UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

GEOFLÁVIA GUILARDUCCI DE ALVARENGA

Uma Abordagem para Tratamento de Regras de Negócio em Sistemas de Informação

Goiânia
2007

GEOFLÁVIA GUILARDUCCI DE ALVARENGA

Uma Abordagem para Tratamento de 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 Ciência da Computação.

Área de concentração: Sistemas de Informação.

Orientador: Prof. Dr. Juliano Lopes de Oliveira

Goiânia
2007

GEOFLÁVIA GUILARDUCCI DE ALVARENGA

Uma Abordagem para Tratamento de 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 Ciência da Computação, aprovada em 27 de Julho de 2007, pela Banca Examinadora constituída pelos professores:

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

Prof. Dr. João Carlos da Silva
Instituto de Informática – UFG

Prof. Dr. Marco Aurélio Souza Mangan
Informática – PUCRS

Geoflávia Guilarducci de Alvarenga

Graduou-se em Ciência da Computação pela Universidade Católica de Goiás e especializou-se em análise e projeto de sistemas de informação pela UFG - Universidade Federal de Goiás. Durante o mestrado na UFG, foi bolsista da CAPES e desenvolveu pesquisa empírica na área de Sistemas de Informação, sob orientação do Prof. Dr. Juliano Lopes de Oliveira.

"Deus quer, o homem sonha, a obra nasce."(Fernando Pessoa)

É com imenso prazer que dedico este trabalho a quem me incentivou e ajudou a edificar esta obra, à minha ***linda mãezinha Emivar Guilarducci***.

Agradecimentos

Agradeço a Deus pelo dom da vida, pela minha saúde e pela força espiritual durante os anos dedicados ao mestrado.

A minha mãezinha Emivar pelo seu amor incondicional, ajuda financeira, apoio e dedicação inestimáveis.

Ao meu orientador Prof. Dr. Juliano Lopes de Oliveira pela excelente orientação e por acreditar no meu trabalho.

A minha irmãzinha Sandra pelo carinho e apoio financeiro.

À Empresa Brasileira de Correios e Telégrafos (ECT), aqui representados, por Djane Mara, Simone Lobão, Marcos Caldeira, Nelson, Gerson Gimenes e Luzimar, pela confiança no meu trabalho, pela aprovação da concessão da minha licença para dedicação ao mestrado e pela compreensão da minha ausência.

A minha amiga Érika pela amizade, pelos conhecimentos compartilhados e por estar ao meu lado com palavras reconfortantes e animadoras.

A minha amiga Lena pela amizade, pela força dada nos meus momentos confusos e pela disponibilidade em ajudar.

Ao colega de laboratório do INF-UFG, Daniel Ribeiro, pela paciência e ajuda dada no laboratório.

Aos colegas de projeto na empresa Estratégia pela ajuda e pelo trabalho em equipe; em especial ao colega Alexandre Claúdio pela atenção e ajudas incansáveis.

Aos colegas de mestrado, Érika, André, Luciana Nishi, Carmen, Rodrigo, Hebert e Marcos, pelos momentos de estudos e de diversão compartilhados.

À equipe de apoio da secretaria do mestrado do INF-UFG pela atenção e pelo trabalho competente prestados aos mestrandos.

À CAPES pelo auxílio financeiro.

"Do the thing you believe in, and believe in the thing you do."

Peter Chen,
Peter Chen Speaks Out; SIGMOD Record, Vol. 33, No. 1, Março 2004.

Resumo

de Alvarenga, Geoflávia Guilarducci. **Uma Abordagem para Tratamento de Regras de Negócio em Sistemas de Informação**. Goiânia, 2007. **??**p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

A construção de Sistemas de Informação (SI) envolve a especificação de três aspectos principais: dados, que são transformados para gerar as informações; funções, que manipulam e transformam dados; e regras, que controlam a manipulação dos dados pelas funções. Tradicionalmente, esta especificação baseia-se na separação entre dados e funções, sendo que as regras são embutidas tanto na especificação dos dados quanto na das funções. Porém, na prática, esta técnica não tem conseguido atender a demanda crescente de evolução dos SI, principalmente, em relação às adaptações exigidas e transformações constantes em regras de negócio. Neste trabalho apresentamos uma nova abordagem para especificação de regras de negócio em SI. Nesta abordagem as regras são especificadas como um aspecto separado de dados e funções. Esta separação não é completa, uma vez que as regras atuam sobre dados e funções; no entanto, não há subordinação da especificação das regras com relação às especificações de dados e funções. A abordagem aqui proposta é compatível com as técnicas utilizadas para implementação de regras nas áreas de Bancos de Dados e de Engenharia de Software. De fato, a idéia aqui apresentada para tratamento de regras de negócio tem como premissa a possibilidade de implementação utilizando os mecanismos disponíveis atualmente para implementação de SI. A abordagem para tratamento de regras de negócio introduzida neste trabalho abrange especificação, modelagem e implementação de regras de negócio em software de forma a facilitar o desenvolvimento e a manutenção de Sistemas de Informação complexos. A fim de demonstrar a viabilidade da abordagem proposta, uma validação empírica foi realizada no tratamento de regras em um Sistema de Informação que visa a otimização da produção de gado de corte. Este sistema está em fase final de desenvolvimento e faz parte de um projeto de pesquisa apoiado pelo CNPq.

Palavras–chave

Regra de Negócio, Requisito de Software, Restrição de Integridade, Sistema de Informação.

Abstract

de Alvarenga, Geoflávia Guilarducci. **An Approach for Treatment of Business Rules in Information Systems** . Goiânia, 2007. **??**p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

To build Information Systems (IS) it is necessary to specify three main aspects: data, which are transformed to generate information; functions, which manipulate and transform data; and rules, which control the manipulation of data by the functions. Traditionally, this specification is based on the separation of data and functions, with rules being embedded on the specification of both data and functions. In practice, however, this technique is not being able to cope with the increasing evolution of IS, which demands continuous modifications on the business rules. In this work we present a new approach to the specification of business rules in IS. In this approach, rules are specified as a separate aspect from data and functions. This separation is not complete, since rules act on data and functions; however, there is no subordination of rules specification with regard to data or functions specifications. The proposed approach is compatible with the techniques adopted to implement rules in the areas of Databases and Software Engineering. In fact, a premise of our approach is the possibility of implementation with the currently available mechanisms for the construction of IS. The approach described in this work involves specification, modeling and implementation of business rules in software, in order to make it easier to develop and maintain complex Information Systems. To demonstrate the viability of our approach, an empiric validation was performed within business rules of an Information System that deals with the optimization of cattle production. This system is now on the final stage of development and is part of a research project supported by CNPq.

Keywords

Business Rule, Software Requirement, Integrity Constraint, Information System.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Códigos de Programas

CAPÍTULO 1

Introdução

Nos dias de hoje, é difícil imaginar uma organização que não faça uso de algum Sistema de Informação (SI) para organizar e controlar todos os dados relacionados ao negócio. Além da necessidade de organização e controle, há também outros motivos que ratificam o uso de SI por uma organização como, por exemplo, o apoio às atividades do negócio, a busca de melhorias na eficiência e eficácia de processos de negócios, o apoio à tomada de decisões, e o fortalecimento da posição competitiva no mercado [29].

Para alcançar seus objetivos, um SI consiste de um conjunto de elementos que, de maneira organizada e cooperada, realizam o processamento e/ou transformação das informações que fluem em uma organização [29, 30]. O funcionamento básico de um SI pode ser resumido em: (a) obter os recursos de dados como entrada; e (b) processar e/ou transformar dados de entrada em produtos de informação como saída. As informações geradas por um SI são disponibilizadas para a organização, e podem servir de entrada para outros SI.

Sistemas de Informação baseados em computador são largamente utilizados no contexto de empresas, onde estes sistemas contêm aplicações operacionais e gerenciais que apóiam as funções básicas do negócio. Por exemplo, no contexto de agronegócio há SI para controle da produção rural; no comércio temos SI do tipo ERP (*Enterprise Resource Planning*); e na indústria existem SI para controle da linha de produção. Todos esses sistemas fazem parte de uma categoria de SI conhecida como sistemas de informação empresarial [29]. Este trabalho aborda prioritariamente os problemas desta categoria, embora os resultados aqui apresentados possam ser aplicados em outros tipos de SI.

1.1 Elementos de um Sistema de Informação

Os elementos que compõem um SI, conforme mostra a Figura 1.1, são [30]: software, programas de aplicação e software básico, por exemplo; hardware, incluindo computadores e outros dispositivos eletrônicos; pessoas, gerentes e operadores de computadores, por exemplo; banco de dados (BD), coleção de informações relacionadas, armazenadas em meio eletrônico e outras mídias; documentação, tais como textos informa-

tivos e manuais de operação do sistema; e procedimentos, que definem o comportamento do SI, tanto no ambiente interno quanto no ambiente externo à organização (por exemplo, definição dos papéis de cada elemento e da forma como eles se comunicam).

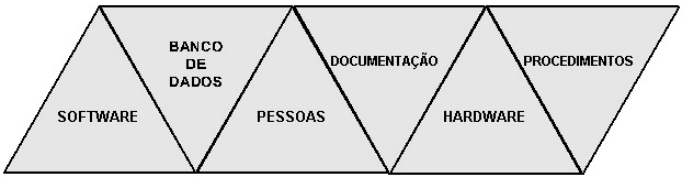

Figura 1.1: *Componentes de um SI.*

Cada elemento de um SI pode conter um conjunto de **regras de negócio**, isto é, regras semânticas que definem o significado e o compromisso do elemento com relação ao negócio. Essas regras estabelecem a forma pela qual o elemento contribui para o atendimento das **necessidades de informação** da organização. Em outras palavras, as regras de negócio governam o processamento de informações em uma organização.

Em **BD**, as regras de negócio expressam como a coleção de informações são estruturadas, em termos de relacionamento e armazenamento, assim como os meios que acessam essas informações. Um exemplo deste tipo de regra de negócio seria: todas as informações do negócio são estruturadas usando o Modelo Relacional de Dados, e o meio de acesso às informações é através de um software de gerenciamento de bancos de dados que implementa este tipo de modelo de dados.

Em **Documentação**, as regras descrevem definições, características ou padrões para elaborar e utilizar os documentos de um SI. Por exemplo: qualquer manual de operação de software deve obedecer o padrão ABNT/NBR-14724.

As regras aplicadas a **Pessoas** definem quem participa do SI e como as pessoas autorizadas devem manipular os elementos do SI. Essas regras também podem atuar em todo o contexto de administração dos recursos humanos envolvidos no SI. Por exemplo: os engenheiros de software devem ser mestres em Ciência da Computação. Neste exemplo, citamos uma regra de conhecimento exigida para as pessoas que participam de uma atividade do SI (construção do software).

Regras sobre **Procedimentos** atuam sobre a definição do comportamento do SI. Um exemplo seria: "o atendimento do suporte técnico em informática deve ocorrer conforme o procedimento definido no manual de operações internas da empresa".

Em **Hardware**, as regras estão direcionadas à infra-estrutura computacional que oferece suporte aos demais elementos do SI. Por exemplo: todos os computadores devem ter dispositivos de multimídia.

Dentre os elementos de um SI, este trabalho analisa os aspectos relacionados ao tratamento de regras de negócio que abrangem o elemento "software". Essas regras podem ser divididas em regras do processo e do produto de software.

As regras de **Processo** de software definem a forma de executar os processos fundamentais do ciclo de vida de um software [19]: aquisição, fornecimento, desenvolvimento, operação e manutenção. Custo estimado para o desenvolvimento e prazo para construção de determinadas funções do software são exemplos de regras de processo de software.

As regras que envolvem o **Produto** de software são relacionadas às características funcionais e de qualidade que devem estar presentes no produto de software. Estas características são definidas, por exemplo, pela norma ISO/IEC 9126 [18]. Um exemplo de regra de produto no contexto de agronegócio seria: "o software deve possibilitar o registro de um nascimento de um animal; este registro pode ser feito em campo com dispositivos portáteis, ou a posteriori, no computador na sede da propriedade". Neste caso temos uma regra definindo uma funcionalidade e outra regra, associada, definindo um aspecto de portabilidade.

Embora todas as regras de negócio aplicadas aos diversos elementos do SI sejam importantes para atender às necessidades de informação da organização, as regras relacionadas ao **produto de software** têm um papel de destaque porque o software é o elemento mais adaptável do SI. A adaptabilidade a mudanças em regras de negócio é crítica nas empresas modernas, o que gera uma tendência cada vez maior de direcionar todas as regras para este elemento.

Tradicionalmente, a especificação de um SI baseia-se na separação entre dados e funções que manipulam esses dados. Porém, esta técnica não tem conseguido atender com eficiência as necessidades de evolução dos SI. De fato, os SI têm se tornado cada vez mais complexos e instáveis devido às mudanças aceleradas pela globalização, principalmente em relação às adaptações e transformações constantes em regras de negócio.

Assim, o desafio para o componente de software de SI é representar e tratar regras de negócio visando minimizar os efeitos dessas mudanças. Este trabalho apresenta uma proposta neste sentido. A abordagem proposta abrange especificação, modelagem e implementação das regras de negócio de uma forma que facilita a sua evolução e adaptação de acordo com as necessidades do negócio.

Esta abordagem foi validada empiricamente no contexto do desenvolvimento de um Sistema de Informação real, voltado para a área de agronegócio e, mais especificamente, para o controle da produção de gado de corte. Este SI está em fase final de desenvolvimento e faz parte de um projeto de pesquisa apoiado pelo CNPq [7]. Como as propostas deste trabalho foram validadas no contexto deste SI, a maior parte dos exemplos apresentados no texto são referentes a este contexto.

1.2 Contribuições e Organização do Trabalho

A principal contribuição desta dissertação é uma metodologia para tratamento das regras de negócio que facilita a fase de manutenção dentro do ciclo de vida de um software. Outras contribuições desta dissertação incluem:

- o estudo e a consolidação dos conceitos de regras de negócio, requisitos e restrições de integridade no contexto de SI;
- uma taxonomia para regras de negócio em SI;
- um estudo sobre o uso dos conceitos de regras de negócio combinado com o paradigma de arquitetura dirigida por modelos;
- um exemplo do uso de um modelo independente de tecnologia computacional para representação de regra de negócio em SI;
- uma proposta para implementação em software de regras de negócio em SI a partir das especificações de um modelo independente de tecnologia.

O restante deste trabalho descreve a proposta para tratamento de regras de negócio em SI e está organizado da seguinte maneira:

O Capítulo 2 trata de conceitos fundamentais para o entendimento do trabalho, abordando regras de negócio, requisitos e restrições em SI. Neste capítulo propomos uma taxonomia para regras de negócio em Sistemas de Informação.

O Capítulo 3 aborda os fundamentos do paradigma MDA (Arquitetura Dirigida por Modelos) e de uma linguagem para representação de regras de negócio em conformidade com os fundamentos dessa arquitetura.

O Capítulo 4 define a proposta para tratamento de regras de negócio em Sistemas de Informação. A especificação é baseada em uma metodologia que é detalhada nos capítulos subseqüentes.

O Capítulo 5 explica a fase de análise de regras de negócio. Esta fase concentra-se no entendimento do que é especificado pela regra de negócio e na compreensão dessa regra por parte do projetista do software.

O Capítulo 6 explica a fase de projeto de regras de negócio. Esta fase consiste na modelagem em alto nível de abstração as estruturas de uma regra de negócio especificada na fase de análise.

O Capítulo 7 explica a fase de implementação de regras de negócio. Esta fase visa representar as estruturas de uma regra de negócio modeladas em alto nível de abstração internamente no software da aplicação.

Finalmente, o Capítulo 8 traz as considerações finais do trabalho de pesquisa desenvolvido, comparando-o com trabalhos correlatos e propondo extensões e direções para a melhoria da abordagem aqui proposta.

CAPÍTULO 2

Regras de Negócio em Sistemas de Informação

Neste capítulo discutimos um dos conceitos principais que balizam nosso trabalho: regras de negócio.

A Seção 2.1 estuda conceitos e características de regras de negócio, e destaca sua importância no contexto de software em SI.

A Seção 2.2 apresenta uma proposta de modelagem de regras de negócio baseada em níveis e camadas de representação da regra. Essas representações permitem a elaboração de regras que expressem a realidade de um negócio de forma mais precisa e consistente, além de viabilizar o processo de automatização das regras (neste caso, em um estágio de implementação de regras).

Para complementar o estudo sobre regras de negócio, a Seção 2.3 descreve uma nova taxonomia para regras de negócio em SI. Esta taxonomia permite um mapeamento dos diversos tipos de regras existentes para a forma com que estas regras são implementadas nos SI atuais.

Neste sentido, a Seção 2.4 analisa e correlaciona as duas principais tecnologias computacionais atualmente empregadas para implementação de regras de negócio em SI: a visão da Engenharia de Software, baseada no conceito de *requisito*; e a visão de BD, baseada no conceito de *restrição de integridade*.

2.1 Conceitos e Características

Regra de Negócio (RN) é toda norma ou tudo aquilo que a lei ou o uso comum determina a respeito de qualquer transação que envolve uma determinada organização. Portanto, regra de negócio é uma diretriz destinada a regulamentar o comportamento do negócio [40, 47].

Sob a perspectiva de SI, regra de negócio é: *"uma declaração que define ou restringe algum aspecto do negócio"* [40]. Neste contexto regras de negócio são usualmente definidas como restrições ou metadados sobre operações de negócio [49], e também são conhecidas como **regras do domínio da aplicação**, ou regras de minimundo. Neste caso,

minimundo é definido como um problema específico de uma organização. As RN expressam declarações que são

> *"elementos-chave na definição das intenções e necessidades do negócio, ou reflexões de como a organização trabalha ou como pretende trabalhar no futuro"* [27].

Apesar da importância das RN, elas têm recebido pouca atenção por parte dos métodos de análise de sistemas, os quais têm dedicado maior esforço para a modelagem das estruturas funcionais e de dados utilizadas por uma organização [40, 47, 48].

De fato, estes métodos geralmente não permitem separar a definição de uma regra de negócio da forma de implementação desta regra. Assim, para definir uma regra é preciso especificar detalhes operacionais, tais como [27]:

- *quem* invoca uma regra: esta informação geralmente é descrita em um caso de uso ou em uma descrição de processo do negócio;
- *quando* a regra é executada: normalmente isto é descrito por um evento, caso de uso ou descrição de processo do negócio;
- *onde* a regra é executada: esta decisão é pertinente ao *design* do sistema e, mais especificamente, ao empacotamento do software; e
- *como* a regra é implementada: também é uma decisão da fase de projeto.

Todas essas informações são importantes para implementar uma RN, mas é essencial que seja possível definir *"o que é"* a RN independentemente da forma como ela é implementada.

Uma característica comum às regras de negócio é que elas fazem uso dos chamados "parâmetros do negócio". Tais parâmetros podem ser definidos e gerenciados da mesma forma que as regras de negócio [27]. Um exemplo de parâmetro do negócio seria, "a condição do parto de um animal deve ser escolhida da seguinte enumeração: regular, prematuro, atrasado".

A Tabela 2.1 resume características de RN encontradas na prática em muitas organizações [27]. Estas características confirmam os princípios introduzidos em [41]: "regras deveriam ser expressas declarativamente, em linguagem natural, bem-formadas e orientadas para as metas do negócio".

Tabela 2.1: *Características de RN (adaptadas de [27]).*

Característica: Uma RN deve ser ...	Justificativa
Atômica.	Caso uma regra seja definida em termos de sub-unidades, estas sub-unidades não serão a mesma RN original. Isto quer dizer que a tentativa de dividir uma RN em outras regras mais simples resultará em perda de informação e também em perda semântica.
Declaração do negócio.	RN deveriam descrever a lógica do negócio e não a tecnologia que irá implementar a regra.
Declarativa.	RN devem contribuir para um objetivo do negócio, ao invés de descrever como o objetivo do negócio é alcançado.
Restritiva.	RN limitam as ações que podem ser aplicadas no contexto do negócio.
Representada em linguagem natural.	RN devem ser expressas através de um idioma natural, pois isto dispensa qualquer tipo de treinamento específico ou uso de ferramentas. Em contra-partida, podem ocorrer expressões ambíguas. Em conseqüência disso, torna-se necessário mapear as regras escritas em linguagem natural para uma expressão matemática formal, como uma linguagem de programação, por exemplo.
Rastreável.	É preciso saber como as RN se encaixam dentro de um SI, e rastreá-las desde sua origem até as suas realizações para manter o acompanhamento de todo o ciclo de vida do SI.
Estruturada.	As regras devem ser escritas de tal maneira que facilitem a leitura e o entendimento. Porém, é necessário restringir o número de opções para se escrever uma regra, isto é, determinar padrões estruturais para representação de regras. Esta prática pode apoiar o processo de automatização da regra, ou seja, o mapeamento da regra para uma implementação desejada.

2.2 Modelagem de Regras de Negócio

A comunidade que pesquisa SI defende a idéia de modelar as RN como uma parte independente da arquitetura do software. Esta idéia consiste em uma divisão do SI em três componentes: dados, funções e regras de negócio [40, 47, 48].

Com essa divisão é possível modelar de forma precisa e independente de implementação as RN de uma determinada realidade. Por exemplo, é possível representar as RN conceitualmente, de maneira a facilitar o processo de compreensão das regras e de automatização da sua execução pelo SI. Além disso, pode-se ter à disposição das pessoas do negócio as regras formalizadas de maneira explícita e declarativa, expressando todo o conhecimento básico do negócio da organização [6, 41].

Para modelar as RN é preciso representá-las de maneira formal. Uma representação em dois níveis, de análise e de projeto, é proposta em [32] para a modelagem de uma RN.

No **nível de análise** a representação de uma RN deve estar de acordo com a compreensão do proprietário do negócio. É recomendado que as declarações usem uma linguagem natural limitada por padrões [27]; diante disso, a abstração usada requer algum tipo de formatação, ou seja, o uso de uma espécie de gabarito (*template*) para moldar a declaração de uma determinada regra.

São sugeridos gabaritos para formatação padrão de textos e de sentenças. Um exemplo simples seria formatar as sentenças que declaram as regras da seguinte forma: [artigo] + [sujeito] + [verbo] + [predicado] + [complemento]. Usando este gabarito no contexto de agronegócio, poderia ser definida a seguinte regra: "o estoque de vacinas deve ser verificado diariamente".

Em [27] são propostos alguns padrões para formatação da declaração de uma RN, e cada padrão obedece uma determinada forma canônica. Por exemplo, uma RN poderia ser declarada assim: "a era de um animal deve ser escolhida da seguinte enumeração fechada: bezerro, bezerra, novilho, novilha, boi, vaca". Nesse exemplo, demonstramos a aplicação do padrão denominado Padrão Enumeração (*Pattern Enumeration*). Este padrão estabelece uma faixa de valores permitidos para um termo do negócio.

O **nível de projeto** define como as regras são representadas dentro do sistema. A abstração usada requer uma combinação de técnica (dados estruturados, operadores e linguagem de restrições, por exemplo) e formalismo (declarações de acordo com uma sintaxe matemática, por exemplo) [27, 32].

No nível de projeto, a representação de uma regra de negócio pode ser definida em três camadas: externa, conceitual e interna. Isto corresponde à proposta da arquitetura de três-esquemas de bancos de dados, conhecida como arquitetura ANSI/SPARC [10]. Completando esta analogia, a Figura 2.1 resume as etapas (níveis e camadas) para

representação de regras de negócio.

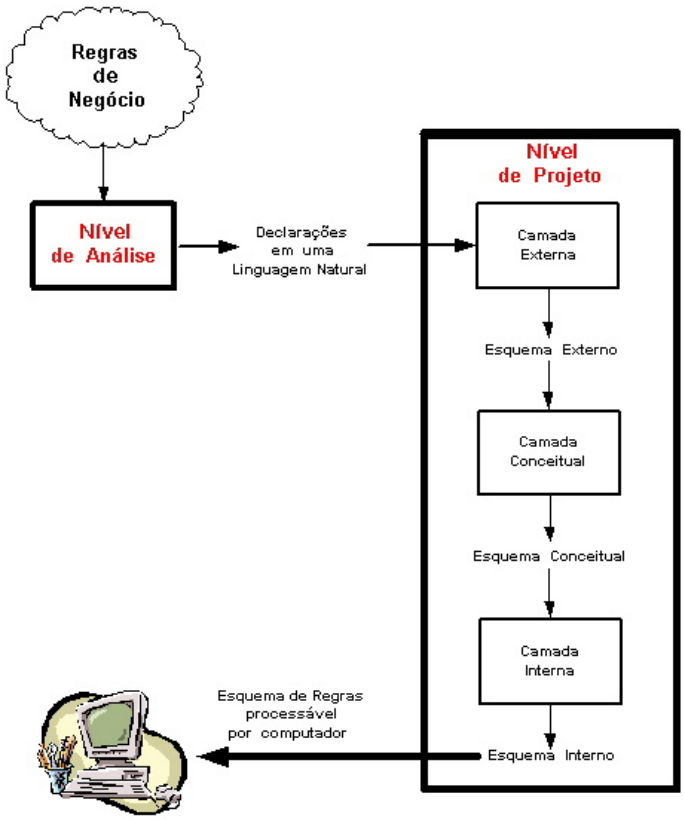

Figura 2.1: *Etapas para representação de regras de negócio (adaptado de [32]).*

A **camada externa** descreve as visões de diferentes categorias de usuários de TI em relação às regras de negócio. Nesta camada concentra-se esforços para modelagem das regras.

As ferramentas de modelagem e especificação tradicionais são úteis para a construção do esquema externo que representa as RN [32]. Entre estas ferramentas pode-se destacar: tabela de decisão, diagrama de transição de estados (DTE), diagrama de fluxo de dados (DFD), modelo Entidade-Relacionamento (MER), diagramas da UML, português estruturado, lista de eventos, dicionário de dados e casos de uso. Independentemente das ferramentas empregadas, a camada externa deve ser escrita usando termos do domínio do problema, de forma a facilitar o seu entendimento pelas pessoas do negócio.

A **camada conceitual** descreve em alto nível as estruturas abstratas das RN. O ponto importante nesta camada é a linguagem usada para representar as regras. O esquema conceitual gerado deve representar as regras de uma maneira que seja possível automatizar a sua execução [32]. Algumas linguagens possuem esta característica como, por exemplo, Lógica de Predicados, ORM (*Object Role Modeling*) e OCL (*Object Constraint Language*).

A **camada interna** descreve as estruturas das RN da forma como são implementadas. O esquema interno será o produto gerado pelo mapeamento realizado do esquema conceitual para uma sintaxe específica de uma linguagem de programação ou de uma linguagem de banco de dados como, por exemplo, SQL (*Structured Query Language*).

2.3 Taxonomia de Regras de Negócio

Em SI complexos existem RN de diversas naturezas. É preciso compreender os vários tipos de RN, tanto para validar o seu significado no contexto do negócio quanto para implementar corretamente as regras em uma determinada tecnologia. Neste sentido, a taxonomia proposta em [40], identifica três classes principais de RN em SI:

Assertiva Estrutural (*Structural Assertion*): inclui regras que definem conceitos ou declarações de fatos que expressam aspectos da estrutura de uma organização. Por exemplo, no contexto de agronegócio existem os seguintes termos: "proprietário, propriedade rural e animal". Um fato inerente ao referido contexto seria: "um proprietário pode possuir várias propriedades rurais, e cada propriedade rural pode conter diversos animais".

Assertiva de Ação (*Action Assertion*): inclui regras que especificam declarações de restrições ou condições que limitam ou controlam as ações de uma organização. Por exemplo: "somente os animais nascidos em uma propriedade rural são identificados com a marca desta propriedade".

Derivação (*Derivation*): inclui regras que especificam declarações de um conhecimento que é derivado de outro conhecimento do negócio. Por exemplo: "a era do animal é obtida de acordo com sexo e a idade em dias do animal; por exemplo, se o animal é macho e sua idade está dentro do intervalo de 1 até 210 dias, então esse animal é um bezerro".

Esta taxonomia pode ser estendida a partir da engenharia reversa da implementação de RN proposta pelo Paradigma de Orientação a Objetos (visão da Engenharia de Software) e pelo Modelo Relacional de Dados (visão de BD). Assim, apresentamos uma nova proposta de taxonomia de RN orientada à implementação de software. Essa taxonomia, que estende a proposta de [40], é composta por classes de regras independentes de tecnologia, e passíveis de implementação utilizando as tecnologias disponíveis atualmente. As novas classes sintetizam os conceitos existentes em Bancos de Dados e em Linguagens de Modelagem para expressar RN.

A Figura 2.2 ilustra esta taxonomia que será utilizada no decorrer deste trabalho. O restante desta seção descreve as classes propostas na taxonomia [1]. Somente serão discutidas as classes diretamente envolvidas com o propósito deste trabalho, ou seja, as subclasses de RN de Produto de Software. As demais classes representam os conceitos comuns à área de SI, discutidos no Capítulo 1.

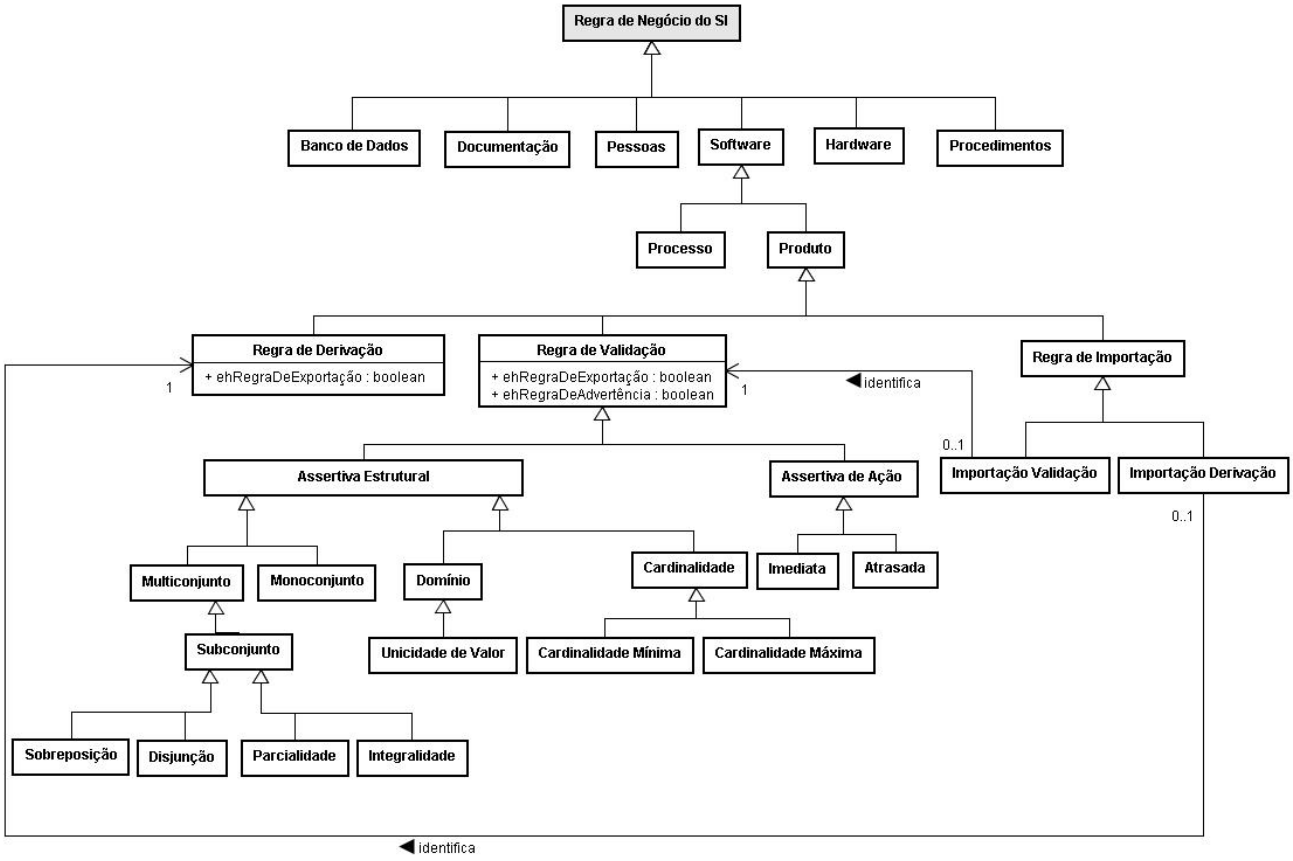

Figura 2.2: *Uma taxonomia de RN orientadas a software.*

2.3.1 Regras de Validação e Derivação

Para compreender a taxonomia apresentada na Figura 2.2 é necessário definir os conceitos de conjunto e elemento sob a visão do software de um SI.

