UNIVERSIDADE FEDERAL DE GOIÁS

INSTITUTO DE INFORMÁTICA

AUTORIZAÇÃO PARA PUBLICAÇÃO DE DISSERTAÇÃO EM FORMATO ELETRÔNICO

Na qualidade de titular dos direitos de autor, **AUTORIZO** o Instituto de Informática da Universidade Federal de Goiás – UFG a reproduzir, inclusive em outro formato ou mídia e através de armazenamento permanente ou temporário, bem como a publicar na rede mundial de computadores (*Internet*) e na biblioteca virtual da UFG, entendendo-se os termos "reproduzir" e "publicar" conforme definições dos incisos VI e I, respectivamente, do artigo 5º da Lei nº 9610/98 de 10/02/1998, a obra abaixo especificada, sem que me seja devido pagamento a título de direitos autorais, desde que a reprodução e/ou publicação tenham a finalidade exclusiva de uso por quem a consulta, e a título de divulgação da produção acadêmica gerada pela Universidade, a partir desta data.

Título: Arquitetura e Implementação de Mecanismos para Suporte a Regras de Negócio em Sistemas de Informação

Autor(a): Glauber Boff

Goiânia, 12 de Abril de 2010.

Glauber Boff – Autor

Juliano Lopes de Oliveira – Orientador

UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

GLAUBER BOFF

Arquitetura e Implementação de Mecanismos para Suporte a Regras de Negócio em Sistemas de Informação

Goiânia
2010

GLAUBER BOFF

Arquitetura e Implementação de Mecanismos para Suporte a Regras de Negócio em Sistemas de Informação

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em Computação.

Área de concentração: Engenharia de Software.

Orientador: Prof. Juliano Lopes de Oliveira

Goiânia
2010

GLAUBER BOFF

Arquitetura e Implementação de Mecanismos para Suporte a Regras de Negócio em Sistemas de Informação

Dissertação defendida no Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Mestre em Computação, aprovada em 12 de Abril de 2010, pela Banca Examinadora constituída pelos professores:

Prof. Juliano Lopes de Oliveira
Instituto de Informática – UFG
Presidente da Banca

Profa. Dra. Ana Paula Laboissière Ambrósio
Instituto de Informática – UFG

Prof. Dr. Marcello Thiry Comicholi da Costa
Universidade do Vale do Itajaí – UNIVALI

Todos os direitos reservados. É proibida a reprodução total ou parcial do trabalho sem autorização da universidade, do autor e do orientador(a).

Glauber Boff

Graduou-se em Ciências da Computação pela Universidade Federal de Goiás. Durante sua graduação, foi pesquisador do CNPq em dois projetos de iniciação científica no Instituto de Informática. Durante o mestrado na UFG, foi bolsista do CNPq e desenvolveu pesquisa empírica na área de Sistemas de Informação, sob orientação do Prof. Dr. Juliano Lopes de Oliveira.

Dedico este trabalho **aos meus pais, Elza e Alfeu**, que sempre me apoiaram e incentivaram a vencer mais este desafio em minha vida!

Agradecimentos

Agradeço a Deus, em primeiro lugar, pela vida e por ter me iluminado, dado saúde e força ao longo desses anos dedicados ao Mestrado.

Aos meus pais, Elza e Alfeu, pelo amor, atenção e apoio constantes, apesar da distância entre Goiânia e Brasília. Sem eles não teria conseguido chegar até aqui!

Ao meu orientador, Prof. Dr. Juliano Lopes de Oliveira, pela orientação ao longo dos anos de universitário, desde a época dos projetos de iniciação científica até a conclusão do Mestrado. Sua orientação foi fundamental não apenas para meu futuro profissional, mas também para minha vida pessoal.

Aos meus irmãos, Gláucia e Alberto, que sempre me apoiaram e torceram pelo meu sucesso. Em todos esses anos morando juntos em Goiânia, o companheirismo do meu irmão foi de grande importância, pois além de irmão, é meu melhor amigo!

À minha amiga Lívia Porto, que esteve sempre presente nesses anos de Mestrado, me acompanhou e apoiou, tanto nos momentos de dificuldade como nos momentos de alegria e diversão.

Aos colegas de Mestrado, Fabiana Freitas, Patrícia Gomes, Alexandre Cláudio, com quem compartilhei momentos relacionados ao trabalho de pesquisa na UFG e viagens a congressos.

Aos colegas de projeto na empresa Estratégia Tecnologia da Informação, pela ajuda e pelo trabalho em equipe, principalmente o colega Alexandre Cláudio, que muito colaborou para a conclusão deste trabalho.

Ao pessoal da secretaria do Mestrado do INF-UFG pela paciência e pelo apoio prestado aos estudantes.

Ao CNPq pelo auxílio financeiro.

"O sucesso vem geralmente àqueles que estão muito ocupados para estar procurando por ele."

Henry David Thoreau,

.

Resumo

Boff, Glauber. **Arquitetura e Implementação de Mecanismos para Suporte a Regras de Negócio em Sistemas de Informação**. Goiânia, 2010. 134p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

O desenvolvimento de Sistemas de Informação (SI) deve ser realizado de forma que as mudanças no negócio do cliente, que podem ser frequentes, possam ser refletidas no sistema, sem que ele perca sua consistência e confiabilidade. Para se atingir essas características, a portabilidade e manutenibilidade, que são desafios para a Engenharia de Software, são importantes. Há dois requisitos essenciais que, quando presentes no processo de desenvolvimento, facilitam o tratamento da complexidade do sistema. O primeiro requisito diz que deveria ser possível descrever as regras de negócio (RN) usando uma linguagem de alto nível, gerando um modelo único e independente de implementação. O segundo requisito demanda que deveria ser possível gerar código fonte a partir do modelo abstrato de RN e integrá-lo ao SI. Neste trabalho é apresentada uma abordagem para atender a esses requisitos. Esta abordagem abrange especificação, modelagem, implementação e avaliação de regras de negócio em software de forma a facilitar o desenvolvimento e a manutenção de SI complexos. Em vez de codificar as RN diretamente nas aplicações, adotou-se a linguagem OCL, uma linguagem abstrata e independente de plataforma, para definir um modelo único de RN para o SI. Para armazenar as regras ao longo do ciclo de vida do SI de forma segura e de fácil acesso foi criado um repositório centralizado no banco de dados, que permite realizar o controle de todas as versões das RN. Na implementação, regras em OCL são convertidas automaticamente para SQL e armazenadas em banco de dados como procedimentos armazenados para posterior avaliação por um componente específico da aplicação. Esta abordagem facilita a manutenção do SI pela descrição centralizada e em alto nível de todas as RN de um sistema. Para demonstrar a viabilidade da abordagem proposta, uma validação empírica foi realizada no contexto de um projeto de pesquisa apoiado pelo CNPq que visa a otimização da produção de gado de corte.

Palavras–chave

Sistemas de Informação, Regras de Negócio, Object Constraint Language (OCL)

Abstract

Boff, Glauber. **An Approach to Support the Life Cycle of Business Rules in Information Systems**. Goiânia, 2010. 134p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

The development of Information Systems (IS) should be performed so that changes in the client's business, which may be common, may be reflected in the system, without reducing its consistency and reliability. To achieve these features, portability and maintainability, which are challenges for Software Engineering, are very important. There are two essential requirements which, when present in the development process, make it easier to deal with the system complexity. The first requirement states that it should be possible to describe the IS business rules (BR) using a high level language, generating a single, implementation independent model. The second requirement demands that it should be possible to generate software source code from the abstract BR model, and to integrate this code to the IS. This work presents an approach to meet these requirements. This approach covers the specification, modeling, implementation and evaluation of business rules in software to facilitate the development and maintenance of complex IS. Instead of hard-coding the BR into the applications, this approach adopts OCL as a platform independent high-level language to define a single BR model for the IS. To store the rules safely throughout the IS life cycle, a centralized repository in the database was developed, which enables control over all BR versions. During implementation, rules in OCL are automatically converted to SQL and stored in the database as *stored procedures* for further evaluation by a specific component of the application. This approach facilitates the maintenance of the IS by the centralized and high-level description of all BR. To demonstrate the feasibility of the proposed approach, an empirical validation was performed in the context of a research project funded by CNPq, which aims to optimize the production of beef cattle.

Keywords

Information Systems, Business Rules, Object Constraint Language (OCL)

Sumário

Lista de Figuras **11**

Lista de Tabelas **12**

Lista de Códigos de Programas **13**

1 Introdução **14**
- 1.1 Objetivos do Trabalho 16
- 1.2 Metodologia da Pesquisa Desenvolvida 17
- 1.3 Organização da Dissertação 19

2 Regras de Negócio em Sistemas de Informação **20**
- 2.1 Conceitos e Características 20
- 2.2 Taxonomias de Regras de Negócio 22
- 2.3 Abordagens para Tratamento de Regras de Negócio 23
  - 2.3.1 Abordagem de Halle 24
  - 2.3.2 Abordagem MDE 26
  - 2.3.3 Comparativo entre as Abordagens 30
- 2.4 Linguagem OCL 31
  - 2.4.1 Características de OCL 32
  - 2.4.2 Tipos de Dados em OCL 33
  - 2.4.3 Tipos de Expressões 34
    - Invariantes 34
    - Pré e pós-condições 34
    - Valores iniciais e regras de derivação 35
    - Expressões de consulta 35
  - 2.4.4 Vantagens da Utilização de OCL 36
- 2.5 Repositório de Regras 37
  - 2.5.1 Requisitos para um Repositório de Regras 38
  - 2.5.2 Formas de Armazenamento 39

3 Engenharia de Requisitos e Regras de Negócio **40**
- 3.1 Processo de Engenharia de Requisitos 41
- 3.2 Análise de Regras de Negócio 43
  - 3.2.1 Modelo Conceitual e Assertivas Estruturais 43
  - 3.2.2 Regras de Validação e Derivação 45
- 3.3 Documentação de Regras de Negócio 46
- 3.4 Exemplos de Regras de Negócio Especificadas na Fase de Análise 48

4 Projeto de Regras de Negócio **52**
4.1 Convenções para a escrita de regras em OCL 52
4.2 Exemplo de RN em OCL 56

5 Construção e Implantação de Regras de Negócio **59**
5.1 Editor OCL 61
5.1.1 Área de Edição de Regras 63
5.1.2 Área de Comandos 64
5.1.3 Área de Mensagens 68
5.2 Mapeamento entre OCL e SQL 69
5.2.1 Mapeamento de Tipos de Dados Básicos e suas Operações 69
5.2.2 Mapeamento de Tipos de Coleção e suas Operações 73
Operações Padrão 73
Operações de Iteração sobre Coleções 75
5.3 Gerador de Código 77
5.3.1 Forma de Implementação das RN 78
5.3.2 Etapas da Geração de Código 78
5.3.3 Exemplos 82
5.4 Implantação de RN 83

6 Avaliação de Regras de Negócio em SI **86**

7 Aplicação da Proposta: Sistema de Informação para o Agronegócio **94**
7.1 Sistema SIA: Visão Geral 94
7.2 Regras de Negócio no SIA 95
7.2.1 Cenário 1 - Ocorrência de Cirurgia de Animal 95
7.2.2 Cenário 2 - Ajuste de Estoque de Produto 99
7.2.3 Cenário 3 - Rejeição de Cria 104
7.2.4 Cenário 4 - Cálculo de Indicadores 108

8 Conclusões **111**
8.1 Considerações Finais 111
8.2 Trabalhos Correlatos 113
8.3 Resultados Obtidos 117
8.4 Trabalhos Futuros 118

Referências Bibliográficas **120**

A Glossário **125**

B DTD do Arquivo de Mapeamento OCL/SQL **127**

C Padrões de Mapeamento OCL-SQL **128**

Lista de Figuras

1.1 Elementos que compõem um Sistema de Informação. 15

2.1 Fases da metodologia de desenvolvimento de sistemas baseados em regras de negócio [34]. 25
2.2 Relação entre os tipos de modelos CIM, PIM e PSM. 28
2.3 Esquema da transformação do modelo PIM para PSM (adaptada de [40]). 28
2.4 Combinação da abordagem de Halle com a MDE. 31
2.5 Modelo conceitual referente ao estoque de produtos em uma propriedade rural. 36

3.1 Atividades do processo de Engenharia de Requisitos (adaptada de [61]). 42
3.2 Esquema conceitual do Modelo de Meta Objetos (adaptado de [20]). 44
3.3 Esquema conceitual do repositório de regras de negócio. 47
3.4 Janela utilizada para o cadastro, atualização ou exclusão de regras de negócio. 49
3.5 Esquema conceitual simplificado de uma organização. 50

5.1 Componentes do *framework* para a geração de SI [18]. 59
5.2 Arquitetura de Componentes do mecanismo de gestão de regras de negócio. 60
5.3 Pacote de Software da camada de interface do mecanismo de regras de negócio. 61
5.4 Pacotes de Software do mecanismo de regras de negócio. 62
5.5 Janela principal do Editor OCL. 63
5.6 Área de edição de regra OCL. 64
5.7 Janela utilizada para a escolha da RN a ser aberta. 65
5.8 Janela utilizada para a escolha da versão da RN a ser aberta. 66
5.9 Diagrama de colaboração para a operação Verificar Consistência. 66
5.10 Diagrama de atividades referente às operações permitidas no Editor OCL. 67
5.11 Janela do Editor OCL com a mensagem de erro de consistência da regra. 68
5.12 Diagrama de Colaboração referente à geração de código SQL. 79
5.13 Diagrama de colaboração para a implantação de uma regra em SQL no SGBD. 84

6.1 Diagrama de atividades referente à avaliação de RN em tempo de execução do SI. 88

7.1 Modelo conceitual referente à ocorrência de Cirurgia. 96
7.2 Modelo conceitual referente ao Ajuste de Estoque de Produtos. 99
7.3 Modelo conceitual referente à ocorrência de Rejeição de Cria. 104
7.4 Modelo conceitual referente à ocorrência de Pesagem de Animal. 108

Lista de Tabelas

2.1 *Quadro comparativo entre as taxonomias.* 24
2.2 *Tipos básicos da OCL [48].* 33
2.3 *Tipos de coleção da OCL [48].* 34

3.1 *Análise da regra RN-3.* 50
3.2 *Análise da regra RN-4.* 51
3.3 *Análise da regra RN-5.* 51

