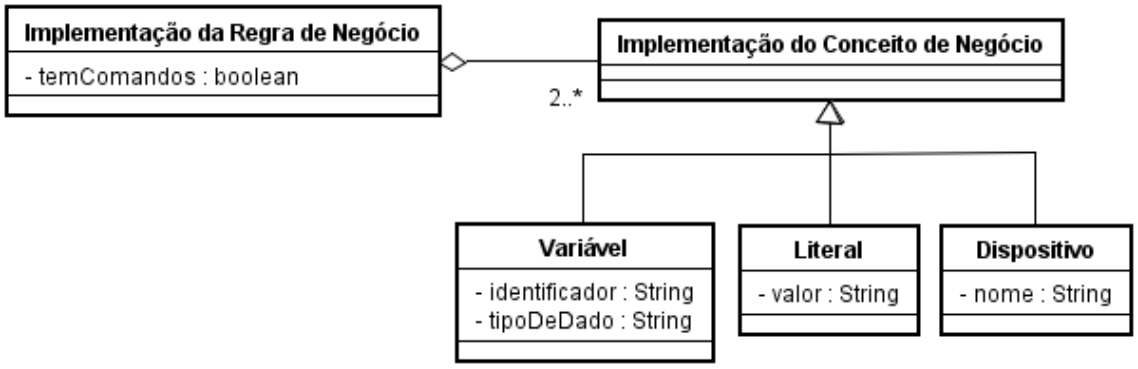
O primeiro mapeamento é da classe *"Conceito de Negócio"*. Como já mencionado anteriormente, um conceito de negócio, geralmente, é implementado no código fonte usando variáveis ou literais. Entretanto, conexões com arquivos ou banco de dados e dispositivos de entrada/saída dados também são conceitos de negócio no código fonte do sistema de informação.

Deste modo, a classe *"Conceito de Negócio"* é mapeada para a classe *"Implementação do Conceito de Negócio"*, com especializações para *"Variável"*, *"Literal"* e *"Dispositivo"*. Ou seja, um conceito de negócio é codificado através de variável, literal ou dispositivo especial da linguagem de programação.

A classe *"Regra de Negócio"*, por sua vez, é mapeada para a classe *"Implementação da Regra de Negócio"*. Esta mesma nomenclatura de nomes é usada para as demais classes apresentadas na ontologia da Figura 3.10.

O relacionamento da classe *"Regra de Negócio"* com *"Conceito de Negócio"* é mapeado para o relacionamento entre as classes *"Implementação da Regra de Negócio"* e *"Implementação do Conceito de Negócio"*.

Figura 4.1: Mapeamento da classe regra de negócio.



(a) Classes de Implementação de uma Regra de Negócio Genérica

(b) Comportamento de uma Regra de Negócio Genérica

Fonte: Elaborado pelo autor.

As regras de negócio são codificadas através de comandos da linguagem de programação que manipulam as variáveis da regra. Assim, os comandos da linguagem de programação equivalem às expressões de ação contidas na classe *"Regra de*

Negócio". Portanto, o atributo *"temAções"* na classe *"Regra de Negócio"* é mapeado para o atributo *"temComandos"* na classe *"Implementação da Regra de Negócio"*.

A Figura 4.1(a) mostra as classes equivalentes para *"Regra de Negócio"* e *"Conceitos de Negócio"*. Já a Figura 4.1(b) apresenta o grafo conceitual para uma implementação de regra de negócio. A seguir são apresentadas as classes equivalentes para as demais classes de regra de negócio da ontologia definida na Figura 3.10.

4.1.2 Mapeamento da Classe Restrição

As sentenças classificadas como regra de restrição, normalmente, são codificadas através de comandos condicionais ou comandos de repetição. Essas regras são codificadas com comandos ***"if"***, ***"switch-case"***, ***"for"*** ou ***"while"*** que restringem a execução de um conjunto de comandos. A expressão *"b"* nos comandos condicionais e de repetição equivale ao atributo *"temTesteConceitos"* da classe *"Restrição"*, enquanto a semântica SE (***"if"***), CASO (***"case"***), ENQUANTO (***"while"***) representam a expressão de obrigação (atributo *"temExpressãoObrigação"*).

A Figura 4.2 apresenta a codificação da regra de restrição definida pela sentença textual "*para efetuar uma compra o cliente* ***deve*** *ter* ***crédito maior que o valor da compra***". O código apresentado na figura usa a sintaxe da linguagem de programação fictícia defina na Seção 2.4.

Figura 4.2: Exemplo de codificação de uma regra de restrição.

```
if (credito > valor_compra) then
  begin sql
    /* restante do código... */
  end sql
end
```

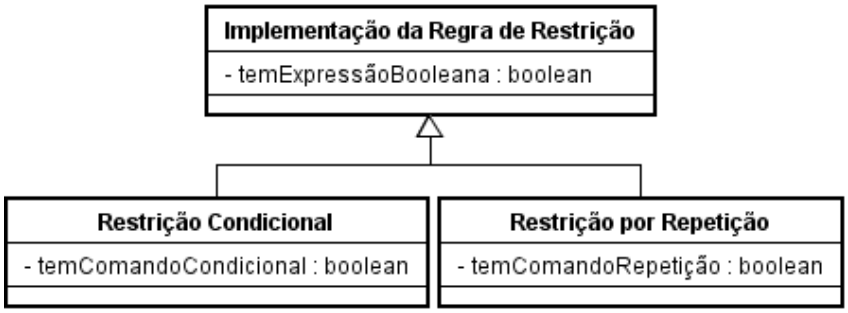
Fonte: Elaborado pelo autor.

Deste modo, a classe *"Implementação da Regra de Restrição"* possui o atributo *"temExpressãoBooleana"* que representa o atributo *"temTesteConceitos"* da classe *"Restrição"* definida na ontologia da Figura 3.10.

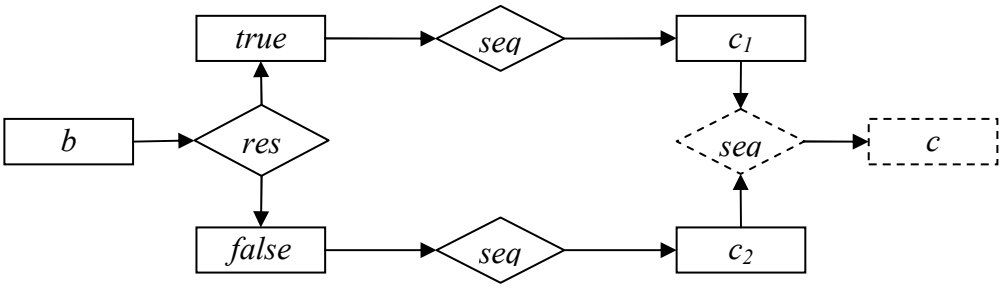
Note também, que o atributo *"temExpressãoObrigação"* da classe *"Restrição"* pode ser codificado por um comando condicional ou de repetição. Deste modo, a classe *"Implementação da Regra de Restrição"* foi especializada para evitar ambiguidades na ferramenta de identificação das regras de negócio (*"Restrição Condicional"* e *"Restrição por Repetição"*). Portanto, a obrigatoriedade de existir uma expressão de obrigação (*"temExpressãoObrigação"*) é mapeada para as subclasses da classe *"Implementação de Restrição"*.

A Figura 4.3 apresenta as classes equivalentes definidas para a implementação de uma regra de restrição, juntamente com seus respectivos grafos conceituais.

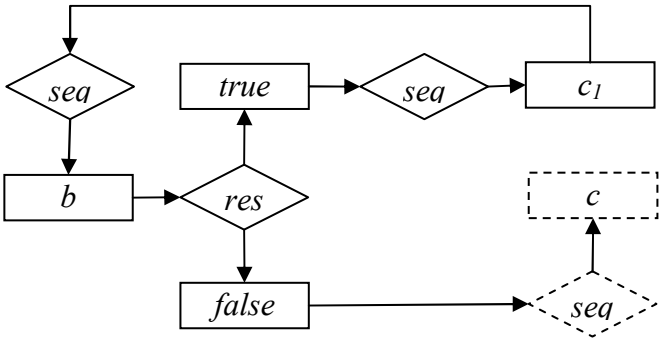
Figura 4.3: Mapeamento da classe restrição.



(a) Classes de Implementação de uma Regra de Restrição



(b) Comportamento da Regra com Comando Condicional



(c) Comportamento da Regra com Comando de Repetição

Fonte: Elaborado pelo autor.

4.1.3 Mapeamento da Classe Computação

A regra de computação foi especializada em regra matemática e regra de persistência. A regra matemática na linguagem de programação, geralmente é codificada por um ou mais comandos de atribuição de expressões aritméticas, como mostra a Figura 4.4. Portanto, o atributo *"temAçãoAritmética"* na classe *"Matemática"* é mapeado para *"temAtribuiçãoComExpressãoAritmética"* na classe *"Implementação da Regra Matemática"*.

Figura 4.4: Exemplo de codificação de uma regra matemática.

Sentença: *O **valor dos impostos** é **5%** do **valor total da compra.***
Codificação: `valor_imposto := valor_total_compra * 0.05;`

Fonte: Elaborado pelo autor.

Por sua vez, a regra de persistência é implementada por um bloco de persistência (*"**begin sql** c **end sql**"*), como ilustra a Figura 4.5. Assim, o atributo *"temAçãoPersistência"* na classe *"Persistência"* é transformado em *"temBlocoDePersistência"* na classe *"Implementação da Regra de Persistência"*.

Figura 4.5: Exemplo de codificação de uma regra de persistência.

Sentença:

*O **usuário exclui** a **compra** do banco de dados.*

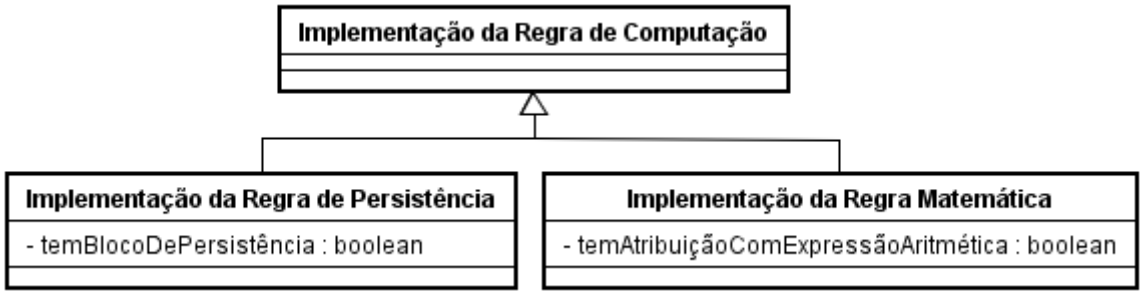
Codificação:

```
begin sql
  exec sql "delete from compras where no = :numero";
end sql
```

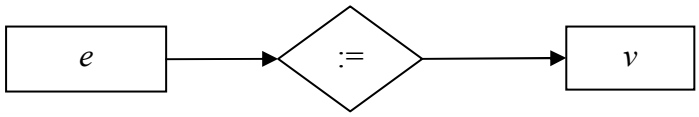
Fonte: Elaborado pelo autor.

A Figura 4.6 mostra as classes equivalentes para a implementação da regra de computação, assim como os grafos conceituais com o comportamento das regras matemática e de persistência.

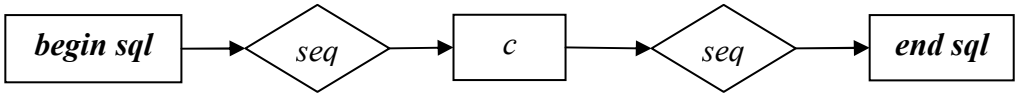
Figura 4.6: Mapeamento da classe computação.



(a) Classes de Implementação de uma Regra de Computação



(b) Comportamento de uma Regra Matemática



(c) Comportamento de uma Regra de Persistência

Fonte: Elaborado pelo autor.

4.1.4 Mapeamento da Classe Derivação

A codificação de uma regra de derivação normalmente é feita através de um comando condicional (comando *"**if**"*). Esta regra pode ser confundida com a regra de restrição, entretanto, nas regras de derivação, os comandos dentro do bloco condicional

são apenas de atribuições de valores (ex.: *"v := 20;"*). Já nas regras de restrição os comandos formam outra regra de negócio, como um bloco de persistência (***"begin sql"***) que caracteriza uma regra de persistência, ou um conjunto de atribuições aritméticas que caracterizam uma regra matemática.

Note que a regra de derivação é caracterizada por atribuições de valores fixos e não valores calculados por uma expressão aritmética. Na regra matemática o valor atribuído a variável é proveniente de um cálculo aritmético (ex.: *"v := e_1 + e_2"*). As regras de derivação são usadas para deduzir um determinado conceito a partir de uma condição. Portanto, o comando verifica uma condição e uma variável recebe um valor literal, que representa a dedução do conceito.

A Figura 4.7 apresenta a codificação da regra de derivação descrita pela sentença: *"o cliente* ***é considerado*** *de alto risco* ***se seu saldo devedor for superior a mil reais***".

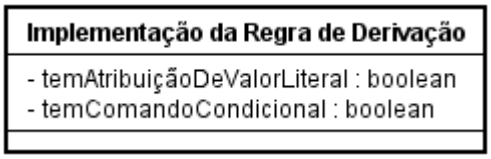
Figura 4.7: Exemplo de codificação de uma regra de derivação.

```
if (saldo_devedor > 1000) then
  risco := 5;
end
```

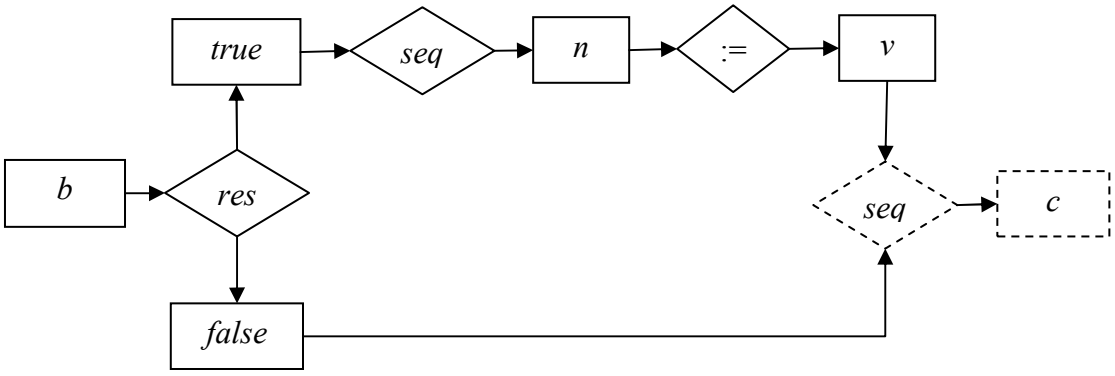
Fonte: Elaborado pelo autor.

O atributo *"temTesteConceitos"* na classe *"Derivação"* é mapeado para o atributo *"temComandoCondicional"* na classe *"Implementação da Regra de Derivação"*. Por sua vez, o atributo *"temAçãoInferência"* na classe *"Derivação"* é mapeado para o atributo *"temAtribuiçãoDeValorLiteral"*. A Figura 4.8 mostra a definição da classe *"Implementação da Regra de Derivação"* e o grafo conceitual que representa o comportamento desta regra.

Figura 4.8: Mapeamento da classe derivação.



(a) Classe de Implementação de uma Regra de Derivação



(b) Comportamento de uma Regra de Derivação

Fonte: Elaborado pelo autor.

4.1.5 Mapeamento da Classe Habilitação

Todas as regras de habilitação encontradas nos sistemas legados estudados foram codificadas através de comandos de entrada de dados. O comando "***read(v)***" ocorre quando o usuário deve informar um novo valor, que é armazenado em uma variável (*"v"*). A Figura 4.9 mostra a codificação da regra de habilitação para a sentença: *"o* ***usuário*** *deve* ***digitar*** *sua* ***senha****"*.

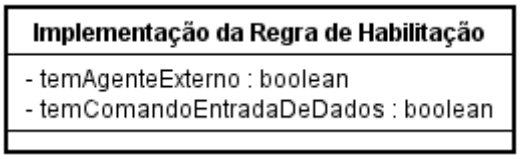
Figura 4.9: Exemplo de codificação de uma regra de habilitação.

```
read(senha);
```

Fonte: Elaborado pelo autor.

A Figura 4.10 apresenta a classe *"Implementação da Regra de Habilitação"*, onde o atributo *"temComandoEntradaDeDados"* é suficiente para mapear os atributos *"temAgenteExterno"* e *"temAçãodeEvento"* da classe *"Habilitação"*. Entretanto, o atributo *"temAgenteExterno"* é mantido na classe de implementação, pois linguagens de programação mais recentes possuem outras formas de entrada de dados, onde o agente externo não é um usuário do sistema (por exemplo, leitura de arquivos).

Figura 4.10: Mapeamento da classe habilitação.



(a) Classe de Implementação de uma Regra de Habilitação

(b) Comportamento de uma Regra de Habilitação

Fonte: Elaborado pelo autor.

4.1.6 Mapeamento da Classe Cópia

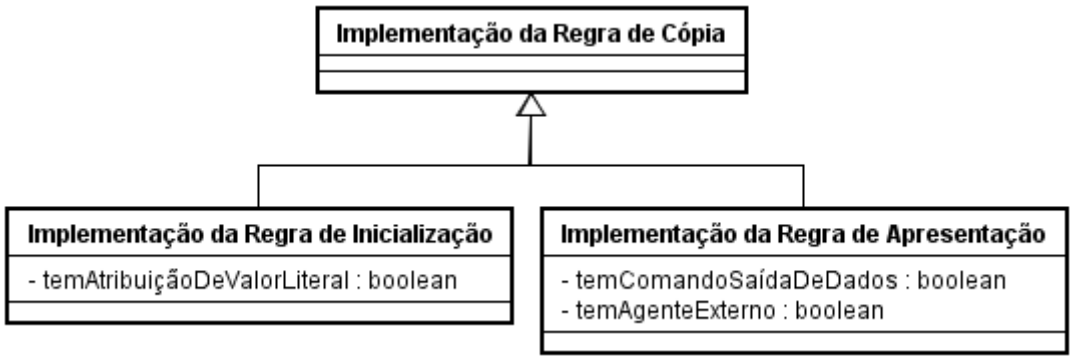
A classe *"Cópia"* foi especializada na ontologia para *"Apresentação"* e *"Inicialização"*. A codificação das regras de apresentação, usualmente é feita através de comandos de saída de dados (*"****print(v)****"*). Já a codificação das regras de inicialização é feita por comandos de atribuição de literais (*"v := n"*).

Da mesma forma que a regra de derivação, as atribuições nas regras de inicialização são de valores literais (ex.: *"v := 3"*), caso contrário, os comandos caracterizam uma regra matemática. Porém, nas regras de inicialização não possuem a condição que caracteriza a dedução de um valor (comando condicional), como ocorre na regra de derivação.

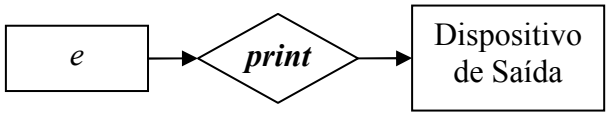
A Figura 4.11 ilustra as classes *"Implementação da Regra de Inicialização"* e *"Implementação da Regra de Apresentação"*. Na classe *"Implementação da Regra de Inicialização"* o atributo *"temAtribuiçãoDeValorLiteral"* substitui o atributo *"temAçãoInicialização"* da classe *"Inicialização"*. Enquanto isso, na classe

"Implementação da Regra de Apresentação" o atributo *"temComandoSaídaDeDados"* substitui o atributo *"temAçãoApresentação"* na classe *"Apresentação"*.

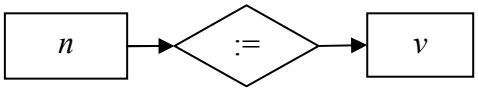
Figura 4.11: Mapeamento da classe cópia.