Um **conjunto** é uma coleção de informações correlatas do negócio. Alguns exemplos de conjuntos no contexto de agronegócio são: propriedades rurais, animais, pastos, manejos. Associações de informações também formam conjuntos como, por

[1] As especializações são Disjuntas e Totais, ou seja, não há sobreposição entre elementos de subclasses, e cada elemento de uma superclasse corresponde a algum elemento de uma de suas subclasses; com exceção da classe Domínio que é Parcial, ou seja, permite-se que um elemento da superclasse não pertença a nenhuma de suas subclasses.

exemplo: as propriedades rurais podem ser divididas em áreas, e as áreas podem conter diversos pastos.

Neste exemplo temos os conjuntos de propriedades, áreas e pastos, e também os conjuntos de áreas que compõem propriedades, e de pastos contidos em áreas. Certas RN são definidas sobre conjuntos; por exemplo: o número de áreas sem pastos deve ser menor que o número de áreas com pastos.

Um **elemento** de um conjunto é um conceito do negócio, que pode ser expresso por um termo ou por um fato [40]. Um termo é uma palavra, expressão, ou frase que define um conceito do negócio. Um fato correlaciona dois ou mais conceitos do negócio. Um elemento (termo ou fato) de um conjunto é um sinônimo para uma informação contida em um conjunto. Existem RN sobre elementos de um conjunto; por exemplo: a área de um pasto individual não pode ser maior que 10 alqueires. Assim, as RN são definidas em função de conjuntos e elementos do negócio. Há duas classes principais de RN: Regras de Validação e Regras de Derivação.

Uma **Regra de Validação (RV)** é uma RN que contém restrições ou condições que determinam a integridade semântica de um conjunto ou elemento do negócio. Nesta classe se enquadram as RN do tipo Assertiva Estrutural ou Assertiva de Ação. A avaliação de uma RV prevê uma descontinuação do comportamento previsto pelo negócio, caso a regra não seja satisfeita em um determinado estado do SI.

Uma RV ainda pode ser caracterizada com conotação **de advertência**. Esta conotação torna o sistema de regras mais flexível, separando as RV em obrigatórias (são as RV padrão, que não podem ser violadas) e de advertência (regras opcionais, cuja violação é notificada, mas sem impedir, necessariamente, a continuidade do processo de negócio).

No caso de uma RV de advertência ser violada, ocorre uma notificação de violação, que possibilita uma interação com o usuário. Este pode aceitar a violação, ignorando a regra e dando continuidade ao comportamento típico definido pelo negócio. O usuário também pode rejeitar a violação, causando o mesmo efeito que a violação de uma RV obrigatória.

Uma **Regra de Derivação (RD)** é uma RN que contém 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, em conformidade com as características dos conceitos do negócio envolvidos na regra.

Como exemplo de RD pode-se citar a era de um animal (que define se o animal é bezerro, novilho, touro, bezerra, novilha ou vaca). A era pode ser obtida a partir da análise da idade e do sexo do animal. A definição da era é importante para a realização de manejos operacionais, tais como desmama, castração, e vacinações específicas.

2.3.2 Regras de Importação e Exportação

Na taxonomia apresentada na Figura 2.2 há uma terceira classe de regras, além das Regras de Validação (RV) e de Derivação (RD). Trata-se das Regras de Importação que são, de fato, *metaregras*, isto é, são regras que definem a forma de obtenção e aplicação das regras básicas (RV e RD).

Uma **Regra de Importação (RImp)** é definida como uma RN cujas declarações expressam como importar uma RN que é propriedade de outro conjunto ou elemento do negócio, isto é, que faz parte de outro contexto.

A avaliação de uma RImp identifica uma RV ou uma RD que deve ser avaliada. Assim, uma RImp pode ser classificada como uma Regra de Importação de RV (representada, na Figura 2.2, pela classe Importação Validação) ou como uma Regra de Importação de RD (representada pela classe Importação Derivação).

Ambas RV e RD podem ser caracterizadas com conotação de **exportação**. Isto indica que a RN é exportada para outra entidade ou conceito do negócio, isto é, para outro contexto. Isso implica que a RN com conotação de exportação nunca deverá ser avaliada no contexto que é proprietário da RN, mas somente no contexto de destino da RN, ou seja, no contexto que faz uso da regra.

Em conseqüência, a avaliação de uma RN de Exportação é dependente de um comportamento de uma Regra de Importação. As declarações contidas nesse tipo de regra expressam significado somente para o contexto que se destina a regra e não para o contexto que é proprietário da regra, e é por este motivo que não devem ser avaliadas no contexto proprietário.

Explicamos a seguir, duas situações que exemplificam a existência de uma Regra de Importação e uma **RN de Exportação (RE)**.

Uma situação seria: o conceito do negócio "Identificação de Proprietário Rural" possui a seguinte RN: "cada elemento ou instância de uma Identificação de Proprietário Rural possui exatamente uma regra de formação que valida a identificação de um Proprietário Rural". Um exemplo desse tipo de instância (ou regra de formação) seria CPF ou CNPJ.

Nesta situação, a entidade que é proprietária da RN é o conceito "Identificação de Proprietário Rural" e a entidade para a qual a RN se destina é o conceito "Proprietário Rural". Portanto, a RN em questão é uma RN de exportação, ou seja, será avaliada em outro contexto, que neste exemplo é a entidade "Proprietário Rural". Para que a entidade "Proprietário Rural" faça uso desta RE ela deve ter uma RImp que expressa como importar esta RE, ou seja, que declara como a RE é encontrada no sistema.

Outra situação semelhante seria no contexto de um SI de Recursos Humanos de uma organização, especificamente no domínio de Folha de Pagamento de Funcionários. O conceito do negócio Provento possui a seguinte RN: "cada instância de um Provento

possui exatamente uma regra salarial que determina o cálculo deste provento em uma folha de pagamento". Exemplos desse tipo de instância (ou regra salarial) incluem décimo terceiro salário, adicional noturno, adicional de insalubridade, e salário-família.

Nesta situação, a entidade proprietária da RN é Provento e a entidade para a qual a RN se destina é Folha de Pagamento. Seguindo o mesmo raciocínio anterior, a RN em questão é uma RN de exportação definida pela entidade Provento, mas que será avaliada em outro contexto (neste exemplo, o contexto da entidade Folha de Pagamento). Para que a entidade Folha de Pagamento faça uso desta RN de exportação ela deve ter uma RImp que expressa como importar esta RE, ou seja, que declara como a RE é encontrada no sistema.

2.3.3 Tipos de Assertiva Estrutural

Regras de Validação são classificadas em assertivas estruturais e de ação, conforme mostra a Figura 2.2. Estas classes de assertivas seguem as mesmas definições propostas em [40] e sintetizadas na Seção 2.3.

Partindo da classe Assertiva Estrutural, dois critérios de classificação são propostos. O primeiro critério é quanto ao número de conjuntos envolvidos na RN: Multiconjunto ou Monoconjunto. Já o segundo critério define a natureza da restrição envolvida na RN: Domínio ou Cardinalidade. Esses critérios são ortogonais, ou seja, uma RV pode ser classificada em ambos os critérios.

Uma Assertiva Estrutural **Multiconjunto**, envolve dois ou mais conjuntos. Por exemplo: "é preciso identificar a propriedade rural à qual pertence um animal". Neste exemplo temos dois conjuntos envolvidos, "propriedade rural" e "animal". Enquanto que na classe **Monoconjunto** a RN somente envolve um único conjunto. Por exemplo: "bezerros devem ter menos que 400 kg".

Ainda em Multiconjunto, temos a subclasse **Subconjunto** que especifica RN de continência em conjuntos e se especializa em dois critérios. No primeiro, essa classe agrupa regras que especificam se dois subconjuntos do mesmo superconjunto podem ter interseção (**Sobreposição**) ou se são mutuamente exclusivos (**Disjunção**). O segundo critério agrupa regras que especificam se todos os elementos do superconjunto precisam ou não estar contidos em algum subconjunto (classe **Integralidade** e **Parcialidade**, respectivamente).

Um exemplo de RN do tipo Integralidade seria: "todo animal pertencente a uma propriedade rural deve ser caracterizado por um dos tipos: animal de propriedade, animal de catálogo, ou animal para informação genealógica"; e um exemplo de Disjunção seria: "um animal ou é animal de propriedade, ou de catálogo, ou para informação genealógica".

A classe **Domínio** retrata as RN que especificam o domínio de uma determinada informação de um conjunto, isto é, especificam a coleção de valores aceitáveis para essa dada informação. Por exemplo, "as condições do parto possíveis para animais são: parto regular, parto prematuro ou parto atrasado".

Uma RN de Domínio ainda pode ter uma especialidade em relação à exclusividade do valor (ou conteúdo) de uma determinada informação de um conjunto. Neste caso, estamos nos referindo à classe **Unicidade de Valor**. Por exemplo, "o nome de um animal em uma propriedade rural deve ser exclusivo, ou seja, não pode haver nomes de animais repetidos".

A classe **Cardinalidade** representa as RN que especificam a obrigatoriedade de uma determinada informação de um conjunto ou a limitação do número de possíveis elementos de conjuntos. Uma regra dessa natureza deve ser ou de cardinalidade mínima ou de cardinalidade máxima.

Quando for de **Cardinalidade Mínima**, uma RN especifica a obrigatoriedade mínima de uma informação de um conjunto, ou a participação mínima entre conjuntos. Por exemplo, "o nome de fantasia de uma propriedade rural é obrigatório", ou seja, deve haver pelo menos um valor para essa informação.

Quando for de **Cardinalidade Máxima**, uma RN especifica a quantidade máxima de uma informação em um conjunto, ou a participação máxima entre conjuntos. Por exemplo, "uma propriedade rural pode conter no máximo dez técnicos em agropecuária".

2.3.4 Tipos de Assertiva de Ação

Além de Assertivas Estruturais (estáticas), as Regras de Validação (RV) podem definir Assertivas de Ação (dinâmicas). Partindo da classe Assertiva de Ação, propomos um critério de classificação baseado no momento (aspecto de tempo) em que uma RV é avaliada.

Na avaliação **Imediata**, a regra é avaliada no mesmo tempo em que a ação do negócio que está sujeita à regra. Por exemplo: "a ocorrência de um nascimento de um animal em uma propriedade rural exige também uma ocorrência de vacinação para o animal nascido". Neste caso, a regra é avaliada de forma simultânea com a ocorrência dos fatos pertinentes à regra.

Na avaliação **Atrasada**, a regra somente é avaliada depois que uma seqüência de ações do negócio é realizada. Por exemplo: "somente após as ocorrências de vacinação contra a Febre Aftosa é que um animal está preparado para seleção de animais para abate". Neste caso, a regra avalia fatos preexistentes no SI, isto é, a regra é avaliada de forma atrasada com relação aos fatos pertinentes à regra.

2.4 Abordagens para Implementação de RN

Esta seção discute as duas principais abordagens atuais para implementação de regras de negócio em SI: linguagens de programação (abordagem de Engenharia de Software) e esquemas de BD (abordagem de BD).

2.4.1 RN na perspectiva da Engenharia de Software

Dada a taxonomia de RN de software em SI (mostrada na Figura 2.2), podemos perceber que a classe principal **Regra de Negócio do SI** tem a mesma finalidade semântica do termo **requisito** utilizado na Engenharia de Software para denominar as necessidades que dão origem a formação de um SI, especialmente na fase de definição do produto de software que irá formar a base deste SI.

De fato, uma definição conhecida para o termo requisito é: *"condição que se exige para certo fim"*. A Tabela 2.2 reúne outras definições sobre o termo "requisito" propostas na literatura.

Tabela 2.2: *Definições do termo "requisito" em SI.*

Autor(es)	Definição
Jackson [20]	*"Requisitos estão sobre os fenômenos do domínio da aplicação, e não sobre os da máquina".*
Sommerville e Sawyer [39]	*"Requisitos são especificações do que deveria ser implementado; estas especificações são descrições de como o sistema deveria se comportar, ou de uma propriedade do sistema ou atributo; elas também podem ser uma restrição sobre o processo de desenvolvimento do sistema".*
Kotonya e Sommerville [22]	*"Requisito é uma declaração de um serviço do sistema ou restrição".*
Sommerville [38]	*"As descrições das funções e das restrições são os requisitos para o sistema".*
IEEE 830 [36]	*"Um requisito especifica uma função externamente visível ou atributo de um sistema".*
IEEE 1233 [35]	*"Um requisito bem formado é uma declaração de funcionalidade do sistema (uma capacidade) que possa ser validada, que deva ser conhecida ou possuída por um sistema para resolver um problema do cliente ou alcançar um objetivo do cliente, e que seja qualificada através de condições mensuráveis e limitada através de restrições".*

A variedade de definições confirma a conclusão de [38]: *"o termo requisito não é utilizado pela indústria de software de modo consistente"*; isto é, a definição está sujeita a contradições. Isto implica que, para alguns, requisito pode ser definido como uma

declaração abstrata e sem muitos detalhes das funcionalidades do sistema e, para outros, pode ser uma especificação formal e detalhada.

Ainda sob a perspectiva de SI, há várias classificações de requisitos encontradas na literatura. Em [24], os autores propõem duas classes principais:

Requisitos de Software são aqueles que representam as necessidades dos clientes e condicionam a qualidade do software. Podem ser divididos em requisitos funcionais e não-funcionais.

Restrições de Projeto são limitações impostas sobre o projeto que fará a construção do sistema como, por exemplo, prazo e custo determinados para construir o software.

Segundo [24], os requisitos funcionais dizem respeito aos aspectos comportamentais do software. Por exemplo, para o controle de uma propriedade rural, é necessário que o software permita a emissão de um relatório mensal, demonstrando todas as transações comerciais envolvendo animais (compra, venda e alienação).

Já os requisitos não-funcionais dizem respeito às condições de comportamento e restrições que devem prevalecer no software como, por exemplo, "o tempo de geração e emissão do relatório mensal que demonstra o histórico das transações comerciais de uma propriedade rural não deve ultrapassar 5 minutos". Neste exemplo, citamos um requisito de performance (ou um atributo de qualidade) que o software deve atender.

Em [39] não se propõe explicitamente uma classificação para requisitos em SI, mas uma orientação para o detalhamento dos requisitos em dois **níveis de detalhe**: primeiramente, os requisitos são expressos em uma descrição abstrata e, posteriormente, desenvolvidos com mais detalhes em uma especificação do sistema.

Esta orientação sobre como expressar requisitos pode ser considerada como um critério de classificação de requisitos:

Requisitos do Usuário (*Stakeholders Requirements*) são os requisitos escritos sob a visão dos usuários do sistema; são expressos sem maiores detalhes e usando uma linguagem natural, por exemplo.

Requisitos de Sistema (*System Requirements*) são especificações detalhadas dos requisitos através de um modelo abstrato do sistema; através de um diagrama de fluxo de dados, por exemplo.

A diferença entre requisitos funcionais e não-funcionais é destacada em [39] assim: *"requisitos funcionais descrevem o que o sistema deveria fazer e os não-funcionais descrevem as restrições sobre como esses requisitos funcionais são implementados"*.

Sommerville, em [38], preconiza os mesmos níveis de detalhes para descrição de requisitos e adiciona outra descrição, denominada **especificação de projeto de software**, cujo intuito é detalhar mais os requisitos de sistema.

O autor afirma que apesar dessas definições simples, a distinção entre os tipos de requisitos não é muito clara, pois ao detalhar um requisito não-funcional pode-se encontrar outros requisitos claramente funcionais. Por exemplo: *"um requisito de usuário relacionado à proteção, que tem todo o aspecto de um requisito não-funcional, contudo, quando desenvolvido com mais detalhes, pode levar a outros requisitos que são claramente funcionais, como a necessidade de incluir recursos de autorização de usuários no sistema"* [38].

Com base nas classificações de requisitos analisadas, constatamos que não há um consenso sobre uma taxonomia padrão (ou modelo) adotada para categorizar requisitos de software em SI. No entanto, existem algumas taxonomias que são de uso freqüente em muitas organizações como, por exemplo, requisito funcional, requisito não-funcional e requisito de projeto.

De acordo com as definições e taxonomias estudadas nesta seção, resumimos a nossa definição de **requisito**, na perspectiva de SI assim: *requisitos especificam as **condições** exigidas para se alcançar o propósito de um SI.*

Essas condições dizem respeito às funcionalidades e às respectivas qualidades subjacentes esperadas de um software em um SI. Assim sendo, o termo requisito tem de fato, a mesma semântica que a classe principal **Regra de Negócio** representada na taxonomia de RN de software em SI mostrada na Figura 2.2. Portanto, a partir desse ponto do trabalho, os termos "RN" e "requisitos" são considerados como sinônimos.

Em relação ao tratamento dado à implementação de requisitos, alguns estágios dentro da Engenharia de Software ocupam-se desta tarefa. Esses estágios contemplam atividades que visam a conversão dos documentos de especificações de requisitos em um software [38]. A abordagem mais explorada na atualidade para atender essa tarefa de conversão é através da escrita de código específico (função ou procedimento) utilizando uma linguagem de programação. Assim, as RN são definidas e gerenciadas conjuntamente com as funções do software.

2.4.2 RN na perspectiva de Bancos de Dados

Segundo [35], *"restrição é uma declaração que expressa limites mensuráveis para um elemento ou função do sistema"*. Neste conceito podemos observar que restrições apresentam uma limitação que restringe o espaço de solução ou atuação de um elemento ou uma função em relação ao modo de atendimento dos propósitos de um SI.

É necessário observar que toda restrição é um requisito, mas nem todo requisito é uma restrição. Em outras palavras, dados os requisitos de um SI, é possível obter um subconjunto destes requisitos que impõem restrições à forma de atendimento de uma necessidade do negócio.

O fato de que restrições também são RN é demonstrado em [27]. O autor afirma que

> *"regras de negócio agem para restringir a operação de um sistema de várias maneiras, seja controlando o comportamento do sistema em algumas situações, ou garantindo que os dados variáveis sempre estarão em conformidade com as estruturas desejadas".*

Como já constatamos que os termos "RN" e "requisito" são sinônimos, e que restrições, caso existam, estão contidas em requisitos, então uma restrição também é uma RN; porém, é uma RN cuja característica inerente é ser restritiva com relação ao comportamento de um elemento do SI. Tais restrições são conhecidas na área de Bancos de Dados como **Restrições de Integridade (RI)** [10]. Estas restrições visam garantir a integridade dos dados em todos os estados da base de dados, além de assegurar que os requisitos de dados do usuário do negócio estejam representados fielmente.

Portanto, uma RI em um esquema de BD especifica uma condição que precisa ser obedecida para garantir a consistência entre o BD e o mundo real por ele representado.

Uma dada RI compromete-se em assegurar que mudanças realizadas no BD por usuários autorizados não resultem na perda da consistência dos dados [33]. Por isso, identificar essas restrições é uma tarefa importante, que o projetista deve considerar durante a fase de projeto conceitual de um esquema de BD.

O conceito de RI aparece também em linguagens de representação do conhecimento, na área de Inteligência Artificial (IA). A idéia básica é que somente certos estados da base de dados/conhecimento são considerados aceitáveis, e uma ou mais RI garantem a legalidade desses estados [31].

Algumas restrições podem ser mapeadas para o modelo de dados operacional e, portanto, são mantidas pelo próprio SGBD (Sistema Gerenciador de Bancos de Dados). Há, porém, restrições que só podem ser tratadas por programas de aplicação, porque são específicas para um determinado contexto de negócio. Por exemplo, para um BD geográfico haverá uma classificação de restrições destinadas a tratar os aspectos topológicos e geométricos dos dados [23].

A abordagem geralmente utilizada em BD para implementar as RN que restringem uma parte do mundo real modelado consiste na implementação dessas regras em esquemas de BD utilizando uma DDL (*Data Definition Language*). Esta abordagem permite que o SGBD realize todo o gerenciamento das RN [10].

É importante notar que nem todas as RN podem ser representadas como RI através de um modelo de dados de um SGBD específico. Por exemplo, o Modelo Relacional representa restrições de domínio, de unicidade e de integridade referencial, mas não oferece suporte para restrições recursivas, tais como: "nenhum animal pode ser ascendente ou descendente de si mesmo".

2.4.3 Análise Comparativa das Abordagens

De acordo com o que foi discutido nas Seções 2.4.1 e 2.4.2, há duas abordagens atuais para implementação de regras de negócio. O diagrama de classes da Figura 2.3 ilustra a correlação entre essas abordagens.

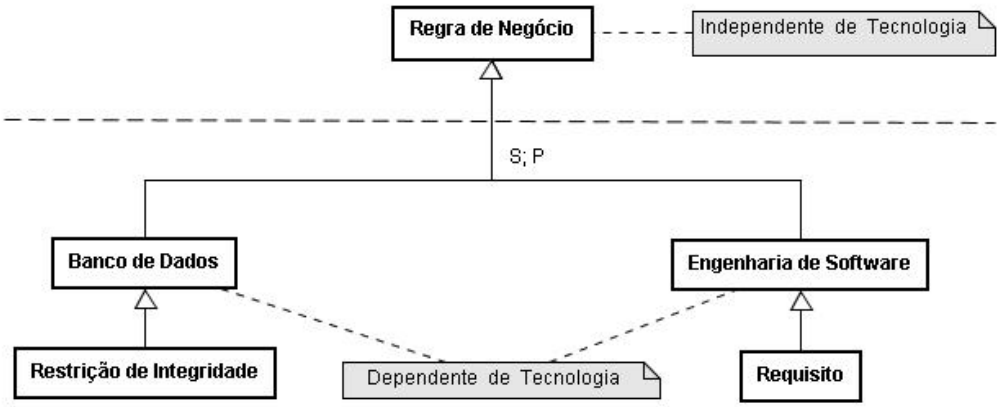

Figura 2.3: *Correlação entre RN, Requisitos e RI.*

Conforme podemos observar na Figura 2.3, uma RN pode ser expressa em ambas as abordagens de implementação, ou em nenhuma delas.

Uma das desvantagens que podemos destacar nas abordagens estudadas é a falta de **portabilidade**, principalmente no que diz respeito ao atendimento de demandas evolutivas em SI. A falta de portabilidade pode ocorrer se a declaração da RN for convertida diretamente para uma ou ambas abordagens, sem que haja um tratamento prévio que a transforme em uma declaração independente de tecnologia ou plataforma específica.

Portabilidade é um atributo de qualidade considerado importante para atender a implementação de uma mesma RN em tecnologias computacionais diferentes, mesmo que a função que implementa essa RN esteja sujeita à perda de eficiência no seu desempenho.

Outra desvantagem que poderia ocorrer está relacionada à falta de **disponibilidade** da RN. Por exemplo: caso uma RN seja implementada em uma ou mais funções (ou através de RI utilizando a DDL do padrão SQL, por exemplo), essas funções (ou o esquema gerado via DDL) podem estar em locais diferenciados no código fonte (ou em esquemas de BD diferentes). Isso acarreta dificuldades na localização, documentação e compreensão da semântica da RN, principalmente para os interessados nas regras de negócio que não são especialistas em Computação. Essas dificuldades geram falhas no processo de manutenção das regras de negócio implementadas.

Os Capítulos 4 a 7 descrevem uma abordagem para tratamento de RN que minimizam as dificuldades de manutenção. A adordagem é baseada em uma arquitetura de SI construída de acordo com o paradigma MDA, discutido no próximo capítulo.

CAPÍTULO 3

Arquitetura de Software Dirigida por Modelo

Modelar regras de negócio que expressem fielmente uma realidade, de maneira precisa e consistente, é um dos principais desafios para a Engenharia de Software. Este capítulo discute o uso da abordagem da Arquitetura Dirigida por Modelos (**MDA** - ***Model-Driven Architecture***) para enfrentar este desafio.

Os fundamentos da MDA são apresentados na Seção 3.1, com foco na criação de modelos para o desenvolvimento de software em SI.

Na Seção 3.2 explicamos as características da linguagem **OCL** que a tornam uma opção adequada para o desenvolvimento dirigido por modelos [4].

O paradigma MDA e a linguagem OCL formam a base para as propostas de tratamento de regras em SI, apresentadas nos capítulos seguintes. A Seção 3.3 correlaciona os dois conceitos: MDA e OCL.

3.1 Fundamentos de MDA

A **arquitetura dirigida por modelos** (**MDA** - ***Model-Driven Architecture***) é um *framework* baseado na UML e em outros padrões propostos pela indústria para visualização, armazenamento e troca de modelos em projetos de software [21].

MDA foi definida e é mantida pelo OMG (*Object Management Group*) para fomentar o processo de desenvolvimento de software centrado na criação de **modelos** do sistema, abstratos e independentes de tecnologia de implementação e de armazenamento [26].

3.1.1 Camadas de Modelagem

É importante compreender as camadas de modelagem que envolvem vários padrões definidos pelo OMG, inclusive MDA. São quatro camadas de modelagem: MO, M1, M2 e M3. A Figura 3.1 ilustra o relacionamento entre essas camadas.

A Camada M0 representa as instâncias reais de um sistema em execução. Por exemplo: durante a modelagem de um negócio (contexto de agronegócio, por exemplo),

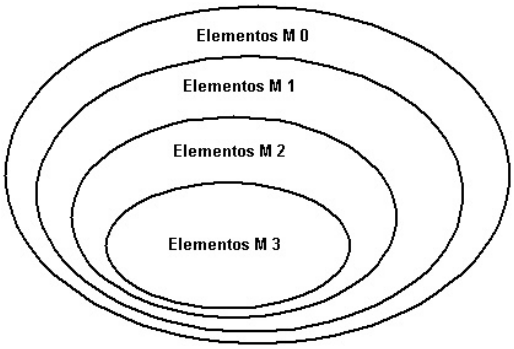

Figura 3.1: *Relacionamento entre as camadas de modelagem (adaptado de [21]).*

as instâncias M0 são os termos do negócio, tais como "propriedade rural e animais"; enquanto que, durante a modelagem de um software para atender esse negócio, as instâncias são as representações reais desses termos, tais como "o programa que controla as faturas das propriedades rurais e as informações sobre histórico dos animais".

A Camada M1 representa o modelo do sistema; neste caso, são os modelos desenhados através da UML. Esta camada visa especificar os elementos da camada M0. Por exemplo, a classe "Proprietário Rural" é definida com as propriedades nome, data de nascimento e endereço.

A Camada M2 representa o modelo do modelo. Um elemento desta camada especifica os elementos da camada M1. O modelo que atua nesta camada é denominado de metamodelo. Quando um projetista constrói um metamodelo, ele está definindo uma linguagem de modelagem para escrever modelos (por exemplo, UML).

A Camada M3 é a camada mais alta, chamada de "metameta". Ela define os conceitos necessários para os elementos da camada M2. O modelo que atua nesta camada é denominado de metametamodelo. Um exemplo de linguagem usada nessa camada é MOF (*Meta Object Facility*), que também é um padrão OMG, cujo propósito é servir como base para construção dos metamodelos de outras linguagens, tais como UML e OCL.

A Figura 3.2 apresenta uma visão geral do relacionamento entre as camadas (de M0 até M3), citando um exemplo em cada camada.

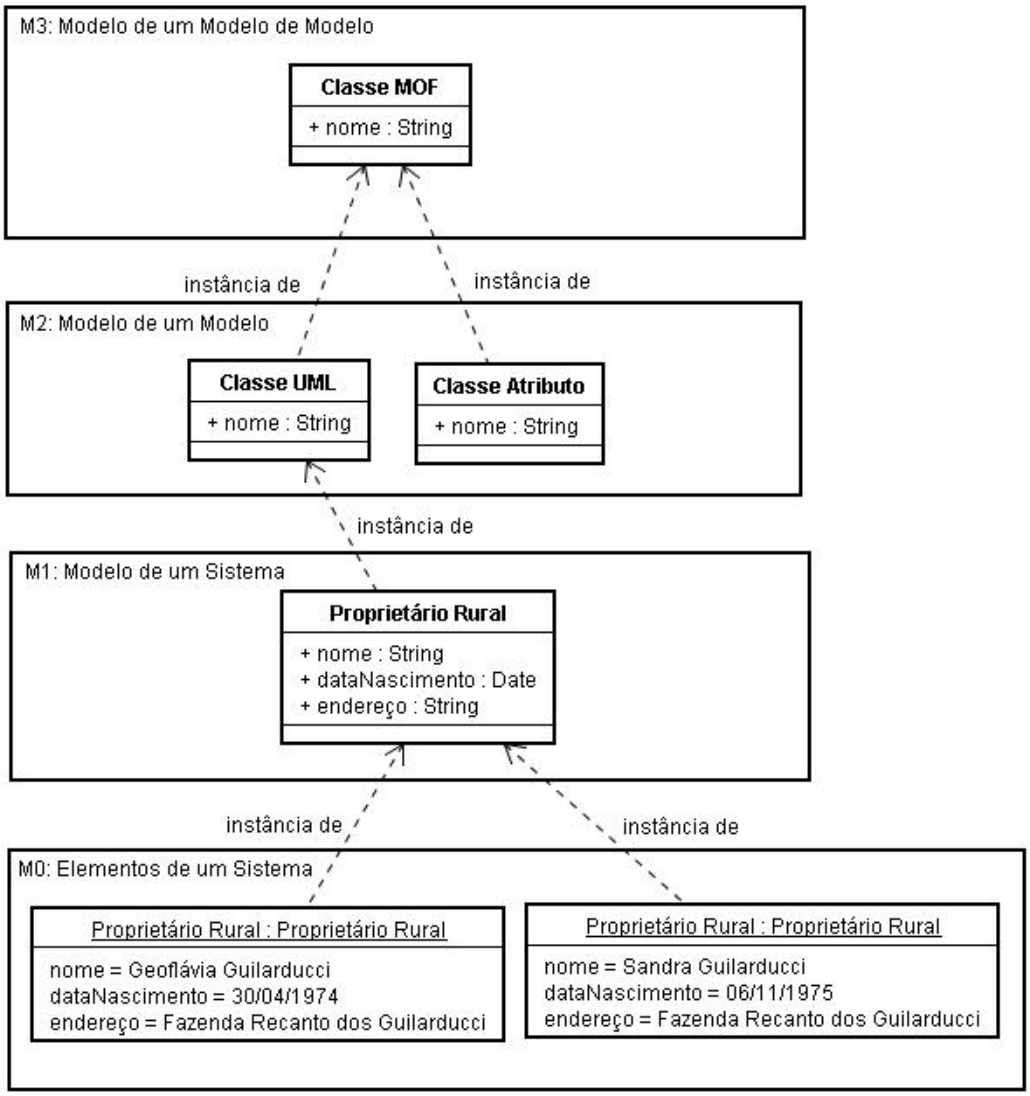

Figura 3.2: *Uma visão geral das camadas de modelagem (adaptado de [21]).*

3.1.2 *Framework* MDA

Tradicionalmente modelos têm sido usados no desenvolvimento de software somente para fins de documentação e comunicação; contrastando com este cenário, os modelos são os artefatos-chave na MDA [45].

MDA direciona seus esforços para a separação da especificação da operação de um sistema dos detalhes de como o sistema usa as capacidades contidas na sua **plataforma** [26]; a idéia básica é separar a essência do sistema da implementação do sistema.