5.1 *Mapeamento entre tipos básicos de OCL e SQL.* 69
5.2 *Operações de tipos booleanos OCL e mapeamentos para SQL.* 70
5.3 *Operações para os tipos Integer e Real e mapeamento para SQL.* 71
5.4 *Operações para o tipo String e mapeamento para SQL.* 72
5.5 *Operações para o tipo Date e mapeamento para SQL.* 73
5.6 *Operações padrão para os tipos de coleção OCL.* 74
5.7 *Exemplo de mapeamento para SQL da operação **count** de OCL.* 74
5.8 *Exemplo de mapeamento para SQL de uma operação includes de OCL.* 75
5.9 *Exemplo de mapeamento para SQL da operação **sum** de OCL.* 75
5.10 *Operações de iteração para os tipos de coleção OCL.* 76
5.11 *Exemplo de mapeamento para SQL da operação de navegação entre entidades do modelo.* 76
5.12 *Exemplo de mapeamento para SQL da operação **exists** de OCL.* 77
5.13 *Exemplo de mapeamento para SQL da operação **forAll** da OCL.* 77

Lista de Códigos de Programas

2.1 Exemplo de expressão de consulta em OCL. 36
4.1 Padrão básico para a definição de regra de validação em OCL. 53
4.2 Padrão básico para a definição de regra de derivação em OCL. 53
4.3 Código OCL para as regras RN-3 e RN-4. 57
4.4 Código OCL para a regra RN-5. 58
5.1 Código JAVA referente ao método responsável por realizar a geração de código SQL. 80
5.2 Código JAVA referente ao método responsável por realizar a geração de código SQL. 81
5.3 Código SQL gerado a partir do código OCL apresentado no Código 4.3. 82
5.4 Código SQL gerado a partir do código OCL apresentado no Código 4.4. 83
5.5 Código SQL da *stored procedure* que implanta uma RN. 85
6.1 Trecho de código utilizado para invocar o método *gerenciarRegras*. 89
6.2 Método responsável por tratar a execução das RN. 90
6.3 Método responsável por gerenciar a avaliação das RN. 91
6.4 Método responsável por executar uma regra de validação. 92
6.5 Método responsável por executar uma regra de derivação. 93
7.1 Regra RN-1 especificada em OCL. 97
7.2 Código gerado referente à RN-1 especificada em OCL. 98
7.3 Regra RN-2.1 especificada em OCL. 100
7.4 Código gerado referente à regra RN-2.1 especificada em OCL. 101
7.5 Regras RN-2.2 e RN-2.3 especificadas em OCL. 102
7.6 Código gerado referente às regras RN-2.2 e RN-2.3 especificadas em OCL. 103
7.7 Regras RN-3.1 e RN-3.2 especificadas em OCL. 105
7.8 Código gerado referente às regras RN-3.1 e RN-3.2 especificadas em OCL. 106
7.9 Código gerado referente às regras RN-3.1 e RN-3.2 especificadas em OCL (continuação). 107
7.10 Regra RN-4 especificada em OCL. 109
7.11 Código gerado referente à regra RN-4 especificada em OCL. 110
B.1 Código DTD do arquivo XML para o mapeamento de OCL para SQL. 127
C.1 Padrões que definem o corpo de *stored procedures* para regras de validação e derivação. 129
C.2 Padrões que definem o mapeamento de tipos de atributos, literais e expressão condicional. 130
C.3 Padrões que definem o mapeamento de expressões lógicas. 131
C.4 Padrões que definem o mapeamento de expressões relacionais. 132
C.5 Padrões que definem o mapeamento de expressões aditivas, multiplicativas e unárias. 133
C.6 Padrões que definem o mapeamento de algumas operações OCL. 134

CAPÍTULO 1

Introdução

A intensa globalização dos negócios que vem ocorrendo desde o final do século XX tem feito com que a concorrência entre as empresas aumente cada vez mais. Isso influencia a forma como as empresas realizam seus negócios, pois são obrigadas a adaptar-se a uma nova realidade em que mudanças são constantes. Para isso é necessário adotar processos de negócio menos burocráticos e mais flexíveis.

Neste cenário, as empresas passaram a fazer uso de Sistemas de Informação para auxiliar a tomada de decisões, otimizar a execução dos seus processos de negócio e, consequentemente, fortalecer sua posição competitiva no mercado.

Um Sistema de Informação (SI) consiste em um conjunto de elementos que, de maneira organizada e cooperada, realizam o processamento das informações que fluem em uma organização. O funcionamento básico de um SI pode ser resumido em duas atividades: i) obter os recursos de dados como entrada e processá-los de modo a obter produtos de informação como saída; ii) disponibilizar as informações geradas por um SI para a organização, com o objetivo de auxiliar a tomada de decisões e também servir como entrada para outros Sistemas de Informação.

A utilização de um Sistema de Informação pode trazer vários benefícios para as organizações, tais como:

- auxílio no suporte à tomada de decisão, identificação de oportunidades de negócios e de vantagens competitivas;
- mais precisão nas informações, ocasionando menos erros;
- valor agregado e melhor qualidade dos produtos, bens e serviços;
- sistemas mais eficientes e com maior eficácia, gerando aumento da rentabilidade;
- diminuição de desperdícios, reduzindo custos.

O objetivo da utilização de um SI em uma organização é garantir a execução correta dos processos com base em regras de negócio, e fornecer apoio para facilitar o fluxo de informação. O SI fornece suporte integrado à entrada e registro de dados, processamento de dados, à produção de informação, à geração de relatórios e às necessidades de saída de dados.

Para atender seus objetivos, em geral um SI abrange um conjunto de elementos organizados de maneira integrada, como mostra a Figura 1.1:

- *pessoas*: podem ser técnicos (analistas de sistemas, programadores e outros que constroem e mantêm o SI) ou usuários (todos os que utilizam, direta ou indiretamente, o SI para processar informações);
- *procedimentos*: representam o conjunto de ações que um sistema deve executar com a finalidade de atingir os objetivos esperados;
- *hardware*: abrange computadores e dispositivos periféricos, como impressoras, processadores, monitores, teclados, dispositivos de mídia, etc.;
- *software*: composto por programas de aplicação que implementam certos procedimentos do SI;
- *bancos de dados*: são estruturas utilizadas para armazenamento de dados relacionados ao negócio;
- *redes*: são sistemas de conexões que permitem a comunicação entre diferentes componentes do SI.

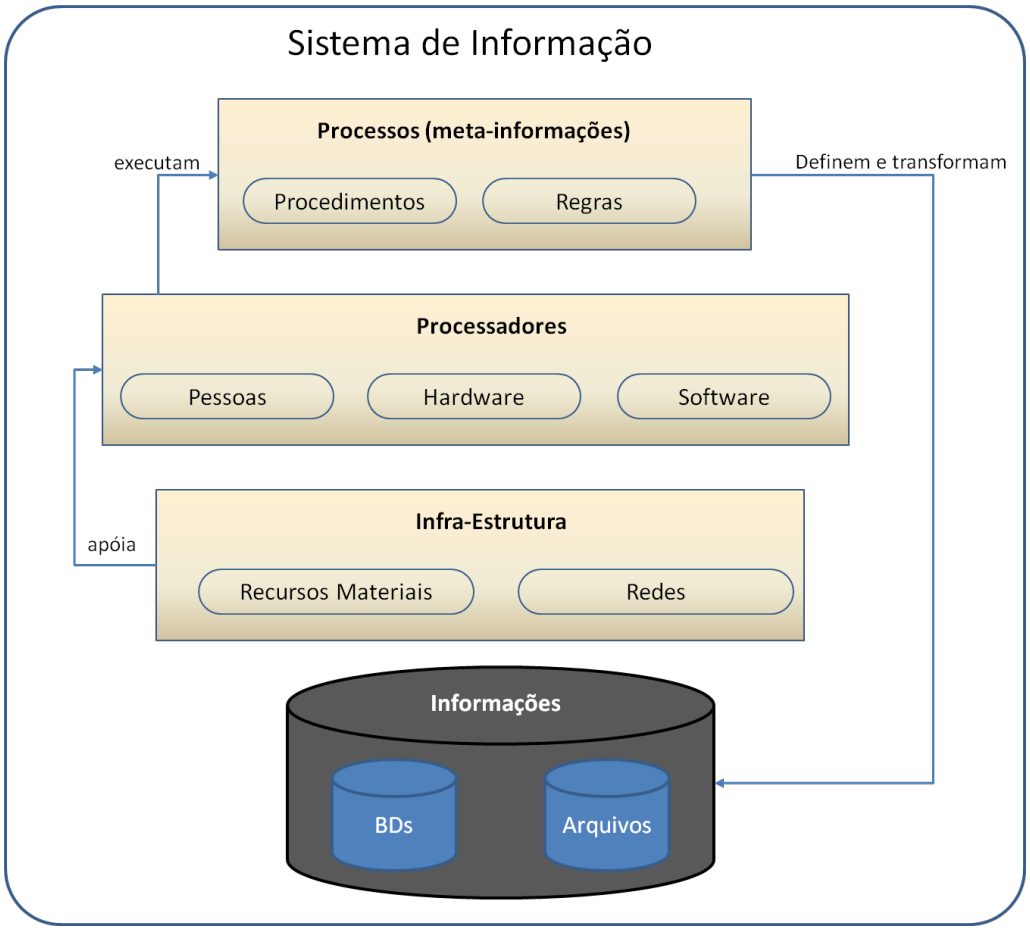

Figura 1.1: *Elementos que compõem um Sistema de Informação.*

Cada componente de um SI pode estar associado a um conjunto de regras de negócio. A função delas é especificar como cada componente contribui para guiar o comportamento do negócio de uma organização.

Como Sistemas de Informação devem refletir de maneira fiel o processo de negócio de uma organização, eles não podem ser estáticos, ou seja, devem ser ajustados à medida que o negócio sofre alterações. A abordagem de desenvolvimento de SI baseado em regras de negócio considera que os sistemas devem ser construídos de forma a permitir que as regras de negócio variem ao longo do tempo, sem que o SI perca sua consistência e confiabilidade [34].

As regras de negócio modeladas por um SI devem ser entendidas não apenas pelos técnicos que as desenvolveram em linguagem técnica, mas também pelos especialistas no negócio [42]. Assim, surge a necessidade de modelar o negócio e suas regras utilizando-se linguagens de alto nível, de modo a facilitar a comunicação entre os envolvidos.

Este trabalho de pesquisa está focado no componente *software* de um SI, e mais especificamente na proposta de uma abordagem para apoiar o desenvolvimento de regras de negócio relacionadas a produtos de *software* em Sistemas de Informação. Essas regras estão relacionadas às características funcionais e de qualidade exigidas por uma organização.

Diversos trabalhos relacionados ao desenvolvimento de regras de negócio em SI podem ser encontrados na literatura. Em [58] é apresentada uma abordagem para a validação de regras de negócio em SI a partir de diagramas UML, de onde as regras são extraídas e processadas. Em [46, 10] discute-se sobre o armazenamento de regras em um repositório e são levantados alguns requisitos, que são levados em consideração para o desenvolvimento do presente trabalho. Em relação a ferramentas, foram encontradas algumas que podem ser utilizadas para auxiliar a representação e implementação de regras de negócio, como o *Dresden OCL Toolkit* [32], *Fujaba Tool Suite* [56], Atenas [17] e Odyssey-PSW [15]. Esses trabalhos são explicados com mais detalhes na Seção 8.2.

1.1 Objetivos do Trabalho

A habilidade dos especialistas do negócio em examinar, revisar e corrigir regras de negócio é limitada quando elas são representadas de maneira embutida no código da aplicação em Sistemas de Informação. Como conseqüência, a habilidade das organizações em modificar tais regras para acompanhar e responder às mudanças do ambiente é consideravelmente prejudicada.

Este trabalho está voltado ao desenvolvimento de uma abordagem que auxilie a comunicação entre todos os envolvidos no desenvolvimento de um SI (desde analistas

de negócio até implementadores) e sua adaptação às constantes mudanças no negócio. O analista de negócio é aquele que conhece detalhes do negócio e é responsável por especificar suas regras. O projetista de regras é o responsável por escrever essas regras em uma linguagem técnica, não ambígua e independente de plataforma, mas que permita efetuar sua transformação automática para linguagens de implementação. Neste cenário, o implementador tem sua função reduzida, já que o código de implementação é gerado automaticamente. Entretanto, podem ocorrer casos em que a ferramenta de transformação não consiga realizar a transformação (no caso de regras complexas), e a atuação de um implementador torna-se necessária.

A principal contribuição da pesquisa aqui apresentada é uma arquitetura de software para organizar o desenvolvimento e o gerenciamento de regras de negócio em SI, de modo a facilitar a comunicação ao longo do processo de desenvolvimento. Entende-se por arquitetura de software "a estrutura de componentes de um programa/sistema, seus inter-relacionamentos, princípios e diretrizes que governam seu projeto e evolução ao longo do tempo" [28].

Outros objetivos desta pesquisa são:

- estudar e consolidar os conceitos de regras de negócio, requisitos e restrições de integridade no contexto de SI;
- definir um processo de desenvolvimento de regras de negócio em SI baseado em um modelo independente de tecnologia computacional;
- utilizar um modelo independente de tecnologia computacional para representação de regras de negócio em SI;
- projetar e implementar um repositório de regras de negócio que permita realizar o registro e acesso às regras de negócio de maneira simples e controlada;
- aplicar as idéias propostas neste trabalho no desenvolvimento de um SI real.

Este trabalho não pretende propor um processo completamente novo para o desenvolvimento de Sistemas de Informação, mas sim incorporar novos conceitos relacionados ao desenvolvimento baseado em modelos, como sugere a abordagem MDE (*Model Driven Engineering*) [53]. A viabilidade de utilização desse processo é avaliada, com foco na capacidade de expressão do mecanismo proposto, e não na eficiência de sua implementação.

1.2 Metodologia da Pesquisa Desenvolvida

Na primeira etapa deste trabalho, a pesquisa se concentrou no levantamento da bibliografia existente e no estudo de diversas abordagens, metodologias, tecnologias e arquiteturas relacionadas aos aspectos que o motivaram. Assim, este estudo abrangeu os

conceitos e taxonomias de regras de negócio, abordagens para o tratamento dessas regras em Sistemas de Informação e, por fim, conceitos de engenharia dirigida por modelos (MDE) e geração automática de código.

Para a pesquisa bibliográfica utilizou-se como principais fontes a biblioteca digital da ACM [1] e do IEEE Xplore [38]. Além dessas bibliotecas, foram pesquisados periódicos relacionados ao tema deste trabalho, como *Transactions on Database Systems - TODS* [2] e *Transactions on Information Systems - TOIS* [3]. Outra fonte bastante utilizada foi o Portal de Periódicos da CAPES [11].