(a) Classes de Implementação de uma Regra de Cópia



(b) Comportamento de uma Regra de Apresentação



(c) Comportamento de uma Regra de Inicialização

Fonte: Elaborado pelo autor.

4.1.7 Mapeamento da Classe Execução

As regras de execução descrevem uma chamada para procedimento ou processo externo ao sistema. Deste modo, estas regras são codificadas na linguagem de programação por uma chamada de subrotina (*"v := **call** func"* ou *"**call** proc"*), como mostra a Figura 4.12.

Figura 4.12: Exemplo de codificação de uma regra de execução.

Sentença:

Notificar-Entrega** é **executado** quando a **compra for finalizada.

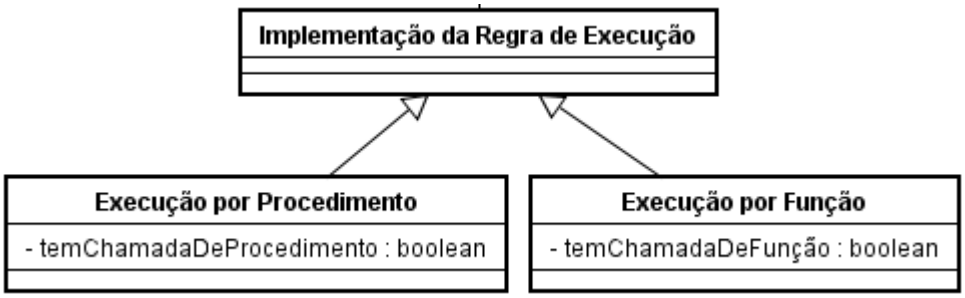
Codificação:

```
call Notificar-Entrega;
```

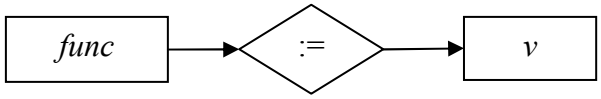
Fonte: Elaborado pelo autor.

Os atributos *"temChamadaDeProcedimento"* ou *"temChamadaDeFunção"* são usados para mapear as características da classe *"Execução"*, como mostra a Figura 4.13. A figura também ilustra que a implementação da regra é especializada em "*Execução por Procedimento*" e "*Execução por Função*" para evitar ambiguidades na ferramenta de identificação das regras de negócio.

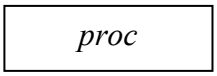
Figura 4.13: Mapeamento da classe execução.



(a) Classes de Implementação de uma Regra de Execução



(b) Comportamento de uma Regra de Execução por Função

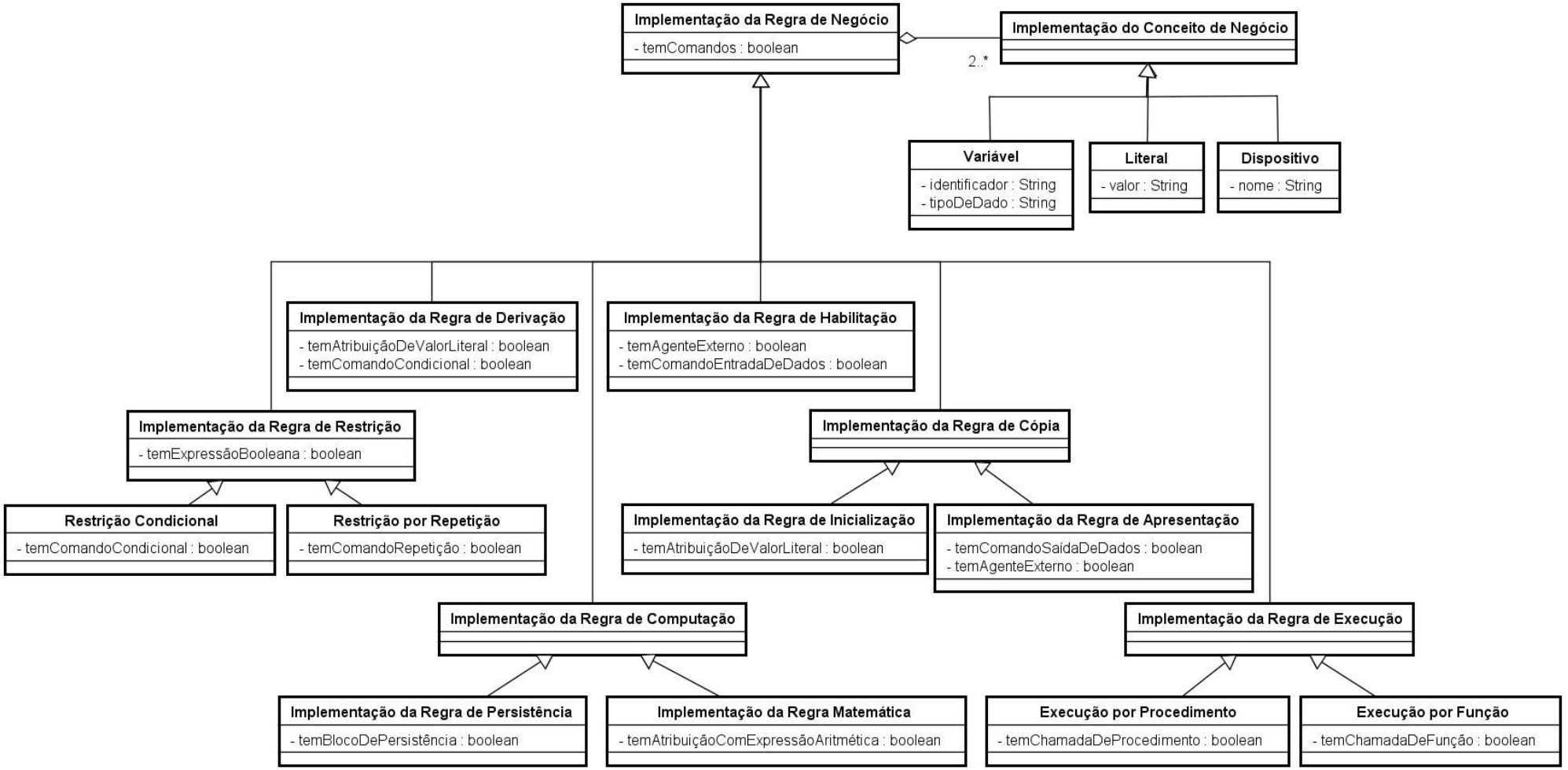


(c) Comportamento de uma Regra de Execução por Procedimento

Fonte: Elaborado pelo autor.

A Figura 4.14 mostra a hierarquia de classes equivalentes gerada para a ontologia de regras de negócio apresentada na Figura 3.10. As classes equivalentes consideram as características de codificação observadas para as 417 regras de negócio encontradas nos sistemas legados selecionados para este trabalho.

Figura 4.14: Classes equivalentes da ontologia de regras de negócio.



Fonte: Elaborado pelo autor.

4.2 Confiança do Mapeamento

Para verificar a confiabilidade das classes de implementação das regras de negócio calculou-se a medida de confiança destas classes (AGRAWAL et. al., 1996). Da mesma forma que a medida de suporte, usada no Capítulo 3, a medida de confiança também é utilizada para verificar a eficiência na mineração de regras de associação em registros em banco de dados (AGRAWAL et. al., 1996).

Nesta tese, a medida de confiança demonstra a probabilidade das regras de negócio estar codificadas usando as características de implementação propostas nas classes equivalentes (Figura 4.14). Por exemplo, a medida de confiança mostra a probabilidade de uma regra de apresentação ser codificada usando comandos de saída de dados, como proposto na classe *"Implementação da Regra de Apresentação"*.

A medida de confiança foi obtida contando o número de ocorrências de um determinado tipo de regra de negócio, que foi codificado com as estruturas sintáticas definidas na sua respectiva classe de implementação. Esse número é dividido pelo total de regras de negócio deste tipo encontradas na documentação dos sistemas legados. O resultado da divisão corresponde à probabilidade do tipo de regra de negócio ser codificado com as estruturas sintáticas definidas na sua classe de implementação.

Por exemplo, a Tabela 3.1 mostra que foram encontradas nove ocorrências de regras matemáticas nos sistemas legados estudados neste trabalho. Destas ocorrências, constatou-se que oito foram codificadas usando comandos de atribuição seguidos de expressões aritméticas, como definido na classe *"Implementação da Regra Matemática"*. Deste modo, a confiança é calculada dividindo oito por nove. O resultado do cálculo é 0,88, ou seja, confiança que 88% das implementações deste tipo de regra de negócio são feitas através de comandos de atribuição seguidos de expressões matemáticas.

A Figura 4.15 apresenta a fórmula usada para o cálculo da medida de confiança.

Figura 4.15: Fórmula para calcular a confiança das classes de implementação.

C = Classe de Regra de Negócio (Figura 3.10).

O(C) = Ocorrências de uma determinada Classe de Regras de Negócio nos Sistema Legados (Tabela 3.1).

OI(C) = Ocorrências de uma determinada Classe de Implementação da Regra de Negócio no Código Fonte dos Sistemas Legados (Figura 4.14).

Confiança (C) = OI(C) / O(C)

Fonte: Elaborado pelo autor.

A Tabela 4.1 mostra os resultados da medida de confiança para cada uma das classes de regra de negócio. Os resultados apresentados consideram o total de regras de negócio identificado nos cinco sistemas legados estudados nesta tese.

Note na Tabela 4.1, que o grau de confiança nas classes de implementação definidas na Figura 4.14, é superior a 90% para quase todas as classes, com exceção da regra de matemática que ficou em 88%. Isso demonstra que as regras de negócio, normalmente,

são implementadas no código fonte usando as estruturas da linguagem de programação mapeadas nas classes equivalentes.

Tabela 4.1: Confiança para as classes de regras de negócio.

Número Total de Regras de Negócio (TS): 417			
Classe de Regra de Negócio (C)	**O(C)**	**OI(C)**	**Confiança**
Regras de Restrição	129	124	124 / 129 = 0,96
Regra Matemática	9	8	0,88
Regra de Persistência	123	120	0,97
Regra de Derivação	32	30	0,93
Regra de Inicialização	8	8	1,00
Regra de Apresentação	38	37	0,97
Regra de Habilitação	21	20	0,95
Regra de Execução	57	54	0,94

Fonte: Elaborado pelo autor.

O grau de confiança não foi maior, pois algumas regras identificadas no código possuem características de duas ou mais classes, tornado a classificação ambígua. Contudo estes problemas podem ser contornados pela análise humana das regras com classificação ambígua. Ou seja, o algoritmo de identificação das regras pode sinalizar para um especialista quais regras tiveram classificação ambígua.

É importante ressaltar que todas as regras de negócio encontradas nos sistemas analisados foram classificadas em classes definidas nesta tese. Entretanto, não é possível afirmar que existem apenas estas classes de regras de negócio. Por este motivo, a ontologia conta com uma classe genérica: *"Regra de Negócio"*. Caso ocorra um trecho de código que não pode ser classificado em uma das classes de regras da Figura 4.14, este trecho será considerado uma regra de negócio genérica. Isso sinaliza que existe um novo tipo de regra de negócio, que ainda não foi classificado na ontologia.

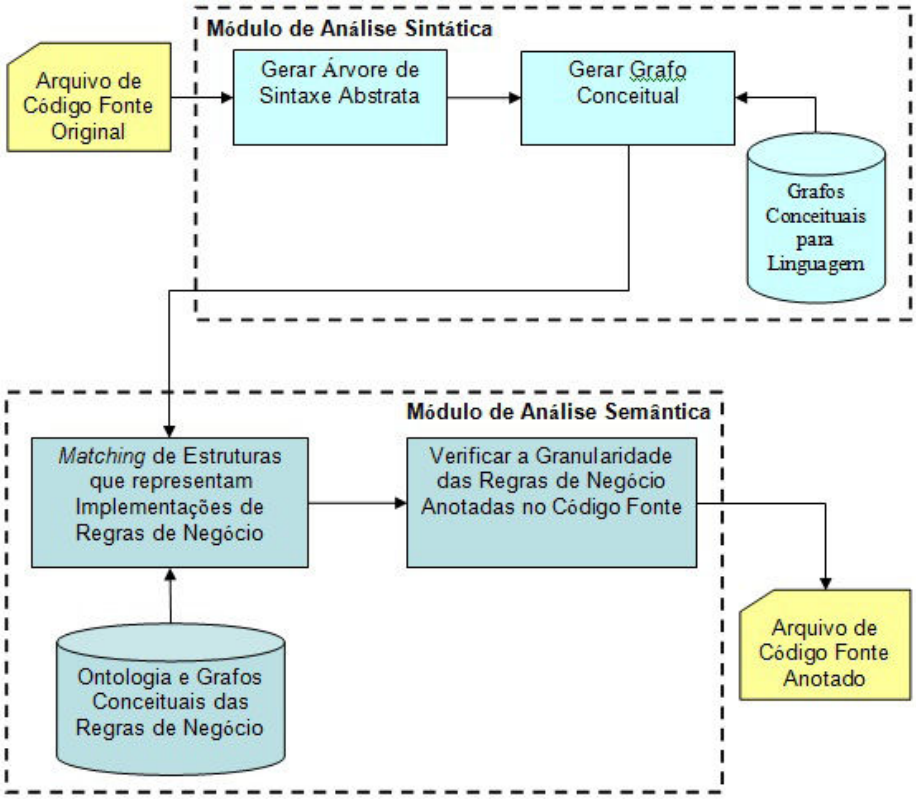
5 FERRAMENTA DE IDENTIFICAÇÃO DAS REGRAS DE NEGÓCIO

Com base na ontologia de regras de negócio e nos grafos conceituais, o próximo passo foi definir uma ferramenta capaz de identificar as regras de negócio contidas nos arquivos de código fonte dos sistemas legados. Este capítulo apresenta a arquitetura da ferramenta, detalhando o algoritmo de identificação das regras de negócio e um conjunto de heurísticas criado para melhorar o resultado final da identificação das regras de negócio.

5.1 Arquitetura da Ferramenta

A ferramenta recebe como entrada um arquivo de código fonte e retorna como saída, o mesmo arquivo com as anotações das regras de negócio contidas neste arquivo. A Figura 5.1 apresenta a arquitetura da ferramenta, dividida em dois módulos: módulo de análise sintática; e módulo de análise semântica.

Figura 5.1: Arquitetura da ferramenta para identificação das regras de negócio.



Fonte: Elaborado pelo autor.

Para demonstrar o funcionamento da ferramenta será usado um algoritmo simples, para cálculo de média aritmética. A Figura 5.2 mostra o código fonte que segue a sintaxe da linguagem de programação proposta na Seção 2.4.

Figura 5.2: Algoritmo para demonstrar a identificação das regras de negócio.

```
...
read(x);
read(y);
soma := x + y;
media := soma / 2;
if media > 5 then
  print("Aprovado");
  print(media);
else
  print("Reprovado");
  print(media);
end
...
```

Fonte: Elaborado pelo autor.

5.2 Módulo de Análise Sintática

O primeiro módulo da ferramenta é usado para executar o *parsing* do código fonte legado. Neste módulo o arquivo de código fonte é interpretado e transformado em um modelo computacionalmente interpretável, que pode ser percorrido pelo módulo de análise semântica para identificar as regras de negócio.

O módulo de análise sintática é composto por duas etapas: **gerar árvore de sintaxe abstrata**; e **gerar grafo conceitual**.

5.2.1 Gerar Árvore de Sintaxe Abstrata

O primeiro passo da ferramenta consiste em gerar uma árvore de sintaxe abstrata (AST – *Abstract Syntax Tree*), que permite realizar o *parsing* do código fonte (PARR; QUONG, 1995). O *parsing* do código fonte consiste em gerar uma estrutura de dados, que facilite a interpretação do código fonte e permita verificar a correção do código, ou seja, se não existem problemas de sintaxe no código fonte (PARR; QUONG, 1995).

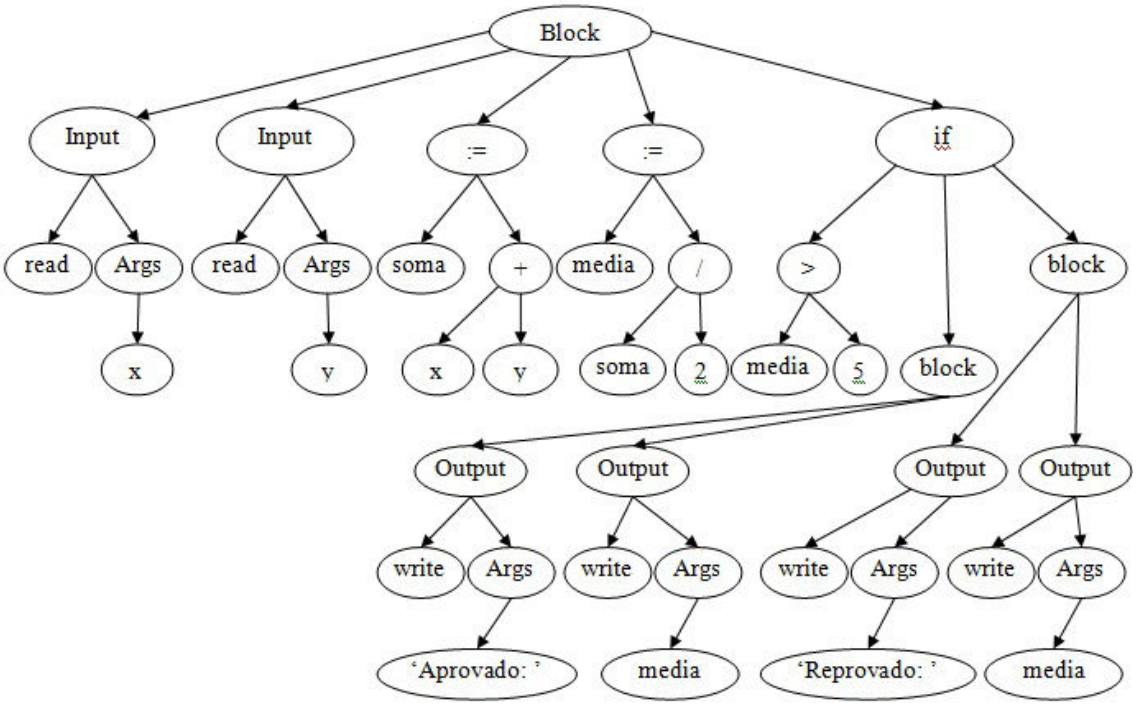
A implementação desta etapa é feita através de algoritmos encontrados na literatura (PARR; QUONG, 1995). Estes algoritmos geram uma árvore *n-ária*, onde os nodos da árvore representam os comandos que compõem o código fonte do sistema legado. Posteriormente, na próxima etapa da ferramenta, a árvore é mapeada para um grafo conceitual que representa o comportamento do código fonte legado.

A biblioteca ANTLR (PARR; QUONG, 1995) foi escolhida para utilização na primeira versão da ferramenta de identificação das regras de negócio. A biblioteca é escrita em Java (mesma tecnologia usada no desenvolvimento da ferramenta) e permite a geração da AST a partir de uma gramática de linguagem de programação parametrizada na ferramenta. Ou seja, a gramática e o arquivo de código fonte são

passados como parâmetros para a biblioteca executar o *parsing*. Ao final da operação a biblioteca gera uma AST que representa o arquivo de código fonte.

Ao executar o *parsing* do código fonte apresentado na Figura 5.2, usando a gramática da linguagem de programação da Seção 2.4 e a biblioteca ANTLR (PARR; QUONG, 1995), obtém-se a árvore de sintaxe abstrata ilustrada na Figura 5.3.

Figura 5.3: Árvore de sintaxe abstrata para o algoritmo da Figura 5.2.



Fonte: Elaborado pelo autor.