Uma plataforma é um conjunto de sub-sistemas e tecnologias que disponibiliza funcionalidades através de interfaces e padrões de uso específicos. Implementações de CORBA (*Common Object Request Broker Architeture*) são exemplos de um tipo de pla-

taforma que estabelece uma padronização para troca de dados entre sistemas distribuídos heterogêneos. Outros exemplos de plataforma incluem sistemas gerenciadores de Bancos de Dados e serviços de comunicação na Internet.

O *framework* MDA é composto pelos seguintes elementos principais: modelos (PIM - *Plataform Independent Model* e PSM - *Plataform Specific Model*); linguagens; definições de transformações; e ferramentas que executam transformações. Uma visão geral desses elementos é ilustrada na Figura 3.3.

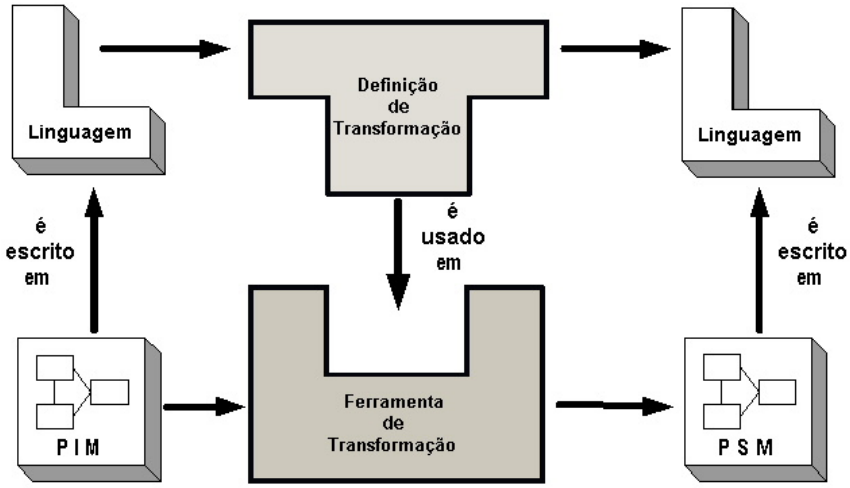

Figura 3.3: *Visão básica do framework MDA (adaptado de [21]).*

O elemento **PIM** é um modelo elaborado em alto nível de abstração, independente de tecnologia de implementação. Este modelo não deve conter nenhuma característica de uma plataforma específica.

O **PSM** é o modelo gerado após a transformação de um PIM em um ou mais modelos que são direcionados para uma tecnologia de implementação específica.

A definição da **Linguagem** é importante para determinar a sintaxe e a semântica para escrita dos modelos. A linguagem deve ser bem definida, de forma que facilite a interpretação automatizada por computador.

A **Definição de Transformação** é dada por um conjunto de regras cujo objetivo é definir como um modelo escrito em uma determinada linguagem pode ser transformado em outro modelo em uma outra linguagem.

A **Ferramenta de Transformação** é um software que permite a execução automática de uma definição de transformação.

O *framework* MDA propõe um processo de desenvolvimento em três etapas, conforme mostra a Figura 3.4 [21, 46]:

A primeira etapa constitui na construção de um PIM em uma determinada linguagem.

Na segunda etapa ocorre a transformação do PIM em um ou mais PSMs através de uma ferramenta de transformação que obedece uma definição de transformação dada.

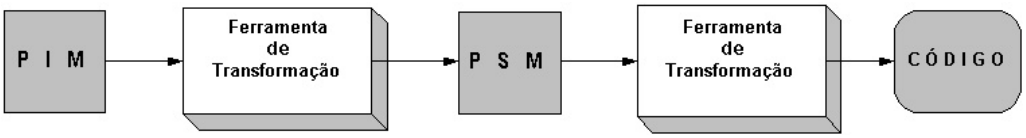

Figura 3.4: *Os três passos principais do processo de desenvolvimento MDA (adaptado de [21]).*

Na terceira e última etapa há uma transformação dos PSMs em código processável por computador. Neste passo também usa-se uma ferramenta de transformação obedecendo uma determinada definição de transformação.

O processo de desenvolvimento MDA tem uma certa semelhança com o processo de desenvolvimento tradicional. A grande diferença está na etapa de transformação de um modelo para outro modelo.

No caso do processo tradicional, esta transformação normalmente ocorre de forma manual, enquanto que as transformações na MDA são sempre executadas automaticamente por ferramentas. Porém, pelo fato de ainda ser considerada uma tecnologia emergente, nem todas as linguagens e ferramentas estão preparadas para alcançar efetivamente o propósito da MDA. Em conseqüência deste fato, poderá ocorrer uma necessidade de mudança manual dentro do código gerado [21].

Em [21, 26, 46] são identificados os **benefícios** que uma aplicação baseada em MDA pode alcançar: portabilidade, produtividade, interoperabilidade, manutenção e documentação.

Dentro da MDA a **portabilidade** é alcançada através do desenvolvimento focado nos modelos PIM, os quais podem ser usados para gerar vários modelos PSM para plataformas diferentes. E na portabilidade depende da disponibilidade das ferramentas de transformação que automatizam a conversão de PIM para PSM.

A **produtividade** trazida pela MDA permite que as equipes de profissionais de TI envolvidos nos processos de desenvolvimento e manutenção do software utilizem linguagens e conceitos comuns entre eles. Isso favorece também a comunicação e integração entre essas equipes. Além disso, um ganho significativo de produtividade pode ser obtido pelo uso de ferramentas que automatizam completamente a geração de código a partir de um PSM.

A **interoperabilidade** entre plataformas pode ser cumprida pelas ferramentas que, além de gerar os modelos PSM, precisam também fazer a ligação entre essas plataformas e possibilitar a interface com outras plataformas.

A facilidade na **manutenção** e **documentação** é obtida pelas informações sobre a aplicação incorporadas nos modelos PIM. Esses modelos cumprem a função de documentação de alto nível de abstração e a construção desses modelos exige menos esforço que a escrita do código fonte. Além disso, esses modelos não são desconsiderados após a

escrita. As mudanças feitas no sistema serão feitas mudando o PIM e reconstruindo, respectivamente, o PSM e o código executável. Nesse ponto, as ferramentas poderão manter a consistência entre PIM e PSM, e vice-versa, logo, tais mudanças serão refletidas na documentação de alto nível que permanecerá consistente com o código atual.

Com o uso da abordagem MDA, os analistas, projetistas e programadores envolvidos nas tarefas de modelagem, passam a valorizar mais a atividade de **modelagem dos requisitos de negócio** da aplicação, já que eles não precisam se preocupar com a plataforma onde serão implementados e armazenados esses requisitos [25].

Os modelos gerados pelo processo de desenvolvimento MDA precisam ser consistentes e precisos, de tal forma que facilitem a tarefa da etapa de transformação de modelos como, por exemplo, na geração de código executável. O código gerado deve estar em conformidade com a definição do seu modelo de origem, e satisfazendo restrições e regras de negócio.

3.2 Fundamentos de OCL

A linguagem de modelagem **OCL** é *"uma linguagem formal utilizada para escrever expressões em modelos UML"* [28].

OCL é também um padrão definido pelo OMG cujo propósito é complementar o entendimento dos modelos construídos com a linguagem UML. A criação do padrão OCL surgiu da lacuna diagnosticada nos diagramas em UML em expressar restrições aplicadas sobre os objetos. Geralmente, estas restrições são escritas em linguagem natural cuja prática, na maioria das vezes, resulta em restrições ambíguas.

Tipicamente, as expressões escritas em OCL especificam condições invariantes que devem ser suportadas pelo sistema, ou especificam consultas sobre objetos descritos no modelo do sistema [28].

Quando uma expressão OCL é avaliada pelo sistema, ela não deve gerar efeitos colaterais, isto é, não deve mudar o estado de nenhum objeto do sistema. Neste caso, é dito que a linguagem é *side effect free* [46].

Segundo a especificação da linguagem OCL [28], as expressões OCL podem ser usadas também para especificar restrições sobre as operações e/ou ações que, quando executadas, alteram o estado do sistema; nesse sentido, as expressões especificam o estado de mudança esperado (uma pós-condição, por exemplo).

As características principais da linguagem OCL que são relevantes para este trabalho são [28, 46]:

- **Linguagem de restrição e de consulta**
  Além de permitir expressar restrições sobre os objetos de um modelo, OCL permite

também a elaboração de expressões de consultas sobre o respectivo modelo. Isto é feito utilizando-se da navegação pelas associações entre as classes e as operações em coleções permitidas pela linguagem, e a definição da derivação de atributos ou associações.

- **Linguagem fortemente tipada**
  Esta é uma característica essencial da linguagem, pois permite que as expressões sejam verificadas durante a fase de modelagem, antes da sua execução. Assim sendo, possíveis erros podem ser removidos do modelo previamente.
- **Linguagem com fundamentação matemática**
  OCL é baseada em lógica de predicados e teoria de conjuntos, mas não faz uso de símbolos matemáticos. O rigor matemático é para garantir que o modelo gerado seja livre de ambigüidade.
- **Linguagem declarativa**
  O compromisso de uma expressão OCL é declarar *o quê* deve ser feito, e não *como* deve ser feito. Isso também permite ao projetista tomar suas decisões em um alto nível de abstração, sem precisar detalhar como será realizada uma determinada operação, por exemplo.
- **Desenvolvida para diferentes propósitos**
  OCL pode ser utilizada como uma linguagem de restrições, de consulta, e também pode ser usada para atender outros propósitos, conforme mostra a Tabela 3.1.

Conforme vimos, expressões OCL podem complementar o entendimento dos diagramas da UML. Para isso, é preciso que seja estabelecido uma ligação entre uma entidade do diagrama UML e uma expressão OCL. Esta ligação é denominada de **definição de contexto** de uma expressão OCL.

Este contexto se refere a uma entidade do modelo: classe, interface, tipo de dado ou componente, ou seja, em termos do padrão UML, são os denominados *classifiers*. Também pode ser uma operação, ou até mesmo uma instância da entidade do modelo [46]. Para definir o contexto da expressão, utiliza-se a palavra-chave *context* [28].

Dentro de uma expressão OCL, pode-se referir aos atributos e operações presentes no contexto utilizando opcionalmente a palavra-chave *self* (referência explícita do contexto de instância; refere-se à instância corrente da classe ou interface, por exemplo; possui uso similar à palavra-chave *this* da linguagem de programação Java).

Ainda dentro de uma expressão OCL é permitido navegar através das associações da classe, utilizando uma notação de pontos. O contexto define o ponto de partida da navegação.

Tabela 3.1: *Diferentes propósitos de OCL.*

Propósito	Descrição
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.
Descrição de pré e pós-condições sobre operações e métodos.	Uma pré-condição precisa ser verdadeira para permitir a execução da operação; já uma pós-condição representa as modificações que devem acontecer 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.
Descrição de condições de guarda.	Condições de guarda em diagramas de estado da UML podem ser especificadas através de uma expressão OCL. Uma condição de guarda informa a condição que deve ser atendida para que uma mudança de estado aconteça.
Especificação de regras de derivação e valores iniciais para atributos.	É importante ressaltar a diferença entre uma regra de derivação e um valor inicial: o conteúdo derivado de um atributo deve sempre conter um valor que expressa a regra de derivação, enquanto que um valor inicial (*default*) 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.
Especificação de expressões que podem complementar o entendimento de diagramas da UML.	O diagrama de classes é beneficiado com expressões OCL, através de valores iniciais, invariantes, e pré e pós-condições sobre métodos, por exemplo. Outros diagramas também podem fazer uso, tais como: 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.

3.2.1 OCL e seus Construtores

Os construtores da linguagem OCL são compostos por elementos básicos divididos em dois grupos: tipos predefinidos (tais como tipos básicos e tipos coleções); e tipos definidos pelo usuário [28, 46].

A Tabela 3.2 apresenta os tipos básicos predefinidos, conjuntamente com os respectivos valores permitidos e exemplos de algumas operações. As definições dos tipos básicos são similares àquelas encontradas na maioria das linguagens de programação conhecidas como, por exemplo, na linguagem Java.

Tabela 3.2: *Tipos básicos de OCL [28].*

Tipo	Valores Permitidos	Exemplos de Operações
Boolean	*true, false* (verdadeiro, falso)	*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()*

O conceito de coleções é muito comum em sistemas orientados a objetos, e também está presente em OCL. Existem cinco tipos de coleções, como mostra a Tabela 3.3. Toda vez que se realiza uma navegação pelas associações contidas no modelo, se obtém uma coleção; o tipo de coleção obtido irá variar de acordo com a navegação efetuada.

Tabela 3.3: *Tipos de coleções de OCL [28, 46].*

Tipo da Coleção	Características
Set	Os elementos da coleção são únicos; não há uma seqüência na recuperação dos elementos.
Bag	Podem haver elementos repetidos na coleção; não há uma seqüência na recuperação dos elementos.
Sequence	Pode haver repetição de elementos; os elementos estão em seqüência.
OrderedSet	Os elementos da coleção são únicos; os elementos estão em seqüência.
Collection	É o supertipo abstrato dos quatro tipos informados nesta tabela; é usado para definir as operações comuns para todos os tipos de coleções.

Os tipos definidos pelo usuário, presentes em um modelo orientado a objetos, também podem ser utilizados em expressões OCL, tais como: atributos e operações de tipos definidos pelo usuário, atributos e operações de classe, associações finais ou papéis, agregações e enumerações.

Além dos elementos básicos explicados, OCL disponibiliza alguns tipos de restrições ou estereótipos [28, 46], os quais analisamos como se segue.

Invariante é um tipo de 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.

Todas as expressões que são utilizadas em invariantes devem ser expressões que retornam o tipo booleano. Uma invariante sempre se refere a uma entidade do modelo (ou conforme termo do padrão UML, *classifier*). Esta entidade é também chamada de contexto da invariante.

Ao escrever uma expressão em OCL que represente uma invariante, deve-se sempre especificar o contexto ao qual a invariante se aplica. A palavra-chave *inv* identifica uma expressão OCL que representa uma invariante.

Pré e pós-condições são restrições que podem estar associadas à uma operação de um *classifier*. Tais restrições expressam as condições que uma operação deve ter para que sua execução seja correta.

Após a execução, esta operação provavelmente produzirá alguns efeitos, por exemplo, criação, alteração ou exclusão de uma ou mais instâncias de uma ou mais classes. Estes efeitos são denominados de pós-condições da operação.

As pré e pós-condições de uma operação podem ser especificadas por expressões OCL. Neste caso, o contexto será identificado pelo nome do *classifier* que contém a operação, seguido do nome da respectiva operação. 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.

Em expressões que representem pós-condições, o valor anterior de um atributo, antes da execução da operação, pode ser obtido usando o sufixo *@pre* posposto ao nome do referido atributo.

Para expressar o **resultado de uma operação de consulta**, OCL permite declarar o corpo de uma operação de consulta. Essa operação deve representar uma consulta às informações contidas no modelo.

Neste caso, a expressão OCL deve retornar o tipo de dado declarado na assinatura da operação. A palavra-chave *body* identifica a expressão OCL que representa a consulta. Opcionalmente, a palavra-chave *result* pode ser utilizada para indicar o retorno da operação.

Conforme já citamos, um dos propósitos de OCL é especificar **regras de derivação e valores iniciais** para atributos. Para isso, a expressão deve retornar um tipo que seja compatível com o tipo do atributo.

Neste caso, o contexto da expressão será o nome da classe seguido do nome do atributo e do seu tipo. As palavras-chave *init* e *derive* são utilizadas para indicar ex-

pressões que especificam valores iniciais e regras de derivação, respectivamente. Ambas expressões podem ser especificadas juntas, no mesmo contexto.

A **definição de novos atributos e operações** pode ser acrescentada ao modelo através de uma expressão OCL.

Neste caso, o contexto da expressão será o nome da classe na qual o atributo ou operação é definida. Para isso, utiliza-se a palavra-chave *def*.

3.3 Uso de OCL em MDA

Conforme definido pela essência da MDA, o processo de desenvolvimento do sistema deve focar na produção de modelos de alto nível. No entanto, também é necessário ter algum tipo de ordem e precisão em relação aos modelos criados.

Warmer e Kleppe definiram em [46] níveis de maturidade para qualificar os modelos de um sistema. Estes níveis podem ser comparados aos níveis do CMMI (*Capability Maturity Model Integration*) [34] para maturidade de processos de software.

No **nível 0** (*No Specification*) a especficação do software concentra-se somente na memória dos programadores. Neste nível, freqüentemente há conflitos de opiniões entre os programadores, e entre programadores e usuários do sistema; os programadores, normalmente, codificam os programas de forma *ad hoc*, o que impossibilita o entendimento do código na ausência desses programadores.

No **nível 1** (*Textual*) a especificação do software é escrita em um ou mais documentos em linguagem natural. Isso favorece a ocorrência de especificações ambíguas. Neste nível, as decisões dos programadores são baseadas em suas próprias interpretações sobre o que foi escrito nas especificações; há grande dificuldade em manter a especificação atualizada depois que ocorrem mudanças no código do programa.

No **nível 2** (*Text with Diagrams*) a especificação do software também provê um ou mais documentos escritos em linguagem natural, porém acrescenta vários diagramas de alto nível para ajudar na compreensão da especificação como, por exemplo, diagramas que detalham a arquitetura do sistema.

No **nível 3** (*Models with Text*) a especificação do software é formada por um conjunto de modelos, isto é, diagramas ou textos formais (texto com um significado muito específico e bem definido). A linguagem natural é usada somente para explicar alguns detalhes dos modelos. Neste nível, os modelos visam a representação real do software; a transformação de um modelo para código fonte é quase toda manual; ainda é muito difícil manter a especificação atualizada após alterações no código fonte, e o programador ainda toma decisões baseadas nas especificações, porém sem interferir na arquitetura do sistema.

No **nível 4** (*Precise Models*) a especificação do software contempla um modelo composto por uma coleção consistente e coerente de textos e/ou diagramas com significados específicos e bem definidos. A linguagem natural também é usada para explicar alguns detalhes dos modelos. Neste nível, os programadores não tomam decisões sobre o projeto do sistema; a manutenção dos modelos e a atualização do código é uma tarefa essencial e fácil; e o desenvolvimento do software é iterativo e incremental e é auxiliado pela transformação direta dos modelos para código fonte. Este é o nível que MDA objetiva com relação aos modelos criados em um processo de desenvolvimento de software.

Um modelo no **nível 5** (*Models Only*) é um modelo completo, consistente, detalhado e com uma descrição precisa do sistema. Neste nível, os modelos são eficientes para habilitar a geração completa de código fonte, dispensando ajustes no código resultante. Essa geração é transparente para o programador, ou seja, ele não precisa modificar o código internamente. Alguns textos ainda são presentes nos modelos deste nível, mas esses textos funcionam como comentários no código fonte. É importante notar que um modelo no nível 5 ainda não é viável com a tecnologia atual.

Os diagramas da UML não possuem toda a capacidade necessária para representar um modelo completo, consistente, formal e sem ambigüidades, que permita seu uso e seu entendimento por ferramentas de transformação, como é exigido pelo nível 4 e pelos fundamentos de MDA [46].

Visando atingir o rigor necessário, OCL é uma ferramenta que disponibiliza os recursos que permitem tornar um modelo livre de ambigüidades, acrescentando a formalidade requerida para o uso das ferramentas que irão realizar as transformações do PIM para os PSMs e, em seguida, para um código executável por computador [21, 46].

Além de adicionar mais precisão aos modelos de um sistema, OCL também pode ser usada na definição das transformações de modelos, ou seja, para especificar como deve ser feito o mapeamento dos elementos de um modelo PIM para os elementos de um modelo PSM. Eventualmente, essas transformações devem ser aplicadas somente sobre determinadas condições, e essas condições também podem ser expressam em OCL [21].

OCL é considerada uma linguagem pequena, porém um ingrediente importante dentro dos propósitos da MDA [46]. O papel da OCL em MDA é ainda mais relevante pelo fato da linguagem ser padronizada pelo OMG, a mesma organização que vem especificando e divulgando a MDA, assim como, proporcionando facilidades de integração entre as diversas tecnologias orientadas a objetos.

O próximo capítulo apresenta nossa proposta de utilização dos conceitos discutidos neste capítulo com o objetivo de implementar regras de negócio em Sistemas de Informação.

CAPÍTULO 4

Tratamento de Regras de Negócio

Este capítulo propõe uma abordagem para tratamento de regras de negócio abrangendo as atividades de especificação, modelagem e realização (implantação e gerenciamento) dessas regras em Sistemas de Informação. Uma versão preliminar desta proposta foi apresentada em [15, 16].

Na Seção 4.1 contextualizamos o problema principal abordado pela proposta, através de uma análise crítica sobre um *framework* para construção de Sistemas de Informação. Este *framework* é baseado no meta-modelo de objetos definido em [7].

Na Seção 4.2 apresentamos uma visão geral da proposta deste trabalho, sintetizada em um ciclo de vida para o tratamento de regras de negócio em SI.

Finalmente, na Seção 4.3 descrevemos o cenário utilizado para validar nossa proposta de tratamento de regras de negócio. Este cenário compreende um Sistema de Informação complexo na área de agronegócio.

4.1 Contextualização do Problema

A Engenharia de Software tem desenvolvido diversas técnicas para construção de software. Uma técnica bastante referenciada propõe o uso de *frameworks* [11].

Um *framework* pode conter componentes de software prontos ou semi-prontos que podem ser instanciados no desenvolvimento do software. Estes componentes podem ser de natureza abstrata como, por exemplo, um diagrama de classes que represente o projeto arquitetural (*design*) de um software, ou de natureza concreta como, por exemplo, códigos fonte implementados em uma linguagem de programação. Em ambas naturezas, o objetivo do *framework* é promover a reutilização.

Esta reutilização pode ocorrer não apenas através do aproveitamento de diagramas ou trechos de código de programas, mas também através da aplicação de boas práticas de processos e do uso de modelos ou padrões durante as fases de desenvolvimento do software.

A tecnologia de *frameworks* possibilita a reutilização e a personalização de uma implementação particular em aplicações de contextos diferentes, além de proporcionar

redução do tempo de desenvolvimento e da complexidade de implementação de um software.

Quanto ao contexto de utilização, a taxonomia proposta em [11] identifica três principais classes de *frameworks*:

1. *Frameworks* de infraestrutura de sistema: compatíveis com o desenvolvimento de interfaces gráficas, sistemas operacionais e compiladores, por exemplo;

2. *Frameworks* de integração com *middleware*: suportam a integração de aplicações distribuídas, a comunicação e a troca de informações entre componentes de arquiteturas iguais ou diferentes (como CORBA e Java Beans, por exemplo); e

3. *Frameworks* de aplicações: se ocupam de domínios específicos de aplicações, tais como sistemas de educacão, de telecomunicações, administrativos e financeiros, por exemplo.

Dentre os esforços para ajudar na criação de *frameworks* de aplicação, destaca-se a abordagem MDA, cujo princípio é a criação de modelos que orientam o desenvolvimento de software, conforme explica o Capítulo 3.

O *framework* de aplicação corporativa proposto em [7] foi desenvolvido com base nos princípios da MDA. No entanto, o *framework* não define um componente para tratamento das regras de negócio do software, o que dificulta o desenvolvimento de uma aplicação corporativa, isto é, um SI, com base neste *framework*.

A Figura 4.1 apresenta uma visão geral da arquitetura do *framework* proposto em [7]. Esta arquitetura representa as camadas (ou pacotes) principais que implementam fisicamente o *framework* proposto em [7]. Cada camada possui responsabilidades bem definidas:

- **Camada Interface** implementa os objetos de apresentação dos dados na interface do usuário.
- **Camada Interface Aplicação** realiza a delegação das tarefas entre as camadas Interface e Aplicação e faz uma preparação dos dados que serão apresentados na interface do usuário.
- **Camada Aplicação** é responsável pelo controle das operações do negócio (ou seja, as transações) e prepara os dados recebidos da Camada Interface Aplicação para as camadas Negócio e Persistência.
- **Camada Negócio** é responsável pela implementação das operações do negócio e da consistência dos dados, isto é, da verificação das regras de negócio.
- **Camada Metadados** implementa a estrutura do modelo conceitual denominado Modelo de Meta-Objetos (MMO).

- **Camada Persistência** efetiva a persistência dos dados em um SGBD.

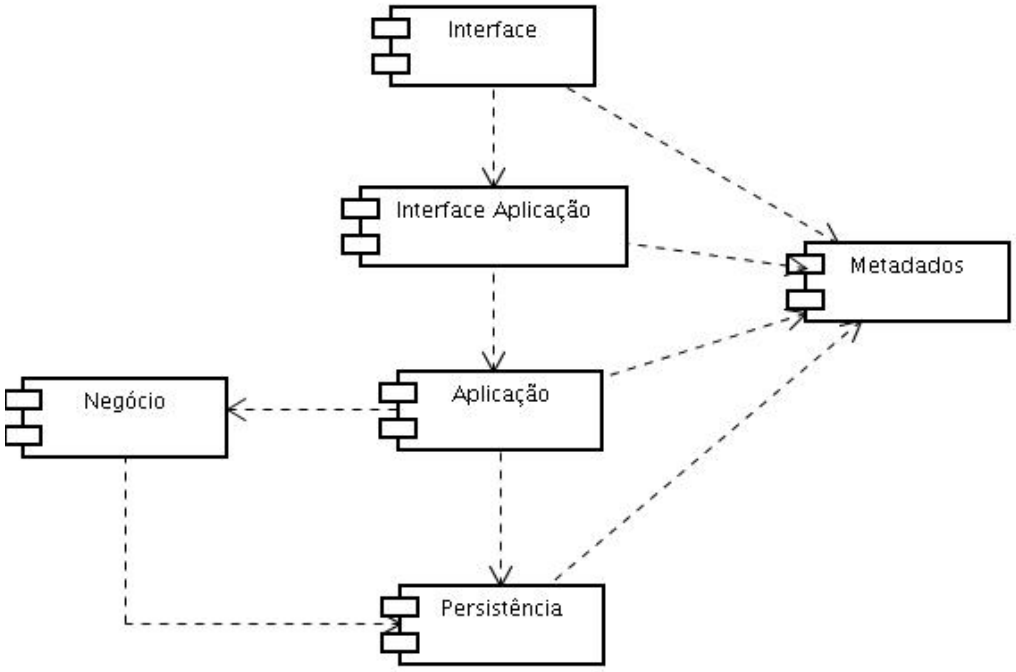

Figura 4.1: *Visão geral da arquitetura do framework proposto em [7].*

4.1.1 Modelo de Meta-Objetos

O sistema para tratameto de regras de negócio proposto neste trabalho está baseado no **Modelo de Meta-Objetos (MMO)** introduzido em [7].

O MMO é composto por um conjunto de objetos complexos que representam metadados típicos de Sistemas de Informação. O MMO está preparado para representar multi-domínios de software dentro da categoria de Sistema de Informação Empresarial (ou Corporativo).

Em outras palavras, o MMO é um modelo de representação de conceitos para um *framework* de aplicação corporativo. Este *framework* define, basicamente, um modelo genérico para informações sobre o negócio.

O software produzido com base no MMO é provido de um **editor de metadados**, sendo que o MMO é o padrão de metadados do sistema.

A idéia do MMO é relevante dentro da perspectiva de SI, haja vista que a etapa de modelagem conceitual do domínio do negócio pode resultar em um conjunto muito grande de entidades do negócio, caracterizadas por seus atributos e associações. Para implementar a manipulação dessas entidades é comum, na etapa de construção do software, codificar um programa individual para manipular cada entidade do negócio.

Isto gera um trabalho bastante repetitivo e dispendioso, pois constata-se na prática que a manutenção dessas entidades possui características muito comuns.

Para exemplificar as características comuns na manutenção das entidades do negócio, temos as operações de manipulação sobre os dados das entidades do negócio, tais como, inserção, alteração, exclusão e consulta. Outra característica comum é a tarefa de garantir a integridade dos dados através de restrições de integridade como, por exemplo, restrições de domínio e de unicidade.

A necessidade de manter muitas entidades de um determinado domínio de negócio em relação ao tratamento da generalidade das operações de manutenção das entidades e à garantia que as restrições de integridade não sejam violadas ainda é um problema para o desenvolvimento de software. O MMO busca resolver este problema.

De acordo com o processo de desenvolvimento MDA (apresentado na Seção 3.1), o primeiro passo é construir um PIM em uma determinada linguagem. O MMO é a linguagem adotada no *framework* adotado neste trabalho. O objetivo principal do MMO é representar as propriedades das entidades de uma determinada realidade de um negócio.

Os principais construtores disponíveis no MMO são: entidade (ou classe), relacionamento (ou associação), atributos, papel, cardinalidade de relacionamento, especialização, generalização e agregação.

A Figura 4.2 apresenta a visão conceitual do MMO, e o Apêndice B apresenta o dicionário de dados que descreve este modelo.

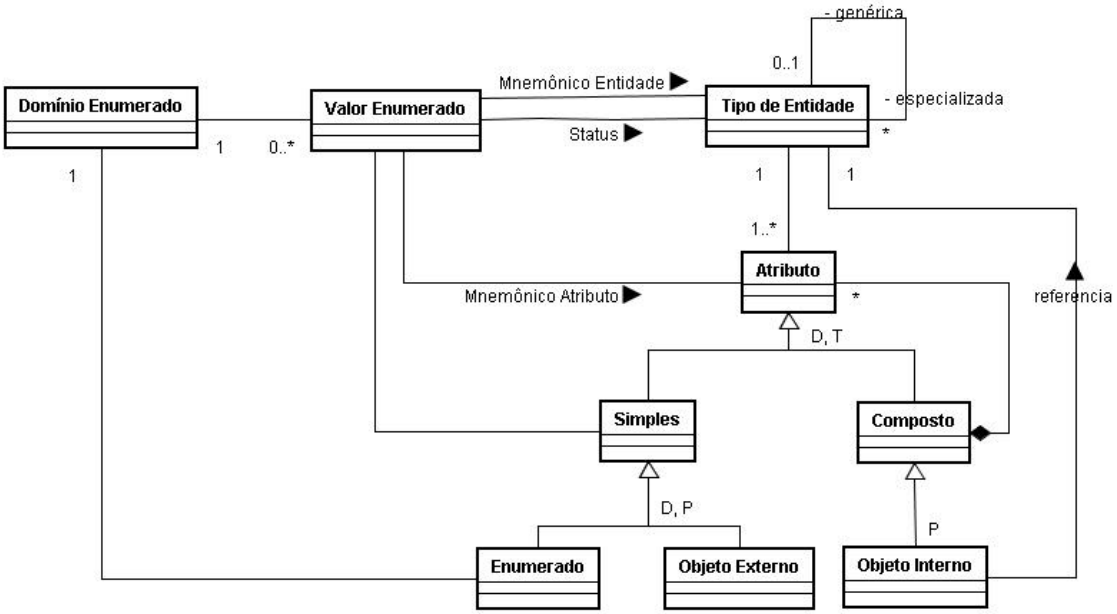

Figura 4.2: *Modelo conceitual do MMO (adaptado de [7]).*

Conforme o padrão de camadas de modelagem definido pelo OMG (descrito na

Seção 3.1), o MMO é um modelo que se enquadra na Camada M2. As instâncias de M1 geradas a partir do MMO são modelos de um Sistema de Informação (isto é, modelos M1).

Com o MMO, a tarefa do projetista se concentra em criar modelos que representam os metadados das entidades do negócio através do editor de metadados. O *framework* faz uso desses metadados e permite a manipulação das entidades de negócio pelo usuário da aplicação via interface gráfica.

O MMO prevê o tratamento de algumas RN. A Tabela 4.1 mostra a cobertura do MMO em relação à taxonomia de RN proposta na Seção 2.3.