Em seguida, uma vez que na literatura pesquisada sobre modelagem de regras de negócio foram encontradas diversas abordagens, decidiu-se concentrar esforços na identificação das principais características de cada uma. Neste sentido foram analisadas desde propostas para a modelagem de regras até ferramentas completas já implementadas. Os requisitos para a abordagem proposta nesta dissertação foram selecionados a partir das limitações observadas nesta análise.

Uma vez definidos os requisitos da nova abordagem, os próximos passos foram o projeto e a implementação de um mecanismo para armazenar, gerenciar e viabilizar a avaliação de regras em Sistemas de Informação. O mecanismo desenvolvido atende os requisitos selecionados e utiliza a linguagem OCL para representar regras em um modelo independente de plataforma.

A última etapa do trabalho envolveu a realização de testes empíricos do mecanismo implementado. Os testes foram realizados no contexto de um SI de grande porte voltado para a melhoria da atividade agropecuária. Foram especificadas, modeladas e implantadas diversas regras utilizando a arquitetura de software desenvolvida neste trabalho. Depois de implantadas, as regras puderam ser avaliadas no SI, com dados reais fornecidos por pessoas que detêm conhecimento do negócio.

Ao longo de todo o desenvolvimento do trabalho, vários artigos descrevendo partes das idéias aqui utilizadas foram submetidos a conferências e periódicos especializados.

Um dos artigos aceitos foi publicado no *VII Workshop de Teses e Dissertações em Bancos de Dados do XXIII Simpósio Brasileiro de Banco de Dados (WTDBD)* [6]. Outro artigo foi publicado no *VI Workshop de Manutenção de Software Moderna do VIII Simpósio Brasileiro de Qualidade de Software* [7]. Um terceiro artigo foi publicado no *XXIII Simpósio Brasileiro de Engenharia de Software (SBES)* [18].

O artigo apresentado no *VI WMSWM* [7] foi premiado dentre os melhores trabalhos do evento, e uma versão expandida foi publicada no periódico INFOCOMP [8].

O artigo [5] foi aceito para publicação em uma conferência internacional. No entanto, o trabalho não foi publicado pois não foi possível que nenhum dos autores

estivessem presentes no local do evento para apresentá-lo.

1.3 Organização da Dissertação

O Capítulo 2 trata de conceitos fundamentais para o entendimento do trabalho, abordando regras de negócio, suas taxonomias, e abordagens para o seu desenvolvimento em Sistemas de Informação. Esse capítulo apresenta também uma introdução à linguagem OCL, que é utilizada para expressar regras de negócio em modelos independentes de plataforma. Além disso, o capítulo discute repositórios de regras e os requisitos para o seu desenvolvimento.

O Capítulo 3 descreve a primeira fase do processo de desenvolvimento de regras de negócio proposto neste trabalho. Esta primeira fase, Engenharia de Requisitos e Regras de Negócio, é descrita em detalhes e um exemplo demonstra a utilização da proposta.

O Capítulo 4 descreve a fase de Projeto (Design) de regras de negócio, mostrando como a linguagem OCL deve ser utilizada para representar, no modelo abstrato, as regras analisadas na fase de Engenharia de Requisitos e Regras de Negócio.

O Capítulo 5 explica a fase de Construção e Implantação de regras de negócio, apresentando a arquitetura completa da solução e descrevendo em detalhes os componentes. O capítulo descreve também a geração de código a partir de um conjunto de padrões de mapeamento da linguagem OCL para SQL, que neste trabalho é a linguagem procedural (*plPgSQL*) do SGBD PostgreSQL, para posterior implantação.

O Capítulo 6 explica a fase de Avaliação de regras de negócio, onde as regras analisadas, projetadas e implementadas nas fases anteriores do processo são incorporadas ao SI e utilizadas para configurar seu comportamento.

O Capítulo 7 apresenta exemplos de utilização do mecanismo proposto neste trabalho em um SI real, utilizado para validação das idéias e resultados obtidos. São descritos cenários reais de utilização da arquitetura implementada para o desenvolvimento de regras em um SI para o Agronegócio.

Finalmente, o Capítulo 8 traz as considerações finais deste trabalho de pesquisa, comparando-o com trabalhos correlatos e propondo extensões e direções para sua melhoria.

CAPÍTULO 2

Regras de Negócio em Sistemas de Informação

O desenvolvimento de Sistemas de Informação envolve a modelagem do domínio do negócio para elicitar os requisitos do sistema. As regras de negócio (RN) são consideradas um tipo de requisito e formalizam as restrições que devem ser impostas para garantir a consistência dos dados e processos de negócio.

Existem abordagens variadas para o desenvolvimento de RN, cada uma com suas peculiaridades. Neste capítulo são descritas duas abordagens nas quais este trabalho está baseado: i) a abordagem de Halle [34] e a abordagem MDE [53].

A Seção 2.1 apresenta os conceitos e características relacionados a RN, independentemente do domínio de aplicação. A Seção 2.2 descreve diferentes taxonomias de RN e como elas se relacionam. A Seção 2.3 apresenta e compara as abordagens para o desenvolvimento de RN em SI. A Seção 2.4 discute o uso da linguagem OCL para especificar outros tipos de regras de negócio que não são suportadas pelo MMO. São apresentados conceitos da linguagem e os tipos de regras que podem ser especificadas. Finalmente, a Seção 2.5 apresenta as principais características e requisitos de um repositório de regras.

2.1 Conceitos e Características

O termo **regra de negócio (RN)** é importante tanto no domínio do negócio de uma organização como no contexto operacional de um SI [57]. Do ponto de vista do negócio, essas regras são políticas que definem restrições ou metadados sobre as operações do negócio [50], enquanto que da perspectiva do SI, essas regras são consideradas restrições sobre os dados e sobre as operações que as manipulam [31].

Diversos autores propõem definições para o termo regra de negócio, todas elas correlacionadas e voltadas para o atendimento das necessidades do negócio de uma organização:

a. [46] trata RN como *"... elementos-chave na definição das intenções e necessidades do negócio, ou reflexões de como a organização trabalha ou como pretende trabalhar no futuro"*.

b. [37] considera que RN seja uma sentença sobre como o negócio é feito; por exemplo, restrições relacionadas aos estados e processos de uma organização.

c. [4] define RN como uma lei ou uso-comum que guia o comportamento ou ações dos atores ligados à organização.

d. [49] considera que RN é uma declaração de políticas ou condições relacionadas ao negócio que devem ser satisfeitas para garantir sua integridade.

e. Para [13], RN reflete as necessidades de uma aplicação, modelando a reação aos eventos que ocorrem no mundo real, e podendo impactar no estado do banco de dados.

Levando em consideração essas variadas definições existentes na literatura, [41] define regras de negócio como *"... declarações sobre o modo como uma organização faz negócio. Elas refletem as políticas do negócio. As organizações possuem políticas com o intuito de: satisfazer os objetivos do negócio, satisfazer seus clientes, fazer bom uso dos recursos, e adequar-se às leis ou convenções gerais do negócio."*

Segundo [33], regras de negócio apresentam características que são encontradas em muitas organizações, e deveriam ser expressas declarativamente, em linguagem natural, bem-formadas e orientadas para as metas do negócio. Essa visão é compartilhada por [46], que diz que toda regra de negócio deve ser:

- **Atômica:** uma regra de negócio não pode ser subdividida em partes menores, pois isso pode causar perda de informação ou perda semântica;

- **Uma declaração sobre o negócio:** a regra deve descrever a lógica do negócio e não pode conter detalhes relacionados com a tecnologia na qual será implementada;

- **Restritiva:** a regra deve limitar o escopo das ações que podem ser aplicadas no contexto do negócio;

- **Representada em linguagem natural:** expressar as regras de negócio não deveria exigir treinamento específico ou uso de alguma ferramenta. No entanto, pode ocorrer ambigüidade na regra expressa em linguagem natural. Em vista disso, torna-se necessário o mapeamento da regra expressa em linguagem natural para uma linguagem formal baseada em princípios matemáticos por um projetista de regras. Assim os especialistas do negócio não precisam entender detalhes técnicos dessa linguagem;

- **Rastreável:** deve ser possível saber como cada regra de negócio está relacionada aos demais componentes do sistema de informação (inclusive as demais regras) e rastreá-las ao longo de seu ciclo de vida (desde a sua criação até o momento de sua

execução, e da sua desativação pelo negócio).

- **Estruturada:** as regras devem ser escritas de tal modo que facilitem sua leitura e entendimento. Padrões estruturais devem restringir o número de opções para se definir uma regra.

2.2 Taxonomias de Regras de Negócio

Diversas taxonomias de RN podem ser encontradas na literatura. A taxonomia apresentada em [31], por exemplo, propõe a divisão de regras de negócio em **Assertivas estruturais**, **Regras de Ação** e **Regras de derivação**. Assertivas Estruturais correspondem a regras que definem conceitos ou declarações sobre a estrutura de uma organização. Regras de Ação definem condições ou restrições que limitam algum aspecto de uma organização; e as Regras de Derivação especificam a maneira de se obter um novo conhecimento relacionado a uma organização a partir de outro conhecimento existente.

A categorização de regras de negócio proposta em [9] divide as RN em **Regras Restritivas**, **Regras Evento-ação** e **Regras de Derivação**. Regras restritivas são utilizadas para manter a integridade das informações referentes ao negócio. As regras do tipo evento-ação expressam as condições em que as atividades devem ser executadas, ou seja, indicam um conjunto de condições que devem ser satisfeitas antes da execução delas. Regras de derivação indicam uma forma de se obter conhecimento do domínio estrutural que não precisa ser armazenado e que seu valor pode ser obtido dinamicamente usando informações existentes.

Outra classificação, descrita em [44], divide as RN em **Regras Restritivas** e de **Derivação** (definições semelhantes às propostas em [9]), enquanto [37] propõe a divisão em **Regras de Integridade** e **Regras de Automação**. As regras de integridade são semelhantes às assertivas estruturais e as regras de automação são semelhantes às regras de ação propostas em [31].

A taxonomia de [34] define três tipos de regras: **restrições**, **cálculos** e **inferências**. Restrições podem ser obrigatórias ou sugeridas em relação ao comportamento do negócio. Uma **restrição obrigatória** é uma declaração que expressa uma circunstância incondicional que pode ser avaliada como verdadeira ou falsa, de forma a manter a integridade do negócio. As **regras de cálculos** são semelhantes às regras de derivação propostas em [31] e [9], e permitem definir algoritmos para calcular o valor de um termo a partir de outros conhecimentos do negócio. As **inferências** são declarações que testam condições que, se satisfeitas, permitem concluir sobre a validade de um novo fato. Regras de cálculos e inferências permitem criar conhecimento enquanto as restrições guiam ou restringem o comportamento do negócio.

Segundo [57], as diversas classificações de regras de negócio apresentam semelhanças que permitem definir, de forma resumida, apenas três tipos de regras de negócio: **Restrições de Integridade** (também denominadas regras restritivas ou regras de integridade), **Regras de Derivação** e **Regras de Reação** (também denominadas regras estímulo-resposta, regras de ação, regras evento-ação ou regras de automação):

- Restrição de integridade é uma assertiva que deve ser satisfeita em todos os estados ou transições entre estados de um sistema de informação [57], garantindo que ele seja consistente e seja manipulado corretamente. Ela pode ser dividida em restrição de estado (estática) ou restrição de processo (dinâmica).

- Regra de derivação é uma declaração de como obter um conhecimento a partir de outro existente, o que pode ser feito utilizando uma inferência ou cálculo matemático. Regras de derivação representam conhecimentos que não precisam ser armazenados explicitamente porque podem ser obtidos ou calculados sob demanda.

- Regras de reação estão relacionadas à invocação de ações em resposta a eventos do sistema. Elas representam as condições para as quais ações devem ser tomadas. Exemplos desse tipo de regra incluem *triggers*, pré-condições e pós-condições.

A Tabela 2.1 apresenta um resumo entre as taxonomias apresentadas, relacionando os autores com os tipos de regras. Os tipos de regras foram agrupados por semântica (indicado nas colunas da tabela) e, apesar de terem nomes diferentes, possuem a mesma finalidade.

Apesar de existirem diversas naturezas de RN, a escolha pela utilização de um tipo de RN em detrimento de outro pode variar de acordo com o domínio do negócio e com o aspecto deste domínio que é modelado. A taxonomia de regras utilizada neste trabalho é a proposta em [31], em que as regras são classificadas em Assertivas Estruturais, Regras de Ação e Regras de Derivação. A nomenclatura das Regras de Ação foi alterada para Regras de Validação para melhor adequação a este trabalho, já que esse tipo de regra é utilizado para a validação de dados relacionados ao negócio.

2.3 Abordagens para Tratamento de Regras de Negócio

Como descrito em [35], os métodos tradicionais de análise de sistemas por muito tempo tenderam a negligenciar regras de negócio e preocupavam-se principalmente em modelar a estruturação dos dados utilizados e os processos executados pelos empreendimentos. Os métodos mais modernos de modelagem de sistemas, principalmente os que dão enfoque a orientação a objetos, já passam a dar maior enfoque às regras de negócio.

Tabela 2.1: *Quadro comparativo entre as taxonomias.*

AUTOR/TIPO	Estrutural, Integridade, Restritiva	Ação, Eventoação, Reação, Automação	Derivação, Cálculo	Inferência
Business Rules Group [31]	X	X	X	
Bubenko et al [9]	X	X	X	
Martin & Odell [44]	X		X	
Herbst [37]	X		X	
Halle [34]	X		X	X
Taveter & Wagner [57]	X	X	X	

Alguns pesquisadores consideram que o desenvolvimento de um SI deve tratar RN como uma parte independente logicamente (ou até fisicamente) de outros componentes do sistema [31, 16, 17]. Isso tem como vantagem uma maior facilidade para incorporar mudanças no SI.

Existem diversas abordagens para a modelagem e implementação de RN como um componente independente em Sistemas de Informação. A abordagem de Halle [34] e a abordagem MDE [53] foram consideradas para o desenvolvimento deste trabalho, e são discutidas no restante desta seção.

2.3.1 Abordagem de Halle

Esta abordagem propõe a divisão do processo de desenvolvimento de SI em seis fases: escopo, descoberta, análise, projeto, implementação e teste [34]. Esta divisão é semelhante à abordagem clássica (Modelo Cascata) proposta pela Engenharia de Software para o desenvolvimento de um SI. No entanto, o processo deve ser iterativo, já que os requisitos e regras de negócio podem variar com o tempo, devendo, assim, ser revistos periodicamente.