5.2.2 Gerar Grafo Conceitual

Agora, a árvore de sintaxe abstrata é convertida para um grafo conceitual que representa o comportamento do código fonte. Para criar o grafo conceitual são usadas as estruturas definidas para cada comando da linguagem de programação na Seção 2.4.2. Para cada comando encontrado na árvore de sintaxe abstrata, o grafo conceitual do código fonte recebe a adição de um subgrafo que representa este comando na linguagem de programação.

É importante ressaltar, que para programas organizados em subrotinas (funções, procedimentos ou métodos), a ferramenta gera um grafo conceitual para cada subrotina. Assim, tanto a geração do grafo conceitual, quanto a análise semântica das regras de negócio, são realizadas por subrotinas implementadas no código fonte legado. Por exemplo, se o arquivo de código submetido conter três subrotinas, então a ferramenta irá gerar três grafos conceituais e analisá-los separadamente.

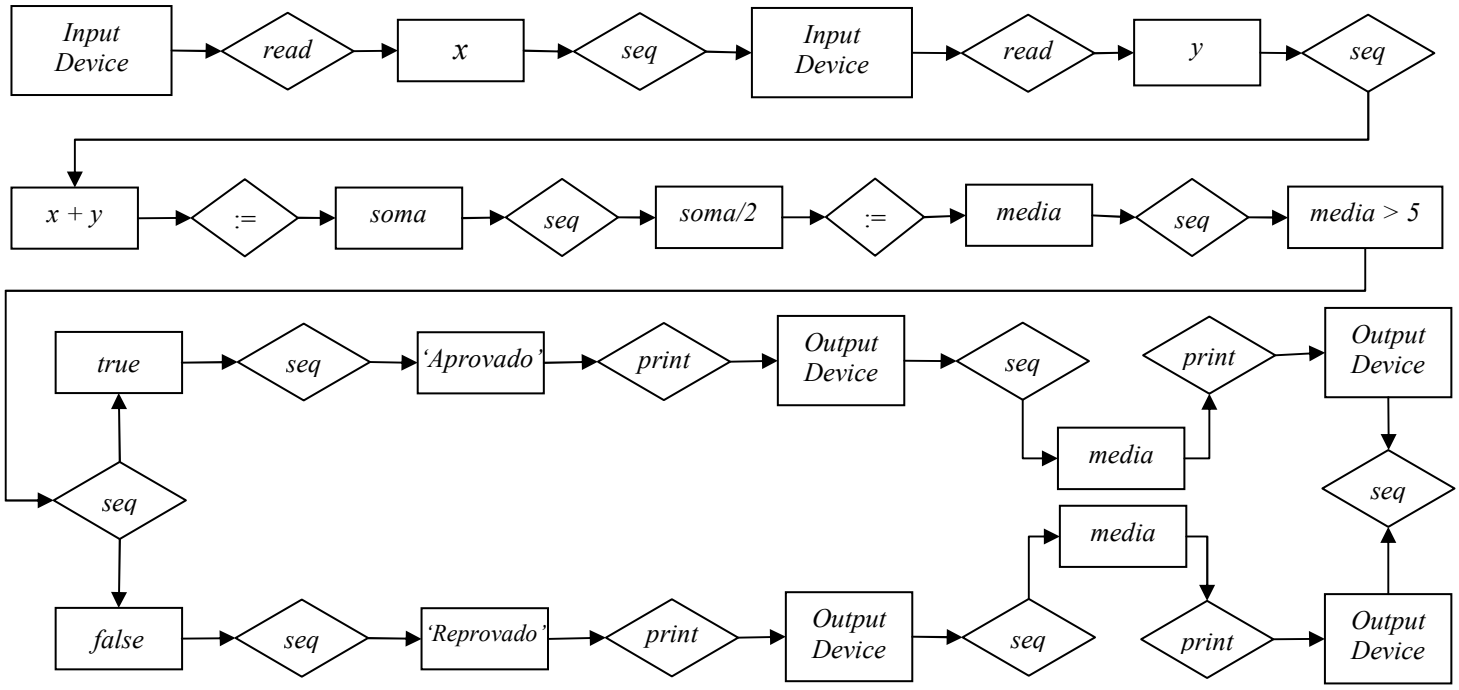
Para gerar o grafo a árvore é percorrida seguindo a ordem: avaliação do **nodo raiz**, avaliação da **subárvore mais a esquerda**, avaliação das **n-subárvores até a subárvore mais a direita**.

Se o nodo avaliado na AST é um dos comandos – *"v := e"*, ***"read(v)"***, ***"print(e)"***, ***"call** proc"*, *"v := **call** func"* e ***"exec sql** query"* – então o grafo conceitual recebe a

adição dos nós correspondentes ao subgrafo do comando correspondente. Caso o nodo avaliado na AST seja um dos comandos – *"**if** b **then** c **end**"*, *"**if** b **then** c_1 **else** c_2 **end**"*, *"**while** b **do** c **end**"* e *"**begin sql** c **end sql**"* – então as subárvores que iniciam com nodo *"Block"* são analisadas recursivamente da esquerda para a direita.

Para o exemplo da Figura 5.3, a árvore é mapeada para o grafo conceitual ilustrado na Figura 5.4. Note que para os comandos atribuição, *"soma := x+y;"* e *"media := soma/2;"*, as expressões aritméticas foram apresentadas como conceitos, ou seja, elas não foram representadas pelo subgrafo da adição e divisão, respectivamente. Isso se deve ao fato de ser desnecessário um detalhamento maior no grafo conceitual para a identificação das regras de negócio. Na próxima etapa da ferramenta, basta conhecer o conceito de expressão aritmética, não sendo necessário identificar o tipo de operação aritmética implementada na expressão.

Figura 5.4: Grafo conceitual gerado a partir da AST da Figura 5.3.



Fonte: Elaborado pelo autor.

5.3 Módulo de Análise Semântica

O módulo de análise semântica é usado para identificar os trechos de código que implementam as classes de regra de negócio definidas na Figura 4.14. Aqui, o grafo conceitual gerado pelo módulo de análise sintática é percorrido, identificando trechos que representam o comportamento das classes de regras de negócio definidas no Capítulo 4.

O módulo de análise semântica é composto por duas etapas: ***matching* das estruturas de regras de negócio**; e **clusterização e anotação das regras de negócio**.

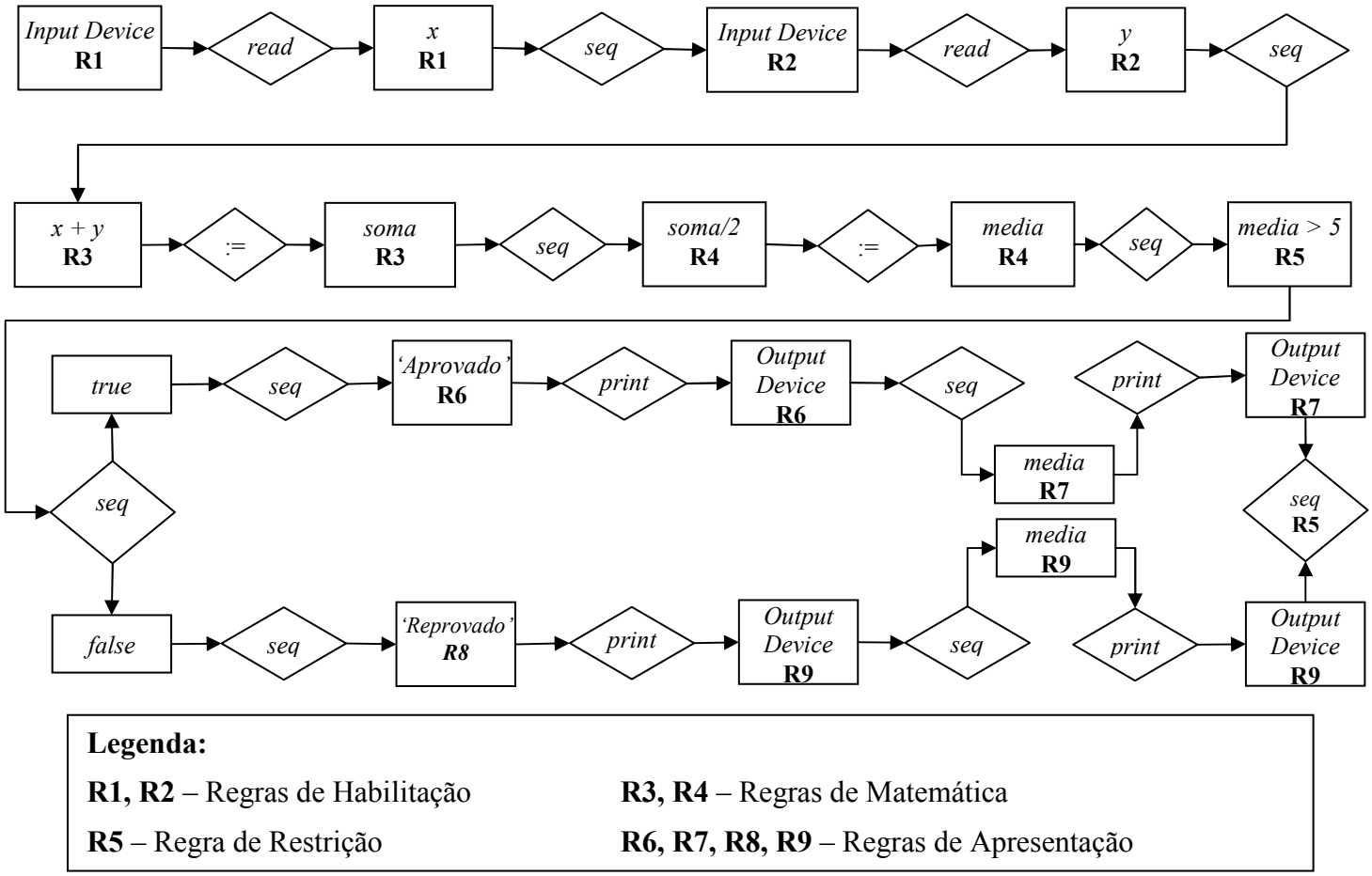
5.3.1 *Matching* das Estruturas de Regras de Negócio

A terceira etapa consiste em percorrer o grafo conceitual do código fonte, buscando por estruturas que representem o comportamento descrito para as classes de regra de negócio. A Seção 4.1 deste trabalho, define as classes de implementação de regras de negócio e os grafos conceituais que representam o comportamento destas classes. A ferramenta procura no grafo do código fonte, subgrafos idênticos aos grafos definidos para cada classe de implementação da Seção 4.1.

Por exemplo, para a classe *"Implementação da Regra de Restrição"* a ferramenta percorre o grafo conceitual do código fonte, identificando subgrafos com as mesmas estruturas definidas para essa classe na Figura 4.3(b) e (c).

Atualmente, a ferramenta percorre o grafo conceitual três vezes. Na primeira passagem são identificadas as regras de persistência e matemática. Já na segunda passagem, identifica-se os trechos que implementam regras de habilitação, apresentação, inicialização e execução. Por último, na terceira passagem, são identificadas as regras de derivação e restrição.

Figura 5.5: Resultado da identificação de trechos do grafo com regras de negócio.



Fonte: Elaborado pelo autor.

Essa ordem é importante, pois alguns tipos de regras, como a regra de persistência, devem se comportar como uma transação única e indivisível (transações atômicas). É muito comum encontrar comandos de controle, como o comando *"if"*, dentro de um

bloco de persistência (*"begin sql"*). Estes comandos de controle não podem ser considerados uma regra de derivação ou restrição, pois fazem parte do bloco de persistência que é tratado como uma transação única, ou seja, uma única regra de negócio.

Deste modo, após identificar as regras de persistência e matemática na primeira passagem pelo grafo, os nodos que formam estas regras são marcados para não serem avaliados nas próximas passagens pelo grafo conceitual do código fonte.

Ao identificar uma regra de negócio no grafo conceitual, a ferramenta marca o início e o fim desta regra, através da anotação da classe da regra e de um código alfanumérico que é adicionado ao componente inicial e final do seu subgrafo. A Figura 5.5 mostra o resultado da anotação das regras de negócio sobre o grafo conceitual apresentado na Figura 5.4.

Durante o processo de comparação das estruturas, podem existir trechos do grafo que não se enquadram em nenhuma das classes de regras de negócio definidas na ontologia (Figura 4.14). Estes trechos podem representar novas classes de regras que ainda não foram mapeadas para a ontologia. Neste caso, a ferramenta anota estes trechos como regras de negócio genéricas, definidas na ontologia pela classe *"Implementação da Regra de Negócio"*. A partir desta anotação, a ferramenta apresenta estes trechos a um especialista, que pode analisar se o trecho representa uma nova classe de regra de negócio, ou se ocorreu uma falha no *matching* das estruturas, ou ainda se o trecho não representa regras de negócio.

5.3.2 Clusterização das Regras de Negócio

Na última etapa a ferramenta avalia a granularidade das regras de negócio, buscando, agrupar duas ou mais regras de negócio em uma única regra com mais significado para o negócio. Por exemplo, uma sequência de cálculos será anotada no grafo com várias regras de negócio matemáticas, mas para o negócio a sequência de cálculos deveria ser uma única regra de negócio, um cálculo indivisível.

No exemplo da Figura 5.2, as operações de adição e divisão são anotadas no grafo conceitual da Figura 5.5 como regras matemáticas distintas (cada operação é uma regra de negócio matemática). Entretanto, o correto é agrupar as duas regras, considerando a existência de uma regra matemática para o cálculo da média. O mesmo ocorre com os comandos *"**print**('Aprovado')"* e *"**print**(media)"* que podem ser agrupados em uma única regra de negócio de apresentação, representando uma tela do sistema para apresentação dos resultados.

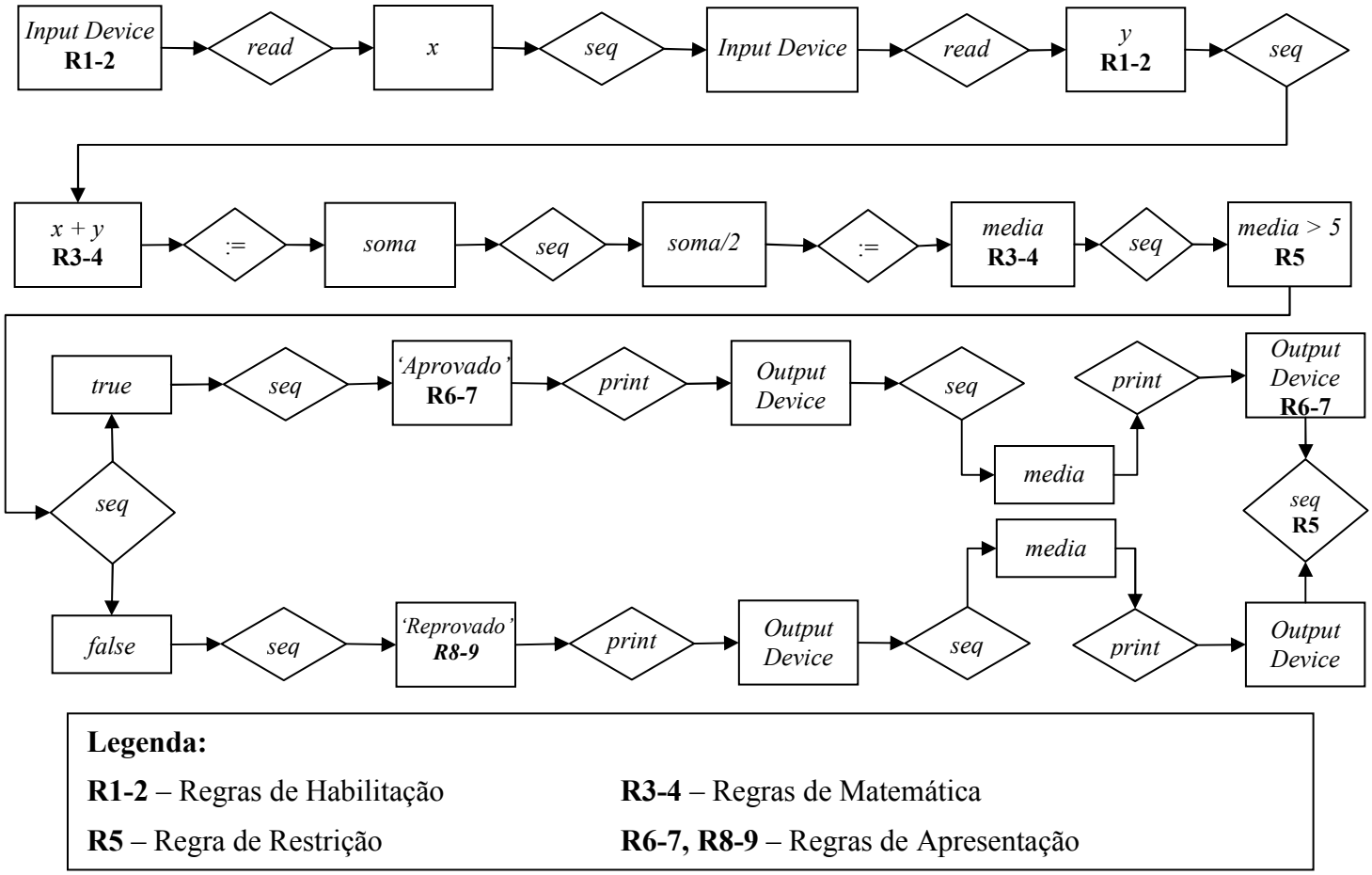
A Figura 5.6 mostra a anotação das regras no grafo conceitual após a clusterização das regras de negócio.

A clusterização das regras de negócio é realizada através da aplicação de heurísticas, que agrupam regras matemáticas, regras de apresentação, regras de inicialização, regras de habilitação, regras de derivação e regras de restrição. Além disso, também são definidas heurísticas que eliminam regras de execução e regras de restrição desnecessárias. Nesta tese foram definidas sete heurísticas:

- **Heurística 01 – Subrotinas implementadas no Código Fonte do Sistema Legado:** chamadas de funções, procedimentos ou métodos que estão definidos no próprio código fonte do sistema legado são descartadas. Durante a etapa de análise do grafo conceitual todas as chamadas de funções,

procedimentos ou métodos são anotadas como regras de execução. Entretanto, quando a função, procedimento ou método está codificada no próprio sistema legado, a sua chamada deve ser desconsiderada, pois em algum momento o corpo da função, procedimento ou método será anotado pela ferramenta.

Figura 5.6: Resultado da clusterização das regras de negócio no grafo conceitual.



Fonte: Elaborador pelo autor.

- **Heurística 02 – Invocações de Bibliotecas da Linguagem de Programação:** normalmente, as plataformas de desenvolvimento disponibilizam bibliotecas para realizar operações comuns no desenvolvimento de software. Por exemplo, biblioteca para manipular *strings*, com operações como concatenação, cálculo de tamanho, comparação, entre outras operações. Tais operações são usadas através de chamadas de funções ou métodos, sendo anotadas como regras de execução. Entretanto, essas invocações não representam execuções de procedimentos ou processos externos ao código fonte, como determina a definição de uma regra de execução. Tais operações manipulam uma determinada informação do sistema legado, assim como operações aritméticas. Deste modo, essas anotações de regras de execução são substituídas por anotações de regras matemáticas.