Tabela 4.1: *Suporte a regras no MMO.*

Tipo de regra	Forma de representação no MMO
Domínio	Tratada através do atributo Tipo Dominio da entidade Atributo e da entidade Valor Enumerado. Tipo Dominio implementa o conceito de tipo de dado.
Unicidade de Valor	Tratada através dos atributos Mnemônico (identificador lógico), EhUnico e EhParteChave da entidade Atributo.
Cardinalidade Mínima	Tratada através do atributo Cardinalidade Mínima da entidade Atributo.
Cardinalidade Máxima	Tratada através do atributo Cardinalidade Máxima da entidade Atributo.
Monoconjunto e Multiconjunto	Tratada pelo tipo do atributo Objeto Interno ou pela combinação dos atributos Cardinalidade Mínima e Cardinalidade Máxima da entidade Atributo.
Assertiva de Ação e Derivação	Essas regras não são tratadas no MMO.

As principais deficiências na modelagem de regras no MMO dizem respeito às RN do tipo Assertiva de Ação e do tipo Derivação.

Estes tipos de regras concentram uma grande parte das declarações sobre os dados e sobre o comportamento do negócio de uma organização, e por isso, são regras importantes para o componente do ativo do negócio de uma organização [41]. Portanto, tais regras deveriam ser explicitamente tratadas no *framework*. Além disso, o *framework* apresenta os seguintes pontos negativos:

- Nenhuma especificação formal, declarativa, estruturada ou padronizada é usada para expressar RN dos tipos Assertiva de Ação e Derivação.
- Não há como expressar RN que envolve mais de uma entidade do negócio.
- Não é possível identificar o contexto da regra no nível de entidade. Isto torna difícil a compreensão da ligação da regra com o modelo.
- Não é possível adicionar restrições ao meta-objeto Tipo de Entidade.

- Não é possível relacionar mais de uma regra a um atributo do meta-objeto Atributo.
- Não é possível rastrear a regra e tampouco manter um completo repositório de regras que possa ser consultado pelos proprietários do negócio.
- Não é possível representar as regras no modelo PIM.

Assim, sob o ponto de vista da taxonomia proposta na Seção 2.3, o MMO trata assertivas estruturais, porém não trata assertivas de ação e derivação. Observamos também a ausência de uma metodologia para o tratamento das regras de negócio no *framework*.

Com o objetivo de contribuir com o uso prático do paradigma de regras de negócio combinado com as diretrizes da MDA, este trabalho apresenta uma proposta para solucionar os problemas aqui identificados.

4.2 Especificação da Proposta

A construção de software envolve a especificação de dados e funções (ou processos) que manipulam esses dados. Tradicionalmente, esta especificação de dados e funções é feita, respectivamente, com base em técnicas da área de Bancos de Dados e de Engenharia de Software.

No entanto, a utilização de técnicas isoladas não permite um tratamento uniforme para as RN. Na prática estas técnicas separadas não têm conseguido atender a demanda crescente de evolução dos SI, especialmente no que diz respeito ao componente de software de um SI. Essa demanda ocorre por diversos motivos como, por exemplo, mudanças nas legislações, alterações de costumes e padrões, e principalmente, na adaptação do software às novas tecnologias.

Em conseqüência dessa demanda, muitos projetos de software fracassam por não atenderem plenamente às necessidades de informação de uma organização. Grande parte dessas falhas envolvem requisitos do software [22, 30].

A modelagem de sistemas baseada em RN visa diminuir os problemas relacionados aos requisitos no processo de desenvolvimento de software [6, 32, 40, 47].

O tratamento de RN em SI proposto neste trabalho está baseado em um paradigma que preconiza a divisão do software em três componentes: dados, funções e regras de negócio [6, 40, 47, 48]; além disso, nossa proposta concentra suas contribuições na combinação das duas abordagens atuais para implementação de regras de negócio estudadas na Seção 2.4: através da codificação de funções em linguagens de programação e através da criação de restrições de integridade em esquemas de bancos de dados.

A proposta para tratamento das RN segue uma metodologia adaptada de [32]. A adaptação consiste na representação de uma RN em dois níveis de abstração, análise e projeto, sendo que no nível de projeto a representação será esboçada somente através

das camadas conceitual e interna. A camada externa não é considerada neste trabalho, pois optamos por não especificar a RN em diferentes formas de representação como, por exemplo, através de uma tabela de decisão, de um DFD ou de uma lista de eventos externos.

Ambos os níveis direcionam esforços para a especificação e/ou modelagem da regra, mas no nível de projeto, especificamente na camada interna, o esforço concentra-se também na realização da regra.

A Figura 4.3 resume a especificação da proposta para tratamento das RN utilizando um diagrama de atividades da UML. O diagrama demonstra o contexto da metodologia abrangendo suas principais atividades e os respectivos produtos gerados em cada atividade e, além disso, mostra a interação da metodologia com outros elementos do SI.

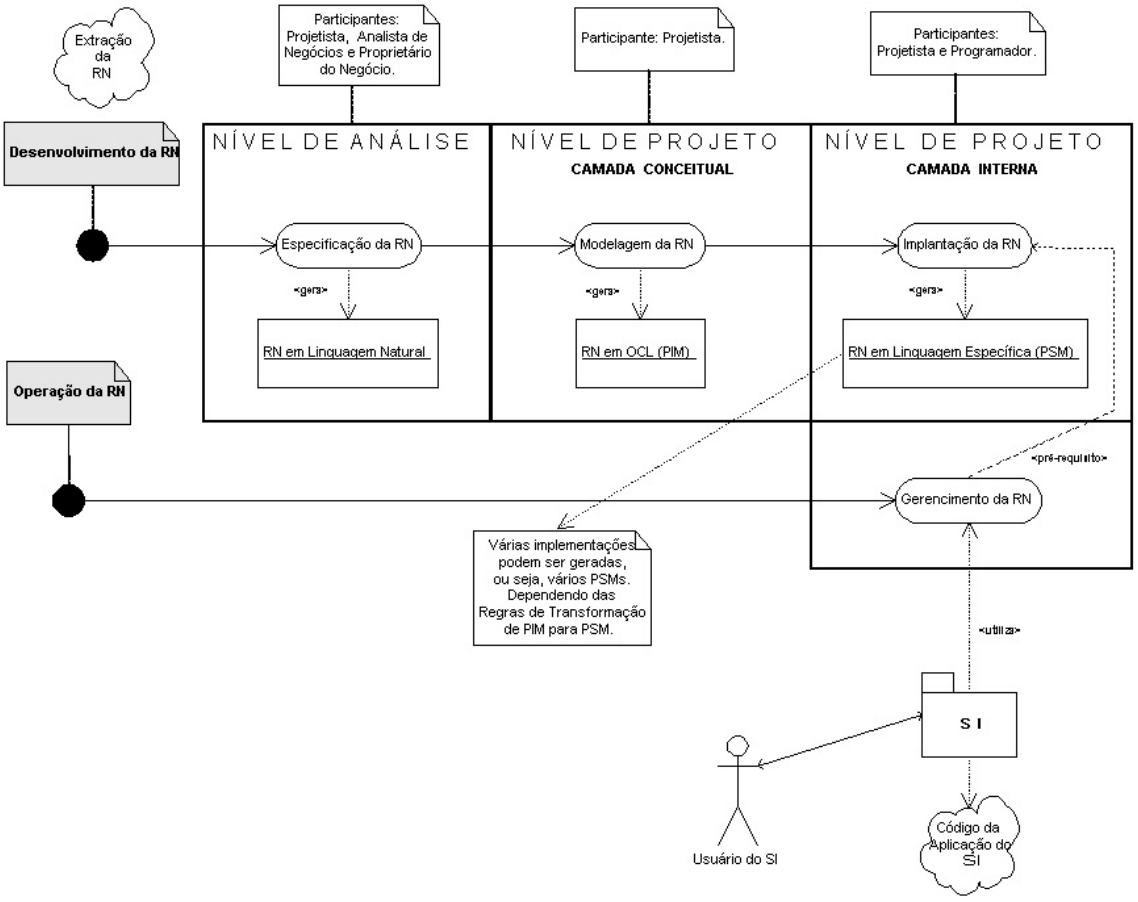

Figura 4.3: *Metodologia para tratamento das RN (adaptado de [32]).*

É importante deixar claro que não faz parte do escopo da metodologia proposta a atividade de extração (ou eliciação) de RN. Para esta atividade de extração existem diversas técnicas propostas pela Engenharia de Requisitos que podem ser empregadas para identificação de RN, tais como *FAST* (*Facilitaded Application Specification Techniques*), *JAD* (*Joint Application Development*), *Brainstorming*, e técnicas de entrevista [39]. O re-

sultado dessa identificação de RN é utilizado como insumo para iniciar as atividades da metodologia aqui proposta.

Os próximos capítulos descrevem o uso desta metodologia. O Capítulo 5 explica o nível de análise; o Capítulo 6 explica a camada conceitual do nível de projeto; e o Capítulo 7 explica a camada interna do nível de projeto. Os exemplos mostrados ao final de cada um destes capítulos estão de acordo com o estudo de caso descrito na próxima seção.

A proposta descrita neste trabalho foi utilizada no desenvolvimento do projeto *SIACorte*, que visa o desenvolvimento de infra-estrutura de Informática para otimização da produção de gado de corte [7]. Um dos produtos deste projeto foi um software produzido com base no *framework* de aplicação corporativa descrito na Seção 4.1. Uma breve introdução ao Sistema de Informação construído neste projeto é apresentada a seguir.

4.3 Um Estudo de Caso - Sistema de Informação para Agronegócio

A proposta descrita neste trabalho foi utilizada no desenvolvimento do projeto ***SIACorte*** (software produzido com base no *framework* de aplicação corporativa proposto em [7]). Uma breve introdução a este projeto é apresentada a seguir.

Apesar da importância do agronegócio na economia mundial, particularmente em países como o Brasil, há pouca utilização de tecnologias que permitam maior eficiência nos diversos elos das cadeias produtivas e a integração destes elos.

Durante os anos de 2004 à 2007, um projeto de pesquisa foi executado com a missão de desenvolver e disponibilizar soluções integradas em tecnologias e serviços para beneficiar o agronegócio. Este projeto foi apoiado pelo CNPq [7] e desenvolveu uma Arquitetura de Soluções Integradas para Agronegócio, ou simplesmente **Arquitetura SIA**. O módulo ***SIACorte*** é um dos componentes desta arquitetura e encontra-se em fase final de desenvolvimento.

O objetivo principal do ***SIACorte*** é permitir a escrituração zootécnica para gado de corte, abrangendo o atendimento dos ciclos produtivos de cria, recria e engorda; são consideradas nestes ciclos as características específicas de rebanhos comerciais e de elite, incluindo os diversos tipos de manejo (nutricional, sanitário e reprodutivo) e as transações comerciais envolvendo animais (venda, aquisição e alienação).

O ***SIACorte*** contempla diversas funcionalidades, entre as quais instanciamos um cenário para validar empiricamente nossa proposta para tratamento de RN. Este cenário envolve o manejo de parto de animais.

A Figura 4.4 apresenta um modelo conceitual baseado no cenário de manejo de parto de animal, o qual contempla alguns dos principais conceitos do negócio envolvidos neste tipo de SI. Os nomes dos conceitos são exatamente os nomes que constam nos metadados do projeto ***SIACorte*** [7]. Para melhor clareza, a Tabela 4.2 apresenta o significado desses nomes.

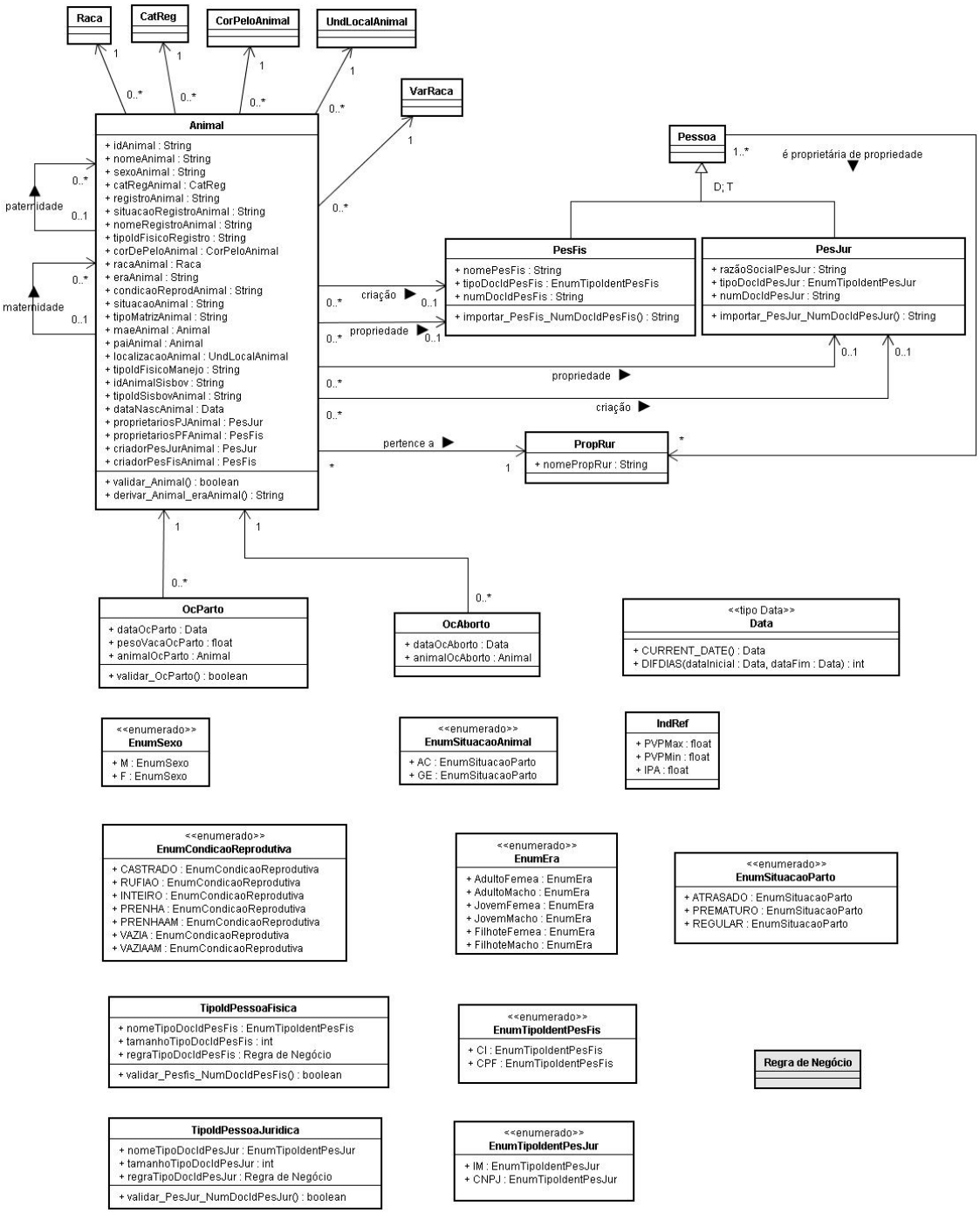

Figura 4.4: *Conceitos envolvidos no manejo de parto de animal.*

Tabela 4.2: *Definições de conceitos de negócio representados na Figura 4.4.*

Conceito do negócio	**Definição**
Animal	Bovino pertencente à propriedade rural.
CatReg	Categoria de registro de animal.
CorPeloAnimal	Cor de pelo de animal.
Data	Caracteriza o tipo data.
EnumEra	Valores para era de animal.
IndRef	Indicador de referência para parto de animal.
EnumSexo	Valores para sexo de animal.
EnumCondicaoReprodutiva	Valores para condição reprodutiva de animal.
EnumSituacaoAnimal	Valores para situação de animal.
EnumSituacaoParto	Valores para situação de parto de animal.
EnumTipoIdentPesFis	Valores para tipo de identificação de pessoa física.
EnumTipoIdentPesJur	Valores para tipo de identificação de pessoa jurídica.
OcAborto	Registro de aborto em um animal.
OcParto	Registro de um parto em um animal.
Pessoa	Pessoa física ou jurídica.
PesFis	Pessoa física.
PesJur	Pessoa jurídica.
PropRur	Propriedade rural.
Raca	Raça de animal.
TipoIdPessoaFisica	Tipo de identificador de pessoa física.
TipoIdPessoaJuridica	Tipo de identificador de pessoa jurídica.
UndLocalAnimal	Unidade de localização de animal.
VarRaca	Variedade de raça de animal.

No cenário modelado na Figura 4.4, o nascimento de um animal em uma propriedade rural é um fato de extrema relevância, registrado através de uma ocorrência de parto. Algumas informações são importantes para completar o registro deste fato, tais como: data do parto, peso da vaca e situação do parto, entre outras informações correlacionadas.

A propriedade rural é um imóvel rural que possui vários animais e, normalmente, é dividida em áreas. Cada área tem uma determinada finalidade: sede, reserva, exploração pecuária, etc.

Uma propriedade rural pode pertencer a várias pessoas. Uma pessoa que possui uma propriedade rural pode ser uma pessoa física ou jurídica (entidade reconhecida

por um país para fins de realização de atos cíveis, notadamente para atos comerciais; por exemplo: empresas privadas e órgãos do governo). Esta pessoa possui um tipo de documento de identificação que pode ser utilizado para sua identificação única.

A classe conceitual **Regra de Negócio** foi inserida no modelo conceitual da Figura 4.4 porque alguns atributos deste modelo são instâncias da classe **Regra de Negócio**, que é um dos componentes propostos para tratamento de RN em Sistemas de Informação. Esta classe será explicada da Seção 5.3.

CAPÍTULO 5

Análise de Regras de Negócio

Este capítulo explica a abordagem proposta em nossa metodologia para a análise de RN. No processo de análise o foco deve ser o entendimento do conceito especificado pela RN e, principalmente, a compreensão da regra por parte do projetista.

A Seção 5.1 apresenta as características de uma RN que devem ser identificadas e documentadas na fase de análise da RN. A seção apresenta a definição de cada característica, de forma a registrar os aspectos essenciais da RN.

A Seção 5.2 sugere um formulário padrão para sistematizar as definições e características de uma RN. O registro das RN neste formulário facilita a consulta e a atualização das definições da RN ao longo do ciclo de vida do SI.

A Seção 5.3 propõe uma forma de automação do formulário padrão proposto, uma vez que o número de regras em um SI tende a ser grande e isto dificulta o tratamento manual da documentação das RN.

Para exemplificar a aplicação das propostas deste capítulo, a Seção 5.4 apresenta exemplos de definição de RN aplicados no cenário descrito na Seção 4.3.

5.1 Definições

Uma vez que as RN tenham sido compreendidas, é necessária a formalização dessas regras, para que o conhecimento sobre elas fique preservado no SI. A fase de análise proposta na nossa metodologia tem o objetivo de formalizar as definições das regras, possibilitando o entendimento comum dessas regras por todos os interessados no SI.

Considerando que as RN já foram eliciadas, inicia-se a formalização dessas regras através do nível de análise conforme proposto pela metodologia (Figura 4.3).

O **nível de análise** tem importância fundamental no tratamento das RN, pois é nesta fase que a compreensão das regras por parte do projetista do software deve se consolidar. O objetivo principal consiste no conhecimento sobre **"o que"** é especificado pela regra de negócio. Conforme mostrado da Figura 4.3 a atividade **Especificação da**

RN é de responsabilidade do projetista do software e deve contar com a participação do analista de negócio e/ou do proprietário do negócio.

Para atingir o objetivo dessa atividade é preciso que os atributos contidos na especificação da RN, além de atenderem as necessidades de conhecimento da regra, também atendam as necessidades requeridas para representação da regra no nível de projeto. Esses atributos são:

1. **Identificação**: esta informação é importante para identificar unicamente uma RN dentro do contexto no modelo do negócio, principalmente para permitir rastrear a RN (conforme definido na Seção 2.1), e para identificar logicamente a regra em um repositório de regras.

   É preciso adotar uma nomenclatura simples e de acordo com a semântica da RN, visando um reconhecimento rápido pelos profissionais envolvidos.

   Juntamente com a identificação da RN é necessário informar o **status** da regra. Este status indica qual é a situação atual que a regra se enquadra, ou seja, em qual processo da metodologia de tratamento de RN a respectiva regra se enquadra.

2. **Tipo**: de acordo com as características e necessidades de SI, a RN deve ser classificada segundo a taxonomia proposta na Seção 2.3.

3. **Comportamento**: uma RN expressa declarações do negócio, incluindo definição ou restrição de algum comportamento (ação ou operação) do negócio. É preciso explicitar quais são os comportamentos do negócio diretamente associados com a RN ou, se for o caso, informar que a declaração da regra não expressa qualquer comportamento específico. Este é o caso típico de uma RN de Exportação (veja Seção 2.3.2) que depende do comportamento de uma Regra de Importação.

   Juntamente com esses comportamentos é importante definir o **aspecto de tempo** correspondente. Isto é feito para melhor caracterizar uma RN do tipo Assertiva de Ação. Um exemplo seria: uma determinada RN associada ao comportamento de alteração das informações de um conceito do negócio somente deve ser avaliada **depois** do respectivo comportamento de alteração.

4. **Linguagem natural**: para a modelagem da regra no nível de análise recomenda-se que a regra use declarações que sejam compreendidas pelo proprietário do negócio, ou seja, declarações de alto nível do domínio do problema.

   Isto implica que a RN deverá ser especificada na linguagem natural conhecida pelos proprietários do negócio (Português ou Francês, por exemplo).

   Caso a RN tenha mais de uma declaração, é necessário identificar essas declarações unicamente.

Essas declarações também serão utilizadas pelos projetistas do software como artefato de entrada para a realização do nível de projeto, de acordo com o processo de desenvolvimento de software adotado.

5. **Contexto proprietário**: identifica o conceito do negócio que possui (ou é proprietário de) uma RN. Isto complementa o quesito de rastreabilidade da regra. O *contexto proprietário* identifica o proprietário da RN. Por exemplo: o conceito do negócio "Propriedade Rural" possui a seguinte RN "uma propriedade rural deve ter 30% de sua área destinada à preservação ambiental"; neste caso, o contexto proprietário da RN é o conceito "Propriedade Rural".

6. **Contexto destinatário**: conforme taxonomia definida na Seção 2.3, uma RN classificada como RN de Exportação precisa especificar para qual conceito do negócio se destina essa regra. Logo, denominamos este conceito de *contexto destinatário*.

   Por exemplo: o conceito do negócio "Identificação de Proprietário Rural" possui a seguinte RN "cada instância de Identificação de Proprietário Rural possui exatamente uma regra de formação que valida a identificação de um Proprietário Rural"; um exemplo desse tipo de instância (ou regra de formação) seria: CPF ou CNPJ. Nesta situação, o contexto proprietário da RN é o conceito "Identificação de Proprietário Rural" e o contexto destinatário é o conceito "Proprietário Rural". A regra de formação definida na "Identificação de Proprietário Rural" é utilizada para validar um atributo de "Proprietário Rural".

5.2 Padrão de Especificação de RN

Para melhor utilizar as definições de RN, sugerimos a adoção de alguns padrões, para especificação da regra de negócio, que é o artefato resultante da atividade **Especificação da RN**. Para registrar os atributos desse artefato, o padrão proposto consiste em um formulário composto por campos que representam as informações necessárias para a especificação da RN. Este formulário é apresentado no Código 5.1.

Código 5.1 Formulário para especificação da RN.

```
1  ---------------------------------
2  Especificação da Regra de Negócio
3  ---------------------------------
4
5  1. Identificação:              Status:
6  2. Tipo:
7  3. Comportamento:              Aspecto de Tempo:
8  4. Linguagem Natural:
9  5. Contexto Proprietário:
10 6. Contexto Destinatário:
```

Para o campo **Identificação** da RN, uma codificação foi padronizada para identificar uma RN: **<XX_YYY>**. Em **XX** deve ser informada a sigla que identifica o tipo da regra (segundo taxonomia da Seção 2.3, composto exatamente por dois caracteres maiúsculos: "RV" para regra de validação; "RD" para regra de derivação, e assim por diante. Em **YYY** deve ser informado o nome do conceito do modelo do negócio que está sujeito à RN. Para o nome do conceito do negócio não limitamos a quantidade de caracteres. Por exemplo no Código 5.2: "RV_Animal", RV informa que é uma regra de validação, e Animal indica um conceito do modelo do negócio.

Sobre o campo **status** da RN, as opções padronizadas são: Especificação (RN no nível de análise), Modelagem (RN na camada conceitual do nível de projeto; este nível será explicado no Capítulo 6) e, Implantação (RN na camada interna do nível de projeto, isto é, a RN está pronta para uso; este nível será explicado no Capítulo 7).

Várias transações podem ser realizadas no contexto de um SI de uma organização. Tais transações (ou comportamentos do negócio) refletem o comportamento esperado desse negócio. Uma parte dessas transações são genéricas e envolvem a manutenção dos conceitos do negócio como, por exemplo, inserção ou alteração de um conceito do modelo do negócio.

Diante disso, os comportamentos padronizados que devem ser informados no campo **Comportamento** são:

1. Alteração: significa que a RN está diretamente associada ao *comportamento de alteração* de um conceito do negócio.

2. Inclusão: significa que a RN está diretamente associada ao *comportamento de inclusão* de um conceito do negócio.

3. Exclusão: significa que a RN está diretamente associada ao *comportamento de exclusão* de um conceito do negócio.

É possível fazer qualquer combinação com os tipos de comportamento padronizados: alteração ou inserção; alteração ou exclusão; inclusão ou exclusão; e alteração, inclusão ou exclusão. Estas combinações indicam que a RN está associada a qualquer um dos comportamentos citados; por exemplo, uma RN com comportamento "alteração ou inserção" deve ser avaliada tanto nas transações que alteram um elemento de negócio existente quanto nas transações que inserem um novo elemento de negócio.

Além disso, o comportamento pode indicar que a RN não apresenta nenhuma associação direta com qualquer comportamento padronizado do negócio. Esta é uma característica de Regra de Exportação, que depende do comportamento de uma Regra de Importação.

Quanto ao **aspecto de tempo**, foram definidas duas opções. Na opção **ANTES** considera-se que uma RN deve ser avaliada imediatamente antes do respectivo comportamento para o qual ela foi definida. Na opção **DEPOIS** considera-se que a RN deve ser avaliada imediatamente depois do respectivo comportamento para o qual ela foi definida.

Para o campo **Linguagem natural**, a formatação das sentenças é opcional, isto é, não padronizamos um gabarito conforme sugerido na Seção 2.2. No entanto, orientamos que as declarações devem ter uma mesma conotação (ou mesma consistência) no que diz respeito ao tipo da regra, ou seja, pertencer ao mesmo tipo de regra conforme a taxonomia da Seção 2.3.

Para manter essa consistência, não é aconselhável definir em um mesmo formulário de especificação de RN (mostrado no Código 5.1), declarações de uma regra de validação e declarações de uma regra de derivação, por exemplo.

Além disso, é importante escrever as declarações em linguagem natural de maneira compreensível a todos os envolvidos a fim de evitar ambigüidades. Consideramos importante observar as recomendações em [27] para expressar melhor as regras de negócio como, por exemplo: "criar regras simples, evitar usar algum evento do negócio como sujeito da regra, e relacionar regras com os elementos visíveis no modelo".

Para o caso de haver mais de uma declaração de uma RN no campo **Linguagem natural**, a identificação única para cada declaração obedece a seguinte nomenclatura:

<identificador lógico da RN>_<número ordinal seqüencial>.

Em **<identificador lógico da RN>**, adotamos a mesma identificação da RN, porém a sigla da regra é escrita em caracteres maiúsculos. Em **<número ordinal seqüencial>**, recomendamos que a numeração siga uma ordem de prioridade, ou seja, da regra que possui maior relevância no **quesito de importância de validação** para a de menor relevância. Este quesito deve ser decidido pelo projetista da RN juntamente com os demais profissionais envolvidos, considerando principalmente a opinião do proprietário do negócio.

Para exemplificar: seja a RN "RV_Animal" vinculada ao conceito "Animal" contendo duas declarações:
1) rv_Animal_1: "a cor do pelo do animal deve ser compatível com o que foi definido na sua raça";
2) rv_Animal_2: "se o registro do animal for informado, deverá ser informada também a filiação (mãe e pai) do animal".

No exemplo anterior, a declaração "rv_Animal_1" possui maior grau de relevância do que a declaração "rv_Animal_2". Isso implica que a primeira declaração deve ser avaliada com prioridade em relação à segunda.

5.3 Forma de Registro das Regras

É importante adotar uma forma de registro eficiente para as definições de RN, pois estas definições serão consultadas e pesquisadas com muita freqüência na atividade posterior da metodologia, que é o nível de projeto da RN. Sugere-se que as definições sejam armazenadas em um banco de dados, de forma que o acesso a estas definições seja eficiente.

No estudo de caso realizado neste trabalho, as definições das RN foram armazenadas em um banco de dados modelado de acordo com o formulário de especificação da RN. Foi desenvolvida uma aplicação de cadastro eletrônico de RN, com base no mesmo *framework* adotado para o desenvolvimento do SI.

Propomos um modelo para o armazenamento (registro) da especificação da RN, ilustrado na Figura 5.1 e seu respectivo dicionário consta no Apêndice C. Neste modelo conceitual foram acrescentadas as classes Regra de Derivação, Regra de Validação e Regra de Importação, que são as mesmas classes representadas na taxonomia definida na Seção 2.3 porém, aqui, suprimimos as propriedades dessas classes.

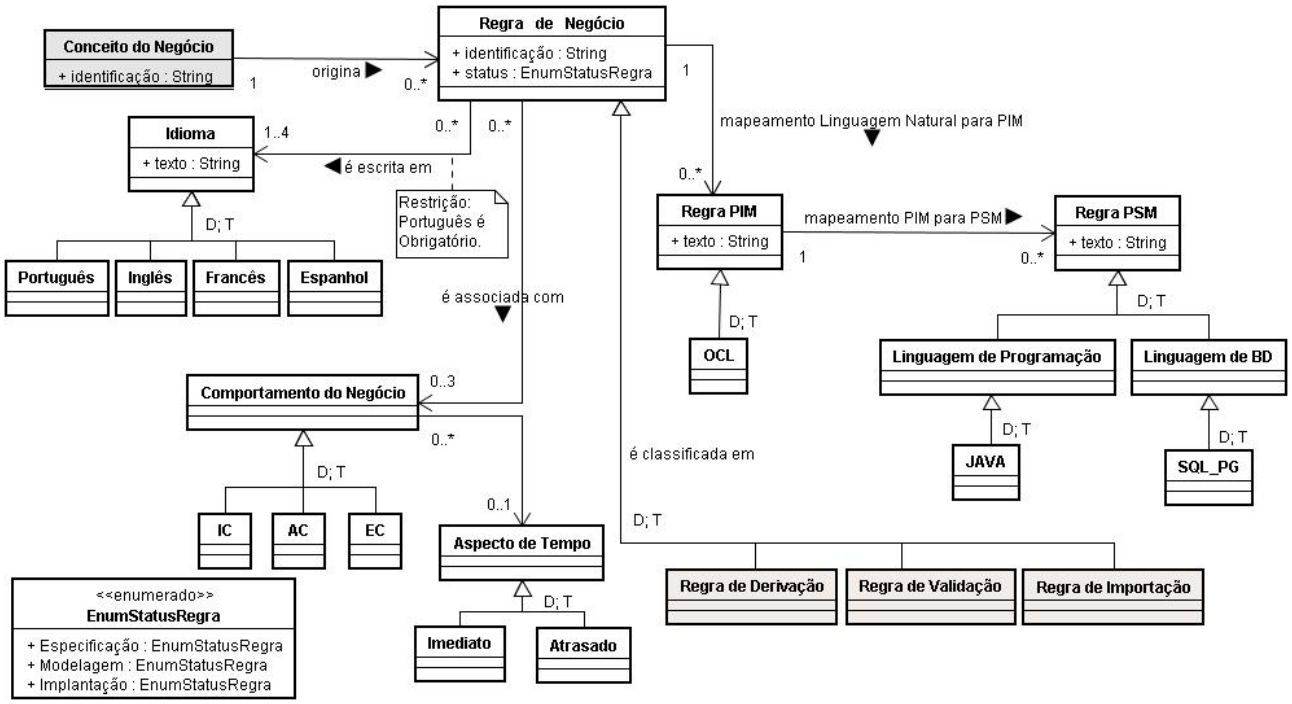

Figura 5.1: *Modelo conceitual para registro das RN.*

5.4 Exemplos

Esta seção tem a finalidade de ratificar o entendimento da representação de uma RN no nível de análise dada pela definição descrita neste capítulo, por meio da apresentação de exemplos aplicados no cenário descrito na Seção 4.3.