A Figura 2.1 mostra que as fases de descoberta, análise, projeto e implementação são divididas em quatro partes: tecnologia, funções, regras e dados. A tecnologia é utilizada para apoiar o desenvolvimento do SI, automatizando atividades e, dessa forma, tornando-o mais ágil; as funções referem-se às operações do negócio que devem ser implementadas no SI; as regras estão relacionadas às funções e determinam como elas

devem ser executadas; e os dados são as informações dos usuários que são mantidas no SI.

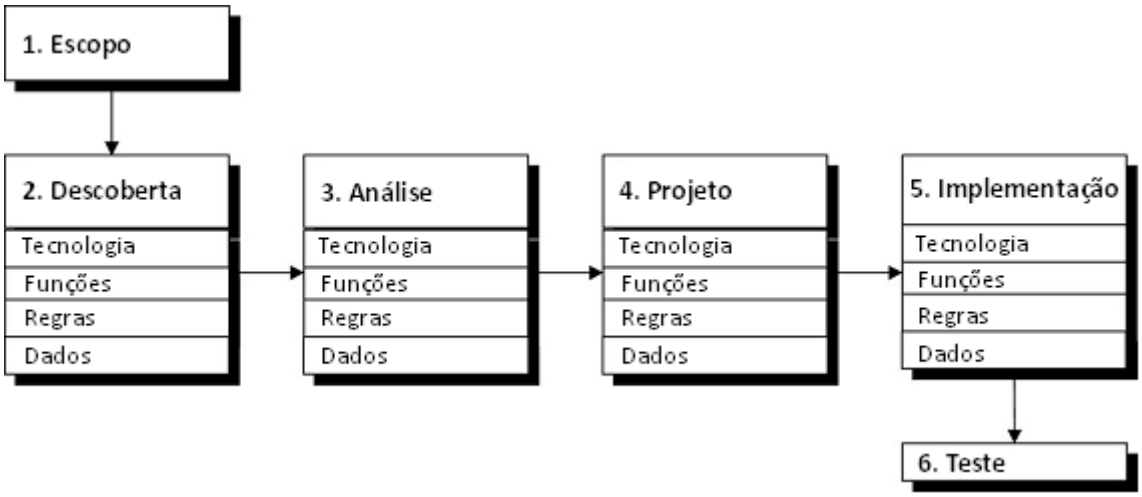

Figura 2.1: *Fases da metodologia de desenvolvimento de sistemas baseados em regras de negócio [34].*

No início do desenvolvimento de um SI, na maioria das vezes não é possível determinar todas as regras de negócio envolvidas, pois elas podem variar ao longo do tempo. Dessa forma, é muito importante determinar o escopo do projeto, isto é, capturar os requisitos de alto nível e os limites abrangidos pelo SI em termos de dados, controles a serem processados, funções de alto nível, principais restrições e interfaces.

Os requisitos de negócio capturados durante a fase de escopo representam os objetivos de alto nível do projeto. Estes requisitos servirão como guias para o desenvolvimento da próxima fase.

A descoberta de requisitos possui dois objetivos principais: a) esclarecer o que o sistema que está sendo projetado deverá fazer e como os interessados no sistema gostariam que o sistema se comportasse; e b) entender as decisões, políticas e regras que estão por trás dos processos do negócio. Nessa fase do desenvolvimento, deve-se descrever os requisitos sem levar em consideração a forma como eles serão implementados em um SI, garantindo que todas as pessoas envolvidas no projeto tenham pleno entendimento dos requisitos do negócio.

A descoberta de regras de negócio consiste no processo de identificar as decisões que devem ser tomadas a partir de eventos do negócio. Nesta fase, os tipos de regras que se pretende capturar e manter ao longo do tempo devem ser definidos e, para isso, é necessário definir uma forma para representar as regras. Neste ponto, a criação de um repositório para as regras identificadas é importante para manter o rastreamento em relação aos requisitos.

A descoberta de regras de negócio deve ser dinâmica e iterativa, pois as regras existentes podem mudar ao longo do tempo, novas regras podem ser criadas e outras

eliminadas. Assim, a descoberta de RN deve ser feita constantemente a partir da interação com os envolvidos no negócio.

Após a descoberta das regras de negócio, acontece a fase de análise. Segundo [34], esta fase corresponde a um conjunto de passos, técnicas e diretrizes para se transformar as regras descobertas na fase anterior para um modelo lógico de regras de negócio. Este modelo compreende um conjunto de regras expressas utilizando termos e fatos que representam a integridade semântica do negócio. É importante destacar que o modelo lógico de regras deve ser totalmente consistente com o modelo do negócio, pois é sobre este modelo que as regras devem ser definidas.

O objetivo da fase de análise é entender cada regra individualmente, e também o relacionamento entre elas. Como resultado desta fase espera-se que as regras apresentem duas características: i)qualidade semântica, isto é, representam de forma fiel a realidade do negócio; e ii) são expressas de forma independente de tecnologia de implementação, de forma que elas possam ser implementadas mais tarde em qualquer tecnologia escolhida. A decisão sobre como serão implementadas as regras de negócio deve ser adiada para a fase seguinte.

Na fase de projeto, o modelo lógico de regras de negócio deve ser transformado em um modelo refinado, com detalhes que serão necessários para a implementação em determinada tecnologia na etapa subsequente do desenvolvimento, mas independente de qualquer produto tecnológico específico. Por exemplo, na fase de projeto o modelo conceitual de banco de dados pode ser convertido para o modelo relacional (independente de tecnologia), que depois (na fase de implementação) será convertido para o esquema físico específico de um determinado SGBD (como o PostgreSQL ou o Oracle).

Ao projetar as regras de negócio, algumas características são essenciais [34]: i)independência das RN em relação à lógica específica da aplicação; ii) independência do sistema gerenciador de bancos de dados; iii) facilidade para realizar modificações nas regras existentes; e iv) eficiência das regras projetadas.

Definidas e projetadas as regras, a próxima etapa deve implementá-las em uma plataforma tecnológica específica. Existem duas formas que são mais freqüentemente utilizadas para implementar RN em SI: no SGBD ou embutidas na aplicação.

2.3.2 Abordagem MDE

MDE (*Model Driven Engineering*) ou Engenharia Orientada por Modelos é uma iniciativa focada na utilização de modelos para o desenvolvimento de SI. No contexto de Engenharia de Software, um modelo é uma descrição de um sistema (ou parte dele) utilizando uma linguagem bem-definida, isto é, uma linguagem com forma (sintaxe) e significado (semântica) precisos [53].

Existe uma outra abordagem de desenvolvimento de software, denominada Arquitetura Dirigida por Modelos (MDA - Model Driven Architecture), que é uma especialização da abordagem MDE [45, 40]. Apesar de darem grande importância aos modelos, elas são sutilmente diferentes, pois MDA está mais focada em gerar artefatos de implementação a partir de modelos abstratos. Neste trabalho as duas abordagens são levadas em consideração para o desenvolvimento do mecanismo proposto.

O desenvolvimento de software baseado em modelos propõe níveis de abstração variados para a modelagem de sistemas, como por exemplo, o Modelo Independente de Computação (*Computational Independent Model* - CIM), Modelo Independente de Plataforma (*Platform Independent Model* - PIM) e Modelo Específico de Plataforma (*Platform Specific Model* - PSM). Ao longo do desenvolvimento do SI, os modelos são mapeados de uma abstração para outra através de processos de transformações sucessivas, durante as quais são incluídos novos elementos nos modelos, reduzindo sua abstração até o nível de dependência da plataforma computacional em que o sistema será implementado [40].

Através dessa abordagem de desenvolvimento é possível que, para um mesmo modelo independente de plataforma, possam ser gerados modelos específicos para plataformas diferentes, ampliando-se a flexibilidade do sistema e as possibilidades de implantação [40].

A Figura 2.2 mostra como os modelos CIM, PIM e PSM estão relacionados. Os requisitos do sistema são modelados no CIM e a partir dele são realizados refinamentos dos requisitos, produzindo o modelo PIM. A transformação do CIM para o PIM pode ser feita manualmente sem nenhuma transformação automática, mas há propostas de transformação automática entre esses modelos [23]. A transformação do PIM para o PSM de uma plataforma específica também pode ser feita de forma automática, por ferramentas de MDE [56, 15, 32], como mostra a Figura 2.3. O código para a plataforma específica é então a partir das transformações realizadas no PSM.

O modelo de negócio não necessariamente modela sistemas de software e, por isso, é também chamado de Modelo Independente de Computação ou **CIM** (*Computation Independent Model*). Para cada modelo de negócio pode haver diversos modelos de software.

Algumas partes do CIM podem ser apoiadas por sistemas de software, mas ele não depende desses sistemas para representar o negócio. O CIM não somente ajuda a entender o domínio do negócio, mas também pode ser utilizado como fonte para definir um vocabulário padrão e para ajudar na comunicação entre a equipe de desenvolvimento e os usuários do SI [42].

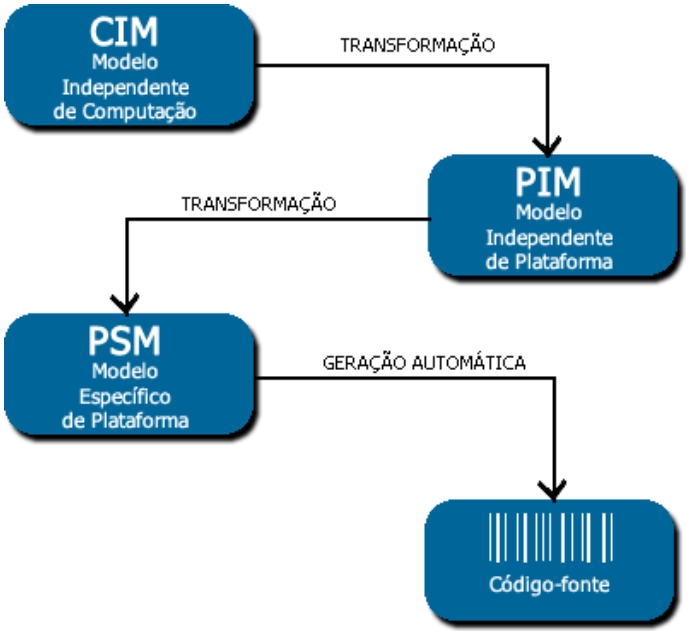

Figura 2.2: *Relação entre os tipos de modelos CIM, PIM e PSM.*

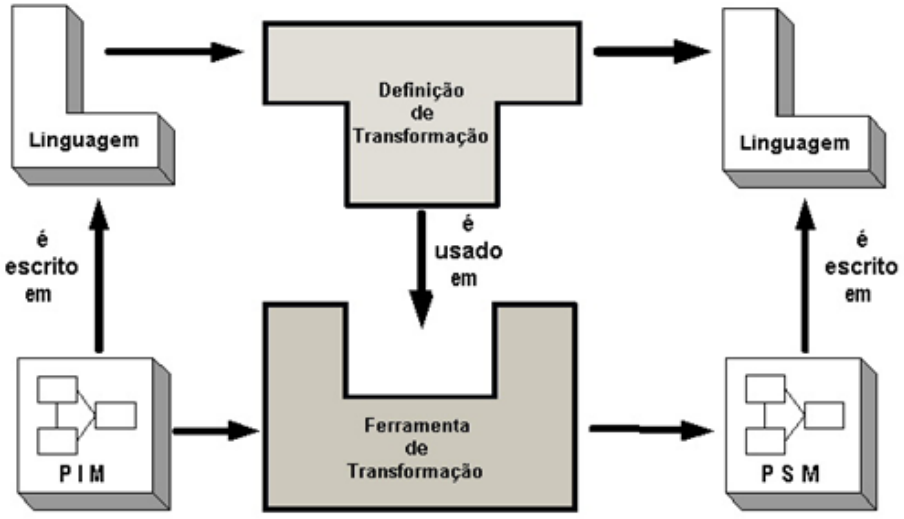

Figura 2.3: *Esquema da transformação do modelo PIM para PSM (adaptada de [40]).*

De acordo com [25], um negócio é formado pelos seguintes componentes: objetivos, recursos, processos e regras. Embora todos eles sejam essenciais para o entendimento do negócio, no contexto do desenvolvimento de um SI os mais importantes são os processos e as regras de negócio, visto que é a partir deles que serão definidos os seus requisitos.

[23, 25] sugerem a utilização de diagramas de atividades da linguagem UML (*Unified Modeling Language* para a representação dos processos de um negócio. Para descrever as regras de negócio, [19] também sugere a utilização da linguagem UML. Em contrapartida, [23] sugere a utilização de linguagem natural, mas de uma forma estruturada. Esse tipo de representação tem a vantagem de ser consistente e não ambígua,

mantendo a aparência de linguagem natural. Ela é baseada em modelos de sentença, que correspondem a uma sequência padronizada de termos usados para descrever regras de negócio.

Um **PIM** descreve o sistema, mas não mostra detalhes sobre uma plataforma específica. Esse modelo deve modelar a relação entre as propriedades de uma entidade e as interações entre entidades de um SI.

Segundo [40], a UML pode ser utilizada para a criação de modelos PIM, pois ela permite criar descrições de sistemas com todos os componentes estruturais e também permite a geração do modelo PSM. O ponto fraco da UML, no entanto, está na modelagem comportamental ou dinâmica dos sistemas, pois os diagramas de representação dinâmica não são formais e completos o suficiente para permitir a geração de qualquer PSM.

Como uma alternativa para a representação de modelos mais completos, a OMG criou uma linguagem denominada OCL (*Object Constraint Language* [48]). A representação de modelos PIM utilizando UML juntamente com OCL faz com que os modelos se tornem mais completos e mais consistentes. Consequentemente, a transformação de PIM para PSM torna-se mais viável de ser realizada. A OCL permite definir restrições, operações de consulta e também aspectos dinâmicos do sistema.

Um **PSM** combina as especificações do PIM com detalhes que descrevem como o sistema usa um tipo particular de plataforma. A diferença entre os modelos PIM e PSM está no nível de abstração em que os dados são representados. No entanto, não há uma fronteira clara entre os dois tipos de modelos. Pode-se dizer apenas que um modelo é mais ou menos voltado para uma plataforma específica do que o outro modelo. Por exemplo, um modelo UML pode ser considerado como um modelo PIM ou PSM dependendo de como ele foi representado.

A principal diferença dessa abordagem em relação à tradicional (conversão manual entre modelos) é que nesta a transformação entre os modelos é feita de forma automática por uma ferramenta de transformação [59]. Além disso, os desenvolvedores do modelo PIM têm menos trabalho de modelagem, visto que os detalhes referentes à tecnologia de implementação não precisam ser analisados e projetados. Esses detalhes já são levados em consideração no momento da transformação do modelo PIM para o PSM. Mais ainda, as ferramentas de transformação e geração de código permitem a redução da quantidade de código que deve ser escrita manualmente por um programador [40].