- **Heurística 03 – Regras de Negócio Vizinhas:** duas regras de negócio de habilitação encontradas em sequência no grafo conceitual são agrupadas em uma única regra. Na Figura 5.5, os comandos *"**read**(x)"* e *"**read**(y)"* aparecem em sequência no grafo conceitual e são anotadas como regras de habilitação. Desta forma, as duas ocorrências de regras de habilitação são unificadas em uma única regra, como mostra a Figura 5.6. Está heurística também é aplicada para regras de apresentação, regras de inicialização e regras de derivação.
- **Heurística 04 – Regras Matemáticas em Sequência:** duas operações aritméticas em sequência devem ser agrupadas, desde que exista uma dependência de variáveis entre as operações. Para regras matemáticas, além da vizinhança, também deve ser analisado as variáveis encontradas nas expressões aritméticas. Se existirem variáveis comuns nas duas ocorrências de regras matemáticas, então, as duas operações fazem parte da mesma regra de negócio. Na Figura 5.5, os comandos *"soma := x + y"* e *"media := soma/2"* são anotadas como regras matemáticas distintas. Porém, as regras ocorrem em sequência e possuem uma variável comum (*soma*). Portanto, devem ser agrupadas em uma única regra de negócio matemática, como mostra a Figura 5.6.
- **Heurística 05 – Variáveis de Domínio:** todas as regras de negócio devem obrigatoriamente manipular variáveis de domínio. As variáveis de domínio são:
    - Variáveis encontradas em comandos de entrada de dados;
    - Variáveis encontradas em comandos de saída de dados;
    - Parâmetros de funções, procedimentos ou métodos;
    - Variáveis usadas em retornos de funções ou métodos;
    - Variáveis cujo valor foi derivado dos valores de variáveis encontradas em comandos de entrada de dados ou parâmetros de funções, procedimentos e métodos;
    - Variáveis que influenciam no valor das variáveis encontradas em comandos de saída de dados ou retorno de funções e métodos.

  No código fonte apresentado na Figura 5.2, todas as variáveis são identificadas como variáveis de domínio. As variáveis x e y são encontradas nos comandos de entrada de dados (o usuário informa seu valor). Já a variável media é de domínio, pois é usada em um comando de saída de dados (resultado do programa). Por último, soma também é considerada variável de domínio, pois é derivada de x e y (variáveis de domínio) e também influencia no valor de media (outra variável de domínio).

  As anotações de trechos que não usam variáveis de domínio são desmarcadas (descartadas). Normalmente, são trechos de código usados para gerar a interface do sistema legado ou realizar algum tipo de controle interno. Deste modo, são trechos que não dizem respeito ao negócio da organização e devem ser desconsiderados na análise das regras de negócio.

- **Heurística 06 – Regras de Restrição transformadas em Regras de Derivação:** durante a aplicação das heurísticas anteriores irão ocorrer rearranjos das regras de negócio. Blocos condicionais antes anotados como regras de restrição podem vir a se transformar em regras de derivação, devido as mudanças de anotações. Como definido na ontologia, uma regra de derivação é caracterizada por um comando condicional seguido de comandos de atribuição de literais, ou seja, regras de inicialização. Se após a aplicação das heurísticas, essa condição for constata para algum comando condicional anotado como regra de restrição, então, a regra é transformada em regra de derivação.
- **Heurística 07 – Regras de Restrição Vazias:** após a aplicação das demais heurísticas, regras de restrição que não possuem outras regras codificadas internamente são descartadas. Ou seja, blocos condicionais ou de repetição que não codificam regras de negócio são desmarcados.

As heurísticas são aplicadas na ordem de precedência definida na Tabela 5.1 A tabela mostra a relação de precedência através dos símbolos “>”, “<” e “=”. O símbolo “>” representa que uma heurística tem precedência sobre outra. Por exemplo, “Heurística 02 > Heurística 03” representa que a segunda heurística tem precedência sobre a terceira heurística. Já o símbolo “<” representa que uma heurística dá precedência à outra (Ex.: Heurística 06 dá precedência a Heurística 05), enquanto “=” representa que as heurísticas tem a mesma precedência (Ex.: Heurística 01 tem a mesma precedência da Heurística 02).

Tabela 5.1: Relações de precedência entre as heurísticas.

	Heurística 01	Heurística 02	Heurística 03	Heurística 04	Heurística 05	Heurística 06	Heurística 07
Heurística 01		=	>	>	>	>	>
Heurística 02	=		>	>	>	>	>
Heurística 03	<	<		=	>	>	>
Heurística 04	<	<	=		>	>	>
Heurística 05	<	<	<	<		>	>
Heurística 06	<	<	<	<	<		>
Heurística 07	<	<	<	<	<	<	

Fonte: Elaborado pelo autor.

Após aplicar as sete heurísticas pela primeira vez, a ferramenta reinicia a aplicação das heurísticas, pois a aplicação de uma heurística pode habilitar a aplicação de outra. Por exemplo, a aplicação da heurística 05, pode habilitar a aplicação da heurística 03, que na primeira vez não causou nenhuma mudança na anotação das regras. A ferramenta reaplica as heurísticas até que nenhuma delas cause mudanças no conjunto de regras de negócio anotadas no grafo conceitual.

Ao final da clusterização a ferramenta anota o arquivo de código fonte legado, com as informações que constam no grafo conceitual, como apresentado na Figura 5.6. Esse é o resultado final da ferramenta, o arquivo de código fonte anotado.

6 MÉTODO DE DESCOBERTA DE PROCESSOS DE NEGÓCIOS

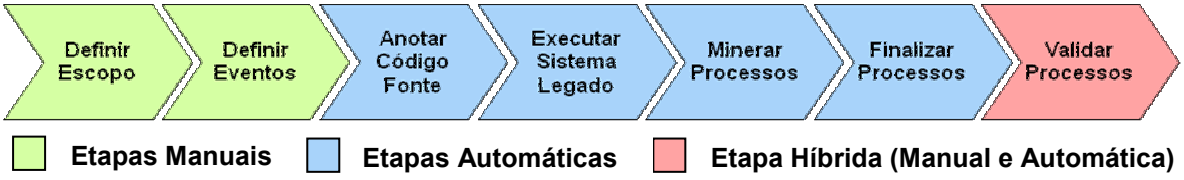
Neste capítulo é definido o método semi-automático para descoberta de processos de negócio codificados em sistemas legados. Conforme proposto na Seção 2.3, o método usa a técnica de anotação das regras de negócio detalhada nos capítulos anteriores, juntamente, com técnicas de mineração de processos a partir de *logs* de eventos do sistema legado.

É importante ressaltar, que um sistema legado pode implementar mais de um processo de negócio. Portanto, o método não propõe necessariamente o mapeamento do sistema legado para um único processo de negócio. Da mesma forma, um sistema legado pode fazer parte de um processo, mas não implementá-lo por completo, ou seja, algumas atividades do processo podem ser executadas fora do sistema legado.

Assim, algumas etapas do método necessitam obrigatoriamente, de interação com os usuários que utilizam o sistema legado na organização. Portanto, a identificação dos processos de negócio no sistema legado, será constituída de algumas etapas que exigem a intervenção humana, e outras que podem ser automatizadas por algoritmos. Por este motivo o método apresentado na tese é semi-automático, pois há necessidade de intervenção humana em algumas etapas da identificação.

O método é composto por sete etapas, que devem ser aplicadas para cada processo de negócio codificado no sistema legado. O profissional que aplicar o método no sistema legado (chamado a partir daqui de projetista), deve obrigatoriamente ter uma noção básica sobre os processos de negócio que podem estar codificados neste sistema. Deve ser uma pessoa com conhecimento mínimo sobre o sistema legado. O projetista deve aplicar as etapas descritas a seguir, para cada identificação de processo de negócio que deseja realizar no sistema legado. A Figura 6.1 apresenta as etapas do método.

Figura 6.1: Etapas do método de descoberta de processos de negócio.



Fonte: Elaborado pelo autor.

6.1 Definição do Escopo do Processo

Esta é a primeira etapa na identificação de um processo de negócio em código fonte legado. Aqui, o projetista deve definir a meta de negócio que é executada pelo processo dentro da organização. O projetista também irá definir o domínio de aplicação do processo de negócio, além de uma lista de palavras chaves sobre o domínio de aplicação deste processo.

Por exemplo, a organização possui um sistema legado para controle de vendas, e deseja-se saber como a ordem de venda é realizada dentro da organização. Para isso, o projetista pode definir como meta do processo a ser identificado, a emissão da ordem de venda. Este processo encontra-se dentro do departamento de vendas, que é o domínio de aplicação do processo e suas palavras chaves são: produto; estoque; cliente; crédito; entre outras.

Estas informações podem ser usadas por um algoritmo para encontrar possíveis arquivos de código fonte que devem ser anotados para a descoberta do processo de negócio.

Um sistema legado é codificado em vários arquivos de código fonte. Cada arquivo pode implementar várias subrotinas do sistema legado. Deste modo, um algoritmo pode usar as palavras chaves para encontrar subrotinas e arquivos que devem ser considerados na anotação das regras de negócio. Por exemplo, através da palavra chave estoque, um algoritmo pode encontrar um procedimento nomeado *"baixarEstoque"*, apontando automaticamente este arquivo como candidato para anotação das regras de negócio.

Além disso, as palavras chaves podem ser usadas para documentar o modelo de processo de negócio após a descoberta do mesmo no código fonte. O modelo de processo obtido a partir do código fonte legado terá rótulos próximos ao nível de programação. Portanto, podem-se usar as palavras chaves para substituir os rótulos das atividades, por palavras mais próximas do negócio da organização.

Também é possível criar uma base de conhecimento para auxiliar o projetista em futuras execuções do método. Na primeira vez que o projetista identificar um processo para o domínio de vendas, a ferramenta pode armazenar o conhecimento adquirido ao longo da identificação do processo de negócio. Desta forma, na segunda identificação de processo dentro do mesmo domínio de aplicação, a ferramenta pode usar o conhecimento adquirido para auxiliar o projetista na nova construção.

6.2 Definição dos Eventos Iniciais e Finais

Na segunda etapa, o projetista aponta como o processo de negócio inicia e finaliza. O projetista deve informar quais eventos no sistema legado são necessários para o processo de negócio iniciar e terminar sua execução.

O evento inicial do processo de negócio pode ser uma tela do sistema legado, ou mesmo o recebimento de um arquivo ou mensagem pela organização. Seguindo o exemplo da ordem de venda, o evento inicial para disparar este processo pode ser um operador do sistema acessar o *menu* Ordens de Venda e selecionar a opção *Criar uma Nova Ordem de Venda*.

Do mesmo modo, o projetista identifica o evento ou condição final do processo de negócio. Em nosso exemplo, o evento final pode ser a gravação da ordem em tabelas do

banco de dados, ou mesmo, o operador do sistema receber na tela uma mensagem de sucesso na criação da ordem de venda.

É importante observar que podem existir mais de um evento inicial ou final. Neste caso o projetista deve identificar todos os eventos iniciais e finais envolvidos no processo de negócio, pois é provável que haja atividades que não são compartilhadas pelos caminhos seguidos após a ocorrência dos eventos iniciais.

Esta etapa ajuda a restringir o escopo de pesquisa sobre o código fonte. Através dos eventos iniciais e finais, podemos restringir os arquivos do código fonte que serão analisados pela ferramenta de identificação das regras de negócio.

Por fim, o projetista também deve definir um identificador de instância, que será usado pela ferramenta de identificação de regras de negócio. O identificador de instância funciona como uma chave primária, uma informação com valor único em cada execução do processo de negócio. Este valor é importante para identificar uma instância de processo no *log* de eventos.

Observe que as duas primeiras etapas do método precisam de intervenção humana, ou seja, é necessário apoio de um usuário especialista no sistema legado, para auxiliar delimitar o escopo do processo de negócio a ser identificado no sistema legado.

6.3 Identificação das Regras de Negócio

A terceira etapa do método consiste em usar a ferramenta de identificação de regras de negócio definida no Capítulo 5. Nesta etapa, o projetista submete à ferramenta de identificação de regras, todos os arquivos de código fonte do sistema legado selecionados nas etapas um e dois do método.

Além dos arquivos, o projetista também informa à ferramenta o identificador de instância do processo de negócio. O identificador de instância foi definido na etapa anterior do método.

Através destas informações, a ferramenta de identificação de regras delimita os trechos de código que implementam regras de negócio, anotando os pontos início e término das codificações das regra de negócio. Essa anotação é uma linha de código responsável por gerar uma entrada de dados no arquivo de *log* do sistema legado.

A Figura 6.2, mostra como as regras de negócio são anotadas pela ferramenta apresentada no Capítulo 5. Nesta figura os comandos *fprint* são anotações adicionadas ao código fonte pela ferramenta de identificação de regras de negócio. Note também, que cada anotação usa a variável *order_number* como identificador de instância do processo. No exemplo citado anteriormente, da ordem de venda, o número da ordem irá identificar uma execução do processo de negócio no sistema legado.

É importante observar que a anotação das regras de negócio pode levar em consideração outros dados, além do identificador de instância e das marcas de início e fim de cada regra. Como por exemplo, a informação do usuário que disparou a execução da regra de negócio. Isso permite ao projetista descobrir o papel que é responsável pela execução deste trecho do sistema legado. Para fazer isso, basta acrescentar mais uma entrada na anotação, com a informação de usuário *"logado"* no sistema legado (ex.: *fprint(logfile, "%d – BEGIN RULE 01 - %s", order_number, username);*).

Figura 6.2: Anotação das regras de negócio no arquivo de código fonte.

```
fprinf(arqlog, "%d – INICIO REGRA HABILITACAO 1", numeroOrdem);
printf("PRODUTO/PREÇO: ");
scanf("%s", produto);
scanf("%f", preco);
fprinf(arqlog, "%d – FIM REGRA HABILITACAO 1", numeroOrdem);

fprinf(arqlog, "%d – INICIO REGRA EXECUCAO 2", numeroOrdem);
aprovado = fin.creditoAnalise(cliente);
fprinf(arqlog, "%d – FIM REGRA EXECUCAO 2", numeroOrdem);

fprinf(arqlog, "%d – INICIO REGRA RESTRICAO 3", numeroOrdem);
if (aprovado && preco > 1000) {

  fprinf(arqlog, "%d – INICIO REGRA MATEMATICA 4", numeroOrdem);
  novoPreco =   preco - (preco * 0.10);
  fprinf(arqlog, "%d – FIM REGRA MATEMATICA 4", numeroOrdem);

} else if  (aprovado) {

  fprinf(arqlog, "%d – INICIO REGRA MATEMATICA 5", numeroOrdem);
  novoPreco =   preco - (preco * 0.05);
  fprinf(arqlog, "%d – FIM REGRA MATEMATICA 5", numeroOrdem);

} else {

  fprinf(arqlog, "%d – INICIO REGRA EXECUCAO 6", numeroOrdem);
  Email.enviar(cliente, "crédito recusado!");
  fprinf(arqlog, "%d – FIM REGRA EXECUCAO 6", numeroOrdem);
  return;
}
fprinf(arqlog, "%d – FIM REGRA RESTRICAO 3", numeroOrdem);

fprinf(arqlog, "%d – INICIO REGRA PERSISTENCIA 7", numeroOrdem);
Query q(db);
q.execute ("INSERT INTO ordens VALUES ()");
fprinf(arqlog, "%d – FIM REGRA PERSISTENCIA 7", numeroOrdem);
```

Fonte: Elaborado pelo autor.

6.4 Execução do Sistema Legado

Agora que o sistema foi anotado, a próxima etapa é executar o sistema legado com objetivo de gerar o arquivo de *log* para posterior mineração. A execução do sistema pode ocorrer em ambiente produção, ou seja, simplesmente deixar os usuários executarem o sistema legado no dia a dia da organização.

Entretanto, a execução em ambiente de produção pode atrasar a aplicação do método, pois podem ser necessários vários dias ou semanas para obter um arquivo de *log* com informações suficientes para a etapa de mineração do processo.

Para agilizar a geração do arquivo de *log*, o projetista pode definir cenários de testes do sistema legado. Os cenários de teste podem ser executados em ambiente de testes, um ambiente controlado, que gera o arquivo de *log* com mais velocidade.

Um cenário de teste consiste de um conjunto de interações dos usuários, com as interfaces do sistema legado. Para cada interface são definidos os dados que devem ser digitados pelo usuário. Desta forma, a sequência de interfaces e os respectivos dados digitados pelo usuário formam um cenário de teste para execução do sistema legado.

Os cenários de testes devem ser definidos por usuários especialistas do sistema legado. Estes usuários são entrevistados para elaborar uma base de dados, com as informações mais utilizadas em cada interface do sistema. Através destas informações, o projetista elabora os cenários de teste para execução do sistema legado.

Após a criação dos cenários de teste, os mesmos devem ser repassados aos usuários especialistas e executados no sistema legado. De acordo com a tecnologia usada no sistema legado, a execução dos cenários pode ser automatizada. Por exemplo, se o sistema legado for construído com tecnologia *web*, então é possível usar uma ferramenta como, Apache JMeter (APACHE, 2012), para executar automaticamente os cenários de teste.

Após a execução do sistema legado é obtido um arquivo de *log* semelhante ao arquivo ilustrado na Figura 6.3.

Figura 6.3: Arquivo de *log* obtido após a execução do sistema legado.

```
1034 – INICIO REGRA HABILITACAO 1
1034 – FIM REGRA HABILITACAO 1
1034 – INICIO REGRA EXECUCAO 2
1034 – FIM REGRA EXECUCAO 2
1034 – INICIO REGRA RESTRICAO 3
1036 – INICIO REGRA HABILITACAO 1
1036 – FIM REGRA HABILITACAO 1
1034 – INICIO REGRA MATEMATICA 5
1034 – FIM REGRA MATEMATICA 5
1036 – INICIO REGRA EXECUCAO 2
...
```

Fonte: Elaborado pelo autor.

6.5 Mineração do Arquivo de Log

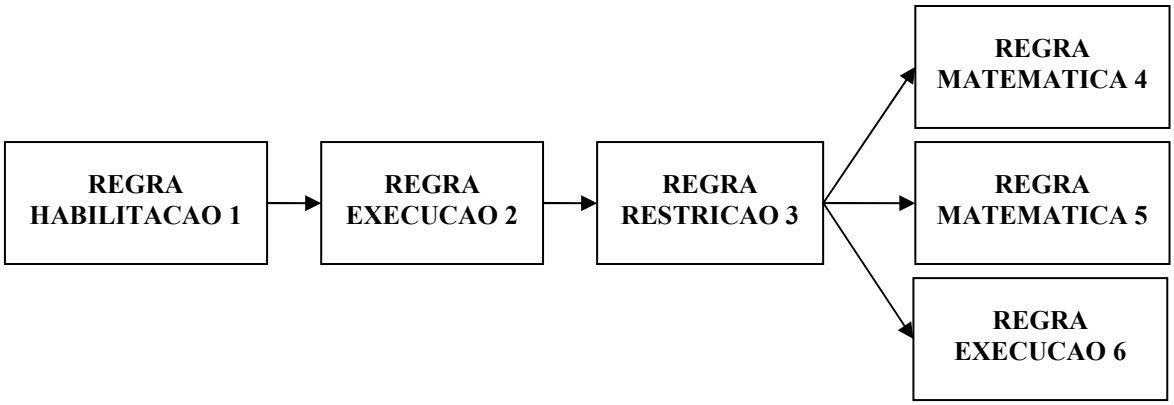
Nesta etapa o arquivo de *log* é minerado usando o algoritmo *Incremental Miner* (KALSING, et. al., 2010a; KALSING, et. al., 2010b). O *Incremental Miner* foi desenvolvido no grupo de pesquisa e é resultado da dissertação de mestrado do aluno André Kalsing. Este algoritmo extrai fragmentos de processos de negócio a partir dos arquivos de *log* com formato apresentado na Figura 6.3.

A principal vantagem do algoritmo definido por Kalsing et. al. (2010b) é a mineração incremental do processo de negócio. Isso significa que o processo pode ser alterado de forma incremental. O algoritmo permite executar a mineração do arquivo de *log* e obter um primeiro modelo do processo de negócio. Logo após a primeira mineração, continua-se a executar cenários de teste, gerando novas entradas de dados no arquivo de log. Após gerar um novo arquivo de *log*, executa-se o *Incremental Miner* novamente. Na nova mineração o algoritmo irá considerar apenas as novas entradas no arquivo de *log*, alterando o modelo do processo já existente, de acordo com as novas entradas encontradas no *log*. Ou seja, o algoritmo não realiza a mineração de todo o *log*

novamente, mas sim, considera apenas as novas entradas de dados geradas na última execução do sistema legado.