Os Códigos 5.2, 5.3 e 5.4 ilustram um exemplo completo de representação de uma RN do tipo Regra de Validação no nível de análise.

Código 5.2 RV da entidade "Animal- parte 1.

```
1  ---------------------------------
2  Especificação da Regra de Negócio
3  ---------------------------------
4
5  1. Identificação:       RV_Animal;  Status: Especificação.
6  2. Tipo:                Regra de Validação; Assertiva de Ação Imediata.
7  3. Comportamento:       Alteração e Inclusão; Aspecto de Tempo: ANTES.
8  4. Linguagem Natural:
9
10  rv_Animal_1_1:
11   Categoria de Registro: Deve ser informada para animais com
12   situação ANIMAL DE CATÁLOGO ou com situação ANIMAL PARA INFORMAÇÃO
13   GENEALÓGICA. Para as demais situações é opcional.
14  rv_Animal_1_2:
15   Categoria de Registro: Se informada e a situação do animal é
16   ANIMAL DE CATÁLOGO ou ANIMAL PARA INFORMAÇÃO GENEALÓGICA, o campo
17   Meio de Identificação Física do Registro do Animal não pode ser
18   preenchido.
19  rv_Animal_1_3:
20   Categoria de Registro: Se informada e a situação do animal é
21   diferente de ANIMAL DE CATÁLOGO ou ANIMAL PARA INFORMAÇÃO GENEALÓGICA,
22   então o campo Meio de Identificação Física do Registro do Animal deve
23   ser preenchido.
24
25  rv_Animal_2:
26  Cor do Pelo: Deve ser compatível com o definido na Raça do Animal.
27
28  rv_Animal_3_1:
29   Condição Reprodutiva: Se é Macho e a Era não é Bezerro, a Condição
30   Reprodutiva deve ser: Castrado, Rufião ou Inteiro.
31   Para Animal de Catálogo ou Animal para Informação Genealógica este campo
32   não deve ser preenchido.
33  rv_Animal_3_2:
34   Condição Reprodutiva: Se é Femêa e a Era não é Bezerra, a Condição
35   Reprodutiva deve ser: Prenha, Pré-Amamentando, Vazia, ou Vazia Amamentando.
36   Para Animal de Catálogo ou Animal para Informação Genealógica este campo
37   não deve ser preenchido.
38  rv_Animal_3_3:
39   Condição Reprodutiva: Se a Era é Bezerro ou Bezerra, a Condição Reprodutiva
40   não deve ser informada. Para Animal de Catálogo ou Animal para Informação
41   Genealógica este campo não deve ser preenchido.
42  rv_Animal_3_4:
43   Condição Reprodutiva: Para Animal de Catálogo ou Animal para Informação
44   Genealógica este campo não deve ser preenchido.
45
```

Código 5.3 RV da entidade "Animal- parte 2.

```
1
2   rv_Animal_4_1:
3    Tipo da Matriz: Se é Femêa e a Era é Bezerra, o Tipo da Matriz não
4    deve ser informado. Para Animal de Catálogo ou Animal para Informação
5    Genealógica este campo não deve ser preenchido.
6   rv_Animal_4_2:
7    Tipo da Matriz: Se é Macho, o Tipo da Matriz não deve ser informado.
8    Para Animal de Catálogo ou Animal para Informação Genealógica este campo
9    não deve ser preenchido.
10  rv_Animal_4_3:
11   Tipo da Matriz: Para Animal de Catálogo ou Animal para Informação
12   Genealógica este campo não deve ser preenchido.
13
14  rv_Animal_5:
15   Filiação: Se o Registro do Animal é informado, deve ser informada
16   também a Filiação: Mãe e Pai do Animal; exceto para Animais de Catálogo
17   e Animais de Informação Genealógica.
18
19  rv_Animal_6:
20   Localização: Deve ser informada a Localização do Animal, exceto quando
21   é Animal de Catálogo ou Animal para Informação
22   Genealógica.
23
24  rv_Animal_7:
25   Identificação do Animal: Deve ser informado o Meio de Identificação Física,
26   exceto quando é Animal de Catálogo ou Animal para
27   Informação Genealógica.
28
29  rv_Animal_8:
30   Identificação do Animal no SISBOV: Se o animal não é registrado no SISBOV,
31   o Meio de Identificação Física do Registro no SISBOV não deve ser informado.
32   Não deve ser informada para animais não pertencentes à Propriedade Rural
33   (Animal de Catálogo ou Animal para Informação Genealógica).
34
35  rv_Animal_9:
36   Registro do Animal: Se a Situação do Animal é Animal de Catálogo ou Animal
37   Para Informação Genealógica, o Meio de Identificação Física do Registro
38   não deve ser informado.
39
40  rv_Animal_10:
41   Data de Nascimento: Deve ser informada para todas as situações do Animal,
42   exceto quando é Animal para Informação Genealógica.
```

Código 5.4 RV da entidade "Animal- parte 3.

```
1
2   rv_Animal_11_1:
3    Proprietários: Todo Animal tem no mínimo um Proprietário,
4    com exceção de Animal para Informação Genealógica.
5   rv_Animal_11_2:
6    Proprietários: Todo Animal tem no mínimo um Criador,
7    com exceção de Animal para Informação Genealógica.
8
9   rv_Animal_12_1:
10   Variedade da Raça do Animal: Se a Raça do Animal tem uma Variedade
11   cadastrada, esta deve ser informada para o Animal daquela Raça.
12  rv_Animal_12_2:
13   Variedade da Raça do Animal: Deve ser compatível com o definido
14   na Raça do Animal.
15
16
17  5. Contexto Proprietário:       Animal.
18  6. Contexto Destinatário:       Animal.
```

O Código 5.5 ilustra um exemplo completo de representação de uma RN do tipo Regra de Derivação no nível de análise.

Código 5.5 RD da entidade "Animal".

```
1  --------------------------------
2  Especificação da Regra de Negócio
3  --------------------------------
4
5  1. Identificação:       RD_Animal_eraAnimal;  Status: Especificação.
6  2. Tipo:                Regra de Derivação; Derivação.
7  3. Comportamento:       Alteração e Inclusão; Aspecto de Tempo: ANTES.
8  4. Linguagem Natural:
9
10  Era do Animal (Faixa Etária):
11  A Era do Animal é dada de acordo com sexo e a idade
12  (quantidade de dias) do Animal.
13   Se o animal é Macho e sua idade é de 1 até 210 dias, então é Bezerro.
14   Se o animal é Fêmea e sua idade é de 1 até 210 dias, então é Bezerra.
15
16   Se o animal é Macho e sua idade é de 211 até 550 dias, então é Novilho.
17   Se o animal é Fêmea e sua idade é de 211 até 550 dias, então é Novilha.
18
19   Se o animal é Macho e sua idade é acima de 550 dias, então é Boi.
20   Se o animal é Fêmea e sua idade é acima de 550 dias, então é Vaca.
21
22 5. Contexto Proprietário:  Animal, especificamente o atributo eraAnimal.
23 6. Contexto Destinatário:  Animal, especificamente o atributo eraAnimal.
```

O Código 5.6 ilustra um exemplo completo de representação de uma RN do tipo RV de Exportação no nível de análise; a RI correspondente é apresentada no Código 5.7.

Código 5.6 RE da entidade "TipoIdPessoaFisica".

```
1  ---------------------------------
2  Especificação da Regra de Negócio
3  ---------------------------------
4
5  1. Identificação:       RE_NumDocIdPesFis;  Status: Especificação.
6  2. Tipo:                RV de Exportação; Assertiva de Ação Imediata.
7  3. Comportamento:       Nenhum; Aspecto de Tempo: Não se aplica.
8  4. Linguagem Natural:
9   A validação dos dígitos verificadores do CPF (Cadastro de Pessoa
10  Física) deve obedecer a regra módulo 11 e o tamanho do CPF deve
11  ser de exatamente 11 dígitos.
12 5. Contexto Proprietário:  Uma instância de TipoIdPessoaFisica.
13 6. Contexto Destinatário:  PesFis, especificamente o atributo NumDocIdPesFis.
```

Código 5.7 RI da entidade "PesFis".

```
1  ---------------------------------
2  Especificação da Regra de Negócio
3  ---------------------------------
4
5  1. Identificação:       RI_NumDocIdPesFis;  Status: Especificação.
6  2. Tipo:                Regra de Importação; Importação Validação.
7  3. Comportamento:       Alteração e Inclusão; Aspecto de Tempo: ANTES.
8  4. Linguagem Natural:
9    Importação da regra para validação do CPF.
10 5. Contexto Proprietário:  PesFis, especificamente o atributo NumDocIdPesFis.
11 6. Contexto Destinatário:  PesFis, especificamente o atributo NumDocIdPesFis.
```

Da mesma forma como foram construídas as representações de RN para as entidades "TipoIdPessoaFisica" (Código 5.6) e "PesFis" (Código 5.7), também é possível construir RN semelhantes para as entidades "TipoIdPessoaJuridica" e "PesJur". Neste caso, as RN validam o CNPJ (Cadastro Nacional de Pessoa Jurídica).

O Código 5.8 ilustra um exemplo completo de representação de uma RN do tipo Regra de Validação no nível de análise.

Código 5.8 RV da entidade "OcParto".

```
1  ---------------------------------
2  Especificação da Regra de Negócio
3  ---------------------------------
4
5  1. Identificação:       RV_OcParto;  Status: Especificação.
6  2. Tipo:                Regra de Validação; Assertiva de Ação Imediata.
7  3. Comportamento:       Alteração e Inclusão; Aspecto de Tempo: ANTES.
8  4. Linguagem Natural:
9
10  rv_OcParto_1: Uma ocorrência de parto somente deve ocorrer para animais
11  do sexo feminino.
12
13  rv_OcParto_2: Peso da vaca no momento do parto: deve ser verificado se
14  o peso da vaca está dentro dos limites definidos pelo índice de referência
15  Peso da Vaca no Parto (PVP).
16
17  rv_OcParto_3: Caso já existam ocorrências de parto ou ocorrências de
18  aborto para a fêmea selecionada, então o período entre a data da ocorrência
19  do parto que está sendo registrado e a data da ocorrência do parto ou aborto
20  imediatamente anterior deve ser igual ou superior ao período definido no
21  índice de referência Intervalo entre Partos e Abortos (IPA).
22
23  5. Contexto Proprietário:  OcParto.
24  6. Contexto Destinatário:  OcParto.
```

O próximo capítulo descreve o método proposto para projetar as RN representadas no formato proposto neste capítulo.

CAPÍTULO 6

Projeto de Regras de Negócio

Este capítulo explica o tratamento da RN na camada conceitual do nível de projeto. Esta camada descreve, em alto nível de abstração, as estruturas das RN especificadas no processo de análise, descrito no Capítulo 5.

A Seção 6.1 explica as definições da camada conceitual do nível de projeto da RN. A Seção 6.2 propõe o uso de padrões em OCL para modelar as RN que foram especificadas no nível de análise.

Para exemplificar a aplicação desses padrões, a Seção 6.3 apresenta as versões complementares dos exemplos que foram apresentados na Seção 5.4.

6.1 Definições

Conforme definido na Seção 4.2, a atividade **Modelagem da RN** é o foco da camada conceitual, cujo objetivo é modelar em alto nível de abstração as estruturas de uma RN de forma que seja possível automatizar a sua execução. Nesta fase essas estruturas são preparadas para serem utilizadas na camada interna.

Para iniciar a elaboração da modelagem da RN, recomendamos a utilização do padrão para especificação de RN que foi proposto no Capítulo 5.

A partir da compreensão da semântica da RN dada pela sua especificação, é preciso formalizar as declarações da RN de maneira independente de plataforma de implementação em conformidade com o modelo de objetos do negócio. Em outras palavras, tais declarações são usadas para gerar o esquema das regras em um modelo PIM. Para isso, a escolha da linguagem usada para modelar as declarações da RN é um ponto fundamental. Neste trabalho, adotamos a OCL, pelas características discutidas na Seção 3.2.

Como OCL permite diversas maneiras para definir regras, consideramos importante observar as diretrizes propostas em [46] para expressar as regras bem definidas. Por exemplo: escolher o contexto sabiamente; evitar expressões com navegação complexa; e, caso possível, dividir as restrições para torná-las mais fáceis de ler e escrever.

6.2 Padrões para Especificação de RN em OCL

OCL permite expressar o esquema conceitual das RN através de seus estereótipos como, por exemplo, *body, pre, post, derive* e *inv*. A linguagem também permite vários caminhos para declarar as expressões contidas nesses estereótipos. Vale lembrar que cada estereótipo tem um propósito bem definido de atuação sobre um conjunto específico de restrições.

Devido à necessidade de gerarmos regras em um modelo PIM, e este ser transformado em um modelo PSM, optamos por generalizar o comportamento das regras a fim de obter uma estrutura que facilite a tarefa de transformar uma regra-PIM em uma regra-PSM.

Portanto, o padrão adotado para escrita da regra em OCL deve seguir o formato especificado em OCL para expressar restrições sobre as operações (ou comportamentos) de um objeto do modelo.

Isto implica que as expressões OCL (que declaram as RN) devem ser escritas sobre as operações do negócio. Tais operações refletem exatamente o comportamento dos objetos de um modelo OO.

Ressaltamos ainda, três aspectos estudados neste trabalho os quais justificam o uso do padrão adotado. O primeiro aspecto foi visto na Seção 2.1 e define que as RN administram o comportamento do negócio; [49] apresenta também a seguinte definição: *"são metadados sobre operações de negócio"*. Logo, podemos também declarar uma regra em OCL no estilo de operações a partir da sua respectiva declaração em linguagem natural.

O segundo aspecto, também discutido na Seção 2.1, tem relação com os parâmetros do negócio. Tais parâmetros são considerados críticos e de influência restritiva, assim como as regras. Entendemos que podemos tratar alguns parâmetros do negócio conjuntamente com o tratamento a ser dado às regras, ou seja, podemos inseri-los na escrita da regra em OCL, conforme seja adequado ao propósito que a regra visa atender. Ainda mais, o estilo de operações em OCL permite a especificação completa de uma lista de parâmetros.

O terceiro aspecto que influenciou o padrão adotado foi analisado na Seção 2.4.1 e diz respeito a uma das práticas comumente utilizadas na Engenharia de Software para o tratamento de RN, que é através da implementação de código específico em uma linguagem de programação. Nesta prática são criadas funções específicas que realizam o tratamento das regras. Em outras palavras, são exatamente as operações sobre os objetos do negócio codificadas em uma linguagem de programação. Por isso, também faremos uso desta prática para expressar e tratar as regras no estilo de operações.

No Código 6.1 apresentamos o padrão básico para escrita da regra em OCL

(conforme [28]).

Código 6.1 Padrão básico para escrita da regra em OCL.

```
1
2  context <nomeDoContexto>::<nomeDaOperação> (
3          <nomeDoParâmetro : tipoDoParâmetro>, ...)
4     : <tipoDoRetornoDaOperação>
5
6  <tipoDoEstereótipo> : <expressões OCL ...>
```

<nomeDoContexto> é exatamente o nome de um conceito ou entidade do modelo do negócio. Todo nome do contexto em OCL deve ser iniciado com letra maiúscula, de acordo com a sintaxe da linguagem OCL.

<nomeDaOperação> é o local destinado para a definição da operação (ou comportamento) de um conceito ou entidade do negócio.

Para cada tipo de RN (conforme taxonomia apresentada na Figura 2.2) definimos uma regra de formação para o nome da operação conforme mostra a Tabela 6.1. Esta regra de formação faz uso de informações contidas no formulário de especificação da RN mostrado no Código 5.1.

Tabela 6.1: *Regra de formação para o nome da operação.*

Tipo da RN	**Aplicado em**	**Regra de Formação**	**Exemplo**
Validação e Exportação	Atributo	validar_<r1>_<r2>	validar_Animal_peso
Validação	Entidade	validar_<r3>	validar_Animal
Importação	Atributo	importar_<r1>_<r2>	importar_ProprietarioRural_nrDocumento
Derivação	Atributo	derivar_<r1>_<r4>	derivar_Animal_eraAnimal

Significados dos argumentos usados nas regras de formação.
r1 = nome do contexto destinatário da RN;
r2 = nome de identificação do atributo do contexto destinatário da RN que será validado;
r3 = nome do contexto proprietário da RN;
r4 = nome de identificação do atributo do contexto destinatário da RN que exige uma derivação.

<nomeDoParâmetro : tipoDoParâmetro> é um par contendo nome de identificação e tipo do parâmetro do negócio necessários para a declaração da RN. Um nome de identificação é o nome de um conceito ou entidade do negócio, podendo ser: nome de identificação de um atributo que se deseja validar ou que faz parte do corpo da expressão da RN (neste caso, está auxiliando de alguma forma na validação) ou nome do identificador lógico de uma entidade. Em outras palavras, um parâmentro nos informa o

quê a declaração da RN precisa internamente para complementar sua validação de forma concisa com seu propósito no negócio.

Os tipos de parâmetros podem ser os tipos primitivos da linguagem OCL ou tipos definidos pelo usuário (tipos que representam conceitos do modelo do negócio). Para nomear um parâmetro definimos também uma regra de formação:
<nomeDoContextoQuePossuiAtributo>_<nomeIdentificaçãoDoAtributo>.

A função de **<nomeDoContextoQuePossuiAtributo>** é atuar como prefixo ou contexto do parâmetro. É o local destinado para escrever o nome de identificação do conceito ou entidade do negócio que possui o atributo. Tal prefixo é importante para diferenciar nomes de atributos iguais em entidades do negócio diferentes e que possivelmente estão envolvidas em uma mesma RN.

<nomeIdentificaçãoDoAtributo> é o local para escrever o nome de identificação do atributo que se deseja atribuir uma RN para validação ou derivação.

<tipoDoRetornoDaOperação> é o tipo do retorno da operação. Por exemplo: no caso de regra de validação e de exportação, o tipo do retorno da operação será sempre *Boolean* (verdadeiro ou falso). Se o retorno da operação for verdadeiro significa que a satisfação de todas as expressões foram alcançadas, caso contrário será falso, pois nesse caso, pelo menos uma expressão não obteve a satisfação desejada.

Em **<tipoDoEstereótipo>** indica-se um estereótipo da OCL que deve ser usado para cada tipo de RN: para regras de validação e exportação, adotamos o estereótipo *post*; para regra de importação empregamos o estereótipo *body*; e para regra de derivação usamos o estereótipo *derive*.

<expressões OCL> contém as expressões em OCL de acordo com as declarações da RN informadas em "linguagem natural" na sua respectiva especificação. Recomendamos o uso da cláusula ***let*** em OCL para escrita das declarações da RN. A cláusula ***let*** permite definir uma variável que pode ser usada em uma restrição e, esta variável pode receber expressões OCL [28]. Exemplos de utilização da cláusula ***let*** serão apresentados na Seção 6.3.

A partir do padrão básico definido no Código 6.1, instanciamos outros padrões para cada tipo de RN. Os Códigos 6.2 a 6.4 demonstram esses padrões.

Código 6.2 Padrão para escrita em OCL de Regra de Validação e Regra de Exportação.

```
1
2  context <nomeConceitoDoNegócio>::validar_<nomeDoConceitoDoNegócio>
3          ( <nomeDoParâmetro : tipoDoParâmetro>, ...) : Boolean
4
5  post: <expressões OCL ...>
```

Código 6.3 Padrão para escrita em OCL de Regra de Importação.

```
1
2  context <nomeConceitoDoNegócio>::importar_
3          <nomeDoConceitoDoNegócio_nomeDoAtributoDoConceitoDoNegócio>
4          ( <nomeDoParâmetro : tipoDoParâmetro>, ...) : String
5
6  body: <expressões OCL ...>
```

Código 6.4 Padrão para escrita em OCL de Regra de Derivação.

```
1
2  context <nomeConceitoDoNegócio>::derivar_
3          <nomeDoConceitoDoNegócio_nomeDoAtributoDoConceitoDoNegócio>
4          ( <nomeDoParâmetro : tipoDoParâmetro>, ...)
5                  : <tipoDoRetornoDaDerivação>
6
7  derive: <expressões OCL ...>
```

É importante destacar que o preenchimento de cada parte dos padrões propostos deve obedecer as regras sintáticas da linguagem OCL especificadas em [28].

Conforme definido no Capítulo 5, é importante definir o aspecto de tempo correspondente ao comportamento da RN. Sobre esse aspecto o projetista da RN, ao utilizar uma das opções (**ANTES**, **DEPOIS**), deve-se modelar as expressões em OCL de forma coerente com a opção escolhida. Por exemplo: caso a opção escolhida seja ANTES, isto implica que as expressões modeladas da RN devem considerar, caso precisem, as propriedades da instância do conceito do negócio (do contexto destinatário da RN) no seu estado corrente, e não depois que as propriedades da instância são persistidas no SGBD.

Para contemplar a especificação das regras em OCL, acrescentamos um novo campo no formulário de especificação da RN proposto no Capítulo 5, denominado **Modelagem da RN**. Este campo é usado para armazenar as declarações da RN escritas em OCL, conforme os padrões aqui definidos.

6.3 Especificação de RN em OCL

Esta seção ilustra a representação de RN na camada conceitual do projeto através da extensão dos exemplos que foram apresentados na Seção 5.4. É importante lembrar que a metodologia prevê a atualização do atributo Status das especificações de RN à medida que as regras fluem nos processos definidos. Assim, os exemplos da Seção 5.4 tiveram seu Status alterado de "Especificação" para "Modelagem".

Os Códigos 6.5 até 6.10 ilustram um exemplo completo de representação de uma RN do tipo Regra de Validação no nível de projeto (camada conceitual). Esses códigos complementam os Códigos 5.2 a 5.4.

Código 6.5 RV da entidade "Animal"(versão estendida) - parte 1.

```
1  ---------------------------------
2  Especificação da Regra de Negócio
3  ---------------------------------
4
5  7. Modelagem da RN:
6  -- Linha 9 define o contexto da RN, neste caso,
7  -- o contexto proprietário é igual ao contexto destinatário (Animal).
8  -- Linhas 10 a 34 definem os parâmetros e o tipo de retorno da RN.
9  context Animal::validar_Animal(
10  animal_catRegAnimal : CatReg;
11  animal_registroAnimal : String;
12  animal_situacaoRegistroAnimal : String;
13  animal_nomeRegistroAnimal : String;
14  animal_tipoIdFisicoRegistro : String;
15  animal_corDePeloAnimal : CorPeloAnimal;
16  animal_racaAnimal : Raca;
17  animal_sexoAnimal : String;
18  animal_eraAnimal : String;
19  animal_condicaoReprodAnimal : String;
20  animal_situacaoAnimal : String;
21  animal_tipoMatrizAnimal : String;
22  animal_maeAnimal : Animal;
23  animal_paiAnimal : Animal;
24  animal_localizacaoAnimal : UndLocalAnimal;
25  animal_tipoIdFisicoManejo : String;
26  animal_idAnimalSisbov : String;
27  animal_tipoIdSisbovAnimal : String;
28  animal_dataNascAnimal : Data;
29  animal_proprietariosPJAnimal : PesJur;
30  animal_proprietariosPFAnimal : PesFis;
31  animal_criadorPesJurAnimal : PesJur;
32  animal_criadorPesFisAnimal : PesFis;
33  animal_varRacaAnimal : VarRaca
34 ) : Boolean
35 post: -- Linha 35 define o estereótipo da RN.
36 let   -- Linhas 36 a 46 descrevem expressões da RN.
37         rv_Animal_1_1 : Boolean =
38         ( if (animal_situacaoAnimal = EnumSituacaoAnimal::AC or
39               animal_situacaoAnimal = EnumSituacaoAnimal::GE)
40           then (animal_registroAnimal <> null and
41                 animal_situacaoRegistroAnimal <> null and
42                 animal_nomeRegistroAnimal <> null and
43                 animal_tipoIdFisicoRegistro <> null)
44           else (true)
45           endif
46         ),
```

Código 6.6 RV da entidade "Animal"(versão estendida) - parte 2.

```
1          -- Linhas 2 a 42 descrevem expressões da RN.
2          rv_Animal_1_2 : Boolean =
3          ( if (animal_catRegAnimal <> null) and
4                (animal_situacaoAnimal = EnumSituacaoAnimal::AC or
5                 animal_situacaoAnimal = EnumSituacaoAnimal::GE)
6            then (animal_tipoIdFisicoRegistro = null)
7            else (true)
8            endif
9          ),
10         rv_Animal_1_3 : Boolean =
11         ( if (animal_catRegAnimal <> null) and
12               (animal_situacaoAnimal <> EnumSituacaoAnimal::AC or
13                animal_situacaoAnimal <> EnumSituacaoAnimal::GE)
14           then (animal_tipoIdFisicoRegistro <> null)
15           else (true)
16           endif
17         ),
18         rv_Animal_2 : Boolean =
19         ( if (animal_corDePeloAnimal <> null)
20           then ( Raca.allInstances()->
21                    select(r1| r1.nomeRaca = animal_racaAnimal and
22                           r1.corPeloRaca->exists(c1| c1.nomeCorPelo
23                           = animal_corDePeloAnimal)
24                           )->notEmpty()
25                 )
26           else (true)
27           endif
28         ),
29         rv_Animal_3_1 : Boolean =
30         ( if ( animal_sexoAnimal = EnumSexo::M and
31                animal_eraAnimal <> EnumEra::FILHOTEMACHO and
32                (animal_situacaoAnimal <> EnumSituacaoAnimal::AC or
33                 animal_situacaoAnimal <> EnumSituacaoAnimal::GE) )
34           then  (animal_condicaoReprodAnimal
35                    = EnumCondicaoReprodutiva::CASTRADO or
36                  animal_condicaoReprodAnimal
37                    = EnumCondicaoReprodutiva::RUFIAO or
38                  animal_condicaoReprodAnimal
39                    = EnumCondicaoReprodutiva::INTEIRO)
40           else (true)
41           endif
42         ),
```

Código 6.7 RV da entidade "Animal"(versão estendida) - parte 3.

```
1          -- Linhas 2 a 42 descrevem expressões da RN.
2          rv_Animal_3_2 : Boolean =
3          ( if (animal_sexoAnimal = EnumSexo::F and
4                animal_eraAnimal <> EnumEra::FILHOTEFEMEA and
5                 (animal_situacaoAnimal <> EnumSituacaoAnimal::AC or
6                  animal_situacaoAnimal <> EnumSituacaoAnimal::GE) )
7            then  (animal_condicaoReprodAnimal
8                     = EnumCondicaoReprodutiva::PRENHA or
9                    animal_condicaoReprodAnimal
10                    = EnumCondicaoReprodutiva::PRENHAAM or
11                   animal_condicaoReprodAnimal
12                    = EnumCondicaoReprodutiva::VAZIA or
13                   animal_condicaoReprodAnimal
14                    = EnumCondicaoReprodutiva::VAZIAAM)
15           else (true)
16           endif
17         ),
18         rv_Animal_3_3 : Boolean =
19         ( if (animal_eraAnimal = EnumEra::FILHOTEMACHO or
20                animal_eraAnimal = EnumEra::FILHOTEFEMEA and
21                (animal_situacaoAnimal <> EnumSituacaoAnimal::AC or
22                 animal_situacaoAnimal <> EnumSituacaoAnimal::GE) )
23           then  (animal_condicaoReprodAnimal = null)
24           else (true)
25           endif
26         ),
27         rv_Animal_3_4 : Boolean =
28         ( if (animal_situacaoAnimal = EnumSituacaoAnimal::AC or
29                animal_situacaoAnimal = EnumSituacaoAnimal::GE)
30           then  (animal_condicaoReprodAnimal = null)
31           else (true)
32           endif
33         ),
34         rv_Animal_4_1 : Boolean =
35         ( if (animal_sexoAnimal = EnumSexo::F and
36               animal_eraAnimal = EnumEra::FILHOTEFEMEA) or
37               (animal_situacaoAnimal = EnumSituacaoAnimal::AC or
38                animal_situacaoAnimal = EnumSituacaoAnimal::GE)
39           then  (animal_tipoMatrizAnimal = null)
40           else (true)
41           endif
42         ),
```

Código 6.8 RV da entidade "Animal"(versão estendida) - parte 4.

```
1          -- Linhas 2 a 42 descrevem expressões da RN.
2          rv_Animal_4_2 : Boolean =
3          ( if (animal_sexoAnimal = EnumSexo::M) or
4                (animal_situacaoAnimal = EnumSituacaoAnimal::AC or
5                 animal_situacaoAnimal = EnumSituacaoAnimal::GE)
6            then  (animal_tipoMatrizAnimal = null)
7            else (true)
8            endif
9          ),
10         rv_Animal_4_3 : Boolean =
11         ( if (animal_situacaoAnimal = EnumSituacaoAnimal::AC or
12                animal_situacaoAnimal = EnumSituacaoAnimal::GE)
13           then  (animal_tipoMatrizAnimal = null)
14           else (true)
15           endif
16         ),
17         rv_Animal_5 : Boolean =
18         ( if (animal_registroAnimal <> null and
19               animal_situacaoRegistroAnimal <> null and
20               animal_nomeRegistroAnimal <> null and
21               animal_tipoIdFisicoRegistro <> null)
22               and
23               (animal_situacaoAnimal <> EnumSituacaoAnimal::AC or
24                animal_situacaoAnimal <> EnumSituacaoAnimal::GE)
25           then  (animal_maeAnimal <> null and animal_paiAnimal <> null)
26           else (true)
27           endif
28         ),
29         rv_Animal_6 : Boolean =
30         ( if (animal_situacaoAnimal <> EnumSituacaoAnimal::AC or
31               animal_situacaoAnimal <> EnumSituacaoAnimal::GE)
32           then  (animal_localizacaoAnimal <> null)
33           else (true)
34           endif
35         ),
36         rv_Animal_7 : Boolean =
37         ( if (animal_situacaoAnimal <> EnumSituacaoAnimal::AC or
38               animal_situacaoAnimal <> EnumSituacaoAnimal::GE)
39           then  (animal_tipoIdFisicoManejo <> null)
40           else (true)
41           endif
42         ),
```

Código 6.9 RV da entidade "Animal"(versão estendida) - parte 5.

```
1          -- Linhas 2 a 36 descrevem expressões da RN.
2          rv_Animal_8 : Boolean =
3          ( if (animal_idAnimalSisbov = null) or
4                (animal_situacaoAnimal = EnumSituacaoAnimal::AC or
5                 animal_situacaoAnimal = EnumSituacaoAnimal::GE)
6            then (animal_tipoIdSisbovAnimal = null)
7            else (true)
8            endif
9          ),
10         rv_Animal_9 : Boolean =
11         ( if (animal_situacaoAnimal = EnumSituacaoAnimal::AC or
12               animal_situacaoAnimal = EnumSituacaoAnimal::GE)
13           then  (animal_tipoIdFisicoRegistro = null)
14           else (true)
15           endif
16         ),
17         rv_Animal_10 : Boolean =
18         ( if (animal_situacaoAnimal <> EnumSituacaoAnimal::GE)
19           then  (animal_dataNascAnimal <> null)
20           else (true)
21           endif
22         ),
23         rv_Animal_11_1 : Boolean =
24         ( if (animal_situacaoAnimal <> EnumSituacaoAnimal::GE)
25           then  (animal_proprietariosPJAnimal <> null or
26                  animal_proprietariosPFAnimal <> null)
27           else (true)
28           endif
29         ),
30         rv_Animal_11_2 : Boolean =
31         ( if (animal_situacaoAnimal <> EnumSituacaoAnimal::GE)
32           then  (animal_criadorPesJurAnimal <> null or
33                  animal_criadorPesFisAnimal <> null)
34           else (true)
35           endif
36         ),
```

Código 6.10 RV da entidade "Animal"(versão estendida) - parte 6.

```
1           -- Linhas 2 a 40 descrevem expressões da RN.
2           rv_Animal_12_1 : Boolean =
3           ( if ( VarRaca.allInstances()
4                   ->select(v1| v1.racaVarRaca = animal_racaAnimal)->notEmpty()
5                 )
6             then (animal_varRacaAnimal <> null)
7             else (true)
8             endif
9           ),
10          rv_Animal_12_2 : Boolean =
11          ( if (animal_varRacaAnimal <> null)
12            then ( VarRaca.allInstances()
13                   ->select(v1| v1.racaVarRaca = animal_racaAnimal)->notEmpty()
14                   )
15            else (true)
16            endif
17          )
18
19  in      -- Linhas 20 a 40 define a expressão lógica resultante da RN.
20          result = rv_Animal_1_1  and
21                   rv_Animal_1_2  and
22                   rv_Animal_1_3  and
23                   rv_Animal_2    and
24                   rv_Animal_3_1  and
25                   rv_Animal_3_2  and
26                   rv_Animal_3_3  and
27                   rv_Animal_3_4  and
28                   rv_Animal_4_1  and
29                   rv_Animal_4_2  and
30                   rv_Animal_4_3  and
31                   rv_Animal_5    and
32                   rv_Animal_6    and
33                   rv_Animal_7    and
34                   rv_Animal_8    and
35                   rv_Animal_9    and
36                   rv_Animal_10   and
37                   rv_Animal_11_1 and
38                   rv_Animal_11_2 and
39                   rv_Animal_12_1 and
40                   rv_Animal_12_2
```

O Código 6.11 ilustra um exemplo completo de representação de uma RN do tipo Regra de Derivação no nível de projeto (camada conceitual). Esse código complementa o Código 5.5.