Com relação à portabilidade, a utilização de MDE permite que um mesmo modelo PIM possa ser transformado em modelos PSM voltados para plataformas variadas. Para isso, basta utilizar outra definição de transformação correspondente à linguagem alvo [40].

No desenvolvimento de SI, tanto a documentação quanto a manutenção são atividades muito dispendiosas e que geralmente causam problemas. A maior parte dos

desenvolvedores não cria documentação dos sistemas de forma apropriada, pois considera que essa atividade não traz nenhum benefício. No entanto, a documentação é essencial para que futuras manutenções no sistema sejam possíveis de serem realizadas.

A utilização do modelo PIM permite que o desenvolvedor trabalhe em um nível mais alto de abstração. Esse modelo deverá ser transformado em um modelo PSM com a utilização de uma ferramenta de transformação. Dessa forma, sempre que alguma alteração seja necessária no sistema, basta modificar o modelo PIM e executar novamente a transformação para o modelo PSM. Isso faz com que a documentação do sistema esteja sempre atualizada, pois qualquer modificação no modelo PIM será refletida no modelo PSM que foi gerado a partir dele, e também no código executável gerado. Além disso, nota-se que o processo de manutenção é simplificado, visto que se resume em reestruturar o modelo PIM e aplicar a transformação de modelos.

2.3.3 Comparativo entre as Abordagens

As abordagens de Halle e MDE para tratamento de RN apresentam focos diferentes. Enquanto a primeira está focada no processo de desenvolvimento, a outra foca em modelos e transformações.

Halle [34] propõe a divisão do processo de desenvolvimento em uma sequência de atividades, executadas por pessoas (analistas, projetistas, desenvolvedores, etc.) até que seja obtido o produto final do desenvolvimento, que é o código fonte.

Por outro lado, a abordagem MDE sugere a criação de diversos modelos, com níveis de abstração variados, desde a fase de descoberta de regras. Esses modelos são sucessivamente transformados (de forma mais ou menos automatizada) em modelos de menor nível de abstração até se obter o código-fonte a ser implementado.

Apesar dessas diferenças, essas abordagens podem ser equiparadas se os modelos abstratos da abordagem MDE forem considerados como os produtos de cada atividade do processo de desenvolvimento proposto por Halle. A Figura 2.4 ilustra essa idéia.

Teoricamente, a abordagem MDE é mais vantajosa que a de Halle, visto que a transformação automática entre modelos permite economizar esforço, tempo e custo de desenvolvimento. No entanto, a adoção dessa abordagem para o desenvolvimento de um SI completo tem se mostrado inviável, visto que a construção de uma ferramenta de transformação entre modelos CIM e PIM é muito complexa. Isso ocorre porque não há linguagens ou tecnologias apropriadas para a representação de modelos CIM tal que seja possível convertê-los em modelos de mais baixo nível sem haver perda na semântica do modelo original.

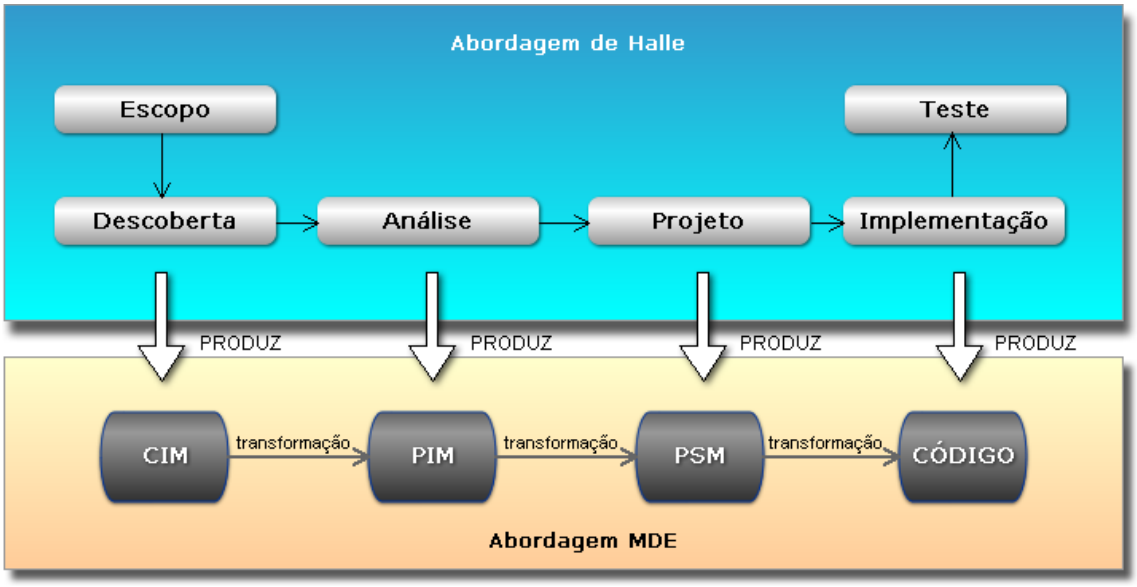

Figura 2.4: *Combinação da abordagem de Halle com a MDE.*

Já a geração de código-fonte a partir de modelo PSM é relativamente mais fácil de ser realizada, pois o PSM apresenta mais detalhes e é representado em nível de abstração mais próximo do código-fonte.

Neste trabalho, a abordagem utilizada para o desenvolvimento de regras de negócio é baseada nas duas abordagens. As fases de descoberta, análise e projeto são feitas de acordo com a abordagem de Halle, sendo que a cada fase as regras são refinadas pela equipe de desenvolvimento. A abordagem MDE é utilizada para a geração automática de código-fonte a partir do modelo PIM de regras (OCL).

2.4 Linguagem OCL

OCL (*Object Constraint Language*) é uma linguagem criada pela OMG [49] para escrever expressões sobre modelos criados usando a UML. No entanto, OCL pode ser utilizada para escrever expressões sobre qualquer modelo conceitual. Por exemplo, outro tipo de modelo que pode ser complementado com OCL é o MER (Modelo Entidade-Relacionamento).

Apesar do alto poder de expressão das linguagens gráficas de modelagem, há propriedades e restrições de negócio que são muito complexas, ou mesmo impossíveis, de serem expressas adequadamente em um diagrama. Mesmo os mecanismos de extensão da UML, como estereótipos (*stereotypes*), valores etiquetados (*tagged values*) e restrições (*constraints*) pré-definidas, podem ser insuficientes.

Um projeto de software envolve dois aspectos do mundo real: estático e dinâmico. O aspecto estático é bem representado pelas linguagens gráficas de modelagem existentes, como a UML (diagrama de classes) e o MER (diagrama entidade-

relacionamento). O aspecto dinâmico, no entanto, na maioria das vezes não é modelado de forma adequada quando se utiliza essas linguagens, já que possuem expressividade limitada. A OCL surgiu para suprir essa necessidade de expressar aspectos dinâmicos do mundo real em modelos abstratos, tornando-os mais completos [14].

2.4.1 Características de OCL

Modelos conceituais passam a ter mais formalidade quando expressões OCL os complementam. Baseada em fundamentos matemáticos para se obter a formalidade, OCL é uma linguagem declarativa fortemente tipada, permitindo que as expressões escritas possam ser verificadas quanto à existência de erros. Além disso, sua formalidade faz com que as expressões sejam não ambíguas, aumentando, assim, a qualidade do modelo.

As expressões OCL, quando avaliadas por um sistema, não devem gerar efeitos colaterais, isto é, não podem alterar qualquer informação do modelo. Por isso ela é uma linguagem *side effect free*. Além disso, utilizando expressões OCL, é possível a navegação entre as classes de um modelo, a partir de uma entidade-base, que fornece o contexto da expressão [60].

Apesar de seu propósito inicial ser a definição de restrições em modelos, OCL evoluiu e a versão atual (2.0) pode ser utilizada em vários contextos com os seguintes propósitos [48]:

1. **Utilização como uma linguagem de consulta**
   OCL permite que se construa expressões de consulta ao modelo subjacente, utilizando navegação pelos relacionamentos entre as classes e as várias operações em coleções permitidas pela linguagem.

2. **Especificação de invariantes**
   Tipicamente, expressões em OCL especificam condições invariantes: uma ou mais expressões booleanas precisam ser verdadeiras para todas as instâncias de uma determinada classe de um modelo do sistema.

3. **Descrição de condições de guarda**
   Uma condição de guarda informa a condição que deve ser satisfeita para que uma mudança de estado aconteça. Também pode ser utilizada para descrever uma condição em um fluxo de atividades ou diagrama de estados.

4. **Descrição de pré e pós-condições**
   Uma pré-condição precisa ser verdadeira para permitir a execução de uma operação; já uma pós-condição representa as condições que devem ser verdadeiras

após a execução da operação. Pré e pós-condições são restrições que, através de expressões booleanas, especificam a aplicabilidade e o efeito de uma operação sem declarar o algoritmo que implementa a respectiva operação.

5. **Especificação de regras de derivação e valores iniciais para atributos**
   O conteúdo de um atributo derivado deve **sempre** conter um valor que expressa a regra de derivação. Já um valor inicial de um atributo precisa ser garantido somente no momento de criação da instância do objeto em questão; após esse momento, o valor pode assumir qualquer conteúdo.

6. **Especificação complementar para entendimento de diagramas da UML**
   O diagrama de classes pode ser beneficiado com expressões OCL, através de valores iniciais, invariantes, e pré e pós-condições sobre métodos, por exemplo. Diagramas de atividades podem especificar condições e valores de parâmetros; diagramas de componentes ou de implantação podem especificar explicitamente suas interfaces; pré e pós-condições dos casos de uso podem ser escritos usando expressões OCL.

2.4.2 Tipos de Dados em OCL

Os tipos de dados de OCL estão divididos basicamente em dois grupos: tipos predefinidos (tais como tipos básicos e tipos coleções) e tipos definidos pelo usuário. Os tipos básicos são mostrados na Tabela 2.2.

Tabela 2.2: *Tipos básicos da OCL [48].*

TIPO	VALORES PERMITIDOS	EXEMPLOS DE OPERAÇÕES
Boolean	true, false	and, or, xor, not, implies, if-then-else-endif
Integer	.. -2, -1, 0, 1, 2, ..	*, +, -, /, abs()
Real	1.5, 3.66, ...	*, +, -, /, floor()
String	'um texto qualquer'	concat(), size(), substring()

Existem quatro tipos diferentes de coleção, como mostra a Tabela 2.3. Quando se realiza uma navegação pelas associações entre entidades do modelo, se obtém uma coleção; o tipo de coleção obtido irá variar de acordo com a navegação efetuada.

Além dos tipos básicos e das coleções, OCL permite que se utilize todos os tipos presentes no modelo de objetos, como entidades, atributos, operações definidas pelo usuário, associações, agregação e enumeração.

OCL permite navegar através dos relacionamentos entre entidades do modelo, utilizando uma notação de pontos. O contexto da expressão define o ponto de partida da navegação. Por exemplo, em um contexto referente à classe *Pessoa*, pode-se definir uma expressão como *self.empresaTrabalha.nomeEmpresa*, considerando que a classe *Pessoa*

Tabela 2.3: *Tipos de coleção da OCL [48].*

TIPO DA COLEÇÃO	CARACTERÍSTICAS
Set	Os elementos da coleção são únicos; não há uma sequência na recuperação dos elementos.
Bag	Podem haver elementos repetidos na coleção; não há uma sequência na recuperação dos elementos.
Sequence	Pode haver repetição de elementos; os elementos estão em sequência.
OrderedSet	Os elementos da coleção são únicos; os elementos estão em sequência.
Collection	É o supertipo abstrato dos outros quatro tipos apresentados nesta tabela; é usado para definir as operações comuns a todos os tipos de coleções.

esteja associada a uma classe *Empresa*, onde *empresaTrabalha* é o nome do papel de *Empresa* nessa associação.

2.4.3 Tipos de Expressões

Dentre os tipos de restrições ou expressões permitidos OCL, as que são consideradas neste trabalho são: invariantes, pré e pós-condições, valores iniciais, regras de derivação e expressões de consulta.

Invariantes

Uma restrição do tipo invariante é uma restrição que se aplica durante toda a vida da entidade a que se refere. Em qualquer momento da existência dessa entidade, as invariantes precisam ser satisfeitas, para todas as instâncias da entidade. Cada expressão utilizada deve retornar um valor booleano, que indica se ela foi satisfeita ou não [60].

Uma invariante obrigatoriamente refere-se a uma entidade do modelo, a qual é chamada de contexto (*context*).

Pré e pós-condições

Toda operação definida para uma entidade do modelo deve executar alguma ação que, geralmente, exige que algumas condições sejam satisfeitas antes do seu início (pré-condições) ou depois de ter sido finalizada (pós-condições). Essas condições devem ser antendidas para garantir a execução correta da operação [60].

Nas pré e pós-condições de uma operação, o contexto será identificado pelo nome da classe ou interface que contém a operação, mais o nome desta. As palavras-chave **pre** e **post** identificam pré e pós-condições, respectivamente. Novamente, todas as expressões

que são utilizadas em pré e pós-condições devem ser expressões que retornem o tipo booleano.

Valores iniciais e regras de derivação

Para definir valores iniciais e regras de derivação de atributos em OCL, utiliza-se, respectivamente, as palavras-chave *init* e *derive*. Nesses casos, a expressão resultante deve possuir tipo compatível com o tipo do atributo de referência [60].

Para esses tipos de expressões, o contexto da expressão será o nome da classe seguido pelo nome e tipo do atributo. Por exemplo:

– Temos uma classe *EstoqueProduto* que possui o atributo *qtdEstocada* com valor inicial 0 (zero)

context EstoqueProduto::qtdEstocada:*Integer*
init: 0

– A classe *EstoqueProduto* pode estar relacionada a várias instâncias da classe *Produto*. O valor do atributo *qtdEstocada* é derivado em função do atributo *quantidade* de cada *Produto*

context EstoqueProduto::qtdEstocada:*Integer*
derive: self.Produto.quantidade->sum()

Expressões de consulta

OCL permite definir expressões de consulta sobre os elementos do modelo, utilizando-se do seu poder de navegação. Nessas expressões, o tipo de retorno deve ser aquele especificado na assinatura da operação [60].

As expressões de consulta são identificadas pela palavra-chave *body* e o contexto é especificado pelo nome da classe que contém a operação, seguido pelo nome da operação. A palavra-chave *result* pode ser utilizada para especificar o retorno da expressão. O Código 2.1 apresenta uma expressão de consulta em OCL referente ao modelo ilustrado na Figura 2.5. A expressão retorna a quantidade de unidades do produto existe no estoque informado.