O *Incremental Miner* extrai um grafo direcionado que representa a ordem parcial de execução das regras de negócio. A Figura 6.4 mostra um exemplo de fragmento de processo extraído por este algoritmo de mineração.

Figura 6.4: Ordem parcial de regras de negócio extraída pelo *Incremetal Miner*.



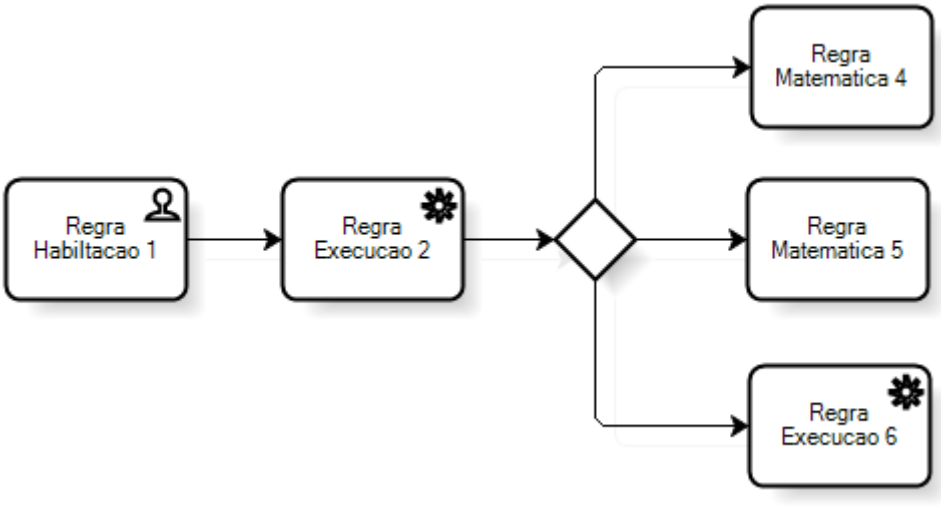
Fonte: Elaborado pelo autor.

O *Incremental Miner* e a ferramenta de suporte foram implementadas usando a linguagem de programação Java. O algoritmo foi publicado com mais detalhes em (Kalsing et. al., 2010a) e (Kalsing et. al., 2010b).

6.6 Finalização do Modelo de Processo de Negócio

Após a mineração do log é obtida uma ordem parcial da execução das regras de negócio no sistema legado. A próxima etapa consiste em substituir cada regra de negócio pela respectiva estrutura de processo que implementa o tipo de regra correspondente em um processo de negócio.

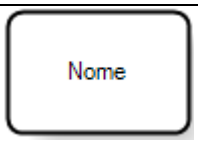
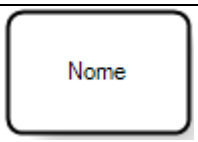
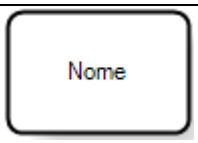
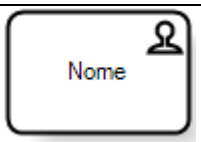
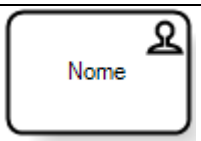
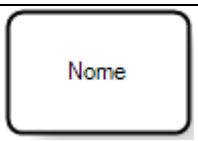
Figura 6.5: Fragmento de processo após a substituição das regras de negócio.



Fonte: Elaborado pelo autor.

A Figura 6.5 mostra a operação realizada sobre a ordem parcial de regras de negócio apresentada na Figura 6.4. Note que cada regra é substituída por um componente de processo que implementa o respectivo tipo de regra em um modelo de processo de negócio. A Tabela 6.1 mostra o componente de processo correspondente para cada tipo de regra de negócio definido em nossa ontologia.

Tabela 6.1: Implementação das classes de regras de negócio em BPMN.

Classe de Regra de Negócio	Componente BPMN	Representação Gráfica
Regra de Restrição	Gateway Exclusivo	
Regra Matemática	Tarefa Automática	
Regra de Persistência	Tarefa Automática	
Regra de Derivação	Tarefa Automática	
Regra de Habilitação	Tarefa Humana	
Regra de Apresentação	Tarefa Humana	
Regra de Inicialização	Tarefa Automática	
Regra de Execução	Tarefa de Serviço	

Fonte: Elaborado pelo autor.

Além disso, outras configurações ainda podem ser feitas no modelo, como por exemplo, a configuração dos papéis que executam as atividades de negócio. Como mencionamos anteriormente (Seção 6.3), o usuário responsável por disparar a execução da regra de negócio, deve ser *logado* como uma informação adicional no arquivo de *log*.

Neste caso, o projetista deve observar quem executou as regras, atribuindo seu papel as atividades correspondentes.

6.7 Validação do Modelo de Processo de Negócio

A última etapa do método apresenta um relatório com informações que permitem verificar se o processo de negócio está correto ou não.

Durante a aplicação do método, as ferramentas coletam o maior número possível de informações para garantir a validade do processo de negócio identificado pelo método. São coletadas estatísticas que podem ser avaliadas pelo projetista para identificar possíveis falhas durante a aplicação do método.

As informações apresentadas no relatório são:

- Número de regras de negócio mapeadas no processo de negócio;
- Quantidade de regras de negócio anotadas no código fonte e que não foram mapeadas para o processo de negócio. Aqui são apresentadas as regras de negócio anotadas no código fonte, mas que não constam no modelo de processo descoberto na etapa de mineração do processo. Estas regras podem representar trechos do processo que não foram executados durante a aplicação do método (não geraram eventos no log), ou ainda, trechos que estão obsoletos, fora de uso na organização.
- Número de trechos do código fonte descartados durante a identificação das regras de negócio. São apresentados os trechos de código que a ferramenta não identificou como regra de negócio, por isso, não constam no modelo de processo identificado na etapa de mineração;
- Grau de confiança em cada transição de atividades. Ou seja, do número total de instâncias de processo encontradas no log de eventos, quantas destas instâncias possuem uma determinada transição de atividades. Por exemplo, um determinado processo possui 15 instâncias registradas no arquivo de log. Destas instâncias, em 13 existe a transição da atividade "A" para o gateway "B". Sendo assim, a confiança na correção desta transição é de 86% (13/15 = 0,86).
- Quantidade de *"livelocks"* existentes no modelo de processo de negócio encontrado na etapa de mineração do processo. *Livelocks* são trechos do processo que não conseguem atingir um evento final, mas que permanecem em um ciclo de execução infinito. Isto é, o processo permanece sempre em execução, mas não pode atingir um evento final.
- Quantidade de "*deadlocks*" existentes no modelo de processo de negócio encontrado na etapa de mineração do processo. Ao contrário do "*livelock*", um "*deadlock*" é um trecho do processo que não atinge um evento final, mas que não permanece em execução, ou seja, o processo fica bloqueado em uma determinada atividade.

Através da análise destes números, o projetista pode verificar a qualidade do processo de negócio identificado pelo método.

7 ESTUDOS DE CASO

Para ilustrar o uso do método, nós apresentamos quatro estudos de caso executados em sistemas legados das áreas de telemedicina, recursos humanos e engenharia civil. Ao todo foram analisados aproximadamente 120 arquivos, totalizando mais 140 mil linhas de código fonte, em sistemas codificados nas linguagens de programação C# e PHP.

Estes sistemas foram escolhidos para os estudos de caso, devido ao fato de possuírem documentação de alguns processos de negócio, além de existir desenvolvedores especialistas em seus códigos fontes. Deste modo, os processos de negócio identificados através do método, foram devidamente validados pelos desenvolvedores de cada sistema legado.

Os estudos de casos permitem analisar a eficiência do método proposto nesta tese. Para isso foi utilizada a seguinte metodologia:

1º. Foram modelados manualmente os principais processos de negócio existentes nos sistemas legados. Em um dos sistemas estudados, os processos estavam desenhados na sua documentação. Para os demais sistemas, usuários especialistas modelaram os principais processos de negócio executados através destes sistemas legados. Este conjunto de modelos foi chamado de conjunto sucesso. Isto é, os métodos de identificação de processos deveriam obter modelos similares ao desenhados no conjunto sucesso. Quanto maior a similaridade, melhor será a eficiência do método.

2º. Desenvolvedores com conhecimentos nas linguagens de programação C e PHP, mas sem conhecimento no negócio relacionado a estes sistemas, realizaram a identificação manual dos processos de negócio codificados. Estes desenvolvedores usaram apenas seu conhecimento na linguagem de programação para analisar o código fone e descobrir os processos de negócio codificados nos sistemas.

3º. Os processos de negócio foram extraídos automaticamente usando as técnicas propostas em Di Francescomarino, Marchetto e Tonella (2009) e Perez-Castillo et. al. (2011).

4º. Os processos de negócio foram identificados automaticamente usando o método proposto nesta tese.

5º. Finalmente, os modelos de processos de negócio encontrados através de nosso método, das técnicas encontradas na literatura e da identificação manual foram comparados com os modelos existentes no conjunto sucesso. Através desta comparação, foi possível evidenciar a eficiência do método proposto na tese em

relação às demais técnicas automáticas e a identificação manual realizada por um desenvolvedor de sistemas.

O restante do capítulo apresenta os resultados obtidos para cada estudo de caso.

7.1 Sistema de Informações Médicas

Este sistema de informação registra todos os atendimentos realizados por um plano de saúde a pacientes com doenças crônicas (ex.: diabetes). Os atendimentos podem ser:

- De rotina: atendimento telefônico, pré-agendado pelo médico responsável, onde uma enfermeira executa uma série de perguntas (protocolo de atendimento) com objetivo de monitorar o quadro clínico do paciente;
- Visita: atendimento realizado por uma enfermeira na casa do paciente. O objetivo aqui é medir informações clínicas, como pressão arterial, batimentos cardíacos, glicose, entre outros;
- Receptivo: o paciente liga para o plano de saúde solicitando informações médicas ou relatando algum problema que está ocorrendo;
- Emergência Hospitalar: o paciente foi atendido emergencialmente em um hospital atendido pelo plano de saúde. Todas as informações dos procedimentos hospitalares são registrados no sistema.

Deste modo, o sistema de informação implementa ao menos quatro processos de negócio, ou seja, um para cada tipo de atendimento. Ambos os processos de negócio foram modelados previamente por usuários especialistas do sistema. Estes modelos foram considerados corretos, ou seja, os métodos de identificação de processos devem obter modelos semelhantes aos modelados pelos usuários especialistas.

Para este estudo de caso, os processos de negócio foram identificados através do método proposto nesta tese, através da técnica proposta por Perez-Castillo et. al. (2011) e através de identificação manual realizada por um desenvolvedor especialista em linguagem de programação C# (mas sem conhecimento do negócio do sistema). Para este sistema não foi possível empregar a técnica proposta por Di Francescomarino, Marchetto e Tonella (2009), pois essa técnica é aplicável apenas a sistemas desenvolvidos com tecnologia web.

O sistema de informação foi desenvolvido na linguagem de programação C#, usando arquitetura cliente-servidor. Neste estudo foram analisados 80 arquivos, totalizando aproximadamente 45 mil linhas de código fonte. A análise destes arquivos foi dividida em três frentes de trabalho:

1. Um desenvolvedor C# analisou os 80 arquivos manualmente, buscando modelar os processos de negócio conforme seu conhecimento na linguagem de programação;
2. Os 80 arquivos foram instrumentalizados usando a ferramenta de identificação de regras de negócio proposta nesta tese;
3. Os 80 arquivos foram instrumentalizados manualmente, por um desenvolvedor C#, conforme a técnica proposta por Perez-Castillo et. al. (2011). Neste caso a instrumentação foi manual, pois não foi encontrada

nenhuma ferramenta para automatizar a instrumentação proposta por Perez-Castillo et. al. (2011).

7.1.1 Preparação para Instrumentação do Código Fonte

Para a instrumentação dos arquivos foi necessário criar um pequeno algoritmo para enviar eventos para um arquivo de log. A Figura 7.1 mostra a classe criada para controlar a gravação de eventos em um arquivo de log.

Figura 7.1: Algoritmo para gerar log de eventos.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
namespace Abstracts {
  public class ProcessLog {
    public static int USUARIO_COD = 0;
    public static String INSTANCIA_ID = "0";
    public static String NOME_PROCESSO = "Default";

    public static void iniciarProcesso(int pescod, int usucod,
                                       string tipoAtendimento) {
      try {
        INSTANCIA_ID = pescod+"_"+usucod+"_"+
                       DateTime.Now.ToString();
        NOME_PROCESSO = tipoAtendimento;
        USUARIO_COD = usucod;
      } catch (Exception e) {
        Console.WriteLine(e.Message);
      }
    }


    public static void escreverEvento(string idRegraNegocio,
                                      byte tipoRegraNegocio) {
      try {
        if (INSTANCIA_ID != "0" && !INSTANCIA_ID.Equals("0")) {
          StreamWriter file = new StreamWriter("Eventos.log",
                                          true, Encoding.ASCII);
          file.WriteLine(INSTANCIA_ID+";"+NOME_PROCESSO+";"+
                         USUARIO_COD+";"+idRegraNegocio+";"+
                tipoRegraNegocio+";"+DateTime.Now.ToString());
          file.Close();
        }
      } catch (Exception e) {
        Console.WriteLine(e.Message);
      }
    }

    public static void terminarProcesso() {
      NOME_PROCESSO = "Default";
      INSTANCIA_ID = "0";
    }
  }
}
```

Fonte: Elaborado pelo autor.

A classe apresentada na Figura 7.1 possui três métodos:

- **inicarProcesso**: este método inicia a execução de um processo de negócio. Ele é executado ao abrir um novo atendimento que caracteriza o início de uma nova instância de processo no sistema. Sendo assim, o método gera o identificador de instância, usando código do paciente a ser atendido ("*pescod*"), código do usuário que presta o atendimento ("*usucod*") e data/hora da abertura do atendimento ("*sysdate*"). Além disso, também é identificado o nome do processo a ser executado ("*tipoAtendimento*"), ou seja, qual dos quatro atendimentos citados anteriormente foi iniciado pelo usuário.
- **escreverEvento**: esse método grava eventos de negócio no arquivo de log do sistema. Para o método proposto nesta tese é enviado para o log um identificador de regra de negócio ("*idRegraNegocio*") e o tipo de regra de negócio ("*tipoRegraNegocio*"), conforme ontologia proposta na Figura 4.14. Além disso, também são gravadas as informações de instância ("*INSTANCIA_ID*"), nome do processo ("*NOME_PROCESSO*") e usuário que executa a instância ("*USUARIO_COD*").
- **terminarProcesso**: ao finalizar um atendimento este método é executado, colocando o valor "0" no identificador de instância. Enquanto o valor de instância é "0" não são gravados eventos no arquivo de log.

Como o sistema de informação é executado na arquitetura cliente-servidor, para cada estação cliente é gerado um arquivo de log, ou seja, são gerados vários arquivos de log no ambiente de produção (são quatro estações de trabalho – quatro arquivos de log). Em uma estação de trabalho é executado apenas um atendimento por vez, ou seja, não são permitidas instâncias paralelas em uma estação de trabalho. Posteriormente, na fase de mineração, os arquivos de log gerados nas estações de trabalho são unificados em um único arquivo para executar a mineração.

Para o método proposto por Perez-Castillo et. al. (2011), a única mudança na classe da Figura 7.1 é a parametrização do método "*escreverEvento*". Neste caso, o método possui apenas um parâmetro, o identificador da subrotina executada no sistema ("*nomeSubRotina*"). Portanto, o nome da subrotina substitui as informações de regra de negócio no log de eventos. Os demais métodos permanecem idênticos.

É importante ressaltar, que as informações para a criação da classe apresentada na Figura 7.1 (instância, nome do processo, etc), assim como a identificação dos arquivos de código fonte que necessitam de instrumentação, foram obtidas a partir das etapas de definição de escopo e definição dos eventos iniciais e finais do método proposto nesta tese (Seções 6.1 e 6.2). Ou seja, através de entrevistas com usuários do sistema foram identificadas as informações iniciais, que permitiram a instrumentação do código fonte.

7.1.2 Instrumentação do Código Fonte

A Figura 7.2 apresenta exemplos de anotações realizadas no código fonte do sistema de informação. A Figura 7.2(a) mostra um exemplo de regra de negócio instrumentalizada usando a ferramenta proposta nesta tese. Já a Figura 7.2(b) mostra um exemplo de anotação de chamadas de subrotina, técnica proposta por Perez-Castillo et. al. (2011).

Note na Figura 7.2, que as instrumentalizações consistem em adicionar uma invocação ao método "***escreverEvento***", apresentado na Figura 7.1. Observe também que a Figura 7.2(c) mostra a invocação do método "***iniciarProcesso***", que foi adicionado aos eventos iniciais de cada tipo de atendimento prestado pelo plano de saúde. O mesmo ocorre com os eventos finais, onde é adicionado a invocação do método "***terminarProcesso***". Ambos os eventos são identificados na segunda etapa do método, conforme apresentado na Seção 6.2.

Figura 7.2: Exemplos de instrumentação do código fonte.

```
...
MotivoCancelamento motivoCancelamento = null;

ProcessLog.escreverEvento("MotivoCancelamento.getById.0", 1);
try {

  ProcessLog.escreverEvento("MotivoCancelamento.getById.1", 3);
  conn = ConnectionLocator.getInstance().getConnection();
  cmd = ConnectionLocator.getInstance().getCommand();
  cmd.Connection = conn;
  cmd.CommandType = CommandType.StoredProcedure;
  cmd.CommandText = SQL_PROC_GET_BY_ID;
  cmd.Parameters.AddWithValue("@MOVCOD", id);
  dr = ConnectionLocator.getInstance().getDataReader(cmd);
  ...
```

(a) Exemplo de anotação com a ferramenta apresentada nesta tese.

```
...
MotivoCancelamento motivoCancelamento = null;

try {
  conn = ConnectionLocator.getInstance().getConnection();
  cmd = ConnectionLocator.getInstance().getCommand();
  cmd.Connection = conn;
  cmd.CommandType = CommandType.StoredProcedure;
  cmd.CommandText = SQL_PROC_GET_BY_ID;
  cmd.Parameters.AddWithValue("@MOVCOD", id);

 ProcessLog.escreverEvento("MotivoCancelamento.getDataReader");
  dr = ConnectionLocator.getInstance().getDataReader(cmd);
  ...
```

(b) Exemplo de Anotação com a Técnica de Perez-Castillo et. al. (2011).

```
...
ProcessLog.iniciarProcesso(this.pessoa.getId(),
                  Global.UsuarioSessao.getId(),
                  "Atendimento_" + cmbTpChamada.SelectedIndex);

ProcessLog.escreverEvento("frmAtendimento.frmAtendimento.0",5);
//Configura a campanha de acordo com o tipo
this.configuraCampanha();
...
```

(c) Método "***inicarProcesso***" para criar uma nova instância de processo.

Fonte: Elaborado pelo autor.

7.1.3 Execução do Sistema de Informação

Após as instrumentações do código fonte, o sistema foi recompilado gerando duas versões do sistema. A primeira versão contendo a instrumentação do código fonte realizada com o método proposto nesta tese e a segunda versão contendo a instrumentação proposta por Perez-Castillo et. al. (2011).

A primeira versão foi disponibilizada no ambiente de produção do plano de saúde, onde atendimentos reais são registrados diariamente. Como mencionado anteriormente, o sistema executa na arquitetura cliente-servidor, com quatro estações de trabalho clientes. Cada estação cliente gera um arquivo de log e executa apenas um atendimento a cada tempo. Isto é, a estação cliente não permite realizar dois ou mais atendimentos ao mesmo tempo.

A primeira versão do sistema executou por três meses no ambiente de produção, gerando arquivos com aproximadamente 10 mil eventos em cada estação de trabalho cliente. A Figura 7.3 mostra um fragmento do arquivo de log gerado no ambiente de produção, usando a instrumentação de código proposta nesta tese.

Figura 7.3: Fragmento de log de eventos.