Código 6.11 RD da entidade "Animal"(versão estendida).

```
1
2  7. Modelagem da RN:
3  -- Linha 6 define o contexto da RN, neste caso,
4  -- o contexto proprietário é igual ao contexto destinatário (Animal_eraAnimal).
5  -- Linhas 7 e 8 definem os parâmetros e o tipo de retorno da RN.
6  context Animal::derivar_Animal_eraAnimal(
7   animal_sexoAnimal : String;
8   animal_dataNascAnimal : Data ) : String
9  derive: -- Linha 9 define o estereótipo da RN.
10 let     -- Linhas 11 a 37 descrevem expressões da RN.
11         qtde_dias : Integer =
12         Data::CURRENT_DATE - animal_dataNascAnimal
13 in
14         if qtde_dias >= 1 and qtde_dias <= 210
15         then
16                 if animal_sexoAnimal = EnumSexo::M
17                 then  EnumEra::FilhoteMacho
18                 else  EnumEra::FilhoteFemea
19                 endif
20         else
21                 if qtde_dias > 210 and qtde_dias <= 550
22                 then
23                         if animal_sexoAnimal = EnumSexo::M
24                         then  EnumEra::JovemMacho
25                         else  EnumEra::JovemFemea
26                         endif
27                 else
28                         if qtde_dias > 550
29                         then
30                             if animal_sexoAnimal = EnumSexo::M
31                             then  EnumEra::AdultoMacho
32                             else  EnumEra::AdultoFemea
33                             endif
34                         else ' '
35                         endif
36                 endif
37         endif
```

Os Códigos 6.12 e 6.13 ilustram um exemplo completo de representação de uma RN do tipo RV de Exportação no nível de projeto (camada conceitual); esses códigos complementam o Código 5.6. A RI correspondente é apresentada no Código 6.14. Este código é um complemento do Código 5.7.

Código 6.12 RE da entidade "TipoIdPessoaFisica"(versão estendida) - parte 1.

```
1  ---------------------------------
2  Especificação da Regra de Negócio
3  ---------------------------------
4
5  7. Modelagem da RN:
6  -- Linha 10 define o contexto da RN, neste caso,
7  -- o contexto proprietário é TipoIDPessoaFisica,
8  -- e o contexto destinatário é Pesfis_NumDocIdPesFis.
9  -- Linhas 11 e 12 definem os parâmetros e o tipo de retorno da RN.
10 context PesFis::validar_Pesfis_NumDocIdPesFis(
11         pesfis_numDocIDPesFis : String;
12         pesfis_tipoDocIDPesFis: String ) : Boolean
13 post: -- Linha 13 define o estereótipo da RN.
14       -- Linhas 15 a 38 descrevem expressões da RN.
15  let d1 : Integer = pesfis_numDocIdPesFis.substring(1,1).toInteger(),
16      d2 : Integer = pesfis_numDocIdPesFis.substring(2,2).toInteger(),
17      d3 : Integer = pesfis_numDocIdPesFis.substring(3,3).toInteger(),
18      d4 : Integer = pesfis_numDocIdPesFis.substring(4,4).toInteger(),
19      d5 : Integer = pesfis_numDocIdPesFis.substring(5,5).toInteger(),
20      d6 : Integer = pesfis_numDocIdPesFis.substring(6,6).toInteger(),
21      d7 : Integer = pesfis_numDocIdPesFis.substring(7,7).toInteger(),
22      d8 : Integer = pesfis_numDocIdPesFis.substring(8,8).toInteger(),
23      d9 : Integer = pesfis_numDocIdPesFis.substring(9,9).toInteger(),
24      dv1 : Integer = pesfis_numDocIdPesFis.substring(10,10).toInteger(),
25      dv2 : Integer = pesfis_numDocIdPesFis.substring(11,11).toInteger(),
26      soma1 : Integer =
27       (d1*10)+(d2*9)+(d3*8)+(d4*7)+(d5*6)+(d6*5)+(d7*4)+(d8*3)+(d9*2),
28      soma2 : Integer =
29       ((d1*11)+(d2*10)+(d3*9)+(d4*8)+(d5*7)+(d6*6)+(d7*5)+(d8*4)+(d9*3))+
30       (dv1*2),
31      resultado1 : Integer = soma1-((soma1.div(11))*11),
32      resultado2 : Integer = soma2-((soma2.div(11))*11),
33      tamanhoTipoDoc : Integer = TipoIDPessoaFisica.allInstances()->select
34       (tp | tp.nomeTipoDocIDPesFis
35        = pesfis_tipoDocIDPesFis).tamanhoTipoDocIDPesFis,
36      re_NumDocIdPesFis_1 : Boolean =
37      if tamanhoTipoDoc <> 11
38      then       false
```

Código 6.13 RE da entidade "TipoIdPessoaFisica"(versão estendida) - parte 2.

```
1       -- Linhas 2 a 38 descrevem expressões da RN.
2       else if pesfis_numDocIdPesFis.size() <> tamanhoTipoDoc
3             then         false
4             else
5                  if resultado1 = 1 or resultado1 = 0
6                  then if dv1 = 0
7                        then
8                            if resultado2 = 1 or resultado2 = 0
9                            then if dv2 = 0
10                                 then    true
11                                 else    false
12                                 endif
13                           else if dv2 = (11-resultado2)
14                                 then    true
15                                 else    false
16                                 endif
17                           endif
18                       else      false
19                       endif
20                 else if dv1 = (11-resultado1)
21                       then
22                           if resultado2 = 1 or resultado2 = 0
23                           then if dv2 = 0
24                                 then    true
25                                 else    false
26                                 endif
27                           else if dv2 = (11-resultado2)
28                                 then    true
29                                 else    false
30                                 endif
31                           endif
32                       else      false
33                       endif
34                 endif
35            endif
36      endif
37         -- Linha 38 define a expressão lógica resultante da RN.
38 in   result = re_NumDocIdPesFis_1
```

Código 6.14 RI da entidade "PesFis"(versão estendida).

```
1
2  7. Modelagem da RN:
3  -- Linha 6 define o contexto da RN, neste caso,
4  -- o contexto proprietário é igual ao contexto destinatário (PesFis_NumDocIdPesFis).
5  -- Linha 7 define o parâmetro e o tipo de retorno da RN.
6  context PesFis::importar_PesFis_NumDocIdPesFis
7          (pesfis_tipoDocIDPesFis: String) : String
8  body: -- Linha 8 define o estereótipo da RN.
9    -- Linhas 10 a 12 descrevem expressões da RN.
10   TipoIDPessoaFisica.allInstances()->
11    select(tp| tp.nomeTipoDocIDPesFis =
12    pesfis_tipoDocIDPesFis).regraTipoDocIDPesFis
```

O Código 6.15 ilustra um exemplo de representação de uma RN do tipo Regra de Validação no nível de projeto (camada conceitual). Este código é complemento do Código 5.8.

Código 6.15 RV da entidade "OcParto"(versão estendida).

```
1  --------------------------------
2  Especificação da Regra de Negócio
3  --------------------------------
4
5  7. Modelagem da RN:
6  -- Linha 9 define o contexto da RN, neste caso,
7  -- o contexto proprietário é igual ao contexto destinatário (OcParto).
8  -- Linhas 10 a 12 definem os parâmetros e o tipo de retorno da RN.
9  context OcParto::validar_OcParto (
10  parto_animalOcParto : Animal;
11  parto_pesoVacaOcParto : Real;
12  parto_dataOcParto : Data ) : Boolean
13 post: -- Linha 13 define o estereótipo da RN.
14       -- Linhas 15 a 41 descrevem expressões da RN.
15 let  sexoAnimal : String = Animal.allInstances()->
16      select(a1| a1.idAnimal = parto_animalOcParto).sexoAnimal,
17 rv_OcParto_1 : Boolean = (if (sexoAnimal = EnumSexo::F)
18                          then true
19                          else false
20                          endif),
21 indValorMinPVP : Real = IndRef.allInstances()->select().PVPMin,
22 indValorMaxPVP : Real = IndRef.allInstances()->select().PVPMax,
23 rv_OcParto_2 : Boolean = (if (parto_pesoVacaOcParto >= indValorMinPVP
24                          and parto_pesoVacaOcParto <= indValorMaxPVP)
25                          then true
26                          else false
27                          endif),
28 dtOcParto : Data = OcParto.allInstances()->
29   select(o1| o1.animalOcParto = parto_animalOcParto).dataOcParto@pre,
30 dtOcAborto : Data = OcAborto.allInstances()->
31   select(o2| o2.animalOcAborto = parto_animalOcParto).dataOcAborto@pre,
32 indValorIPA : Integer = IndRef.allInstances()->select().IPA,
33 dif_1 : Integer = Data::DIFDIAS(parto_dataOcParto,dtOcParto),
34 dif_2 : Integer = Data::DIFDIAS(parto_dataOcParto,dtOcAborto),
35 rv_OcParto_3 : Boolean = (if (dif_1 >= indValorIPA or dif_2 >= indValorIPA)
36                          then true
37                          else false
38                          endif)
39       -- Linha 41 define a expressão lógica resultante da RN.
40 in   result = rv_OcParto_1 and rv_OcParto_2 and rv_OcParto_3
```

Este capítulo descreveu o método proposto para representar as RN (especificadas na fase de análise) no nível de projeto (camada conceitual). O próximo capítulo descreve o método proposto para implementar as RN representadas no formato proposto neste capítulo.

CAPÍTULO 7

Implementação de Regras de Negócio

Este capítulo explica o tratamento da RN na camada interna do nível de projeto. Esta camada visa mapear as estruturas de RN que foram modeladas na camada conceitual do nível de projeto para um modelo que possa ser executado pelo software do SI.

A Seção 7.1 explica as atividades de transformação envolvidas na construção da camada interna do nível de projeto da RN, e na execução das RN pelo SI.

A Seção 7.2 apresenta os padrões utilizados para sistematizar as transformações propostas na Seção 7.1.

Com base nas definições e padrões apresentados, a Seção 7.3 propõe uma forma de implementação da RN.

Finalmente, os exemplos apresentados na Seção 6.3 são concluídos na Seção 7.4, fechando o ciclo de vida proposto pela metodologia de tratamento de RN proposta neste trabalho.

7.1 Definições

O principal objetivo da camada interna no nível de projeto é definir a forma de implementação das RN. Para isso é preciso representar as estruturas da RN internamente no software da aplicação. Conforme mostrado da Figura 4.3 isto é feito pelas atividades **Implantação da RN** (realizada em tempo de desenvolvimento da RN) e **Gerenciamento da RN** (realizada em tempo de operação ou execução da RN). A atividade de implantação é pré-requisito para a atividade de gerenciamento da RN.

É importante ressaltar que o cadastro eletrônico proposto na Seção 5.3 deve estar disponível antes de iniciar as atividades de implantação e gerenciamento da RN. A aplicação do sistema deve viabilizar uma forma de recuperação das informações contidas no cadastro eletrônico, bem como serviços de conexão e transação com SGBD. As informações da RN registradas via cadastro eletrônico devem ser disponibilizadas em uma forma de metadados que armazena a RN associada ao conceito do modelo do negócio. Em outras palavras, a RN deve ser relacionada ao seu respectivo contexto proprietário.

A Figura 7.1 mostra os passos definidos para realização de uma implantação de RN:

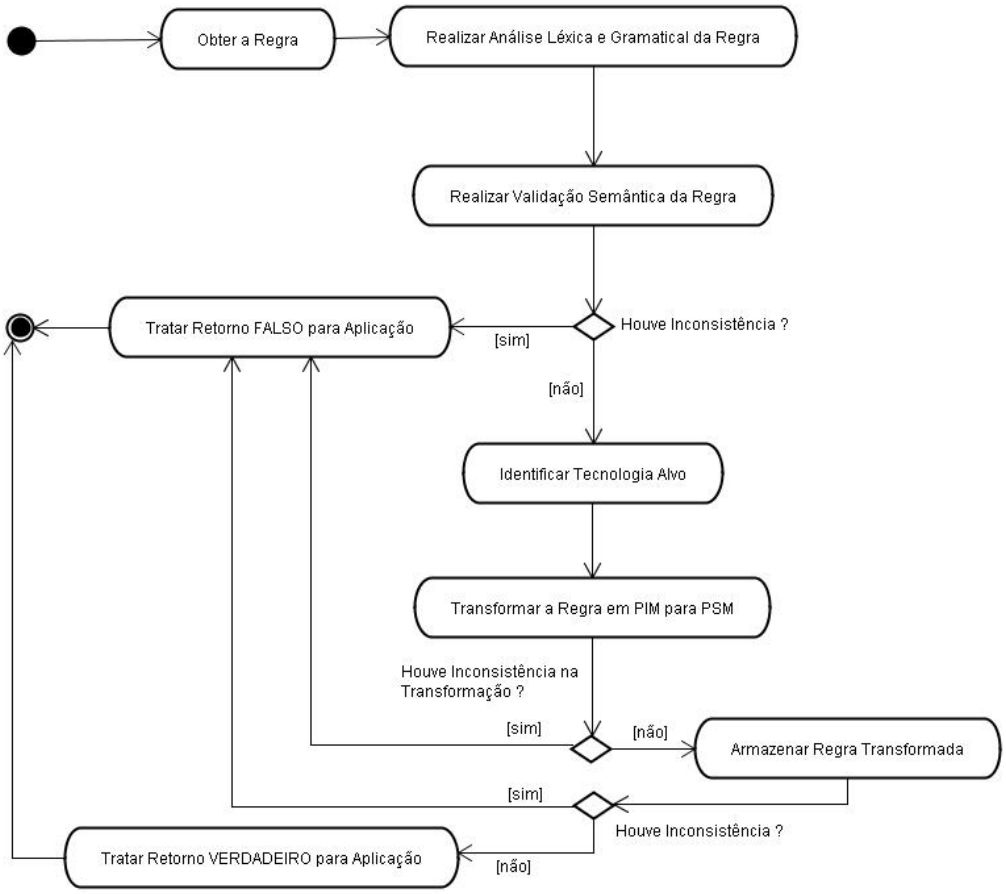

Figura 7.1: *Passos da atividade Implantação da RN.*

Obter a Regra: neste passo inicia-se a atividade de implantação da RN, cujo objetivo é obter todos os dados da RN já armazenada (via cadastro eletrônico de regras), principalmente a declaração da regra em PIM (ou regra-PIM para o restante deste capítulo). Todos os dados da RN devem ser disponibilizados em uma estrutura de dados para os passos seguintes desta atividade, quando solicitado.

É importante ressaltar que somente as RN cujo atributo *Status* é igual a Implantação estão prontas para serem implantadas (esta atualização deve ter sido feita previamente pelo projetista via cadastro eletrônico de regras).

Realizar Análise Léxica e Gramatical da Regra: a tarefa deste passo é realizar a análise léxica e gramatical da regra-PIM, de acordo com a gramática da linguagem PIM adotada (neste caso é a OCL). Esta regra é armazenada em uma estrutura de dados tipo árvore, facilitando a automatização dessas análises. Esta estrutura deve ser disponibilizada para os passos subseqüentes quando solicitado.

Realizar Validação Semântica da Regra: conforme definido no Capítulo 6, a declaração da RN é feita sobre o modelo de objetos do negócio. No entanto, para que os tipos de objetos declarados em uma regra-PIM estejam consistentes com os tipos definidos no modelo de

objetos em questão, este passo deve realizar uma validação sintática desses tipos. Caso haja alguma inconsistência, deve-se corrigir a regra-PIM e não realizar sua implantação (no diagrama da Figura 7.1 este procedimento é simbolizado pelo passo "Tratar Retorno FALSO para Aplicação").

Identificar Tecnologia Alvo: caso nenhuma inconsistência com a declaração da RN tenha ocorrido no passo anterior, neste passo deve-se obter a linguagem PSM alvo. Esta informação é necessária para a transformação da regra-PIM para PSM (neste trabalho adotamos a linguagem SQL).

Transformar a Regra em PIM para PSM: após ter-se identificado a linguagem PSM no passo anterior, este passo deve realizar os procedimentos para transformação da regra-PIM para PSM. Este passo é essencial para a realização da atividade "Implantação da RN", pois o resultado dessa transformação é uma regra processável por computador.

Os procedimentos de transformação devem obedecer regras de transformação, e estas regras são totalmente direcionadas para o mapeamento do modelo de objetos do negócio para o modelo dependente de plataforma correspondente (ou PSM). É muito importante o conhecimento exato desses modelos para definir a regra de transformação. Um exemplo de regras de transformação são as regras para mapeamento do Modelo OO para o Modelo Relacional. Caso ocorra alguma inconsistência nesta transformação, a implantação da RN não deve ser concretizada.

Devido a sua importância, este passo é detalhado na Figura 7.2.

Armazenar Regra Transformada: caso nenhuma inconsistência tenha ocorrido na transformação da RN em PIM para PSM (passo anterior), então este passo deve implantar a regra na aplicação, ou seja, deve armazenar de forma persistente para que possa ser executada quando for invocada pela aplicação. Se nenhuma inconsistência acontecer neste passo, segue-se para o passo "Tratar Retorno VERDADEIRO para Aplicação" e, a atividade "Implantação da RN" é realizada com sucesso na aplicação.

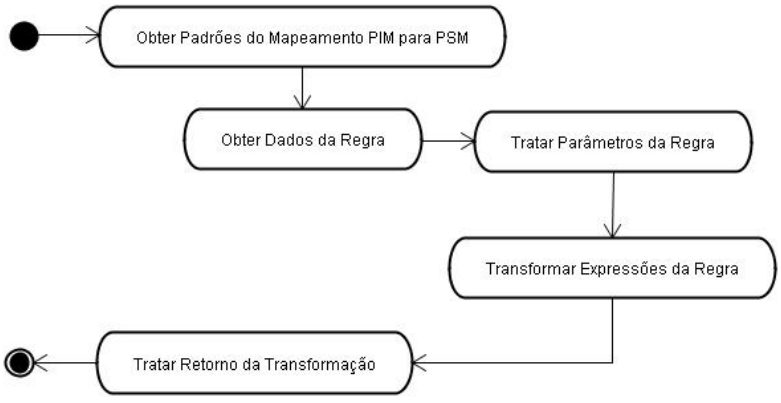

Figura 7.2: *Transformação da RN em PIM para PSM.*

Obter Padrões do Mapeamento PIM para PSM: neste passo, deve-se obter as regras ou padrões previamente definidos para realizar a transformação ou mapeamento da declaração da RN expressa em PIM para um PSM específico. Por exemplo: um padrão que represente toda a estrutura da regra em PSM conforme o tipo da RN.

Esses padrões devem ser disponibilizados para os passos subseqüentes quando solicitado.

Obter Dados da Regra: neste passo deve-se obter dados específicos (ou partes específicas) da regra-PIM (disponibilizada no passo "Obter a Regra") e torná-los disponíveis para realização do passo "Transformar Expressões da Regra". Isto facilita a busca por partes específicas da regra-PIM como, por exemplo, os parâmetros declarados na regra.

Tratar Parâmetros da Regra: este passo consiste em obter os parâmetros do negócio contidos da declaração de parâmetros da regra-PIM e seus respectivos valores (ou conteúdos) que devem ser fornecidos dinamicamente pela aplicação. Algum tratamento nesses valores pode ser feito neste passo como, por exemplo, a conversão dos conteúdos de um vetor para um tipo *string*.

Transformar Expressões da Regra: este passo realiza a transformação das expressões da regra-PIM para PSM. O funcionamento básico é: a partir da estrutura de dados disponibilizada no passo "Obter Dados da Regra" deve-se navegar nesta estrutura e, na seqüência, identificar as expressões da regra. De posse das expressões da regra e da regra de transformação ou mapeamento de PIM para PSM, deve-se construir um esquema PSM equivalente.

Tratar Retorno da Transformação: após concluído com sucesso o passo anterior, a regra transformada em um PSM deve ser disponibilizada para o passo "Armazenar Regra Transformada".

Após a implantação da RN na aplicação, finaliza-se o desenvolvimento da RN. Logo, a RN está disponível para operação (ou execução). Esta operação é realizada pela atividade "Gerenciamento da RN", cujo objetivo é gerenciar a tarefa de execução da regra implantada quando for solicitada pela aplicação. A Figura 7.3 apresenta os passos para realizar este gerenciamento.

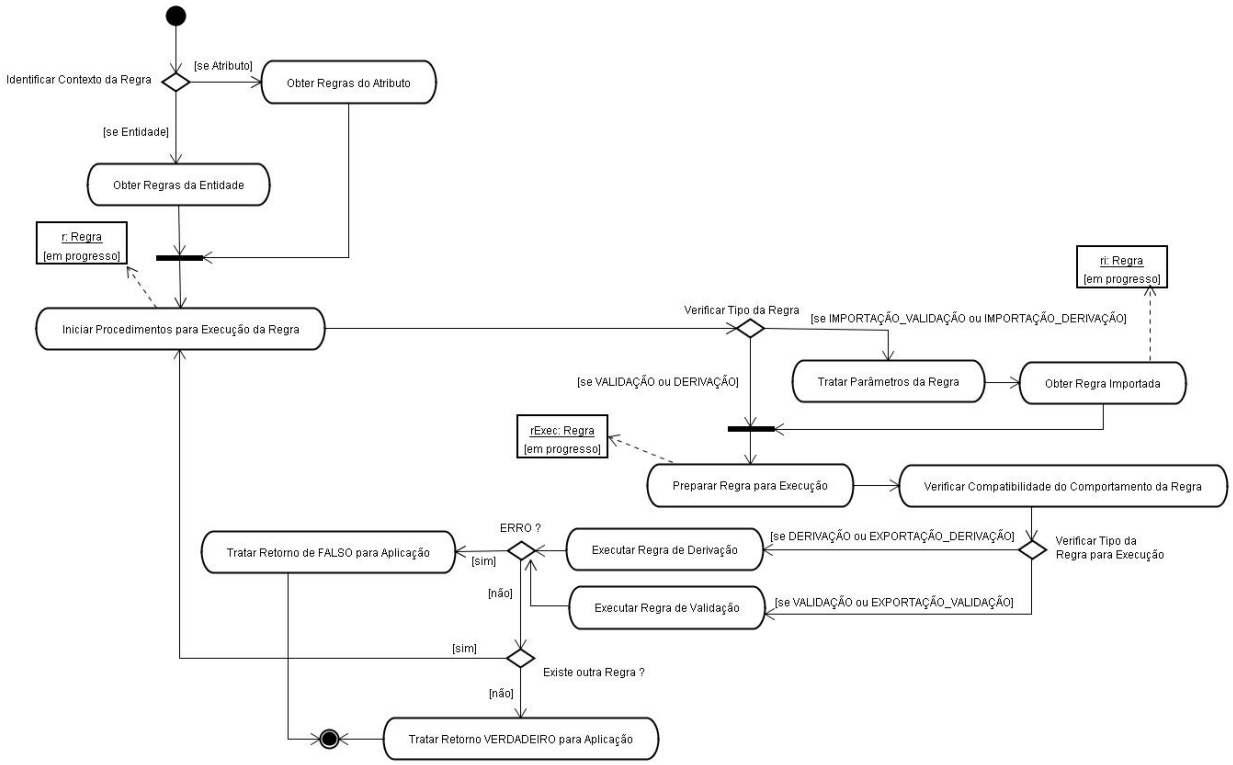

Figura 7.3: *Passos da atividade Gerenciamento da RN.*

Identificar Contexto da Regra: as responsabilidades deste passo são: 1) identificar o contexto proprietário da regra, isto é, se o proprietário da regra que será gerenciada é uma entidade do negócio ou um atributo de uma entidade do negócio; 2) obter todos os dados da(s) regra(s) (dados já armazenados via cadastro eletrônico de regras) associadas com o contexto em questão. No diagrama da Figura 7.3 este procedimento é simbolizado pelos passos "Obter Regras da Entidade" ou "Obter Regras do Atributo", de acordo com o contexto identificado.

A obtenção dos dados da regra é necessária pois o momento de realização deste passo é em tempo de operação da regra, conforme ilustrado na Figura 4.3. Portanto, deve-se obter os dados da regra que será gerenciada.

Esses dados devem ser disponibilizados em uma estrutura de dados para os passos seguintes desta atividade, quando solicitado. É recomendado também armazenar a regra-PIM em uma estrutura de dados tipo árvore.

Iniciar Procedimentos para Execução da Regra: pode haver mais de uma regra associada a uma entidade ou atributo do negócio para ser gerenciada conforme define o passo anterior. Logo, a responsabilidade deste passo é disponibilizar e controlar a verificação individual de cada uma dessas regras para os passos seguintes. Para cada uma dessas regras deve-se criar um objeto do tipo Regra cujas propriedades contêm os dados de uma regra disponibilizados pelo passo anterior.

Verificar Tipo da Regra: o objetivo deste passo é verificar qual é o tipo da RN. Esta verificação é importante porque as regras de importação têm um tratamento diferenciado antes de serem executadas.

O funcionamento deste tratamento é: se a regra for de importação, seja com propósito dedicado para uma regra de validação ou para uma regra de derivação, dois procedimentos devem ser feitos: primeiramente, obter os parâmetros do negócio contidos da declaração de parâmetros da regra e seus respectivos valores (ou conteúdos) que devem ser fornecidos dinamicamente pela aplicação (no diagrama da Figura 7.3 este procedimento é simbolizado pelo passo "Tratar Parâmetros da Regra"); e depois, obter a RN importada de outro contexto proprietário (no diagrama da Figura 7.3 este procedimento é simbolizado pelo passo "Obter Regra Importada"). Esta importação resulta em um identificador lógico de uma RN que será insumo para o passo seguinte. Um objeto do tipo Regra deve ser criado com as propriedades da regra importada.

É importante lembrar que o tipo da RN deve ter sido disponibilizado pelo passo "Identificar Contexto da Regra".

Preparar Regra para Execução: este passo é responsável pela criação de um novo objeto do tipo Regra para realizar a execução da regra propriamente dita. Este novo objeto deve receber as propriedades de outro objeto, isto é, caso a regra verificada no passo anterior não seja uma regra de importação, então o objeto deve receber as propriedades do objeto tipo Regra criado no passo "Iniciar Procedimentos para Execução das Regras" e, caso contrário, deve receber as propriedades do objeto tipo Regra criado no passo anterior.

Verificar Compatibilidade do Comportamento da Regra: a tarefa deste passo é verificar se o comportamento e o aspecto de tempo da regra (esta regra provém do passo anterior) que será executada estão de acordo com o comportamento e o estado atual da aplicação; caso contrário, significa que a regra não está apta e não será executada.

Verificar Tipo da Regra para Execução: caso a regra esteja apta para ser executada conforme passo anterior, este passo deve testar qual é o tipo da regra (dado obtido no passo "Verificar Tipo da Regra") e encaminhar para o passo de execução correspondente, isto é, ou para execução de uma regra de derivação ou para execução de uma regra de validação.

Executar Regra de Derivação e Executar Regra de Validação: o funcionamento desses passos são semelhantes, porém com algumas variações. Antes da execução, deve-se obter os parâmetros do negócio contidos na declaração de parâmetros da regra e seus respectivos valores (ou conteúdos) que devem ser fornecidos dinamicamente pela aplicação.

Em seguida, deve-se executar a regra de derivação ou de validação, ou seja, invocar o código executável da regra já implantado previamente. Caso ocorra algum erro durante este passo, é preciso tratar seu insucesso (no diagrama da Figura 7.3 isto é simbolizado pelo passo "Tratar Retorno FALSO para Aplicação"); caso contrário, se foi executada uma regra de derivação, então deverá retornar para aplicação um tipo de objeto conforme foi modelado

na regra e a aplicação deve tratar este objeto retornado; caso foi uma regra de validação, então a execução foi com sucesso e isto significa que não houve violação de RN (ambos procedimentos são simbolizados no diagrama da Figura 7.3 pelo passo "Tratar Retorno VERDADEIRO para Aplicação").

Ao término, caso haja outra regra para ser executada, deve-se retornar para o passo "Iniciar Procedimentos para Execução da Regra".

O passo "Tratar Retorno FALSO para Aplicação" retorna para a aplicação a identificação da regra que foi violada ou retorna uma informação que não foi possível realizar a derivação; e a aplicação deverá estar preparada para receber esta informação e tratá-la de maneira conveniente.

7.2 Padrões Utilizados

Alguns padrões foram utilizados para realização da atividade **Implantacão da RN**. Em **Obter a Regra** adotamos uma estrutura de dados em árvore para o armazenamento temporário da declaração da regra em PIM. Esta escolha visa auxiliar as demais atividades que necessitam dos dados descritos na regra. Isto torna a busca por partes específicas da declaração da regra mais eficiente, pois cada parte da regra é armazenada em um nó da árvore, e cada nó representa uma sintaxe abstrata correspondente à estrutura da gramática OCL (linguagem PIM adotada). Esse tipo de estrutura facilita a navegabilidade entre as partes da regra, e também a análise léxica e gramatical.

Para **Transformar a Regra em PIM para PSM** é muito importante um bom conhecimento dos vários modelos envolvidos: modelo de objetos do negócio, PIM e PSM. Para definir a regra de transformação de PIM para PSM, uma boa prática é realizar a transformação entre os modelos manualmente até atingir um "certo" grau de maturidade e confiança no processo de transformação e, após este periódo de prática, definir os passos que tornarão esse processo de transformação automático.

Conforme já definimos, o modelo PIM adotado é a linguagem OCL e o modelo PSM é a linguagem SQL. Como OCL é uma linguagem OO e SQL uma linguagem relacional, temos um mapeamento objeto-relacional. Em [21] são propostas algumas diretrizes para este mapeamento, incluindo a geração de código SQL a partir de OCL. Por exemplo: o nome de uma classe no modelo de objetos do negócio e também em OCL será transformada em uma tabela em SQL; os atributos desta classe correspondem aos campos da tabela.