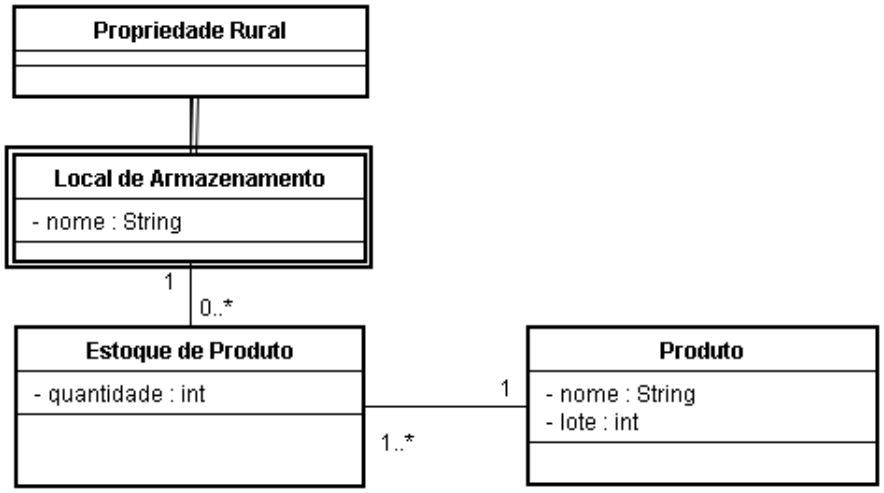

Figura 2.5: *Modelo conceitual referente ao estoque de produtos em uma propriedade rural.*

Código 2.1 Exemplo de expressão de consulta em OCL.

```
1  context EstoqueProduto::obterQtdDisponivel(
2    produto_nome : String;
3    produto_lote : String;
4    locArmaz_codigoPropRur : String;
5    locArmaz_nome : String
6  ) : String
7  body:
8  let
9    qtdAnteriorProduto : Integer = EstoqueProduto.allInstances()
10     ->select(
11       produto.nome = produto_nome and
12       produto.lote = produto_lote and
13       propRur.codigoPropRur = locArmaz_codigoPropRur and
14       locArmaz.nome = locArmaz_nome).quantidade
15
16 in
17   result = qtdAnteriorProduto
```

2.4.4 Vantagens da Utilização de OCL

A utilização de UML em conjunto com OCL permite a especificação, em modelos de alto nível, tanto de restrições estruturais quanto comportamentais, desde as mais simples até regras complexas do negócio. Desta forma, é possível representar assertivas de ação e regras de derivação sobre os elementos do modelo.

De acordo com [21], a utilização de UML em conjunto com OCL para definir RN tem várias vantagens:

- UML é uma linguagem padrão para a definição de modelos orientados a objetos;
- OCL é uma linguagem precisa e abstrata para especificar restrições de integridade. O esquema de navegação pelo modelo é semelhante aos conceitos de bancos de dados;
- O uso de OCL na definição de regras de negócio permite alcançar independência em relação à tecnologia de implementação;
- A geração automática de código SQL a partir do código OCL reduz o tempo de desenvolvimento e facilita a manutenção do SI;
- A especificação de operações com pré e pós-condições é um bom ponto de início para a geração automática de código SQL para consultas em banco de dados, atualizações, *triggers* ou *stored procedures* correspondentes àquelas operações.

Apesar dessas vantagens, a utilização de OCL dificulta o entendimento das regras por parte dos especialistas de negócio. Pensando nisso, propõe-se neste trabalho representação de RN em linguagem natural estruturada, que pode ser entendida por especialistas de negócio na fase de Engenharia de Requisitos e Regras de Negócio, e a definição do papel do projetista de regras, que tem a função de converter essas regras escritas em linguagem natural para OCL.

2.5 Repositório de Regras

No processo de desenvolvimento de software, é comum encontrar regras de negócio especificadas de diversas maneiras e espalhadas em documentos variados. Isto dificulta o controle sobre a criação ou modificação das regras e também a rastreabilidade entre as regras e os conceitos do negócio que as originaram.

Para lidar com esses e outros problemas relacionados ao tratamento de regras de negócio, é importante a criação de um repositório centralizado e consistente de regras de negócio. Um repositório de regras é qualquer sistema utilizado para documentar, atualizar e manter as necessidades e regras do negócio relacionadas a um projeto. O IEEE define repositório de regras de negócio como “a organização básica de um sistema, incorporada em seus componentes, relacionamentos e ambiente, e os princípios que direcionam seu projeto e evolução.” [43].

O repositório de regras deve apoiar todas as fases do ciclo de vida das regras. A cada fase, elas são representadas em um nível maior de detalhamento, e essa evolução deve ser registrada no repositório. A utilização de um repositório central permite que tanto os proprietários do negócio quanto os técnicos desenvolvedores possam acessá-las a qualquer momento e fazer modificações, caso sejam necessárias, no nível apropriado de abstração, e de acordo com mecanismos de controle de acesso.

Um repositório pode ser projetado de várias formas, mas todas tem o mesmo objetivo final de apoiar o tratamento de regras de negócio.

2.5.1 Requisitos para um Repositório de Regras

O propósito da construção de um repositório de regras é reunir todas as informações relevantes para o negócio e para as pessoas nele envolvidas. Um repositório de regras deve atender os seguintes requisitos de qualidade específicos [46, 10, 12]:

- Permitir armazenar as regras em diferentes níveis de detalhamento, desde a fase de descoberta, onde as regras podem ser definidas em alto nível utilizando linguagem natural, até a fase de projeto, onde elas podem ser expressas utilizando linguagens formais;

- Permitir consultar as definições de todas as regras de negócio, com a possibilidade de gerar relatórios para realizar análise de impacto de modificações;

- Garantir rastreabilidade entre as regras e os artefatos de software onde elas são referenciadas;

- Garantir segurança e integridade das regras por meio de mecanismos de controle de acesso e de proteção de informações;

- Assegurar que uma regra: não é duplicada; não está sendo tratada no contexto de outra regra; e não é conflitante com outra regra;

- Permitir criar novas versões das regras, como parte de um processo de mudança das regras;

- Permitir o acompanhamento de mudanças propostas ao longo do processo de manutenção;

- Guardar o estágio de desenvolvimento em que a regra se encontra (em modelagem, em implementação ou desativada, por exemplo).

2.5.2 Formas de Armazenamento

Um repositório de regras pode ser implementado de duas maneiras distintas [10]:

1. No banco de dados: as regras são armazenadas no banco de dados e são caracterizadas por seus atributos. O repositório pode ser projetado como um banco de dados independente ou como parte do próprio banco de dados da aplicação principal.

2. No código de aplicação: essa abordagem é adotada nas metodologias tradicionais onde as regras são implementadas diretamente no código da aplicação. A diferença é que o código que representa as regras é armazenado independentemente das outras camadas do SI e elas são especificadas uma única vez no sistema.

A primeira abordagem, com o repositório implementado no banco de dados, tem a vantagem de que a verificação das regras pode ser feita no próprio banco de dados, utilizando-se restrições de integridade ou até mesmo mecanismos de gatilho (*triggers*), *stored procedures* ou *views*. Além disso, no contexto de aplicações distribuídas que compartilham um mesmo banco de dados, a centralização do repositório facilita o acesso e a manutenção das regras de negócio. No entanto, esta abordagem pode causar a sobrecarga do banco de dados, tornando-o um gargalo no SI, devido ao grande número de acessos que devem ser realizados para avaliar as regras.

A segunda abordagem, com o repositório implementado no código de aplicação, tem a vantagem de minimizar acessos ao banco de dados para se avaliar as regras de negócio. Entretanto, como as regras estão inseridas no código de aplicação, qualquer modificação em uma regra exige que o código-fonte seja recompilado, dificultando a manutenção do SI.

Neste capítulo foram apresentados os principais conceitos e taxonomias de regras de negócio, bem como as características da linguagem OCL e os requisitos para um repositório de RN.

O Capítulo 3 apresenta a fase de Engenharia de Requisitos e como as regras de negócio são analisadas. Depois dessa fase, as regras de negócio são projetadas utilizando-se a linguagem OCL, como é explicado no Capítulo 4. O Capítulo 5 mostra como as regras em OCL são convertidas para a linguagem de implementação.

CAPÍTULO 3

Engenharia de Requisitos e Regras de Negócio

Na Engenharia de Software existem diversas definições para o termo **requisitos**. Sommerville e Sawyer [55] definem requisitos como ".. *uma especificação do que deveria ser implementado. Eles são descrições de como o sistema deveria se comportar, ou a descrição de uma propriedade ou atributo. Eles podem ser uma restrição no processo de desenvolvimento do sistema*". De acordo com essa definição, os requisitos definem o que será produzido quando o projeto estiver finalizado. Quando menciona os atributos de um sistema, a definição se refere a como o sistema atende seus objetivos, mas não ao que deve ser feito (funcionalidades).

Segundo a norma IEEE 830 [47], "*Um requisito especifica uma função externamente visível ou atributo de um sistema*". Goldsmith [29] define requisitos como "*O quê deve ser entregue para agregar valor*". A partir desta definição deve-se entender que os requisitos só agregam valor ao negócio se eles atendem aos objetivos do próprio negócio.

Os requisitos podem ser classificados em **requisitos de usuário** e **requisitos de sistema (ou de domínio)**, dependendo da forma e nível de abstração em que são definidos [54]:

- Requisitos de usuário são declarações de quais serviços deverão ser oferecidos pelo sistema e quais as restrições de seu uso. Eles devem especificar apenas o comportamento externo do sistema, não se envolvendo em questões de projeto;
- Requisitos de sistema especificam detalhes das funções do sistema, serviços e restrições operacionais. Esses requisitos definem o que deve ser implementado.

Outra taxonomia típica divide os requisitos em duas categorias: requisitos funcionais e não-funcionais.

Os **requisitos funcionais** são declarações de serviços que o sistema deveria fornecer, como o sistema trata determinadas entradas e como o sistema deveria se comportar em determinadas situações. Eles referem-se a "o que" o sistema deveria fazer ou deveria deixar o usuário fazer [61].

Os **requisitos não-funcionais** expressam restrições sobre os serviços ou funções oferecidas pelo sistema, como restrições de tempo, restrições sobre o processo de

desenvolvimento e uso de determinadas normas. Eles não estão diretamente relacionados a funções que o sistema deve desempenhar para atender os processos de domínio do negócio.

No contexto de requisitos funcionais aparece uma outra categoria de requisitos, que são as **regras de negócio (RN)**. Gottesdiener afirma em [30] que "*Regras de negócio estão no núcleo dos requisitos funcionais*". Elas são o que um requisito funcional "sabe", como: decisões, diretrizes e controles que estão por trás da funcionalidade. As regras de negócio, assim como os demais requisitos funcionais, devem ser capturadas durante a fase de Engenharia de Requisitos, permitindo sua validação e verificação junto aos proprietários do negócio.

3.1 Processo de Engenharia de Requisitos

O processo de Engenharia de Requisitos envolve atividades de elicitação, análise, documentação, validação e gerenciamento de requisitos, segundo Wiegers [61].

Wiegers propõe a separação do processo de Engenharia de Requisitos em duas atividades genéricas concorrentes: desenvolvimento de requisitos e gerenciamento de requisitos. O objetivo principal da primeira atividade é identificar e listar todos os requisitos funcionais e características do produto a ser desenvolvido, enquanto que o objetivo da segunda atividade é, principalmente, o gerenciamento de mudanças ao longo do tempo de requisitos identificados.

A atividade de desenvolvimento de requisitos pode ser dividida em 4 subatividades: elicitação, análise, documentação e validação de requisitos. A Figura 3.1 ilustra o esquema do processo de Engenharia de Requisitos.

Na atividade de Elicitação de requisitos, os analistas de negócio trabalham juntamente com o cliente, a fim de descobrir informações sobre o domínio da aplicação, quais serviços o sistema deve fornecer, o desempenho exigido do sistema, as restrições de hardware, entre outros.

A atividade de Análise de requisitos tem a finalidade de obter detalhes a respeito dos requisitos de alto nível obtidos na atividade de Elicitação e realizar a modelagem deles. A Análise também envolve a negociação de prioridades relativas aos requisitos, busca por requisitos ainda não descobertos, avaliação de viabilidade técnica e riscos. Ao executar esta atividade, caso os requisitos ainda não estejam bem claros para os analistas, é possível retornar para a atividade de elicitação para melhorar o entendimento das necessidades do cliente.

Os requisitos especificados durante a atividade de análise devem ser documentados na atividade de Documentação. Esses requisitos devem ser descritos utilizando linguagem natural e diagramas, para que sejam compreendidos por todos os *stakeholders*.

O fato de se ter uma boa documentação dos requisitos melhora a manutenibilidade de um sistema, pois através dos detalhes apresentados é mais fácil entender cada requisito e modificá-lo, caso necessário. Nesse sentido, a criação de um repositório consistente é importante, pois permite armazenar todos os requisitos, incluindo as regras de negócio, conforme discute a Seção 2.5.