```
...
1215_1607_2012100315:43:55;Atendimento_0;1607;MedicoDAO.getById
.1;3;2012-10-03 15:44:40.843
1215_1607_2012100315:43:55;Atendimento_0;1607;frmCadPessoaMedic
o.ValidaFormulario.0;4;2012-10-03 15:45:20.013
1215_1607_2012100315:43:55;Atendimento_0;1607;PessoaMedicoDAO.i
nsert.1;3;2012-10-03 15:45:20.843
1215_1607_2012100315:43:55;Atendimento_0;1607;ConsultaDAO.inser
t.1;3;2012-10-03 15:45:20.843
1215_1607_2012100315:43:55;Atendimento_0;1607;ObjetivoAtividade
DAO.getById.1;3;2012-10-03 15:45:20.857
1215_1607_2012100315:43:55;Atendimento_0;1607;ObjetivoAtividade
DAO.update.1;3;2012-10-03 15:45:20.857
...
```

Fonte: Elaborado pelo autor.

Por sua vez, a segunda versão do sistema, instrumentalizada com a proposta de Perez-Castillo et. al. (2011), foi disponibilizada em um ambiente de testes. Todos os atendimentos realizados no ambiente de produção do plano de saúde foram replicados no ambiente de testes, gerando um arquivo de log com a instrumentação proposta por Perez-Castillo et. al. (2011). Deste modo, os arquivos de log das duas propostas foram gerados a partir das mesmas informações.

7.1.4 Mineração e Modelagem dos Processos de Negócio

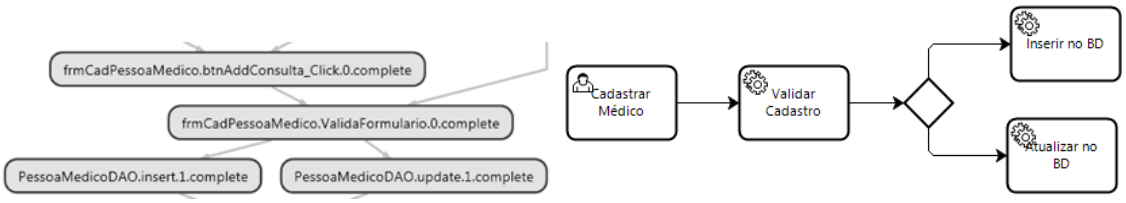
Para a etapa de mineração dos processos os arquivos de log gerados no ambiente de produção (quatro arquivos – quatro estações de trabalho) foram unificados em um único arquivo. Sendo assim, foram minerados dois arquivos de log: o primeiro contendo os eventos provenientes da instrumentação de código proposta nesta tese; e o segundo contendo os eventos provenientes da instrumentação de código proposta por Perez-Castillo et. al. (2011).

Como a técnica proposta por Perez-Castillo et. al. (2011) não especifica um algoritmo para mineração de processos, nós usamos o algoritmo *Incremental Miner* para minerar os dois arquivos de log gerados neste estudo de caso.

A partir do arquivo de log da Figura 7.3 (instrumentação proposta nesta tese), o algoritmo *Incremental Miner* gera quatro grafos que representam as ordens parciais de execução das regras de negócio dos processos implementados no sistema de informação.

Logo após a mineração, os grafos foram mapeados para modelos de processos (notação BPMN), conforme mapeamento proposto na Tabela 6.1. Cada regra de negócio encontrada no grafo foi mapeada para um dos fragmentos de processos definidos na Tabela 6.1 (conforme o tipo da regra). A Figura 7.4 mostra um fragmento de grafo gerado pelo *Incremental Miner*, assim como seu respectivo modelo de processo mapeado a partir da Tabela 6.1.

Figura 7.4. Fragmento de processo minerado pelo *Incremental Miner*.



(a) Ordem Parcial de Regras de Negócio

(b) Fragmento de Processo mapeado da Ordem Parcial das Regras

Fonte: Elaborado pelo autor.

Da mesma forma, o arquivo de log gerado a partir da instrumentação proposta por Perez-Castillo et. al. (2011) foi minerado pelo *Incremental Miner*, identificando quatro grafos que representam as ordens parciais de execução das chamadas de subrotinas dos processos implementados no sistema de informação. Após a mineração, os nós dos grafos são mapeados atividades dos processos de negócio.

7.1.5 Resultados do Estudo de Caso

Durante o tempo de execução do sistema de informação (três meses), um desenvolvedor especialista na linguagem de programação C#, analisou o código fonte manualmente, buscando identificar os quatro processos de negócio apenas com conhecimento humano.

Deste modo, a última etapa do estudo de caso busca comparar os modelos de processos de negócio identificados pelo desenvolvedor C#, pelo método proposto nesta tese e pela técnica proposta por Perez-Castillo et. al. (2011). A Tabela 7.1 apresenta os resultados obtidos no estudo de caso.

A segunda coluna da Tabela 7.1 mostra o número de processos modelados pelos usuários especialistas do sistema. Antes de iniciarmos o estudo de caso com o referido sistema, usuários especialistas modelaram quatro processos de negócio que são executados através do sistema de informação. Os quatro modelos foram considerados corretos e juntos possuem 76 atividades de negócio, 23 gateways e 4 atores.

As três últimas colunas da Tabela 7.1, "**Método Manual**", "**Técnica de Perez-Castillo et. al. (2011)**" e "**Método Proposto na Tese**" apresentam os resultados obtidos

nos três métodos de identificação de processos utilizados no estudo de caso. O método manual de mapeamento (desenvolvedor C#), o método de instrumentação do código através das invocações de subrotinas apresentado em (Perez-Castillo et. al., 2011) e o método de instrumentação por regras de negócio proposto nesta tese.

Tabela 7.1: Resultados do estudo de caso com o sistema de informações médicas.

Processos Modelados por Usuários Especialistas no Sistema			**Métodos de Identificação dos Processos de Negócio**		
			Método Manual	**Técnica de Perez-Castillo et. al. (2011)**	**Método Proposto na Tese**
Nº de Processos	04		4	4	4
Nº de Atividades	76	Total	83	171	152
		Acertos	68	43	75
		Outras	15	128	77
Nº de Gateways	23	Total	20	34	32
		Acertos	19	18	23
		Outros	1	16	9
Atores	04	Total	4	0	4
		Acertos	4	0	4
		Outros	0	0	0
Fluxos Alternativos			5	2	3

Fonte: Elaborado pelo autor.

Observe que os três métodos identificaram os quatro processos de negócio implementados no sistema. Note também, que o método de identificação manual obteve modelos mais próximos dos processos desenhados pelos usuários especialistas. Neste método foram identificadas 83 atividades de negócio, sendo que 68 atividades são idênticas às atividades encontradas nos modelos dos usuários especialistas, isto é, 89% de atividades identificadas em relação aos modelos dos usuários especialistas. Outras 15 atividades são modeladas de forma incorreta ou são atividades identificadas a mais em relação ao modelo apresentado pelos usuários especialistas (18% das 83 atividades identificadas pelo desenvolvedor).

A Tabela 7.1 também mostra que os métodos automáticos identificaram um número maior de atividades de negócio: 171 atividades usando a técnica de instrumentação proposta em (Perez-Castillo et. al., 2011); e 152 atividades usando o método de instrumentação por regras de negócio (proposto nesta tese). Isso se deve ao fato das ferramentas identificarem atividades mais próximas da implementação dos processos, ou seja, em um nível de abstração mais baixo do que a análise humana (mais voltada para o negócio da organização).

Entretanto, observe que a técnica de instrumentação proposta na tese obteve mais sucesso na identificação das atividades. Das 152 atividades identificadas, 75 são idênticas as modeladas pelos usuários especialistas, isto é, 98% das atividades foram identificadas no modelo extraído através deste método. Isso se deve ao fato das regras de negócio possuírem semântica, enquanto as invocações de subrotinas dependem de análise posterior de um desenvolvedor para descobrir se a rotina implementa ou não uma atividade de negócio.

Por exemplo, um trecho de código fonte que cria um formulário para entrada de dados é anotado como regra de habilitação, que posteriormente através da semântica da regra é modelado como atividade humana. A análise é automática, realizada por uma ferramenta e não por um usuário que precisa analisar o conteúdo da subrotina para identificar o significado do evento que gerou a atividade no modelo.

Por este motivo o método proposto em (Perez-Castillo et. al., 2011) obteve um número tão expressivo de outras atividades (128). Elas não representam exatamente uma identificação incorreta, mas sim, representam atividades que não puderam ser comparadas diretamente com o modelo dos usuários especialistas. Elas ainda necessitam de uma análise do desenvolvedor C# no corpo das subrotinas anotadas como eventos de negócio. Isso representa um passo extra na aplicação deste método, que não ocorre em relação ao método proposto neste trabalho.

Em nosso método foram identificadas 77 atividades de negócio que não estavam nos modelos dos usuários especialistas. Entretanto, são atividades automáticas que dizem respeito ao nível de implementação do processo. Não são atividades totalmente descartáveis, pois provavelmente, ao implementar o processo em um Sistema Gerenciador de Processos essas atividades também serão implementadas. Além disso, um analista humano pode extrair dos modelos a visão humana dos processos de negócio (mais próxima do negócio).

Essa mesma análise realizada para as atividades de negócio também é válida para a identificação dos gateways (*XOR-Split* e *AND-Split*). Já os atores foram identificados apenas através do método proposto nesta tese, já que no método proposto em (Perez-Castillo et. al., 2011) não referencia este tipo de componente.

O número de fluxos alternativos refere-se a caminhos encontrados pelos métodos de identificação e que não foram modelados pelos usuários especialistas. Estes caminhos estão implementados no código fonte e não estão documentados pelos analistas de negócio. Normalmente são fluxos que surgiram através das manutenções corretivas do sistema de informação.

Em sistemas legados é muito comum ocorrer manutenções corretivas que não são atualizadas nas documentações do sistema. Observe que na análise manual foram identificados dois fluxos alternativos a mais que o método proposto nesta tese. Isso se deve pelo fato dos fluxos não serem executados durante os três meses em que o sistema permaneceu em produção gerando os logs de eventos. Deste modo, estes fluxos não foram *logados* e consequentemente não aparecem na mineração dos processos.

Por fim, ainda observamos que a relação custo/beneficio para utilização de um método automático é melhor do que a utilização de um método manual. A análise manual foi realizada por um desenvolvedor C#, que dedicou 20 horas semanais de trabalho, durante seis semanas para apresentar o resultado da modelagem dos quatro processos de negócio. Enquanto isso, para os métodos automáticos foram utilizadas 16 horas de trabalho de um desenvolvedor C#, sendo 12 horas para preparação e instrumentação dos arquivos de código fonte e 4 horas para minerar os logs de eventos.

Contudo, os métodos automáticos obtiveram resultado em 12 semanas, enquanto o método manual obteve resultado em 06 semanas. Isso se deve ao tempo de geração dos arquivos de log. Este tempo não possui trabalho humano envolvido, mas foi um tempo de espera necessário, pois não foi possível automatizar a execução de cenários de teste para este sistema de informação. Entretanto, o custo financeiro de hora trabalhada foi

menor para os métodos automáticos – 16 horas nos métodos automáticos contra 120 horas no método manual.

7.2 Sistemas de Administração Pública

Neste estudo de caso foram usados dois sistemas administrativos de uma instituição pública de ensino. Ambos foram desenvolvidos com a mesma tecnologia, por este motivo foram agrupados na mesma seção.

O primeiro sistema de informação é usado para controlar o trâmite de processos administrativos da instituição de ensino. Para o estudo de caso foi utilizado especificamente o módulo de recursos humanos, identificando dois processos de negócio que são implementados através deste sistema:

- Afastamentos: processo utilizado para registro e aprovação de solicitações de afastamento de servidores para capacitação. Por exemplo, um docente deseja solicitar afastamento para realizar pós-graduação no exterior. Aqui, o docente dispara o processo de afastamento, que irá passar por todas aprovações de chefias, nomeação de substitutos, até a conclusão do processo com o registro de afastamento do docente no diário oficial da união.
- Aproveitamento de concursos: processo usado para registro e aprovação de pedidos de aproveitamento de concursos entre dois órgãos institucionais. Por exemplo, o câmpus "A" gostaria de aproveitar o concurso para docente realizado pelo câmpus "B". Neste caso, o câmpus "A" inicia o processo de aproveitamento, que coleta todas as aprovações necessárias e é finalizado com a nomeação ou não do docente.

Da mesma forma que o estudo de caso anterior, os dois processos de negócio deste sistema foram modelados previamente por usuários especialistas. Os usuários especialistas foram entrevistados e seus modelos foram considerados corretos para o estudo de caso.

Por sua vez, o segundo sistema de informação é usado para liberação de recursos financeiros destinados as obras de uma instituição pública de ensino. Neste sistema, chamado de sistema de engenharia civil, foi identificado um único processo negócio. Esse processo é executado para liberar os recursos financeiros de um programa especial, de ampliação da estrutura física das instituições públicas de ensino. As tarefas do processo consistem de aprovações realizadas em um determinado projeto, até a liberação completa dos recursos necessários para execução da obra.

Diferente do sistema de recursos humanos, o sistema de engenharia civil possui documentação e seu processo de negócio está devidamente modelado e documentado nos manuais de instruções do sistema. Desta forma, não foram necessárias entrevistas com usuários especialistas para desenhar o processo de negócio de liberação de recursos financeiros. O modelo encontrado nos manuais de instrução do sistema foi considerado correto.

Ambos os sistemas de informação usam plataforma web e foram desenvolvidos na linguagem de programação PHP. Sendo assim, os processos foram identificados a partir do método proposto nesta tese, da técnica proposta por Perez-Castillo et. al. (2011), da técnica proposta por Di Francescomarino, Marchetto e Tonella (2009) e a partir da

identificação manual realizada por um desenvolvedor especialista em linguagem de programação PHP.

Durante o estudo de caso foram analisados 36 arquivos, totalizando aproximadamente 16 mil linhas de código fonte. A análise dos arquivos seguiu a mesma lógica empregada no estudo de caso anterior, ou seja, enquanto o desenvolvedor PHP analisou manualmente os arquivos, outro desenvolvedor instrumentalizou os arquivos procurando identificar os processos a partir dos métodos automáticos.

7.2.1 Preparação para Instrumentação do Código Fonte

Para a instrumentação dos arquivos foi criado um algoritmo para enviar eventos para um arquivo de log (idêntico ao estudo de caso anterior). A Figura 7.5 mostra a função PHP criada para controlar a gravação de eventos em um arquivo de log.

Note que neste estudo de caso existe apenas a função "***escreverEvento***", diferente da classe apresentada na Seção 7.1.1, que possui três métodos. A diferença aqui encontra-se na plataforma web, que permite a execução do processo em diferentes máquinas e por vários usuários. Assim, a execução do processo pode se dar em várias máquinas e não em uma única estação de trabalho, como ocorria no experimento anterior. Deste modo, o identificador de instância não pode ser um número gerado aleatoriamente e armazenado na memória.

Figura 7.5: Algoritmo para gerar log de eventos em PHP.

```
<?php
  function escreverEvento($idInstancia, $nomeProcesso,
                          $idRegraNegocio, $tipoRegraNegocio=0) {
    if ($idInstancia != '' && !empty($idInstancia)) {
       $usuario = $_SESSION['username'];
       $file = fopen('Eventos.log','w+');
       fwrite($file, $idInstancia .';'. $nomeProcesso .';'.
                     $usuario .';'. $idRegraNegocio .';'.
                  $tipoRegraNegocio .';'. date("Y-m-d H:i:s"));
       fclose($file);
    }
  }
?>
```

Fonte: Elaborado pelo autor.

Para o sistema de recursos humanos o identificador de instância é o número de protocolo gerado ao abrir uma solicitação de afastamento ou aproveitamento de concurso. Já para o sistema de engenharia civil o identificador de instância é o número de projeto gerado ao solicitar liberação de recursos. Esses números são itens de trabalho que constam em todas as tarefas dos processos. Portanto, os números de protocolo e de projeto são passados como parâmetro na função "***escreverEvento***", assim como o nome do processo. Essa diferença fica mais clara na próxima seção.

7.2.2 Instrumentação do Código Fonte

Cada arquivo instrumentalizado recebeu a adição de uma instrução de inclusão da função apresentada na Figura 7.5. Essa instrução foi adicionada no inicio de cada arquivo, conforme mostra o exemplo de código da Figura 7.6.

Figura 7.6: Instrução de inclusão da função "***escreverEvento***".

```
<?php include "funcao_escrever_log.inc"; ?>
<HTML>
...
```

Fonte: Elaborado pelo autor.

Na Figura 7.7(a) é apresentado um exemplo de anotação de uma regra de negócio, usando a ferramenta de instrumentação proposta nesta tese. A Figura 7.7(b) apresenta um exemplo de anotação de chamada de subrotina, conforme proposta de Perez-Castillo et. al. (2011).

Observe na Figura 7.7, que o primeiro parâmetro da invocação de "***escreverEvento***" é o identificador de instância, ou seja, a variável "*$protocolo*" que armazena o número de protocolo gerado para uma solicitação de afastamento. O valor "*$protocolo*" foi informado como parâmetro de entrada de nossa ferramenta de instrumentação. Deste modo, o valor pode ser alterado dependendo do arquivo que será anotado (por exemplo, nos arquivos do sistema de engenharia civil o identificador é o número do projeto). O mesmo ocorre com o segundo parâmetro da invocação de "***escreverEvento***", que é o nome do processo executado no arquivo de código fonte ("*AFASTAMENTO*").

Figura 7.7: Exemplos de instrumentação do código fonte em PHP.

```
...
escreverEvento($protocolo, "AFASTAMENTO", "dadoPessoal.5", 3);
global $db;
$sql = "UPDATE gestaopessoa.anexogp SET agpstatus = 'I' WHERE
fdpcpf = '".$_SESSION['fdpcpf']."'";
$db->executar( $sql );
$db->commit();
...
```

(a) Exemplo de anotação com a ferramenta apresentada nesta tese.

```
...
$sql = "UPDATE gestaopessoa.anexogp SET agpstatus = 'I' WHERE
fdpcpf = '".$_SESSION['fdpcpf']."'";

escreverEvento($protocolo,    "AFASTAMENTO",    "dadoPessoal.db.
executar");
$db->executar( $sql );

escreverEvento($protocolo,    "AFASTAMENTO",    "dadoPessoal.db.
commit");
$db->commit();
...
```

(b) Exemplo de Anotação com a Técnica de Perez-Castillo et. al. (2011).

Fonte: Elaborado pelo autor.

Note também, que as Figuras 7.7(a) e 7.7(b) mostram o mesmo trecho de código, sendo que (a) possui apenas uma anotação, enquanto (b) possui duas anotações. A Figura 7.7(a) mostra a anotação de um bloco de persistência, uma regra de persistência, que no sistema de recursos humanos é iniciado com "`global $db`" e finalizado com

"`$db->commit()`". Assim, nossa ferramenta anota o bloco completo como uma regra de negócio, enquanto na Figura 7.7(b) cada chamada de função ("`$db->executar()`" e "`$db->commit()`") foi anotada como um evento de negócio, como proposto por Perez-Castillo et. al. (2011).

7.2.3 Execução dos Sistemas de Informação

Neste estudo de caso foram gerados dois ambientes de testes. No primeiro ambiente foram disponibilizadas as versões dos sistemas instrumentalizados com o método proposto nesta tese. Já no segundo ambiente de testes foram disponibilizadas as versões dos sistemas com a instrumentação proposta por Perez-Castillo et. al. (2011).

Ambos os experimentos foram executados em ambientes de testes, pois a instituição não permitiu executar os sistemas instrumentalizados no ambiente de produção. Desta forma, os dados de afastamentos, aproveitamentos de concursos e liberação de recursos para obras foram coletados nos sistemas de produção e replicados nos ambientes de testes, gerando assim, os arquivos de log necessários para a mineração destes processos.

Além da geração dos logs de eventos a partir da instrumentação do código fonte, também foi habilitada a geração do arquivo "***access.log***" do servidor web (Apache 2.2). O arquivo "***access.log***" registra as páginas acessadas durante a navegação dos sistemas. A partir deste arquivo, nós aplicamos a mineração de processos proposta por Di Francescomarino, Marchetto e Tonella (2009).

Os arquivos de log de eventos gerados neste estudo de caso tem o mesmo formato dos fragmentos apresentados na Figura 7.3. A Figura 7.8 mostra um fragmento do arquivo "***access.log***".

Figura 7.8: Fragmento do log de acesso de um servidor web.