Adotamos dois gabaritos básicos para o formato do mapeamento de OCL para SQL. O **Gabarito 1** é para ser usado no mapeamento de regra de validação, importação ou exportação, e o **Gabarito 2** é específico para regra de derivação. O formato padronizado define um procedimento armazenado (função) que pode ser implantado em um SGBD relacional. Adotamos o dialeto de SQL do SGBD PostgreSQL [42]. Os Códigos 7.1 e 7.2 ilustram estes gabaritos.

Código 7.1 Gabarito 1.

```
1
2  CREATE OR REPLACE FUNCTION <nomeDaFuncao> ( <listaDeArgumentos> )
3  RETURNS VARCHAR AS $$
4  DECLARE
5          resultado VARCHAR;
6          <variáveis>
7  BEGIN
8          resultado := '';
9          <expressões>
10
11 RETURN resultado;
12 END;
13 $$ LANGUAGE plpgsql;
```

Código 7.2 Gabarito 2.

```
1
2  CREATE OR REPLACE FUNCTION <nomeDaFuncao> ( <listaDeArgumentos> )
3          RETURNS <tipoDoRetorno>  AS $$
4  DECLARE
5          <variaveis>
6  BEGIN
7          <expressões>
8  END;
9  $$ LANGUAGE plpgsql;
```

Ambos gabaritos mostrados apresentam campos que devem ser preenchidos durante o processo de transformação da regra-PIM (OCL) para regra-PSM (função SQL). A seguir, detalhamos esses campos:

Em **<nomeDaFuncao>** deve aparecer o nome da função, que deve ser o mesmo nome da operação em OCL da RN em questão.

Por exemplo: seja o seguinte trecho da RN em OCL mostrada no Código 6.5, "context Animal::validar_Animal (... animal_racaAnimal: Raça) : Boolean"; o nome da operação "validar_Animal" é o mesmo nome da função SQL; no caso dos Gabaritos 1 e 2, o nome desta operação preenche o campo **<nomeDaFuncao>**; o argumento da operação "animal_racaAnimal: Raça" forma a lista de parâmetros da função SQL; no caso dos Gabaritos 1 e 2, preenche o campo **<listaDeArgumentos>**. O tipo do parâmetro deve obedecer os tipos básicos de SQL: o tipo *string* de OCL será mapeado para *VARCHAR* de SQL; o tipo *integer* de OCL será mapeado para *INTEGER* de SQL; o tipo *boolean* de OCL será mapeado para *BOOL* de SQL; o tipo *real* de OCL será mapeado para *DECIMAL(15,2)* de SQL.

Em OCL não existe um tipo de domínio para datas, porém, é preciso criar o tipo Data como tipo definido no modelo de objetos do negócio, juntamente com os métodos pertinentes ao

tipo Data que forem necessários como, por exemplo, o método *agora*(), cujo comportamento deve ser o retorno da data atual do sistema. Diante disso, os parâmetros em OCL do tipo data utilizam o tipo definido **data**, e o seu correspondente em SQL é mapeado para *DATE*; o formato da data utilizado é *AAAA-MM-DD* (ano-mês-dia).

Em **<tipoDoRetorno>**, no caso de RN do tipo validação, importação e exportação o tipo mapeado para SQL é *VARCHAR*. Este tipo alfanumérico pode receber uma lista de identificações lógicas de regras violadas, da mesma forma que uma única identificação lógica de uma regra importada. No caso de regra de derivação, o tipo de retorno deve obedecer o tipo modelado em OCL.

No entanto, no caso de tipos definidos no modelo de objetos do negócio como, por exemplo, "Raça" (declarado no Código 6.5), o mapeamento também é para *VARCHAR*, pois um tipo definido no modelo de objetos do negócio contido em uma lista de parâmetros nos informa uma referência para uma instância desse tipo em questão, e esta referência é exatamente um identificador lógico desse tipo. Assim, o tipo alfanumérico *VARCHAR* é preparado para receber qualquer tipo de identificador lógico, isto é, um identificador do tipo básico inteiro ou *string*, por exemplo.

Em **<variáveis>** devem ser incluídas as variáveis modeladas em uma cláusula *let* de OCL ou quaisquer outras variáveis necessárias à execução de algum procedimento escrito no corpo da função.

<expressões> devem conter as expressões SQL equivalentes em OCL. Recomendamos que este mapeamento seja feito com o auxílio da estrutura tipo árvore criada em **Obter a Regra**. Com esta estrutura é possível navegar sobre as partes das expressões e a cada passo na navegação deve-se identificar a sintaxe de OCL equivalente em SQL e realizar o mapeamento. É importante notar que a cada subexpressão em OCL identificada resulta ou um valor booleano ou um objeto de qualquer tipo definido em OCL.

Em relação à atividade **Gerenciamento da RN**, alguns padrões também foram utilizados. Em **Identificar Contexto da Regra** o padrão adotado para identificar se uma regra é aplicada a uma entidade do negócio ou a um atributo de uma entidade do negócio é através do contexto destinatário da RN. Este contexto é o mesmo contexto explicitamente descrito no nome da operação declarada em OCL, conforme o padrão sugerido para nomenclatura da operação em OCL.

Por exemplo: no Código 6.5 temos, "context Animal::validar_Animal (...)"; neste exemplo temos uma RN aplicada a uma entidade "Animal" cujo contexto destinatário é a entidade do negócio "Animal". Isto implica que toda instância da entidade Animal deve obedecer a RN em questão.

Outro exemplo seria: "context Animal::validar_Animal_DataDeNascimento (...)"; neste exemplo temos uma RN aplicada ao atributo "DataDeNascimento" da entidade do negócio "Animal"; portanto, toda instância deste atributo da entidade Animal deve obedecer a RN em questão.

7.3 Forma de Implementação

Com base nas definições e padrões apresentados, esta seção explica uma forma de implementação para os principais passos apresentados nas atividades de implantação e gerenciamento da RN.

A Figura 7.4 ilustra o diagrama de classes de projeto que implementa o tratamento de RN. Uma visão geral deste projeto de classes, juntamente com outros componentes que auxiliam a implementação, é mostrada na Figura 7.5. Para uma melhor compreensão destes diagramas, o Apêndice G apresenta o dicionário dos componentes representados nos diagramas.

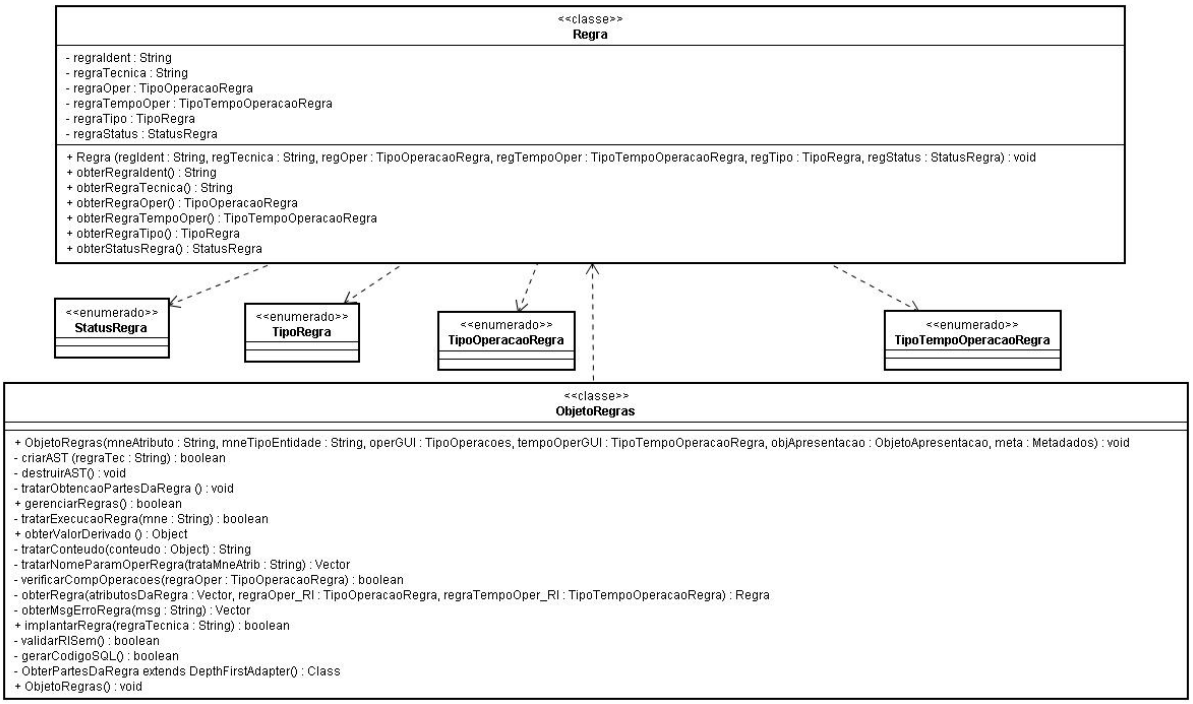

Figura 7.4: *Diagrama de classes de projeto para tratamento de RN.*

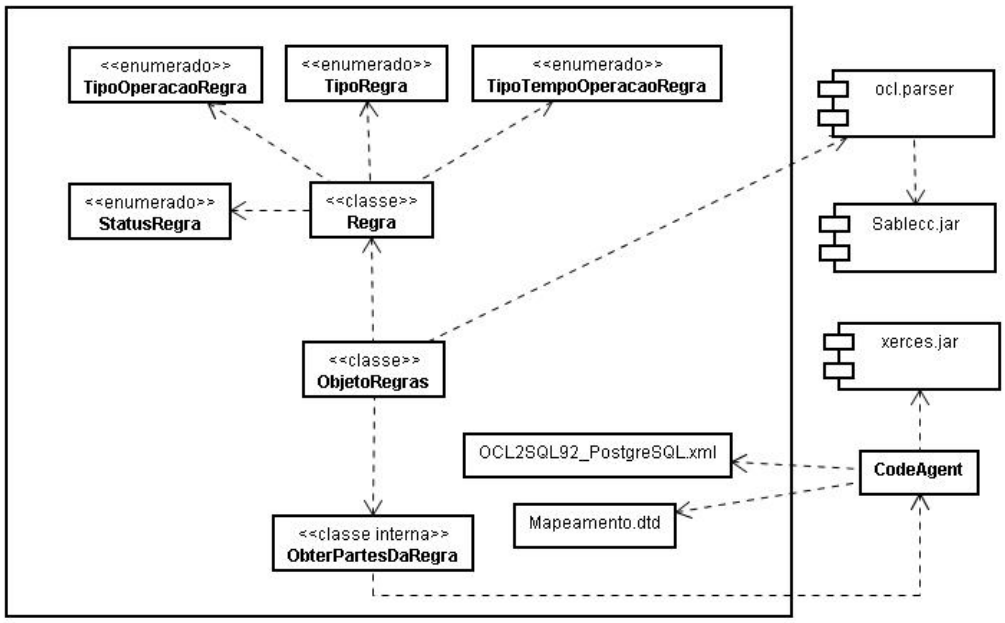

Figura 7.5: *Visão geral do projeto de classes para tratamento de RN.*

Em relação ao armazenamento persistente da especificação da RN, especificamente as declarações da RN em linguagem natural nos vários idiomas, a forma de implementação adotada foi através de um serviço de cadastramento de mensagens da aplicação, no qual cada tipo de RN foi associada a um tipo de estereótipo de mensagem destinada ao usuário da aplicação.

Partindo da atividade **Implantação da RN**, os passos **Obter a Regra** e **Realizar Validação Semântica da Regra** foram implementadas com o auxílio de um gerador de *parser* OO denominado ***SableCC*** [13]. Este gerador permite a criação de classes JAVA que implementam a análise léxica e gramatical da declaração de uma RN em OCL. A regra é convertida em uma estrutura de dados tipo árvore, conhecida como **AST** (*Abstract Syntax Tree*) [12, 17], a partir da sintaxe da gramática OCL.

Através da AST gerada é possível navegar na estrutura da regra e obter partes da regra que forem necessárias a qualquer etapa da atividade de realização da RN, tais como:

(A) em **Realizar Validação Semântica da Regra** um procedimento foi implementado para navegar na AST criada e identificar os termos que correspondem às entidades do negócio e, juntamente com um serviço de pesquisa que a aplicação deve fornecer (pesquisa aos metadados, entidades/conceitos do modelo de objetos do negócio), este procedimento faz a validação dos termos da regra verificando se estão consistentes com o modelo de objetos do negócio.

(B) em **Transformar a Regra em PIM para PSM** um procedimento foi implementado para identificar na AST os *tokens* correspondentes ao padrão de transformação de OCL para SQL padronizados (Gabaritos 1 e 2 apresentados nos Códigos 7.1 e 7.2, respectivamente). Exemplos de RN em OCL transformadas para SQL são apresentados na Seção 7.3.

Em **Obter Padrões do Mapeamento PIM para PSM** a forma implementada para obtenção dos padrões previamente criados foi através do armazenamento desses padrões em um arquivo XML (*Extensible Markup Language*) [43]. Por exemplo: os Gabaritos 1 e 2 mostrados

nos Códigos 7.1 e 7.2, respectivamente; e os tipos básicos de OCL para SQL. Ambas configurações do arquivo XML e da DTD (*Document Type Definition*) foram adaptadas das configurações disponíveis no *Dresden ToolKit* [37]. O Código D.1 do Apêndice D mostra a DTD utilizada na interpretação do arquivo XML mostrado nos Códigos E.1 e F.1 dos Apêndices E e F, respectivamente. Este arquivo XML demonstram a implementação do armazenamento dos Gabaritos 1 e 2, e dos tipos básicos OCL para SQL do SGBD PostgreSQL [42].

Em relação a forma de atribuição de valores nos Gabaritos 1 e 2 (arquivos em XML) foi utilizada uma classe do *Dresden ToolKit* denominada *CodeAgent* juntamente com o pacote *xerces.jar* [37]. Estes componentes trabalham em conjunto para permitir a leitura do arquivo XML que armazena os padrões (Gabaritos 1 e 2, por exemplo), bem como as atribuições de parâmetros nos locais especificados dentro dos gabaritos.

O resultado da implementação do passo **Armazenar Regra Transformada** (este passo finaliza a implantação de uma RN) é uma função armazenada no SGBD PostgreSQL [42]. Codificamos uma função em SQL que pode ser utilizada pela aplicação para armazenar uma função que implementa uma RN; para o parâmetro de entrada desta função deve ser atribuído o respectivo código SQL da regra que foi transformada. O Código 7.3 mostra a codificação desta função.

Código 7.3 Código SQL para armazenar uma função que implementa uma RN.

```
1
2  CREATE OR REPLACE FUNCTION implantar_regra(regra VARCHAR)
3    RETURNS bool AS
4  BEGIN
5          EXECUTE regra;
6          RETURN TRUE;
7  END;
8  $$ LANGUAGE plpgsql;
```

Com relação à atividade **Gerenciamento da RN**, em **Iniciar Procedimentos para Execução da Regra** concentra-se o início dos procedimentos para execução das regras. Neste ponto é criado um **objeto do tipo Regra** projetado e implementado para encapsular toda a especificação de uma RN.

Em ambos passos, **Executar Regra de Derivação** e **Executar Regra de Validação**, a aplicação invoca a função que executa o teste de validação ou de derivação da regra que rege a entidade de negócio.

No caso de regra de validação é feito um teste para cada expressão da RN modelada, isto é, cada variável da cláusula *let* de OCL corresponde a um possível teste de regra de validação que precisa ser verificado. Se o teste não for satisfatório, a variável resultado do teste deve acumular a identificação lógica do teste correspondente à regra que foi violada.

O passo **Tratar Retorno FALSO para Aplicação** implementa o tratamento do retorno falso; neste caso, se o teste foi originado por uma regra de derivação, a aplicação receberá um

retorno característico de erro e, se foi originado por uma regra de validação, a aplicação receberá um conjunto de identificadores lógicos juntamente com os respectivos textos declarativos em linguagem natural das RN que foram violadas, ou seja, aquelas cuja satisfação é falsa.

No passo **Tratar Retorno VERDADEIRO para Aplicação**, se o teste foi originado por uma regra de derivação, conforme definição, a aplicação receberá um retorno do tipo que foi modelado; e se foi originado por uma regra de validação, a aplicação receberá um retorno booleano verdadeiro.

Com relação à utilização do tratamento de RN aqui proposto no software de uma aplicação de um SI (na Figura 4.3 isto é simbolizado pelo relacionamento denominado ***<utiliza>***), os Códigos 7.4 a 7.6 apresentam exemplos de utilização dos métodos que devem ser inseridos no código fonte da aplicação. Esses exemplos foram instanciados no módulo ***SIACorte*** (cenário descrito na Seção 4.3).

Código 7.4 Método responsável pelo gerenciamento da RN, com exceção da RD.

```
1  /*
2  * Método responsável pela invocação do tratamento
3  * das RN, com exceção da RD.
4  */
5  private boolean verificarRegras(
6          String mneAt,
7          TipoOperacoes oper,
8          TipoTempoOperacaoRegra tempoOper){
9
10         ObjetoRegras objRegras = new ObjetoRegras
11          (mneAt, mneEnt, oper, tempoOper, objAp, metadados);
12
13         // Realiza a atividade "Gerenciamento da RN",
14         // com exceção da RD.
15         if ( !(objRegras.gerenciarRegras()) ) {
16                 return false;
17         }
18         return true; // Verificação OK!
19 }
```

Código 7.5 Método responsável pela implantação da RN.

```
1  /*
2  * Método responsável pela implantação de uma RN.
3  */
4  private boolean implantarRegra(String regraTecnica){
5      ObjetoRegras objRegras = new ObjetoRegras ();
6
7          // Realiza a atividade "Implantação da RN".
8          if ( !(objRegras.implantarRegra(regraTecnica)) ) {
9                  return false;
10         }
11         return true; // Implantação OK!
12 }
```

Código 7.6 Método responsável pelo gerenciamento da RD.

```
1  /*
2  * Método responsável pela invocação de uma RD.
3  */
4  private Object obterValorDerivado (
5          String mneAt,
6          ObjetoApresentacao objAp,
7          Metadados meta ){
8
9          ObjetoRegras objRegras = new ObjetoRegras
10         (mneAt, null, TipoOperacoes.ALTERAR,
11         TipoTempoOperacaoRegra.ANTES, objAp, meta);
12
13         // Realiza a atividade "Gerenciamento da RN" para RD.
14         if ( !(objRegras.gerenciarRegras()) ) {
15                 return false;
16         }
17         return objRegras.obterValorDerivado(); // Derivação OK!
18 }
```

7.4 Exemplos

Com o propósito de demostrar a representação de uma RN no nível de projeto na camada interna, conforme definido neste capítulo, apresentamos os exemplos estendidos que foram mostrados na Seção 6.3.

Os Códigos 7.7 até 7.15 ilustram um exemplo completo de representação de uma RN do tipo Regra de Validação no nível de projeto (camada interna). Esses códigos representam o mapeamento de OCL para SQL dos Códigos 6.5 a 6.10.

Código 7.7 RV da entidade "Animal"(versão estendida) - parte 1.

```
1  --------------------------
2  Mapeamento da RN para SQL:
3  --------------------------
4
5  CREATE OR REPLACE FUNCTION validar_Animal (
6   animal_catRegAnimal VARCHAR,
7   animal_registroAnimal  VARCHAR,
8   animal_situacaoRegistroAnimal VARCHAR,
9   animal_nomeRegistroAnimal VARCHAR,
10  animal_tipoIdFisicoRegistro VARCHAR,
11  animal_corDePeloAnimal VARCHAR,
12  animal_racaAnimal VARCHAR,
13  animal_sexoAnimal VARCHAR,
14  animal_eraAnimal VARCHAR,
15  animal_condicaoReprodAnimal VARCHAR,
16  animal_situacaoAnimal VARCHAR,
17  animal_tipoMatrizAnimal VARCHAR,
18  animal_maeAnimal VARCHAR,
19  animal_paiAnimal VARCHAR,
20  animal_localizacaoAnimal VARCHAR,
21  animal_tipoIdFisicoManejo VARCHAR,
22  animal_idAnimalSisbov VARCHAR,
23  animal_tipoIdSisbovAnimal VARCHAR,
24  animal_dataNascAnimal  DATE,
25  animal_proprietariosPJAnimal VARCHAR,
26  animal_proprietariosPFAnimal VARCHAR,
27  animal_criadorPesJurAnimal VARCHAR,
28  animal_criadorPesFisAnimal VARCHAR,
29  animal_varRacaAnimal VARCHAR
30 )
31 RETURNS VARCHAR AS $$
```

Código 7.8 RV da entidade "Animal"(versão estendida) - parte 2.

```
1
2  DECLARE
3          pkPesquisa      INTEGER;
4          rv_Animal_1_1   BOOL;
5          rv_Animal_1_2   BOOL;
6          rv_Animal_1_3   BOOL;
7          rv_Animal_2     BOOL;
8          rv_Animal_3_1   BOOL;
9          rv_Animal_3_2   BOOL;
10         rv_Animal_3_3   BOOL;
11         rv_Animal_3_4   BOOL;
12         rv_Animal_4_1   BOOL;
13         rv_Animal_4_2   BOOL;
14         rv_Animal_4_3   BOOL;
15         rv_Animal_5     BOOL;
16         rv_Animal_6     BOOL;
17         rv_Animal_7     BOOL;
18         rv_Animal_8     BOOL;
19         rv_Animal_9     BOOL;
20         rv_Animal_10    BOOL;
21         rv_Animal_11_1  BOOL;
22         rv_Animal_11_2  BOOL;
23         rv_Animal_12_1  BOOL;
24         rv_Animal_12_2  BOOL;
25         resultado       TEXT;
26 BEGIN
27
28         resultado := '';
29
30         IF ( UPPER(animal_situacaoAnimal) = 'AC' OR
31              UPPER(animal_situacaoAnimal) = 'GE' )
32         THEN rv_Animal_1_1 := (animal_registroAnimal NOTNULL AND
33                                animal_situacaoRegistroAnimal NOTNULL AND
34                                animal_nomeRegistroAnimal NOTNULL AND
35                                animal_tipoIdFisicoRegistro NOTNULL);
36         ELSE rv_Animal_1_1 := TRUE;
37         END IF;
38         RAISE NOTICE 'Resultado rv_Animal_1_1 = %', rv_Animal_1_1;
39         IF rv_Animal_1_1 = FALSE
40         THEN    resultado := resultado || '* RV_Animal_1_1 ';
41         END IF;
```

Código 7.9 RV da entidade "Animal"(versão estendida) - parte 3.

```
1
2           IF (animal_catRegAnimal NOTNULL) AND
3               (UPPER(animal_situacaoAnimal) = 'AC' OR
4                UPPER(animal_situacaoAnimal) = 'GE')
5           THEN rv_Animal_1_2 := (animal_tipoIdFisicoRegistro NOTNULL);
6           ELSE rv_Animal_1_2 := TRUE;
7           END IF;
8           RAISE NOTICE 'Resultado rv_Animal_1_2 = %', rv_Animal_1_2;
9           IF rv_Animal_1_2 = FALSE
10          THEN    resultado := resultado || '* RV_Animal_1_2 ';
11          END IF;
12
13          IF (animal_catRegAnimal NOTNULL) AND
14              (UPPER(animal_situacaoAnimal) <> 'AC' OR
15               UPPER(animal_situacaoAnimal) <> 'GE')
16          THEN rv_Animal_1_3 := (animal_tipoIdFisicoRegistro NOTNULL);
17          ELSE rv_Animal_1_3 := TRUE;
18          END IF;
19          RAISE NOTICE 'Resultado rv_Animal_1_3 = %', rv_Animal_1_3;
20          IF rv_Animal_1_3 = FALSE
21          THEN    resultado := resultado || '* RV_Animal_1_3 ';
22          END IF;
23
24          IF (animal_corDePeloAnimal NOTNULL)
25          THEN  SELECT INTO pkPesquisa r1.pk
26                FROM Raca r1
27                JOIN ValorEnumerado ON ValorEnumerado.pk = r1.pknomeraca
28                WHERE ValorEnumerado.mnemonico = animal_racaAnimal
29                AND  EXISTS
30                     (SELECT 1 FROM CorPeloRaca c1
31                       JOIN CorPeloAnimal c2
32                       ON c1.pkcorpeloanimal = c2.pk
33                       JOIN ValorEnumerado
34                       ON ValorEnumerado.pk = c2.pknomecorpelo
35                      WHERE c1.pkraca = r1.pk AND
36                       ValorEnumerado.mnemonico = animal_corDePeloAnimal);
37                  IF FOUND
38                  THEN rv_Animal_2 := TRUE;
39                  ELSE rv_Animal_2 := FALSE;
40                  END IF;
41          ELSE rv_Animal_2 := TRUE;
42          END IF;
43          RAISE NOTICE 'Resultado rv_Animal_2 = %', rv_Animal_2;
44          IF rv_Animal_2 = FALSE
45          THEN    resultado := resultado || '* RV_Animal_2 ';
46          END IF;
```

Código 7.10 RV da entidade "Animal"(versão estendida) - parte 4.

```
1
2           IF (UPPER(animal_sexoAnimal) = 'MASCULINO' AND
3               UPPER(animal_eraAnimal) <> 'FILHOTEMACHO') AND
4               (UPPER(animal_situacaoAnimal) <> 'AC' OR
5                UPPER(animal_situacaoAnimal) <> 'GE')
6           THEN rv_Animal_3_1 :=
7                                (UPPER(animal_condicaoReprodAnimal)
8                                  = 'CASTRADO' OR
9                                 UPPER(animal_condicaoReprodAnimal)
10                                 = 'RUFIAO' OR
11                                UPPER(animal_condicaoReprodAnimal)
12                                 = 'INTEIRO' );
13          ELSE rv_Animal_3_1 := TRUE;
14          END IF;
15          RAISE NOTICE 'Resultado rv_Animal_3_1 = %', rv_Animal_3_1;
16          IF rv_Animal_3_1 = FALSE
17          THEN    resultado := resultado || '* RV_Animal_3_1 ';
18          END IF;
19
20          IF (UPPER(animal_sexoAnimal) = 'FEMININO' AND
21              UPPER(animal_eraAnimal) <> 'FILHOTEFEMEA') AND
22              (UPPER(animal_situacaoAnimal) <> 'AC' OR
23               UPPER(animal_situacaoAnimal) <> 'GE')
24          THEN rv_Animal_3_2 :=
25                               (UPPER(animal_condicaoReprodAnimal)
26                                 = 'PRENHA' OR
27                                UPPER(animal_condicaoReprodAnimal)
28                                 = 'PRENHAAM' OR
29                                UPPER(animal_condicaoReprodAnimal)
30                                 = 'VAZIA' OR
31                                UPPER(animal_condicaoReprodAnimal)
32                                 = 'VAZIAAM');
33          ELSE rv_Animal_3_2 := TRUE;
34          END IF;
35          RAISE NOTICE 'Resultado rv_Animal_3_2 = %', rv_Animal_3_2;
36          IF rv_Animal_3_2 = FALSE
37          THEN    resultado := resultado || '* RV_Animal_3_2 ';
38          END IF;
```

Código 7.11 RV da entidade "Animal"(versão estendida) - parte 5.

```
1
2           IF (UPPER(animal_eraAnimal) = 'FILHOTEMACHO' OR
3               UPPER(animal_eraAnimal) = 'FILHOTEFEMEA') AND
4               (UPPER(animal_situacaoAnimal) <> 'AC' OR
5                UPPER(animal_situacaoAnimal) <> 'GE')
6           THEN rv_Animal_3_3 := (animal_condicaoReprodAnimal ISNULL);
7           ELSE rv_Animal_3_3 := TRUE;
8           END IF;
9           RAISE NOTICE 'Resultado rv_Animal_3_3 = %', rv_Animal_3_3;
10          IF rv_Animal_3_3 = FALSE
11          THEN    resultado := resultado || '* RV_Animal_3_3 ';
12          END IF;
13
14          IF (UPPER(animal_situacaoAnimal) = 'AC' OR
15              UPPER(animal_situacaoAnimal) = 'GE')
16          THEN rv_Animal_3_4 := (animal_condicaoReprodAnimal ISNULL);
17          ELSE rv_Animal_3_4 := TRUE;
18          END IF;
19          RAISE NOTICE 'Resultado rv_Animal_3_4 = %', rv_Animal_3_4;
20          IF rv_Animal_3_4 = FALSE
21          THEN    resultado := resultado || '* RV_Animal_3_4 ';
22          END IF;
23
24          IF (UPPER(animal_sexoAnimal) = 'FEMININO' AND
25              UPPER(animal_eraAnimal) = 'FILHOTEFEMEA') OR
26             (UPPER(animal_situacaoAnimal) = 'AC' OR
27              UPPER(animal_situacaoAnimal) = 'GE')
28          THEN rv_Animal_4_1 := (animal_tipoMatrizAnimal ISNULL);
29          ELSE rv_Animal_4_1 := TRUE;
30          END IF;
31          RAISE NOTICE 'Resultado rv_Animal_4_1 = %', rv_Animal_4_1;
32          IF rv_Animal_4_1 = FALSE
33          THEN    resultado := resultado || '* RV_Animal_4_1 ';
34          END IF;
```

Código 7.12 RV da entidade "Animal"(versão estendida) - parte 6.

```
1
2           IF (UPPER(animal_sexoAnimal) = 'MASCULINO') OR
3               (UPPER(animal_situacaoAnimal) = 'AC' OR
4                UPPER(animal_situacaoAnimal) = 'GE')
5           THEN rv_Animal_4_2 := (animal_tipoMatrizAnimal ISNULL);
6           ELSE rv_Animal_4_2 := TRUE;
7           END IF;
8           RAISE NOTICE 'Resultado rv_Animal_4_2 = %', rv_Animal_4_2;
9           IF rv_Animal_4_2 = FALSE
10          THEN    resultado := resultado || '* RV_Animal_4_2 ';
11          END IF;
12
13          IF (UPPER(animal_situacaoAnimal) = 'AC' OR
14              UPPER(animal_situacaoAnimal) = 'GE')
15          THEN rv_Animal_4_3 := (animal_tipoMatrizAnimal ISNULL);
16          ELSE rv_Animal_4_3 := TRUE;
17          END IF;
18          RAISE NOTICE 'Resultado rv_Animal_4_3 = %', rv_Animal_4_3;
19          IF rv_Animal_4_3 = FALSE
20          THEN    resultado := resultado || '* RV_Animal_4_3 ';
21          END IF;
22
23          IF (animal_registroAnimal NOTNULL AND
24              animal_situacaoRegistroAnimal NOTNULL AND
25              animal_nomeRegistroAnimal NOTNULL AND
26              animal_tipoIdFisicoRegistro NOTNULL)  AND
27              (UPPER(animal_situacaoAnimal) <> 'AC' OR
28               UPPER(animal_situacaoAnimal) <> 'GE')
29          THEN rv_Animal_5 := (animal_maeAnimal NOTNULL
30                          AND animal_paiAnimal NOTNULL);
31          ELSE rv_Animal_5 := TRUE;
32          END IF;
33          RAISE NOTICE 'Resultado rv_Animal_5 = %', rv_Animal_5;
34          IF rv_Animal_5 = FALSE
35          THEN    resultado := resultado || '* RV_Animal_5 ';
36          END IF;
```