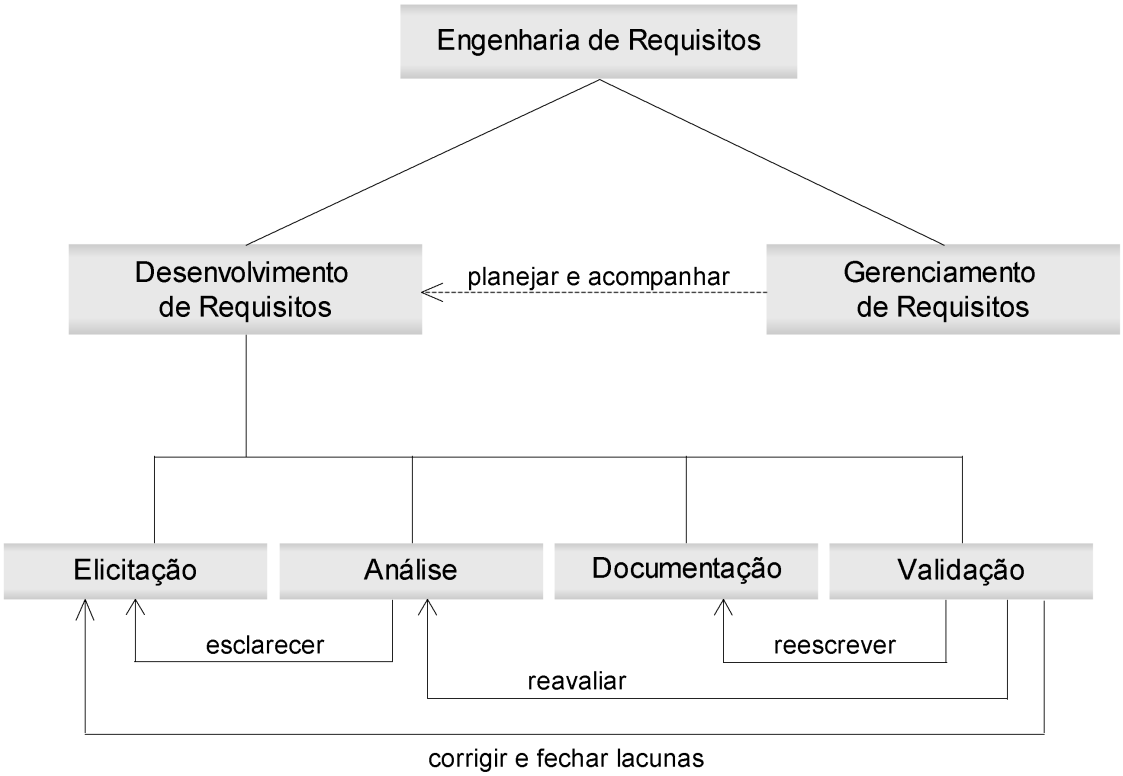

Figura 3.1: *Atividades do processo de Engenharia de Requisitos (adaptada de [61]).*

A Validação de requisitos tem o objetivo de mostrar se os requisitos elicitados e analisados foram bem entendidos pelos analista e se condizem com a real necessidade do cliente. Além disso, deve ser avaliado se esses requisitos apresentam algum problema que comprometa a funcionalidade do software. Esta validação é importante porque a ocorrência de erros na especificação de requisitos pode levar a grandes custos relacionados ao retrabalho, quando esses erros são descobertos durante o desenvolvimento ou depois que o sistema estiver em operação. O custo de fazer uma modificação no sistema, resultante de um problema de requisito, é muito maior do que reparar erros de projeto ou de codificação.

O Gerenciamento de Requisitos é uma atividade constante ao longo do ciclo de vida do SI. Requisitos não são estáticos e evoluem ao longo do tempo. Dessa forma, esta atividade engloba o desenvolvimento de requisitos, ou seja, não são atividades isoladas. O gerenciamento de requisitos planeja e acompanha o desenvolvimento deles.

No presente trabalho, são abordadas as atividades de Análise, Documentação e Gerenciamento de requisitos, e mais especificamente dos requisitos que se referem às regras de negócio. As atividades de Elicitação e Validação também são apoiadas pelas propostas deste trabalho, embora não sejam o foco principal. Informações a respeito dessas fases podem ser encontradas em [55, 61].

3.2 Análise de Regras de Negócio

Considerando que a atividade de Elicitação tenha sido concluída e que as regras de negócio tenham sido compreendidas pelos analistas de negócio, a próxima etapa consiste em detalhar e formalizar todas as regras. Esta atividade é importante no desenvolvimento para que haja um entendimento comum dessas regras por todos os interessados no SI, e para que esse conhecimento seja consolidado como um ativo do negócio.

A Seção 3.2.1 apresenta o modelo conceitual utilizado neste trabalho e como as assertivas estruturais são representadas nesse modelo, enquanto a Seção 3.2.2 explica como é feita a análise das regras de validação e derivação.

3.2.1 Modelo Conceitual e Assertivas Estruturais

A modelagem conceitual neste trabalho é feita com base no Modelo de Meta-Objetos (MMO), descrito inicialmente em [18, 20]. O MMO consiste em uma implementação orientada a objetos do Modelo Entidade-Relacionamento (MER) e permite representar os aspectos estruturais do domínio de um SI, como conceitos (ou entidades) de negócio, relacionamentos entre conceitos e restrições de integridade.

A utilização do MMO permite representar os conceitos de um SI de maneira independente da tecnologia que será utilizada para a sua implementação, ou seja, apenas as informações e os conceitos relevantes para o negócio são modelados. A Figura 3.2 mostra o esquema conceitual dos conceitos de modelagem presentes no MMO.

Como pode ser visto na Figura 3.2, o MMO permite representar dois tipos de entidades: Entidade Forte e Entidade Fraca.

Para caracterizar essas entidades, podem ser especificados atributos, os quais podem ser classificados de acordo com o seu tipo em: **Básico**, **Composto**, **Enumerado** e **Objeto Externo**. O atributo Composto pode ser do tipo **Objeto Interno**, que no modelo representa um relacionamento entre entidades. Além do tipo, os atributos de tipo Básico são classificados quanto ao seu domínio em: **Alfanumérico**, **Data**, **Decimal** ou **Inteiro**.

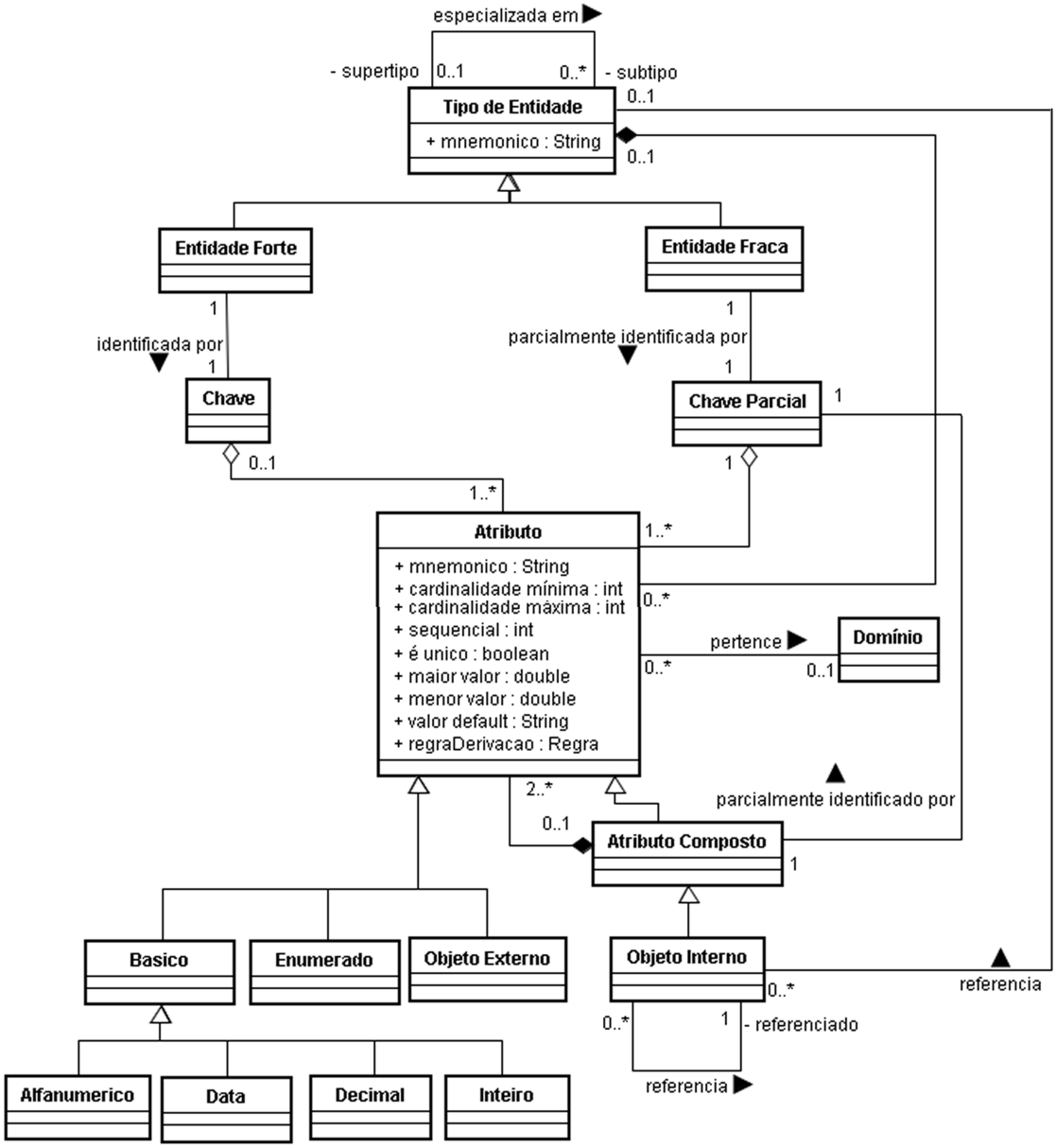

Figura 3.2: *Esquema conceitual do Modelo de Meta Objetos (adaptado de [20]).*

Os relacionamentos entre entidades no MMO são definidos como atributos, isto é, para definir um relacionamento entre duas entidades é necessário criar um atributo do tipo Objeto Interno em uma das entidades, referenciando a outra entidade.

É possível também representar um relacionamento entre uma entidade do negócio e uma instância de relacionamento já existente, o que configura uma Agregação no modelo. Para isso o Objeto Interno de uma entidade deve referenciar o Objeto Interno de outra entidade.

Quanto à especificação de regras, o MMO é limitado, pois permite expressar apenas restrições estruturais. São elas:

- **domínio:** representada através do “Tipo de Domínio” da entidade “Atributo”;

- **chave:** representada através do conceito “Chave” (no caso de Entidade Forte) e “Chave Parcial” (no caso de Entidade Fraca);

- **unicidade:** representada através do atributo ehUnico da entidade “Atributo”;

- **cardinalidade mínima de atributo:** representada através do atributo Cardinalidade Mínima da entidade “Atributo”;

- **cardinalidade máxima de atributo:** representada através do atributo Cardinalidade Máxima da entidade “Atributo”;

Regras dinâmicas, como Assertivas de Ação e Regras de Derivação de atributos, não podem ser representadas neste modelo. Como discute o Capítulo 4, a solução proposta neste trabalho para essa limitação é a utilização da linguagem OCL de forma a complementar o MMO.

3.2.2 Regras de Validação e Derivação

Como as Assertivas Estruturais podem ser representadas diretamente no MMO, apenas as Regras de Validação e Derivação precisam ser especificadas em detalhes. As informações que devem ser obtidas sobre estes tipos de regras são:

1. **Identificação:** cada regra deve ter um nome único, relacionado ao seu contexto no modelo de negócio. Para padronizar essa nomenclatura, o seguinte formato para os nomes de regras foi adotado: <XX_YYY>. Neste formato, “XX” pode assumir o valor “RV”, no caso de uma regra de validação, ou “RD”, no caso de uma regra de derivação. O código “YYY” deve ser substituído pelo contexto proprietário da regra em questão. Por exemplo, no caso de uma regra de validação para a entidade Pessoa, teremos a identificação RV_Pessoa.

2. **Tipo:** corresponde ao tipo da regra, que pode ser VALIDAÇÃO ou DERIVAÇÃO.

3. **Comportamento:** indica a operação do negócio à qual a regra deve estar relacionada. Essa operação pode ser de inserção, alteração, exclusão ou consulta a dados referentes às entidades do negócio. É possível que uma regra esteja relacionada a mais de uma dessas operações ao mesmo tempo, ou seja, qualquer combinação delas, como inserção e alteração ou inserção, alteração e exclusão. Isso significa que uma regra deve ser avaliada sempre que for realizada qualquer das operações

às quais está relacionada.

4. **Aspecto de Tempo:** indica o momento em que uma regra deve ser avaliada no SI. Foram definidos dois momentos para a execução das regras, ANTES (pré-condição) e DEPOIS (pós-condição), relativos ao comportamento do negócio definido para a regra.

5. **Contexto Proprietário:** identifica o conceito do negócio ao qual uma regra está relacionada. Por exemplo, o conceito do negócio "Pessoa" possui uma regra que diz que "Pessoas distintas não podem possuir o mesmo número de CPF". Neste caso, o contexto proprietário da regra é o conceito "Pessoa".

6. **Regra em linguagem natural:** as regras capturadas junto aos proprietários do negócio são escritas em linguagem natural e podem ser armazenadas em idiomas variados (Português ou Inglês, por exemplo). No entanto, a utilização de linguagem natural apresenta pontos negativos, pois as regras podem ser interpretadas de diferentes maneiras (ambigüidade) entre usuários e técnicos. Uma forma de amenizar esse problema é a especificação das regras utilizando-se linguagem natural estruturada. Recomenda-se que seja utilizada uma linguagem natural limitada por padrões, ou seja, é necessário definir algum tipo de formatação, como uma espécie de gabarito (*template*) [46, 52]. Um exemplo simples seria formatar as sentenças que declaram as regras da seguinte forma: [artigo] + [sujeito] + [verbo] + [predicado] + [complemento].

Depois de obtidos todos os detalhes a respeito das regras de negócio, é necessário documentá-los de maneira segura, de forma que estejam disponíveis ao longo de todo o ciclo de vida do SI. Essas informações são utilizadas, por exemplo, na fase projeto, quando os projetistas de regras convertem as regras em linguagem natural para a linguagem OCL (Capítulo 4).

Neste trabalho, todas as regras de negócio são documentadas e armazenadas em um repositório central, descrito na Seção 3.3.

3.3 Documentação de Regras de Negócio

Como visto na Seção 2.5, existem duas grandes abordagens para o armazenamento de regras de negócio. Uma delas propõe o armazenamento das regras em um banco de dados, enquanto a outra propõe o armazenamento no código da aplicação. O mecanismo para armazenamento de regras utilizado neste trabalho utiliza a primeira abor-

dagem, ou seja, o armazenamento das regras no banco de dados, onde o repositório é modelado juntamente com o esquema da aplicação principal.

Esta forma de armazenar as regras tem a vantagem de isolá-las completamente do código fonte da aplicação, facilitando, assim, a manutenção do SI. No entanto, o armazenamento das regras no banco pode impactar o desempenho do sistema, já que constantes acessos ao SGBD devem ser realizados. Ao optar por uma das abordagens, deve-se analisar todos os aspectos relacionados ao sistema (por exemplo, a quantidade de usuários simultâneos) e verificar qual é o mais apropriado.

A Figura 3.3 mostra o esquema conceitual simplificado do repositório de regras de negócio.