```
...
127.0.0.1 - - [02/Jun/2013:00:26:24 -0300] "GET /login.php
HTTP/1.1" 500 96
127.0.0.1 - - [02/Jun/2013:00:56:44 -0300] "GET
/muda_sistema.php?sisid=12 HTTP/1.1" 200 102
127.0.0.1 - - [02/Jun/2013:00:56:44 -0300] "GET
/reuni/reuni.php?modulo=inicio&acao=C HTTP/1.1" 404 213
127.0.0.1 - - [02/Jun/2013:01:00:04 -0300] "GET
/tarefa/tarefa.php?modulo=inicio&acao=C HTTP/1.1" 404 215
127.0.0.1 - - [02/Jun/2013:01:00:07 -0300] "GET
/obras/obras.php?modulo=inicio&acao=A HTTP/1.1" 200 49923
...
```

Fonte: Elaborado pelo autor.

7.2.4 Mineração e Modelagem dos Processos de Negócio

Para este estudo de caso foram minerados três arquivos de log:

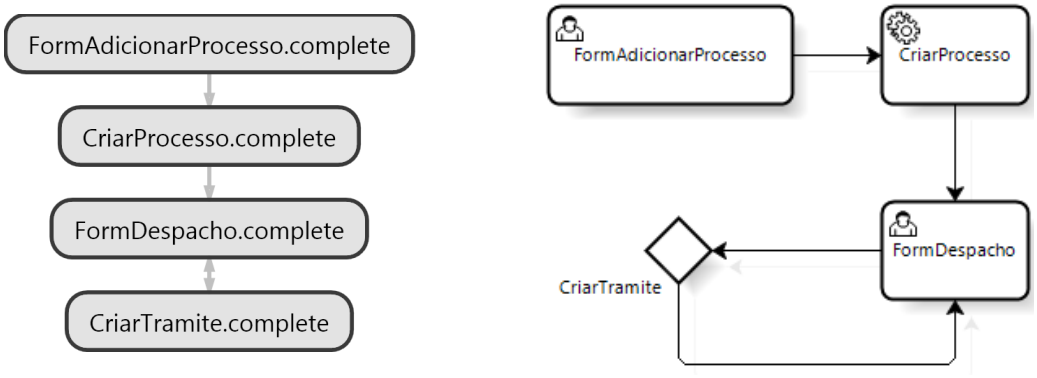
1. Arquivo com a instrumentação do código proposta nesta tese;
2. Arquivo com a instrumentação do código proposta por Perez-Castillo et. al. (2011);
3. Arquivo com os acessos registrados pelo servidor web, "***access.log***", conforme proposto por Di Francescomarino, Marchetto e Tonella (2009).

Os três arquivos foram minerados com o algoritmo *Incremental Miner*, como ocorreu no estudo de caso anterior.

Entretanto, o arquivo com registros de acesso do servidor web, "***access.log***", necessitou de adaptação para uso no *Incremental Miner*. Neste caso, os registros de acesso foram mapeados para o formato aceito pelo algoritmo.

Durante a etapa de mineração foram obtidos sete grafos, que representam a ordem parcial de execução dos processos identificados nos sistemas de informação. Após a mineração, os grafos foram mapeados para modelos de processos, conforme proposto na Tabela 6.1. A Figura 7.9 mostra um fragmento de processo extraído neste estudo de caso.

Figura 7.9. Fragmento de processo minerado pelo *Incremental Miner*.



(a) Ordem Parcial de Regras de Negócio (b) Fragmento de Processo mapeado da Ordem Parcial das Regras

Fonte: Elaborado pelo autor.

7.2.5 Resultados do Estudo de Caso

Os processos de negócio identificados neste estudo de caso são mais simples, ou seja, possuem um número menor de atividades. Entretanto, o experimento mostra que o método proposto nesta tese pode ser aplicado em diferentes tecnologias. Além disso, esse estudo de caso permite a comparação de nosso método, com o método proposto por Di Francescomarino, Marchetto e Tonella (2009), que é aplicado apenas a sistemas web. Os resultados obtidos neste estudo de caso são apresentados na Tabela 7.2.

Note que a técnica proposta por Di Francescomarino, Marchetto e Tonella (2009) recupera processos mais próximos dos modelos desenhados pelos usuários especialistas. As 24 atividades dos processos de negócio mapeados por esta técnica encontram-se nos processos modelados pelos usuários especialistas. Isso porque, o log registra apenas as páginas acessadas pelo usuário, ou seja, registra somente as atividades humanas do processo de negócio. Desta forma, o processo extraído é mais próximo do nível de negócio, da visão humana sobre o processo de negócio.

Entretanto, a visão de implementação do processo não é descoberta através da técnica de Di Francescomarino, Marchetto e Tonella (2009). Normalmente, as atividades automáticas não são registradas no log de acesso dos servidores web. Sendo assim, cálculos aritméticos, persistências, entre outras atividades automáticas não são descobertas por esta técnica. Neste experimento 08 atividades automáticas não foram descobertas pela técnica de Di Francescomarino, Marchetto e Tonella (2009).

Tabela 7.2: Resultados do estudo de caso com sistemas da administração pública.

Processos Modelados por Usuários Especialistas no Sistema			**Métodos de Identificação dos Processos de Negócio**			
			Método Manual	**Técnica de Perez-Castillo et. al. (2011)**	**Técnica de Di Francescomarino et. al. (2009)**	**Método Proposto na Tese**
Nº de Processos	03		3	2	3	3
Nº de Atividades	32	Total	34	39	24	58
		Acertos	30	18	24	30
		Outras	4	21	0	28
Nº de Gateways	9	Total	7	5	7	10
		Acertos	7	3	6	8
		Outros	0	2	1	2
Atores	11	Total	11	0	0	10
		Acertos	11	0	0	10
		Outros	0	0	0	0
Fluxos Alternativos			0	2	1	1

Fonte: Elaborado pelo autor.

Além disso, o log de acesso ("***access.log***") do servidor web necessitou de um pré-processamento para adaptar seu formato, ao formato exigido pelo algoritmo de mineração de processos. Esse pré-processamento aumentou em oito horas o tempo de execução da etapa de mineração.

Outra observação importante do estudo de caso é o fato da técnica proposta por Perez-Castillo et. al. (2011) descobrir apenas dois processos de negócio. Isso porque, o sistema de engenharia civil não foi organizado em subrotinas. Assim, não foi possível instrumentar seu código fonte usando a técnica proposta por Perez-Castillo et. al. (2011). Esta característica é muito comum em sistemas legados mais antigos, onde o código é obsoleto e não segue as boas práticas de programação.

Por último, observe que os atores foram identificados apenas pelo método proposto nesta tese. Entretanto, o método não encontrou um dos papeis de um processo de negócio, pois este papel é usado no fluxo alternativo que não foi executado em nosso experimento (os dados usados não habilitaram a execução deste fluxo). O restante da análise é similar ao estudo de caso com o sistema de informações médicas. A Figura 7.10 mostra um fragmento de processo de negócio minerado com o método proposto nesta tese.

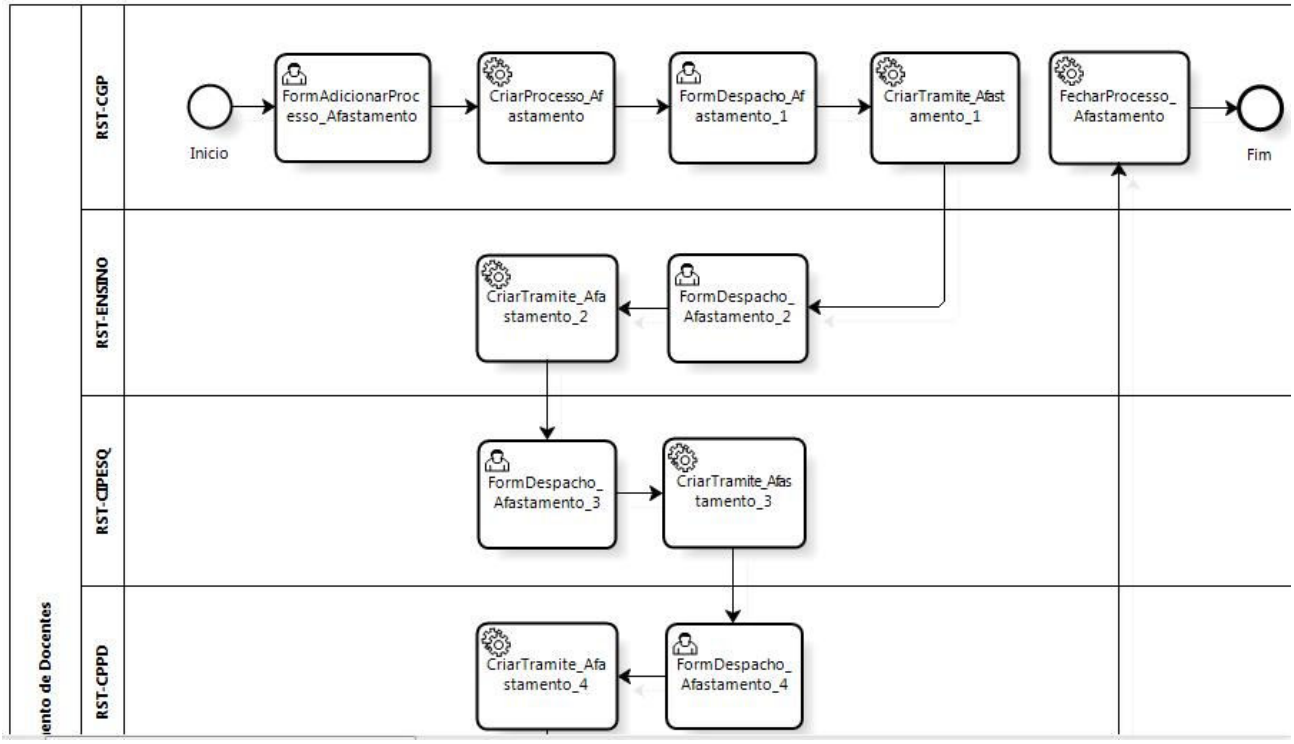
7.3 Ameaças e Dificuldades

Durante a execução dos experimentos foram encontradas algumas dificuldades que podem interferir no resultado dos experimentos, ou seja, tais dificuldades podem se configurar em ameaças nos resultados obtidos nos estudos de casos.

A instrumentação do código fonte através da técnica de Perez-Castillo et. al. (2011) foi uma das dificuldades enfrentadas nos estudos de casos. Isso porque, os autores não disponibilizam uma ferramenta para instrumentação automática do código fonte. Sendo assim, a instrumentação desta técnica foi realizada manualmente, por um desenvolvedor

especialista nas respectivas linguagens de programação empregadas nos estudos de caso.

Figura 7.10: Processo de negócio identificado pelo método proposto na tese.



Fonte: Elaborado pelo autor.

A instrumentação manual da técnica de Perez-Castillo et. al. (2011) pode representar uma ameaça aos resultados obtidos nos experimentos. A instrumentação torna-se dependente do nível de conhecimento (tanto do código fonte, quanto da linguagem de programação) e da capacidade de atenção do desenvolvedor. Uma interpretação incorreta do desenvolvedor pode interferir no resultado final do experimento.

Para minimizar esse problema, a instrumentação manual da técnica de Perez-Castillo et. al. (2011) foi executada por dois desenvolvedores, ou seja, enquanto o primeiro desenvolvedor instrumenta o código fonte, o segundo revisa a instrumentação a procura de falhas no trabalho realizado pelo primeiro desenvolvedor.

Outra dificuldade importante foi executar os sistemas da administração pública em ambientes de testes, pois a instituição não permitiu o uso do ambiente de produção. Com isso, os processos foram gerados a partir de situações simuladas em ambientes de testes (cenários de testes). Os processos de negócio não foram mapeados de situações reais executadas nestes sistemas de informação.

Desta forma, a geração dos cenários de testes pode representar outra ameaça aos resultados destes experimentos. Caso os cenários de testes não sejam criados corretamente ou sejam insuficientes para simular as situações de um ambiente real, então, os resultados obtidos podem ser incompletos ou distorcidos da realidade organizacional.

Para minimizar o impacto do uso de ambientes de teste, foram gerados cenários de teste a partir de entrevistas com usuários especialistas nos respectivos sistemas de informações. Ou seja, estes usuários determinaram cenários de testes a partir de situações reais executadas nos sistemas de informação.

8 CONCLUSÃO

A presente tese defende a hipótese que é possível definir um método para, pelo menos, semi-automatizar os procedimentos para a descoberta de processos de negócio implementados implicitamente no código fonte de sistemas legados.

Para provar esta hipótese, o primeiro objetivo da pesquisa foi investigar diferentes técnicas de extração de conhecimento aplicadas a sistemas legados. Com base na literatura sobre extração de conhecimento de sistemas legados, concluímos que as técnicas de análise estática podem ser usadas na identificação das atividades de negócio (codificadas no sistema legado), enquanto as técnicas de análise dinâmica (mineração de *log*) podem descobrir a ordem parcial das atividades de negócio (modelo do processo).

A partir deste estudo, demonstramos que o comportamento dos sistemas legados é determinado pelas regras de negócio implementadas em seu código fonte. Da mesma forma, outros pesquisadores demonstram que as atividades de um processo, também podem ser especificadas e implementadas através de regras de negócio.

Portanto, as regras de negócio podem ser usadas como invariáveis no mapeamento de processos de negócio codificados em sistemas legados. As regras de negócio tornam possível a identificação de trechos de código fonte que representam atividades de um processo de negócio codificadas no sistema legado. Deste modo, respondemos o segundo objetivo desta tese, conforme apresentado na Seção 1.2.

Na sequência da tese, nós propomos uma técnica de análise estática capaz de anotar as regras de negócio implementadas no código fonte de sistemas legados. Para definir a técnica foram minerados cinco sistemas legados, buscando observar como as regras de negócio foram codificadas nestes sistemas legados. Através deste estudo, nós observamos que existem padrões sintáticos que podem ser usados para identificar a implementação de regras de negócio no código fonte.

Sendo assim, nós definimos uma ontologia de regras de negócio, que é usada como base de conhecimento para uma ferramenta de análise estática identificar as regras de negócio codificadas no sistema legado.

A definição da ferramenta para identificação das regras de negócio corresponde ao terceiro objetivo definido para a pesquisa, conforme apresentado na Seção 1.2. Esse objetivo é completado com a definição da ferramenta de mineração de processos. O algoritmo *Incremental Miner* foi desenvolvido na dissertação de mestrado de André Cristiano Kalsing (Kalsing, 2012). O tema desta dissertação surge da necessidade de uma ferramenta de mineração para completar o método proposto nesta tese. Portanto, essa tese gerou ao menos um trabalho de mestrado até o presente momento.

Após a definição das ferramentas de instrumentação do código fonte e mineração de processo, o documento apresenta os passos do método para mapeamento de processos de negócio codificados em sistemas legados. O método corresponde ao quinto objetivo de pesquisa definido na Seção 1.2.

O método é composto de sete etapas, onde algumas são executadas por usuários especialistas e outras são automatizadas. Deste modo, o método é semi-automático, pois prevê intervenção humana durante a aplicação das etapas.

A partir desta pesquisa, concluímos que não é possível definir um método automático para descoberta de processos de negócio em sistemas legados. Existem informações específicas, como os eventos de inicio e fim de um processo de negócio ou o identificador de instância, que não são encontradas no código fonte ou banco de dados do sistema legado. Somente usuários especialistas podem fornecer tais informações.

Entretanto, um método semi-automático acelera o mapeamento inicial de processos de negócios em uma organização. Normalmente, o mapeamento inicial é a etapa mais lenta e de maior custo na implantação da Gestão por Processos de Negócio. Neste sentido, nosso método permite iniciar o trabalho de mapeamento a partir de uma base de conhecimento existente na organização, ou seja, a partir de um conjunto de processos descobertos de forma semi-automática em sistemas legados.

Atualmente, o conhecimento implícito nos sistemas legados é descartado ao mapear os processos de negócio em uma organização. Nosso método semi-automático permite reutilizar esse conhecimento, diminuindo o tempo necessário para o mapeamento dos processos de negócio em uma organização.

Em relação aos demais métodos existentes na literatura, os estudos de caso demonstram que nosso método pode ser aplicado em diferentes sistemas legados, independente da plataforma ou linguagem de programação.

A técnica proposta por Di Francescomarino, Marchetto e Tonella (2009) não foi aplicada ao sistema de informações médicas, pois o referido sistema não foi implementado na plataforma web. Por sua vez, a técnica proposta por Perez-Castillo et. al. (2011) não foi aplicada ao sistema de engenharia civil, devido ao sistema não estar organizado em subrotinas. Enquanto isso nosso método foi utilizado em ambos os sistemas.

Além disso, nosso método obteve resultados mais completos que a técnica proposta por Perez-Castillo et. al. (2011). Isso porque, os trechos de código fonte anotados como atividades possuem um significado para o negócio da organização. São unidades de processamento atômicas, que manipulam estruturas de dados e possuem semântica (ex.: cálculos aritméticos, blocos de persistência, tomadas de decisão). Na técnica proposta por Perez-Castillo et. al. (2011) não há garantias que uma chamada de subrotina representa uma atividade de negócio.

Os estudos de caso também mostram que o método encontra processos de negócio mais próximos do nível de implementação. Ou seja, o modelo identificado pelo método está na visão de implementação do processo de negócio. Entretanto, um analista de pode refinar o modelo, extraindo um modelo mais próximo da visão de negócio.

8.1 Publicações da Tese

Os resultados da tese geraram uma dissertação de mestrado (já defendida), sete publicações em congressos e uma em periódico (ainda sobre avaliação).

A dissertação de mestrado trata do algoritmo de mineração de processos de negócio e foi desenvolvida pelo aluno André Cristiano Kalsing (Kalsing, 2012), integrante do grupo de pesquisa em workflow do Instituto de Informática da UFRGS. A dissertação "*Uma Abordagem Incremental para Mineração de Processos de Negócio*" defendida em julho de 2012, contempla a quinta etapa do método de identificação de processos de negócio em sistemas legados (Seção 6.5).

As demais publicações apresentam resultados parciais da pesquisa e focam em uma determinada parte da tese:

- **Ontologia de Regras de Negócio:**
  - Nascimento, G. S. do, Iochpe, C., Thom, L. H., Kalsing, A., Moreira, A. F.: **Identifying Business Rules to Legacy Systems Reengineering Based on BPM and SOA**. In: Computational Science and Its Applications - ICCSA 2012 - 12th International Conference. Salvador, Bahia, Brazil, pp. 67-82, 2012.
  - Junior, W. A. R., Nascimento, G. S. do, Iochpe, C.: **Survey and Proposal of a Method for Business Rules Identification in Legacy Systems Source Code and Execution Logs**. In: Proceedings of the 13th International Conference on Enterprise Information Systems (ICEIS). Beijing, China, pp. 207-213, 2011.
- **Ferramenta de Identificação das Regras de Negócio:**
  - Nascimento, G. S. do, Iochpe, C., Thom, L. H., Kalsing, A.: **Discovering Business Processes in Legacy Systems using Business Rules and Log Mining.** In: 10th IEEE International Conference on e-Business Engineering (ICEBE). Coventry, United Kingdom, 2013. (aguardando publicação dos anais).
- **Algoritmo de Mineração de Processos:**
  - Kalsing, A. C., Iochpe, C., Thom, L. H., Nascimento, G. S. do: **Evolutionary Learning of Business Process Models from Legacy Systems using Incremental Process Mining.** In: Proceedings of the 15th International Conference on Enterprise Information Systems (ICEIS). Angers, France, pp. 58-69, 2013.
  - Kalsing, A., Nascimento, G., Iochpe, C., Thom, L. H.: **An Incremental Process Mining Approach to Extract Knowledge from Legacy Systems.** Proceedings of the 14th IEEE International Enterprise Distributed Object Computing Conference (EDOC). Vitoria, Brazil, pp. 79-88, 2010.
- **Definição do Método de Identificação de Processos de Negócio:**
  - Nascimento, G. S. do, Iochpe, C., Thom, L. H., Kalsing, A.: **Mapeando Processos de Negócio à partir de Regras de Negócio implementadas em Sistemas Legados.** In: VII Workshop Brasileiro

em Gestão de Processos de Negócio (WBPM). João Pessoa, Brazil, 2013.

- Nascimento, G. S. do, Iochpe, C., Thom, L. H., Reichert, M.: **A Method for Rewriting Legacy Systems using Business Process Management Technology.** In: Proceedings of the 11th International Conference on Enterprise Information Systems (ICEIS). Milan, Italy, pp. 57-62, 2009.

- **Tese completa e Estudos de Caso:**