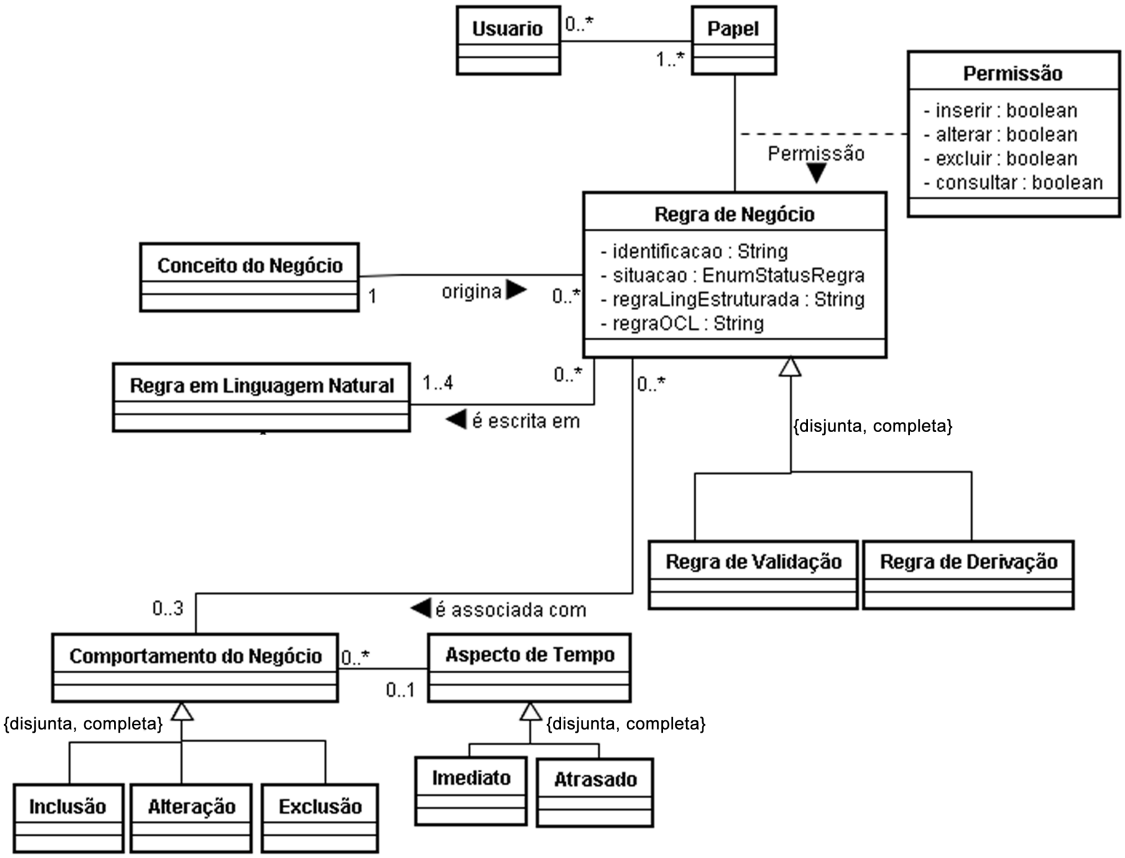

Figura 3.3: *Esquema conceitual do repositório de regras de negócio.*

Como mostra a Figura 3.3, toda regra de negócio está relacionada a algum conceito do negócio. As regras possuem identificação única e padronizada para evitar que duas regras iguais sejam armazenadas no repositório.

Regras de validação representam restrições ou condições que determinam a integridade semântica de um conceito do negócio. Regras de derivação contêm declarações sobre a forma de obter um conhecimento a partir de outros conhecimentos existentes no

negócio. O tipo do dado que representa um conhecimento (conteúdo) derivado é definido na própria declaração da regra.

O repositório implementado neste projeto permite armazenar as regras especificadas de duas diferentes maneiras ao longo do ciclo de vida: em linguagem natural (fase de engenharia de requisitos) e em OCL (fase de projeto). A representação das regras em OCL descrita no Capítulo 4. As outras informações obtidas na atividade de Análise (Seção 3.2) são também armazenadas no repositório, como mostra a Figura 3.3.

Além dessas informações, o repositório deve guardar a situação atual da regra, ou seja, em que etapa do ciclo de vida do desenvolvimento da RN ela se encontra. As possíveis situações para uma regra são: "EM ESPECIFICAÇÃO", "EM MODELAGEM", "EM USO" ou "DESATIVADA".

Como é descrito na Seção 2.5, um dos requisitos de um repositório é que o acesso a ele seja bem controlado, para evitar que pessoas não autorizadas modifiquem as regras de negócio indevidamente. O esquema da Figura 3.3 contempla este requisito, pois cada usuário deve possuir pelo menos um papel no sistema, e cada papel tem permissões específicas para acessar as regras de negócio do repositório. A permissão de acesso limita as operações que podem ser realizadas pelos usuários que possuem determinado papel, ou seja, permissão para inserção, alteração, exclusão, consulta ou qualquer combinação dessas operações.

O cadastro das RN no repositório é feito através de uma janela específica, como ilustra a Figura 3.4. As informações obtidas durante a atividade de análise podem ser registradas no repositório utilizando essa janela.

3.4 Exemplos de Regras de Negócio Especificadas na Fase de Análise

Na Seção 3.2 foram descritas as informações relevantes para a análise de RN. Para exemplificar a análise de RN, é apresentado nesta seção um cenário simples, mas suficiente para permitir o entendimento.

A Figura 3.5 ilustra o modelo conceitual simplificado para uma organização fictícia. Nesse modelo estão envolvidas as entidades *Pessoa*, *Empregado* e *Departamento*. Um empregado é, na verdade, uma pessoa, contendo, além das propriedades desta (*número de CPF, nome, sexo, data de nascimento*), as propriedades *matrícula*, *data de contratação* e *salário*. Uma pessoa é identificada pelo seu número de CPF, o empregado pelo número de matrícula. O departamento corresponde ao local de trabalho de um empregado e possui as seguintes propriedades: *código*, *nome*, *salário base de seus*

funcionários e o *bônus ao salário base do funcionário que o gerencia*. Um departamento é identificado pelo seu código.

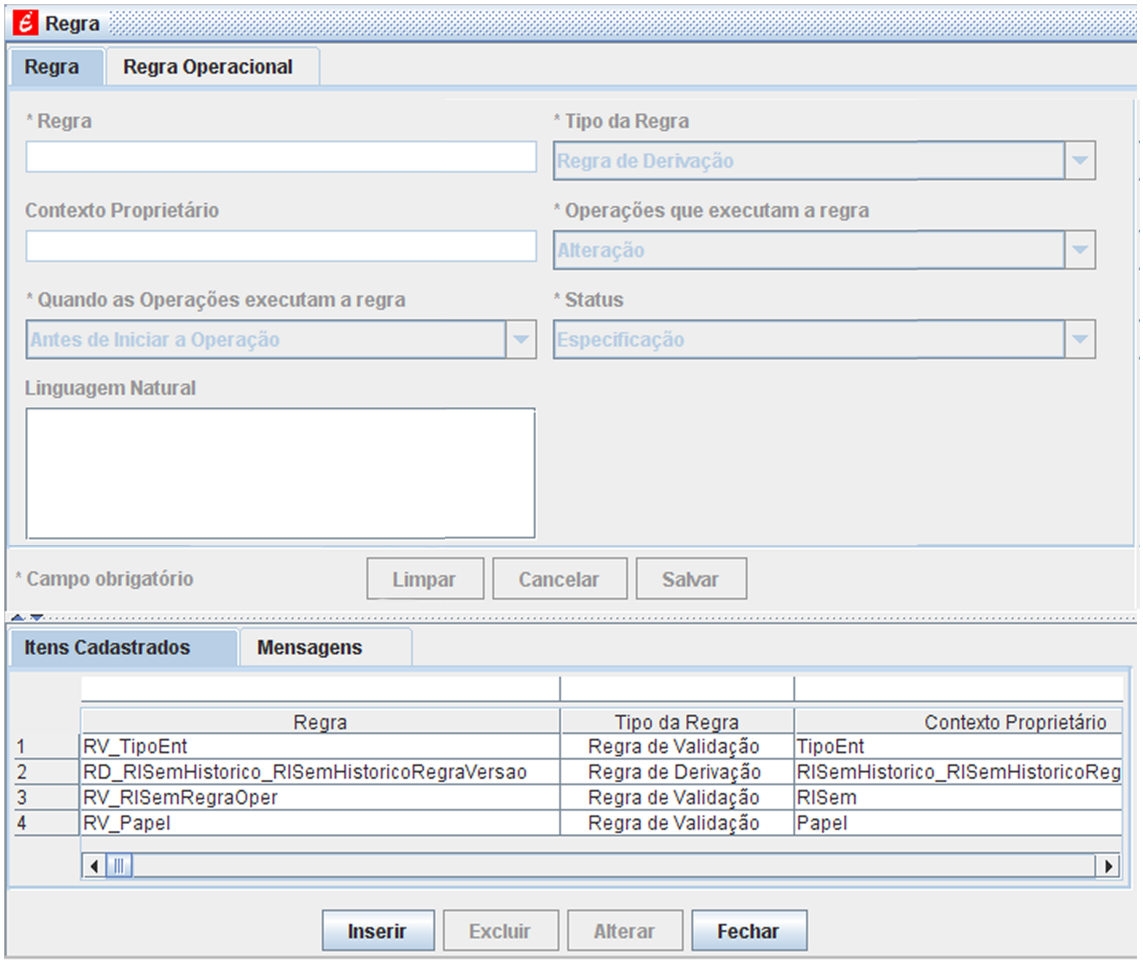

Figura 3.4: *Janela utilizada para o cadastro, atualização ou exclusão de regras de negócio.*

Para o cenário da organização descrito no modelo conceitual da Figura 3.5 poderiam ser criadas algumas RN:

- RN-1: Um empregado só pode trabalhar em um único departamento;
- RN-2: Um empregado pode gerenciar no máximo um departamento;
- RN-3: Um empregado só pode gerenciar o departamento no qual ele trabalha;
- RN-4: A data de contratação de um empregado não pode ser anterior à sua data de nascimento;
- RN-5: O salário de um empregado deverá ser igual ao salário-base definido para o departamento no qual ele trabalha. Se o empregado gerenciar o seu departamento, então ele tem um acréscimo no salário equivalente ao bônus do gerente definido para o departamento.

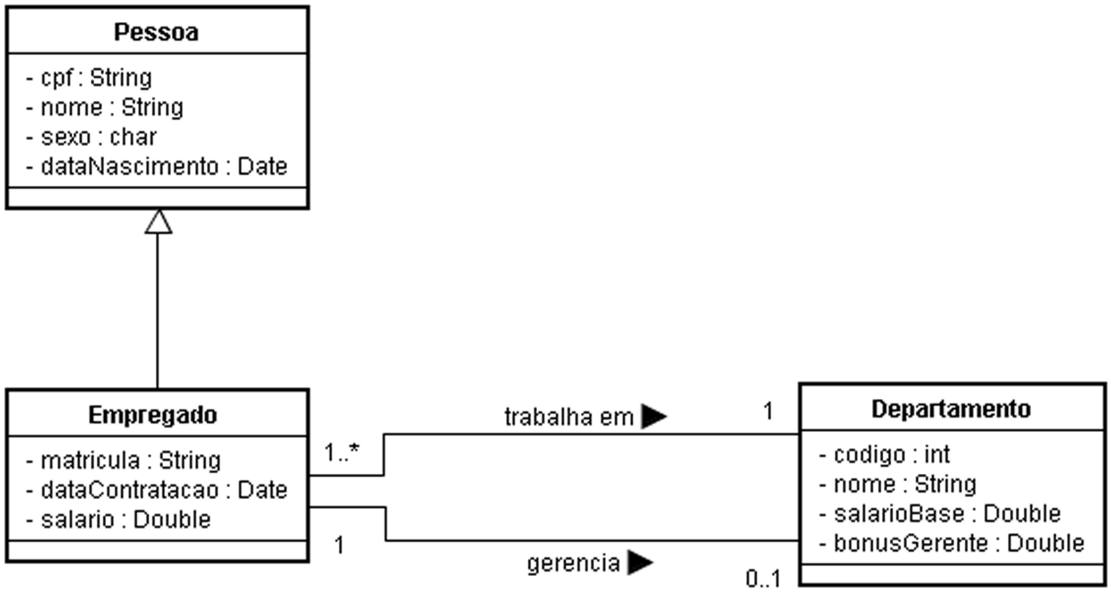

Figura 3.5: *Esquema conceitual simplificado de uma organização.*

Das RN citadas, apenas as regras estruturais (RN-1 e RN-2) estão representadas no modelo conceitual da Figura 3.5. As outras regras (regras de validação e de derivação) não podem ser representadas devido à limitação da linguagem UML. Para solucionar este problema, as regras de ação e derivação são representadas utilizando-se a linguagem OCL, como detalha o Capítulo 4.

Neste caso, apenas as regras RN-3, RN-4 e RN-5 são analisadas e as informações relevantes no contexto do negócio são extraídas para cada uma. As Tabelas 3.1, 3.2 e 3.3 mostram os detalhes relacionados às regras RN-3, RN-4 e RN-5, respectivamente. O significado de cada propriedade presente na tabela é descrito na Seção 3.2.

Tabela 3.1: *Análise da regra RN-3.*

PROPRIEDADE	**VALOR**
Identificação	RV_Empregado_1
Tipo	VALIDAÇÃO
Comportamento	INCLUSÃO ou ALTERAÇÃO
Aspecto de tempo	ANTES
Contexto proprietário	Empregado
Regra em linguagem natural	RN-3: Um empregado só pode gerenciar o departamento no qual ele trabalha.

Tabela 3.2: *Análise da regra RN-4.*

PROPRIEDADE	**VALOR**
Identificação	RV_Empregado_2
Tipo	VALIDAÇÃO
Comportamento	INCLUSÃO ou ALTERAÇÃO
Aspecto de tempo	ANTES
Contexto proprietário	Empregado
Regra em linguagem natural	RN-4: A data de contratação de um empregado não pode ser anterior à sua data de nascimento.

Tabela 3.3: *Análise da regra RN-5.*

PROPRIEDADE	**VALOR**
Identificação	RD_Empregado_salario
Tipo	DERIVAÇÃO
Comportamento	INCLUSÃO ou ALTERAÇÃO
Aspecto de tempo	ANTES
Contexto proprietário	Atributo *salario* de Empregado
Regra em linguagem natural	RN-5: O salário de um empregado deverá ser igual ao salário-base definido para o departamento no qual ele trabalha. Se o empregado gerenciar o seu departamento, então ele tem um acréscimo no salário equivalente ao bônus do gerente definido para o departamento.

Neste capítulo foram descritas as atividades do processo de Engenharia de Requisitos e como é feita a análise de regras de negócio na abordagem aqui proposta. Como exemplo da análise de regras, foi apresentado um modelo conceitual de uma empresa fictícia e os detalhes de algumas regras relacionadas. Nos capítulos 4 e 5 são descritas, respectivamente, as fases de projeto e implementação de regras a partir da análise, como descrito neste capítulo.