  - Nascimento, G. S. do, Iochpe, C., Thom, L. H., Kalsing, A.: **A Legacy System based Framework to Business Processes Mapping.** IET Software. (aguardando avaliação do periódico).

8.2 Trabalhos Futuros

Como trabalhos futuros, para continuidade da tese, podemos citar os seguintes:

1. Definição de uma ferramenta para geração de casos de teste;
2. Estudo de viabilidade técnica para o uso de ferramentas de execução automática de testes;
3. Estudo de viabilidade técnica para inclusão de outras fontes de informações sobre o sistema legado;
4. Construção de um framework para descoberta de processos de negócio em sistemas legados.

Atualmente, o ponto mais crítico para aplicação do método encontra-se na execução do sistema legado para geração do log. O método não dispõe de ferramenta ou técnica capaz de acelerar a geração do arquivo de log. Após a instrumentação automática do código fonte, o analista de negócio precisa optar por deixar o sistema legado instrumentalizado executando no ambiente de produção (com dados reais da organização) ou executá-lo em um ambiente de testes, criado especificamente para geração do log de eventos.

No primeiro caso, a execução em ambiente de produção necessita de um tempo maior para gerar um arquivo de log adequado para a mineração dos processos de negócio. Além disso, ao injetar novas instruções no código fonte, corre-se o risco de produzir falhas na execução do sistema, que podem comprometer o bom funcionamento da organização.

Na segunda situação, a execução em ambiente de testes necessita da geração de casos de testes para execução do sistema legado. Os casos de testes são conjuntos de tarefas que devem ser executadas no sistema legado, a fim de cumprir uma determinada meta de negócio através do sistema. Atualmente, o método não prevê algoritmos ou ferramentas para gerar os casos de teste de forma automática. Sendo assim, os casos de teste são gerados manualmente, através de entrevistas com usuários do sistema ou através de análise dos dados gravados no banco de dados da organização.

A geração do log de eventos em um ambiente de testes é mais segura, pois evita possíveis problemas operacionais na organização, causados pela instrumentação do código fonte legado. Entretanto, torna-se necessário acrescentar ao método, um algoritmo ou ferramenta de geração automática dos casos de teste, facilitando a

execução do sistema legado em ambientes de teste. Além disso, a geração manual dos casos de teste não garante que o log de eventos terá todas as possibilidades de execução do sistema legado.

Em conjunto com a geração dos casos de teste, também é importante estudar a viabilidade técnica de utilizar ferramentas para automação de casos de teste, ou seja, uma ferramenta capaz de executar os casos de teste de forma automática. A automação dos casos de testes aumenta a velocidade de geração do log de eventos, diminuindo o tempo de aplicação do método.

Outro trabalho futuro consiste em estudar a viabilidade de incluir outras fontes de dados na análise do sistema legado. Por exemplo, o log de transações de um banco de dados, como apresentado no trabalho de Junior (2002), pode ser agregado ao log de eventos gerado pelo sistema legado. Isso pode enriquecer a análise das regras de persistência, permitindo verificar os dados gravados no banco de dados. Deste modo é possível acrescentar a semântica as regras de persistência, ou seja, que tabela e que campos são alterados, quais dados são gerados na transação, entre outras informações.

Por fim, todas as ferramentas propostas no método podem ser compiladas em um único pacote, criando um framework para descoberta de processos de negócio e outras informações importantes sobre o negócio da organização. Além da descoberta de processos, o framework também pode ser usado como base de conhecimento para a reengenharia do sistema legado, como proposto na Seção 1.4.

REFERÊNCIAS

AALST, W. M. P. van der; REIJERS, H.; WEIJTERS, A. Business Process Mining: An Industrial Application. **Journal of Information Systems**, Oxford, UK, v. 32, n.5, p. 713–732, jul. 2007.

AGRAWAL, R.; IMIELINSKI, T.; SWAMI, A. Mining association rules between sets of items in large databases. In: ACM SIGMOD International Conference on Management of Data, 25., 1993, Ney York, USA. **Proocedings…** New York: ACM, 1993. p. 207–216.

AGRAWAL, R.; MANNILA, H.; SRIKANT, R.; TOIVONEN, H.; VERKAMO, A. Fast discovery of association rules. In: _____. **Advances in Knowledge Discovery and Data Mining**. Menlo Park, CA: AAAI Press, 1996. p. 307–328.

APACHE, Software Foundation. **Apache JMeter.** Disponível em: <http://jmeter.apache.org>. Acesso em: nov. 2012.

BAJEC, M.; KRISPER, M. A Methodology and Tool Support for Managing Business Rules in Organizations. **Journal of Information Systems**, Oxford, UK, v.30, n.6. p. 423–443, set. 2005.

BELL, J.; BROOKS, D.; GOLDBLOOM, E.; SARRO, R.; WOOD, J. **Re-Engineering Case Study** – Analysis of Business Rules and Recommendations for Treatment of Rules in a Relational Database Environment. 1990. Technical Report, Information Technologies Group, Bellevue Golden, USA, 1990.

BIGGERSTAFF, T. J. Design Recovery for Maintenance and Reuse. **Journal Computer**, New York, USA, v.22, n.7, p. 36–49, jul. 1989.

BORGES, M. R. S.; VINCENT, A. F.; PENADÉS, C.; ARAUJO, R. M. Introducing Business Process into Legacy Information Systems. In: Business Process Management, 2., 2005, Nancy, France. **Proceedings...** Nancy: Springer, 2005, p. 452–457.

BREUER, P.T.; LANO, K. Creating specifications from code - reverse engineering. **Journal of Software Maintenance**, Oxford, USA, v. 3, n. 3, p. 145-162, 1991.

CAI, Z.; YANG, X.; WANG, W. Business Process Recovery for System Maintenance: An Empirical Approach. In: International Conference on Software Maintenance, 25., 2009, Edmonton, Canada. **Proceedings…** Canada: IEEE Press, 2009, p. 399-402.

CHARFI, A.; MEZINI, M. Hybrid Web Service Composition: Business Processes meet Business Rules. In: International Conference on Service Oriented Computing, 2., 2004, New York City, NY, USA. **Proceedings…** New York City: ACM, 2004. p. 30-38.

CORBI, T. Program Understanding – Challege for the 1990s. **IBM Systems Journal**, New York City, NY, USA, v. 28, n. 2, p. 294-306, abr. 1989.

DI FRANCESCOMARINO, C.; MARCHETTO, A.; TONELLA, P.. Reverse Engineering of Business Processes exposed as Web Applications. In: European Conference on Software Maintenance and Reengineering, 13., 2009, Kaiserslautern, Germany. **Proceedings…** Germany: IEEE, 2009. p. 139-148.

ERNST, M. D. Static and dynamic analysis: synergy and duality. In: International Conference on Software Engineering, 25., 2003, Portland, OR, USA. **Proceedings…** Portland: IEEE, 2003. p. 24-27.

GALLIANO, A. G. **O método científico:** teoria e prática. 1. ed. São Paulo: Harbra, 1979.

GHOSE, A.; KOLIADIS, G.; CHEUNG, A. Process Discovery from Model and Text Artefacts. In: IEEE Congress on Services, 4., 2007, Salt Lake City, UT, USA. **Proceedings…** Salt Lake City: IEEE, 2007. p. 167-174.

GUNTHER, C. W.; AALST, W. M. P. van der. A generic import framework for process event logs. In: International Conference on Business Process Management, 4., 2006, Vienna, Austria. **Proceedings…** Vienna: Springer, 2006. p. 81-92.

HALLE, B. Von. Business Rules Applied: Building Better Systems Using the Business Rules Approach. 1. ed. Malden: Wiley, 2001.

HARTMAN, J. Under-standing Natural Programs using Proper Decomposition. In: International Conference on Software Engineering, 13., 1991, Austin, Texas, USA. **Proceedings…** Austin: IEEE, 1991. p. 62-73.

HASEGAWA, R.; KITAMURA, M.; KAIYA, H.; SAEKI, M.: Extracting Conceptual Graphs from Japanese Documents from Software Requirements Modeling. In: Asia-Pacific Conference on Conceptual Modeling, 6., 2009, Wellington, New Zealand. **Proceedings…** Wellington: Australian Computer Society, 2009. p. 87-96.

HAUSLER, P.; PLESZKOCH, M.; LINGER, R.; HEVNER, A. Using Function Abstraction to understand program behavior. **IEEE Software**, Los Alamitos, CA, USA, v. 7, n. 1, p. 55-63, jan. 1990.

HUANG, H.; TSAI, W. T.; BHATTACHARYA, S.; CHEN, X. P.; WANG, Y.; SUN, J. Business Rule Extraction from Legacy Code. In: Computer Software Applications Conference, 20., 1996, Seul, Korea. **Proocedings…** Seal: IEEE, 1996. p. 162-167.

INGVALDSEN, J. E.; GULLA, J. A. Preprocessing support for large scale process mining of SAP transactions. In: International Conference on Business Process Management, 5., 2007, Brisbane, Australia. **Proceedings…** Brisbane: Springer, 2007. p. 30-41.

JASMINE, N. **BPM and SOA: Better Together**, IBM White Paper, New Yourk, USA, jul. 2005. Disponível em: < http://www-05.ibm.com/hu/news/events/2007/akademia/pdf /SOA-BPM.pdf>. Acesso em: ago. 2012.

JUNIOR, A. S. S. **Mineração de Dados Endógenos**. 2002. 98 f. Tese (Doutorado em Engenharia de Sistemas e Computação) – COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2002.

KALSING, A.C.; IOCHPE, C.; THOM, L. H. An Incremental Process Mining Algorithm. In: International Conference on Enterprise Information Systems, 12., 2010, Madeira, Portugal. **Proceedings...** Madeira: Springer, 2010. p. 263-268.

KALSING, A. C.; NASCIMENTO, G. S.; IOCHPE C.; THOM, L. H.; REICHERT, M. An Incremental Process Mining Approach to Extract Knowledge from Legacy Systems. In: International IEEE EDOC Conference, 14., 2010, Vitória, ES, Brasil. **Proceedings...** Vitória: IEEE, 2010. p. 79-88.

KALSING, A. C. **Uma Abordagem Incremental para Mineração de Processos de Negócio.** 2012. 110 f. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2012.

KAMOUN, F. A Roadmap towards the Convergence of Business Process Management and Service Oriented Architecture. **ACM Ubiquity**, New York City, NY, USA, v. 8, n. 14, p. 79–84, abr. 2007.

KARAKOSTAS, V. Modelling and Maintenance of Software Systems at the teleological level. **Journal of Software Maintenance**, New York City, NY, USA, v. 2, n. 1, p. 47-59, mar. 1990.

KARAKOSTAS, V. Intelligent Search and Acquisition of Business Knowledge from Programs. **Journal of Software Maintenance**, New York City, NY, USA, v. 4, n. 1, p. 1-17, mar. 1992.

KETTINGER, W.; TENG, J. T. C.; GUHA, S. Business process change: a study of methodologies, techniques, and tools. **MIS Quarterly**, Minneapolis, MN, USA, v. 21, n. 1, p. 55-98, mar. 1997.

KNOLMAYER, G.; ENDL, R.; PFAHRER, M. Modeling Processes and Workflows by Business Rules. In: International Conference on Business Process Management**,** Models, Techniques, and Empirical Studies, 1., 2000, London, UK. **Proceedings…** London: Springer, 2000. p. 16-29.

KOLBER, A.; HAY, D.; HEALY, K. A.; et al. **Defining Business Rules - What Are They Really?.** 2000. 77 f. Relatório Técnico – Business Rules Group, Seatle, 2000. Disponível em: <http://www.businessrulesgroup.org>. Acesso em: set. 2012.

LEE, S.; KIM, T-Y.; KANG, D.; KIM, K.; LEE, J. Y. Composition of Executable Business Process Models by Combining Business Rules and Process Flows. **Expert Systems with Applications**, Tarrytown, NY, USA, v. 33, n. 1, p. 221-229, jul. 2007.

LIU, K.; ALDERSON, A.; QURESHI, Z. Requirements Recovery from Legacy Systems by Analysing and Modelling Behaviour. In: IEEE International Conference on

Software Maintenance, 17., 1999, Oxford, England, UK. **Proceedings…** Oxford: IEEE, 1999. p. 3-12.

MILLHAM, R. Migration of a Legacy Procedural System to Service-Oriented Computing Using Feature Analysis. In: International Conference on Complex, Intelligent and Software Intensive Systems, 4., 2010, Krakow, Poland. **Proceedings…** Krakow: IEEE. 2010. p. 538 - 543.

MORGAN, T. **Business Rules and Information Systems:** Aligning IT with Business Goals. 1. ed. Indianapolis: Addison-Wesley Professional, 2002.

MPOG. **Portal do Software Público Brasileiro.** Ministério do Planejamento, Orçamento e Gestão. Brasília, abr. 2007. Disponível em: <http://www.softwarepublico.gov.br>. Acesso em: nov. 2011.

NASCIMENTO, G. S.; IOCHPE, C.; THOM, L. H.; REICHERT, M. A Method for Rewriting Legacy Systems using Business Process Managemet Technology. In: International Conference on Enterprise Information Systems, 11., 2009, Milan, Italy. **Proceedings...** Milan: Springer, 2009. p. 57-62.

NASCIMENTO, G. S. do; IOCHPE, C.; THOM, L. H.; KALSING, A.; MOREIRA, A. Identifying business rules to legacy systems reengineering based on BPM and SOA. In: International Conference on Computational Science and Its Applications, 12., 2012, Salvador, BA, Brasil. **Proceedings…** Salvador: Springer, 2012. p. 67-82.

NASCIMENTO, G. S. do; IOCHPE, C.; THOM, L. H.; KALSING, A. Discovering Business Rules in Legacy Systems using Business Rules and Log Mining. In: IEEE International Conference on e-Business Engineering, 10., 2013, Coventry, UK. **Proceedings…** Coventry: IEEE, 2013. p. 207-212.

OMG. **Business Process Modeling Notation (BPMN)**. 2011. 508 f. Relatório Técnico – Object Management Group, Needham, MA, USA, 2011.

PARR, T. J.; QUONG, R. W. ANTLR: A Predicated-LL(k) Parser Generator. **Software: Practice and Experience**, Nova Jersey, v. 25, n. 7, p. 789-810, jul. 1995.

PAPAZOGLOU, M. P.; HEUVEL, W. J. Service oriented architectures: approaches, technologies and research issues. **Journal on Very Large Data Bases**, Secaucus, NJ, USA, v. 16, n. 3, p. 389-415, jul. 2007.

PARADAUSKAS, B.; LAURIKAITIS, A. Business Knowledge Extraction from Legacy Information Systems. **Journal of Information Technologies and Control**, Kaunas, LITHUANIA, v. 35, n. 3, p. 214-221, set. 2006.

PÉREZ-CASTILLO, R.; GUZMÁN, I. de G. R.; ÁVILA-GARCÍA, O.; PIATTINI, M. MARBLE: a modernization approach for recovering business processes from legacy systems. In: Workshop on Reverse Engineering Models from Software Artifacts, 16., 2009, Lille, France. **Proceedings…** Lille: IEEE, 2009. p. 17–20.

PÉREZ-CASTILLO, R.; WEBER, B.; GUZMÁN, I.G-R de; PIATTINI, M. Process Mining through Dynamic Analysis for Modernising Legacy Systems. **IET Software**, London, UK, v. 5, n. 3, p. 304–319, jul. 2011.

PLOTKIN, G. D. A Structural Approach to Operational Semantics. **Journal of Logic and Algebraic Programming**, Philadelphia, PA, USA, v. 60-61, n. 1, p. 17-139, jul. à dez. 2004.

PUTRYCZ, E.; KARK, A. **Recovering Business Rules from Legacy Source Code for System Modernization.** In: International Conference on Advances in Rule Interchange and Applications, 1., 2007, Orlando, Florida, USA. **Proceedings…** Orlando: Springer, 2007. p. 107-118.

RICH, C.; WILLS, L. Recognizing Program's Design – A Graph-Parsing Approach. **IEEE Software**, New York, NY, USA, v. 7, n. 1, p. 82–89, jan. 1990.

ROSS, R. G. **The Business Rule Book:** Classifying, Defining and Modeling Rules. 2. ed., Houston: Business Rule Solutions Inc., 1997.

ROSS, R. G. **Principles of the Business Rule Approach.** 1. ed., Indianapolis: Addison-Wesley, 2003.

SNEED, H. M.; ERDOS, K. Extracting business rules from source code. In: IEEE International Workshop on Program Comprehension, 4., 1996, Berlin, Germany. **Proceedings…** Berlin: IEEE, 1996. p. 240-247.

SOMMERVILLE, J. **Software Engineering**. 8. ed., Indianapolis: Addison-Wesley, 2007.

SOWA, J. F. **Conceptual Graphs.** 2004. Disponível em: <http://www.jfsowa.com/cg>. Acesso em: mar. 2011.

STINEMAN, B. IBM WebSphere ILOG Business Rule Management Systems: The Case for Architects and Developers, IBM White Paper, New Yourk, USA, nov. 2009. Disponível em: < http://public.dhe.ibm.com/common/ssi/rep_wh/n/WSW14091USEN/ WSW14091USEN.PDF>. Acessado em: nov. 2012.

WANG, X.; SUN, J.; YANG, X.; HE, Z.; MADDINENI, S. Business Rules Extraction from Large Legacy Systems. In: European Conference on Software Maintenance and Reengineering, 8., 2004, Tampere, Finland. **Proceedings...** Tampere: IEEE, 2004. p. 249-253.

WARD, M. P.; BENNETT, K. H. Formal methods for legacy systems. **Journal of Software Maintenance and Evolution**, Malden, MA, USA, v. 7, n. 3, p. 203-219, mai. à jun. 1995.

WEIDEN, M.; HERMANS, L.; SCHREIBER, G.; ZEE, S. van der. **Classification and Representation of Business Rules.** 2002. 14 f. Technical Report, Social Science Informatics, University of Amsterdam, Amsterdam, 2002.

WESKE, Mathias. **Business Process Management:** Concepts, Languages, Architectures. 2ª ed. Berlin: Springer, 2012.

WfMC. **Workflow Management Coalition - Terminology & Glossary**. 1999. 65 f. Relatório Técnico – Workflow Management Coalition, Hampshire, UK, 1999.

WOODLEY, T.; GAGNON, S. BPM and SOA: Synergies and Challenges. In: International Conference on Web Information Systems Engineering, 6., 2005, New York, NY, USA. **Proceedings...** New York: Springer, 2005. p. 679–688.

ZOU, Y.; HUNG, M. An approach for extracting workflows from e-commerce applications. In: International Conference Program Comprehension, 14., 2006, Atenas, Grécia. **Proceedings…** Atenas: IEEE, 2006, p. 127-136.