Código 7.13 RV da entidade "Animal"(versão estendida) - parte 7.

```
1
2           IF (UPPER(animal_situacaoAnimal) <> 'AC' OR
3               UPPER(animal_situacaoAnimal) <> 'GE')
4           THEN rv_Animal_6 := (animal_localizacaoAnimal NOTNULL);
5           ELSE rv_Animal_6 := TRUE;
6           END IF;
7           RAISE NOTICE 'Resultado rv_Animal_6 = %', rv_Animal_6;
8           IF rv_Animal_6 = FALSE
9           THEN    resultado := resultado || '* RV_Animal_6 ';
10          END IF;
11
12          IF (UPPER(animal_situacaoAnimal) <> 'AC' OR
13              UPPER(animal_situacaoAnimal) <> 'GE')
14          THEN rv_Animal_7 := (animal_tipoIdFisicoManejo NOTNULL);
15          ELSE rv_Animal_7 := TRUE;
16          END IF;
17          RAISE NOTICE 'Resultado rv_Animal_7 = %', rv_Animal_7;
18          IF rv_Animal_7 = FALSE
19          THEN    resultado := resultado || '* RV_Animal_7 ';
20          END IF;
21
22          IF (animal_idAnimalSisbov ISNULL) OR
23             (UPPER(animal_situacaoAnimal) = 'AC' OR
24              UPPER(animal_situacaoAnimal) = 'GE')
25          THEN rv_Animal_8 := (animal_tipoIdSisbovAnimal ISNULL);
26          ELSE rv_Animal_8 := TRUE;
27          END IF;
28          RAISE NOTICE 'Resultado rv_Animal_8 = %', rv_Animal_8;
29          IF rv_Animal_8 = FALSE
30          THEN    resultado := resultado || '* RV_Animal_8 ';
31          END IF;
32
33          IF (UPPER(animal_situacaoAnimal) = 'AC' OR
34              UPPER(animal_situacaoAnimal) = 'GE')
35          THEN rv_Animal_9 := (animal_tipoIdFisicoRegistro ISNULL);
36          ELSE rv_Animal_9 := TRUE;
37          END IF;
38          RAISE NOTICE 'Resultado rv_Animal_9 = %', rv_Animal_9;
39          IF rv_Animal_9 = FALSE
40          THEN    resultado := resultado || '* RV_Animal_9 ';
41          END IF;
```

Código 7.14 RV da entidade "Animal"(versão estendida) - parte 8.

```
1
2           IF (UPPER(animal_situacaoAnimal) <> 'GE')
3           THEN rv_Animal_10 := (animal_dataNascAnimal NOTNULL);
4           ELSE rv_Animal_10 := TRUE;
5           END IF;
6           RAISE NOTICE 'Resultado rv_Animal_10 = %', rv_Animal_10;
7           IF rv_Animal_10 = FALSE
8           THEN    resultado := resultado || '* RV_Animal_10 ';
9           END IF;
10
11          IF (UPPER(animal_situacaoAnimal) <> 'GE')
12          THEN rv_Animal_11_1 := (animal_proprietariosPJAnimal NOTNULL OR
13                                  animal_proprietariosPFAnimal NOTNULL);
14          ELSE rv_Animal_11_1 := TRUE;
15          END IF;
16          RAISE NOTICE 'Resultado rv_Animal_11_1 = %', rv_Animal_11_1;
17          IF rv_Animal_11_1 = FALSE
18          THEN    resultado := resultado || '* RV_Animal_11_1 ';
19          END IF;
20
21          IF (UPPER(animal_situacaoAnimal) <> 'GE')
22          THEN rv_Animal_11_2 := (animal_criadorPesJurAnimal NOTNULL OR
23                                  animal_criadorPesFisAnimal NOTNULL);
24          ELSE rv_Animal_11_2 := TRUE;
25          END IF;
26          RAISE NOTICE 'Resultado rv_Animal_11_2 = %', rv_Animal_11_2;
27          IF rv_Animal_11_2 = FALSE
28          THEN    resultado := resultado || '* RV_Animal_11_2 ';
29          END IF;
30
31          SELECT INTO pkPesquisa v1.pk
32          FROM  VarRaca v1 JOIN RacaVarRaca r1 ON v1.pk = r1.pkvarraca
33                JOIN Raca r2 ON r2.pk = r1.pkraca
34                JOIN ValorEnumerado ON ValorEnumerado.pk = r2.pknomeraca
35          WHERE ValorEnumerado.mnemonico = animal_racaAnimal;
36          IF FOUND
37          THEN rv_Animal_12_1 := (animal_varRacaAnimal NOTNULL);
38          ELSE rv_Animal_12_1 := TRUE;
39          END IF;
40          RAISE NOTICE 'Resultado rv_Animal_12_1 = %', rv_Animal_12_1;
41          IF rv_Animal_12_1 = FALSE
42          THEN    resultado := resultado || '* RV_Animal_12_1 ';
43          END IF;
```

Código 7.15 RV da entidade "Animal"(versão estendida) - parte 9.

```
1
2           IF (animal_varRacaAnimal NOTNULL)
3           THEN
4                SELECT INTO pkPesquisa r1.pk
5                FROM  VarRaca v1 JOIN RacaVarRaca r1 ON v1.pk = r1.pkvarraca
6                      JOIN Raca r2 ON r2.pk = r1.pkraca
7                      JOIN ValorEnumerado ON ValorEnumerado.pk = r2.pknomeraca
8                WHERE ValorEnumerado.mnemonico = animal_racaAnimal;
9                IF FOUND
10               THEN rv_Animal_12_2 := TRUE;
11               ELSE rv_Animal_12_2 := FALSE;
12               END IF;
13          ELSE rv_Animal_12_2 := TRUE;
14          END IF;
15          RAISE NOTICE 'Resultado rv_Animal_12_2 = %', rv_Animal_12_2;
16          IF rv_Animal_12_2 = FALSE
17          THEN    resultado := resultado || '* RV_Animal_12_2 ';
18          END IF;
19
20          RETURN resultado;
21
22  END;
23  $$ LANGUAGE plpgsql;
```

O Código 7.16 ilustra um exemplo completo de representação de uma RN do tipo Regra de Derivação no nível de projeto (camada interna). Este código representa o mapeamento de OCL para SQL do Código 6.11.

Código 7.16 RD da entidade "Animal"(versão estendida).

```
1  --------------------------
2  Mapeamento da RN para SQL:
3  --------------------------
4
5  CREATE OR REPLACE FUNCTION derivar_Animal_eraAnimal (
6   animal_sexoAnimal VARCHAR,
7   animal_dataNascAnimal DATE
8  ) RETURNS VARCHAR AS $$
9  DECLARE
10         qtde_dias       INTEGER;
11         era             VARCHAR;
12 BEGIN
13         SELECT INTO qtde_dias (CURRENT_DATE - animal_dataNascAnimal);
14         RAISE NOTICE 'Qtde Dias -> %', qtde_dias;
15         IF qtde_dias >= 1 AND qtde_dias <= 210
16         THEN
17               IF animal_sexoAnimal = 'M'
18               THEN era := 'FilhoteMacho';
19               ELSE era := 'FilhoteFemea';
20               END IF;
21         ELSE  IF qtde_dias > 210 and qtde_dias <= 550
22               THEN
23                     IF animal_sexoAnimal = 'M'
24                     THEN era := 'JovemMacho';
25                     ELSE era := 'JovemFemea';
26                     END IF;
27               ELSE
28                     IF qtde_dias > 550
29                     THEN  IF animal_sexoAnimal = 'M'
30                           THEN era := 'AdultoMacho';
31                           ELSE era := 'AdultoFemea';
32                           END IF;
33                     ELSE era := NULL;
34                     END IF;
35               END IF;
36         END IF;
37         RETURN era;
38 END;
39 $$ LANGUAGE plpgsql;
```

Os Códigos 7.17 até 7.19 ilustram um exemplo completo de uma representação de uma RN do tipo RV de Exportação no nível de projeto (camada interna); esses códigos representam o mapeamento de OCL para SQL dos Códigos 6.12 e 6.13. A RI correspondente é apresentada no Código 7.20. Este código representa o mapeamento de OCL para SQL do Código 6.14.

Código 7.17 RE da entidade "TipoIdPessoaFisica"(versão estendida) - parte 1.

```
1  --------------------------
2  Mapeamento da RN para SQL:
3  --------------------------
4  CREATE OR REPLACE FUNCTION validar_PesFis_NumDocIdPesFis (
5          pesfis_numDocIDPesFis VARCHAR,
6          pesfis_tipoDocIDPesFis VARCHAR) RETURNS VARCHAR AS $$
7  DECLARE
8          resultado               VARCHAR;
9          re_NumDocIdPesFis_1     BOOL;
10         d1              INTEGER;
11         d2              INTEGER;
12         d3              INTEGER;
13         d4              INTEGER;
14         d5              INTEGER;
15         d6              INTEGER;
16         d7              INTEGER;
17         d8              INTEGER;
18         d9              INTEGER;
19         dv1             INTEGER;
20         dv2             INTEGER;
21         soma1           INTEGER;
22         soma2           INTEGER;
23         resultado1      INTEGER;
24         resultado2      INTEGER;
25         tamanhoTipoDoc  INTEGER;
26         pkTipoIDPessoaFisica_var INTEGER;
27 BEGIN
28         resultado := '';
29         d1 := 0;
30         d2 := 0;
31         d3 := 0;
32         d4 := 0;
33         d5 := 0;
```

Código 7.18 RE da entidade "TipoIdPessoaFisica"(versão estendida) - parte 2.

```
1
2  d6 := 0;
3  d7 := 0;
4  d8 := 0;
5  d9 := 0;
6  dv1 := 0;
7  dv2 := 0;
8  soma1 := 0;
9  soma2 := 0;
10 resultado1 := 0;
11 resultado2 := 0;
12 tamanhoTipoDoc := 0;
13
14 SELECT INTO pkTipoIDPessoaFisica_var TipoIDPessoaFisica.pk
15 FROM TipoIDPessoaFisica JOIN ValorEnumerado
16 ON ValorEnumerado.pk = TipoIDPessoaFisica.pkNomeTipoDocIDPesFis
17 WHERE ValorEnumerado.mnemonico = pesfis_tipoDocIDPesFis;
18
19 IF pkTipoIDPessoaFisica_var ISNULL THEN RETURN FALSE; END IF;
20
21 SELECT INTO tamanhoTipoDoc TipoIDPessoaFisica.tamanhotipodocidpesfis
22 FROM TipoIDPessoaFisica
23 WHERE TipoIDPessoaFisica.pk = pkTipoIDPessoaFisica_var;
24
25 IF tamanhoTipoDoc <> 11
26 THEN re_NumDocIdPesFis_1 := FALSE;
27 ELSE  IF LENGTH(pesfis_numDocIdPesFis) <> tamanhoTipoDoc
28       THEN  re_NumDocIdPesFis_1 := FALSE;
29       ELSE  d1 := CAST ((SUBSTR(pesfis_numDocIdPesFis,1,1)) AS INTEGER);
30             d2 := CAST ((SUBSTR(pesfis_numDocIdPesFis,2,1)) AS INTEGER);
31             d3 := CAST ((SUBSTR(pesfis_numDocIdPesFis,3,1)) AS INTEGER);
32             d4 := CAST ((SUBSTR(pesfis_numDocIdPesFis,4,1)) AS INTEGER);
33             d5 := CAST ((SUBSTR(pesfis_numDocIdPesFis,5,1)) AS INTEGER);
34             d6 := CAST ((SUBSTR(pesfis_numDocIdPesFis,6,1)) AS INTEGER);
35             d7 := CAST ((SUBSTR(pesfis_numDocIdPesFis,7,1)) AS INTEGER);
36             d8 := CAST ((SUBSTR(pesfis_numDocIdPesFis,8,1)) AS INTEGER);
37             d9 := CAST ((SUBSTR(pesfis_numDocIdPesFis,9,1)) AS INTEGER);
38           dv1 := CAST ((SUBSTR(pesfis_numDocIdPesFis,10,1)) AS INTEGER);
39           dv2 := CAST ((SUBSTR(pesfis_numDocIdPesFis,11,1)) AS INTEGER);
40         soma1 := (d1*10)+(d2*9)+(d3*8)+(d4*7)+(d5*6)+(d6*5)
41                  +(d7*4)+(d8*3)+(d9*2);
42         soma2 := ((d1*11)+(d2*10)+(d3*9)+(d4*8)+(d5*7)+(d6*6)
43                  +(d7*5)+(d8*4)+(d9*3))+(dv1*2);
```

Código 7.19 RE da entidade "TipoIdPessoaFisica"(versão estendida) - parte 3.

```
1
2       resultado1 := soma1-soma1/11*11;
3       resultado2 := soma2-soma2/11*11;
4           IF resultado1 = 1 OR resultado1 = 0
5           THEN   IF dv1 = 0
6                  THEN    IF resultado2 = 1 OR resultado2 = 0
7                          THEN    IF dv2 = 0
8                                  THEN re_NumDocIdPesFis_1 := TRUE;
9                                  ELSE re_NumDocIdPesFis_1 := FALSE;
10                                 END IF;
11                         ELSE    IF dv2 = (11 - resultado2)
12                                 THEN re_NumDocIdPesFis_1 := TRUE;
13                                 ELSE re_NumDocIdPesFis_1 := FALSE;
14                                 END IF;
15                         END IF;
16                 ELSE re_NumDocIdPesFis_1 := FALSE;
17                 END IF;
18          ELSE   IF dv1 = (11 - resultado1)
19                 THEN   IF resultado2 = 1 OR resultado2 = 0
20                        THEN    IF dv2 = 0
21                                THEN re_NumDocIdPesFis_1 := TRUE;
22                                ELSE re_NumDocIdPesFis_1 := FALSE;
23                                END IF;
24                         ELSE   IF dv2 = (11 - resultado2)
25                                THEN re_NumDocIdPesFis_1 := TRUE;
26                                ELSE re_NumDocIdPesFis_1 := FALSE;
27                                END IF;
28                         END IF;
29                 ELSE re_NumDocIdPesFis_1 := FALSE;
30                 END IF;
31           END IF;
32         END IF;
33  END IF;
34
35  RAISE NOTICE 'Resultado re_NumDocIdPesFis_1 = %', re_NumDocIdPesFis_1;
36  IF re_NumDocIdPesFis_1 = FALSE
37  THEN   resultado := resultado || '* RE_NumDocIdPesFis_1 ';
38  END IF;
39
40  RETURN resultado;
41
42 END;
43 $$ LANGUAGE plpgsql;
```

Código 7.20 RI da entidade "PesFis"(versão estendida).

```
1  --------------------------
2  Mapeamento da RN para SQL:
3  --------------------------
4
5  CREATE OR REPLACE FUNCTION importar_PesFis_NumDocIdPesFis  (
6          pesfis_tipoDocIDPesFis VARCHAR ) RETURNS REFCURSOR AS $$
7  DECLARE
8          csr REFCURSOR;
9          pkTipoIDPessoaFisica_var INTEGER;
10 BEGIN
11         SELECT INTO pkTipoIDPessoaFisica_var TipoIDPessoaFisica.pk
12         FROM TipoIDPessoaFisica JOIN ValorEnumerado ON ValorEnumerado.pk =
13          TipoIDPessoaFisica.pkNomeTipoDocIDPesFis
14         WHERE ValorEnumerado.mnemonico = pesfis_tipoDocIDPesFis;
15
16         OPEN csr FOR
17
18         SELECT
19          RISem.RISemRegraIdent,
20          RISem.RISemRegraTecnica,
21          enumRISemRegraOper.mnemonico,  enumRISemRegraTipo.mnemonico
22         FROM TipoEntAtributoRISem JOIN RISem
23              ON TipoEntAtributoRISem.pkRISem = RISem.pk
24              JOIN ValorEnumerado AS enumRISemRegraOper
25              ON RISem.pkRISemRegraOper =  enumRISemRegraOper.pk
26              JOIN ValorEnumerado AS enumRISemRegraTipo
27              ON RISem.pkRISemRegraTipo =  enumRISemRegraTipo.pk
28         WHERE
29          TipoEntAtributoRISem.pkTipoIdPessoaFisica
30          = pkTipoIDPessoaFisica_var;
31
32         RETURN csr;
33 END;
34 $$ LANGUAGE plpgsql;
```

Os Códigos 7.21 e 7.22 ilustram um exemplo completo de representação de uma RN do tipo Regra de Validação no nível de projeto (camada interna). Esses códigos representam o mapeamento de OCL para SQL dos Código 6.15.

Código 7.21 RV da entidade "OcParto"(versão estendida) - parte 1.

```
1  --------------------------
2  Mapeamento da RN para SQL:
3  --------------------------
4
5  CREATE OR REPLACE FUNCTION validar_OcParto (
6   ocParto_animalOcParto VARCHAR,
7   ocParto_pesoVacaOcParto DECIMAL(15,2),
8   ocParto_dataOcParto DATE
9  )
10 RETURNS VARCHAR AS $$
11 DECLARE
12   sexoAnimal      VARCHAR;
13   indValorMinPVP  DECIMAL(15,2),
14   indValorMaxPVP  DECIMAL(15,2);
15   dtOcParto       DATE;
16   dtOcAborto      DATE;
17   indValorIPA     INTEGER;
18   dif_1           INTEGER;
19   dif_2           INTEGER;
20   rv_OcParto_1    BOOL;
21   rv_OcParto_2    BOOL;
22   rv_OcParto_3    BOOL;
23   resultado       TEXT;
24 BEGIN
25
26    resultado := '';
27
28    SELECT INTO sexoAnimal a1.sexoAnimal
29    FROM Animal a1
30    WHERE a1.idAnimal = ocParto_animalOcParto;
31    rv_OcParto_1 := (UPPER(sexoAnimal) = 'FEMININO');
32
33    RAISE NOTICE 'Resultado rv_OcParto_1 = %', rv_OcParto_1;
34    IF rv_OcParto_1 = FALSE
35    THEN resultado := resultado || '* RV_OcParto_1 ';
36    END IF;
```

Código 7.22 RV da entidade "OcParto"(versão estendida) - parte 2.

```
1
2     SELECT INTO indValorMinPVP IndRef.PVPMin FROM IndRef;
3     SELECT INTO indValorMaxPVP IndRef.PVPMax FROM IndRef;
4     rv_OcParto_2 := (ocParto_pesoVacaOcParto >= indValorMinPVP
5                     AND ocParto_pesoVacaOcParto <= indValorMaxPVP);
6
7     RAISE NOTICE 'Resultado rv_OcParto_2 = %', rv_OcParto_2;
8     IF rv_OcParto_2 = FALSE
9     THEN resultado := resultado || '* RV_OcParto_2 ';
10    END IF;
11
12    SELECT INTO dtOcParto o1.dataOcParto
13    FROM OcParto o1
14    WHERE o1.animalOcParto = parto_animalOcParto;
15    SELECT INTO dtOcAborto o2.dataOcAborto
16    FROM OcAborto o2
17    WHERE o2.animalOcAborto = parto_animalOcParto;
18    SELECT INTO indValorIPA IndRef.IPA FROM IndRef;
19    dif_1 := SELECT DATE parto_dataOcParto - DATE dtOcParto;
20    dif_2 := SELECT DATE parto_dataOcParto - DATE dtOcAborto;
21    rv_OcParto_3 := (dif_1 >= indValorIPA OR dif_2 >= indValorIPA);
22
23    RAISE NOTICE 'Resultado rv_OcParto_3 = %', rv_OcParto_3;
24    IF rv_OcParto_3 = FALSE
25    THEN resultado := resultado || '* RV_OcParto_3 ';
26    END IF;
27
28    RETURN resultado;
29
30 END;
31 $$ LANGUAGE plpgsql;
```

Os Códigos 7.23 a 7.25 exemplificam como os métodos apresentados nos Códigos 7.4 a 7.6 foram instanciados no módulo ***SIACorte*** (cenário descrito na Seção 4.3).

Código 7.23 Um exemplo de uso do método apresentado no Código 7.4.

```
1
2  public class ObjetoNegocio {
3
4   // Foi suprimido esta parte código.
5
6          // Verifica se os atributos do metadados estão consistentes
7          public boolean ehConsistente(TipoOperacoes oper){
8
9                  // Tratamento da consistência no Contexto de Atributos.
10                 for (int i = 0; i < this.metadados.obterNumAtributos();  i++)
11                         if (!consistir(metadados.obterAtributoEm(i),
12                                 null, this.objAp.obterCampoEm(i), oper )){
13                                 System.out.println(metadados.
14                                   obterAtributoEm(i).obterMnemonico());
15                                 return false; // Dado Inconsistente!
16                         }
17
18                 // Verificação da consistência no Contexto de Entidades.
19                 // Realiza a invocação da atividade
20                 // "Gerenciamento da RN", com exceção da RD.
21                 if (!verificarRegras("", oper, TipoTempoOperacaoRegra.ANTES)){
22                         return false;
23                 }
24
25                 // Todos os dados estão consistentes
26                 return true;
27         }
28
29   // Foi suprimido o restante do código.
30 }
```

Código 7.24 Um exemplo de uso do método apresentado no Código 7.5.

```
1
2  public class ObjetoNegocio {
3
4   // Foi suprimido esta parte código.
5
6          public boolean inserir(String mneEnt){
7
8                  // Foi suprimido esta parte código.
9
10                 // Procedimentos para implantação de uma RN.
11                 if ( metadados.obterAtributoEm(i).
12                         obterMnemonico().equalsIgnoreCase("RISemRegraTecnica") ){
13
14                         // Realiza a invocação da atividade "Implantação da RN".
15                         if ( !implantarRegra(objAp.obterCampoEm(i).toString()) ){
16                                 return false;
17                         }
18
19                 }
20
21                 // Foi suprimido o restante do código.
22
23                 // OK!
24                 return true;
25         }
26
27   // Foi suprimido o restante do código.
28 }
```

Código 7.25 Um exemplo de uso do método apresentado no Código 7.6.

```
1
2  public class AplicacaoCadastro extends Aplicacao{
3
4   // Foi suprimido esta parte código.
5
6          // Método responsável pelo tratamento do Atributo
7          // que possui Regra de Derivação relacionada.
8          private void tratarDerivadosEmObjetoApresentacao(
9                  Metadados metadados, ObjetoApresentacao ent){
10
11                 String mne = "";
12                 Object valorDerivado = null;
13
14                 for (int k = 0; k < metadados.obterNumAtributos(); k++) {
15                         if( (metadados.obterAtributoEm(k).obterTipo()
16                                  == TipoAtributo.BASICO ||
17                                  metadados.obterAtributoEm(k).obterTipo()
18                                  == TipoAtributo.ENUMERADO)
19                                   && ((AtributoSimples)metadados.
20                                   obterAtributoEm(k)).ehDerivado() ) {
21
22                                 mne = metadados.obterAtributoEm(k).
23                                 obterMnemonico();
24
25                         // Realiza a invocação da atividade
26                         // "Gerenciamento da RN" para RD.
27                                 valorDerivado =
28                                  obterValorDerivado(mne, ent, metadados);
29
30                                 ent.removerCampo(k);
31                                 ent.adicionarCampoEm(
32                                   k, tratarDados(metadados.obterAtributoEm(k),
33                                   valorDerivado));
34
35                           // Foi suprimido o restante do código.
36                         }
37         }
38
39   // Foi suprimido o restante do código.
40
41 }
42
```

CAPÍTULO 8

Conclusões

Neste capítulo descrevemos as considerações finais sobre nosso trabalho (Seção 8.1), analisamos trabalhos correlatos na Seção 8.2, destacamos as contribuições na Seção 8.3, e discutimos possíveis extensões e trabalhos futuros na Seção 8.4.

8.1 Considerações Finais

Este trabalho apresentou uma proposta para tratamento de regras de negócio no contexto do produto de software em Sistemas de Informação. Este tratamento abrange especificação, modelagem e implementação da RN. O foco do tratamento é o acréscimo de uma forma de representação do conhecimento do negócio, ou seja, uma incorporação do conhecimento do negócio no software da aplicação.

Na abordagem proposta para especificação de RN introduzimos uma taxonomia para caracterizar RN em SI. Esta taxonomia permite compreender a natureza das RN, facilitando a sua validação com relação ao negócio subjacente ao SI, e também a sua implementação no software do SI.

A abordagem proposta para implementação de RN faz uso de uma arquitetura de software que dedica uma parte da sua estrutura exclusivamente às RN.

O principal benefício desta proposta é a portabilidade e a disponibilidade das RN.

Nas abordagens existentes atualmente, as regras se transformam em código fonte de programas computacionais e/ou em esquemas de bancos de dados, sem que haja um tratamento prévio que expresse a RN em uma declaração independente de implementação específica.

Uma prática típica que reflete este fato é implementar RN diretamente em uma tecnologia de BD (por exemplo: algum dialeto de SQL de um SGBD específico), ou em uma tecnologia da Engenharia de Software (por exemplo, criação de funções via uma linguagem de programação). Neste contexto, caso seja preciso atender uma demanda evolutiva que exija a inclusão de outro SGBD ou de uma nova linguagem de programação, as RN devem ser reescritas, porque as regras foram escritas em tecnologias computacionais específicas. Isto reflete a falta de uma declaração da regra em uma linguagem ou modelo independente de tecnologia e de uma etapa que transforme essa regra em PIM na nova tecnologia desejada.

Nossa proposta permite que a RN seja declarada em OCL, uma linguagem PIM, de forma a definir o algoritmo da regra em alto nível de abstração. Isso facilita o entendimento do modelo do negócio, pois a linguagem exige um conhecimento técnico comum entre os profissionais de TI (OO, Teoria de Conjuntos, Lógica de Primeira Ordem), e também possibilita melhor leitura, validação e otimização da regra.

A RN escrita em PIM possibilita a geração automática para qualquer PSM desejado, que é uma das diretrizes recomendadas pela abordagem MDA [46].

A falta de disponibilidade da RN pode acarretar deficiências em atributos adjacentes às RN, tais como, localização, documentação e compreensão da semântica da RN. Tais atributos nas implementações tradicionais costumam ser atendidos através de documentos textuais contendo as especificações das RN ou via declarações de regras fixas em algum local dentro do código fonte da aplicação. Essas documentações dificilmente são mantidas atualizadas com o código fonte durante o processo de desenvolvimento ou manutenção do software. Além disso, tais documentações não possuem uma localização lógica da regra que disponibilize o acesso à RN quando requisitada por algum interessado do negócio. Assim, a tendência é que essas documentações se percam pela codificação do software, e isso gera regras não documentadas apropriadamente e de difícil acesso.

Uma solução para estes problemas é dada em nossa proposta, a qual viabiliza a manutenção de um repositório de regras parametrizáveis, adaptáveis para vários idiomas e modeladas através de uma linguagem PIM. Esta solução proporciona uma melhor qualidade das RN sem deixar de manter a consistência dessas regras com o contexto do negócio.

A separação das RN do restante do código fonte é outro benefício da solução proposta, pois as regras não ficam espalhadas pelo código fonte da aplicação. Com esta organização centralizada das regras, é possível conhecer e gerenciar as regras contidas no sistema, identificando as que se aplicam em cada conceito (entidade ou atributo) de negócio, e proporcionando mais clareza para manutenção dessas regras. Com isso é possível agilizar o processo de desenvolvimento ou manutenção das RN.

Nossa proposta permite utilizar as duas abordagens tradicionais (Banco de Dados e Engenharia de Software), pois apenas acrescentamos uma forma de representação do conhecimento do negócio em um nível de abstração mais alto, porém diretamente ligado ao código que implementa a RN. Sob este ponto de vista, nossa proposta é mais eficiente, porque a abordagem proposta traz as RN mais próximas do software, isto é, há um estreitamento da distância existente entre a modelagem e a implementação dessas regras, sem perder a portabilidade e a disponibilidade da RN.

O mecanismo proposto permite que a aplicação controle a execução das RN através de chamadas a um componente de regras, e não através da incorporação das regras ao código fonte. Para modificar uma regra, não há necessidade de compilar o código fonte do sistema, basta compilar a regra.

A proposta também inclui uma metodologia para o tratamento das RN que pode ser inserida dentro de um processo de desenvolvimento ou manutenção de software. Esta metodologia pode trazer maior clareza na definição e mais agilidade na modificação de regras dentro do ciclo de vida de um SI.

8.2 Trabalhos Correlatos

Alguns trabalhos têm demonstrado a aplicabilidade de abordagens semelhantes à solução aqui proposta [9, 27, 32, 40, 41, 48]. Tais trabalhos confirmam as vantagens de modelagem abstrata de regras de negócio ou resultados satisfatórios obtidos na construção de uma parte do software dedicado às regras de negócio.

Os trabalhos [8, 9, 47, 48] ratificam o fato de que é possível a construção de um mecanismo para gerenciamento de regras de negócio que envolve a representação das regras em linguagem formal e independente de tecnologia, como OCL. Estes trabalhos também discutem como gerar o código fonte para a implantação da regra e permitir a verificação da consistência dessas regras sob o controle da aplicação ou de um SGBD específico.

Os trabalhos [47, 48] apresentam o projeto e a implementação da ferramenta Atena; em [47] foi descrito o primeiro protótipo da ferramenta, e em [48] é apresentada uma versão mais atualizada do sistema implementado. A ferramenta é uma máquina de execução de RN escritas em uma linguagem formal (OCL, por exemplo), construída sobre um SGBD Relacional. Esta ferramenta permite acoplar uma camada de regras de negócio em uma aplicação já existente, com todo o gerenciamento das RN centralizado na ferramenta.

Em resumo, as principais características da ferramenta Atena são: 1) armazenamento de uma base formal para a documentação das RN de um SI; 2) um extrator automático de regras (extração via esquema do BD, via cadastro pelos usuários e via inferência a partir das outras duas); 3) um editor de RN (esta funcionalidade também permite a criação de referências para o código fonte que avalia a regra em questão, isto é, armazena o nome do arquivo e a linha onde a regra deve ser avaliada); 4) um gerador de lista de eventos (inserção, alteração, etc) versus RN (ou seja, encontra os eventos que disparam as regras e colocam no código em locais previamente armazenados onde cada evento ocorre); 5) um gerador do código da regra, isto é, as regras escritas em OCL são convertidas para a linguagem SQL, no nível de DML ou DDL, utilizando um compilador de OCL para SQL (as regras mais complexas não podem gerar automaticamente código SQL eficiente para validá-las, e para tratar essa situação, é previsto na ferramenta a geração de código procedural, com o auxílio de programadores); 6) para regras consideradas simples são criados *check constraints* no SGBD e para os demais casos é a aplicação que deve invocar o procedimento de validação da ferramenta Atena (este mecanismo de verificação da regra atua antes de efetivar uma transação, e caso alguma regra seja violada ele retorna uma identificação do erro para ser tratado pela aplicação); 7) a consulta base para regras restritivas verifica quais tuplas violam a regra, ou seja, verifica se existe alguma tupla onde a negação da expressão OCL é verdadeira, neste caso indica que a regra foi violada.

Da mesma forma que a ferramenta Atena, nossa proposta também viabiliza a manutenção de um repositório de RN. O modelos conceitual e de implementação para realização de uma RN comprovam essa característica. As regras também são modeladas através da OCL e, além disso, são especificadas em linguagem natural e adaptáveis para vários idiomas. Nosso modelo conceitual para realização de uma RN não viabiliza o armazenamento dos arquivos e linhas no código fonte da aplicação onde as regras deverão ser validadas como é feito na ferramenta Atena.

Em nossa abordagem identificamos o comportamento e o aspecto de tempo associados com a RN. Quanto aos locais de invocação da regra, o projetista do software é quem deve decidir o local no código fonte da aplicação que deverá instanciar nosso componente que trata as RN. O mecanismo desse componente faz o acionamento das funções armazenadas de acordo com a taxonomia da regra, faz o tratamento dos parâmetros do negócio explicitados na declaração da RN, verifica se a RN está em conformidade com o comportamento e o tempo para ser validada, e se tudo estiver correto, realiza o teste de validação ou derivação da regra.

A validação das RN na ferramenta Atena é feita pelo SGBD nos casos de regras simples e nos demais casos a aplicação deve invocar o mecanismo de validação da ferramenta. Em nossa abordagem padronizamos o comportamento de validação das regras somente através da aplicação.

A aplicação que faz uso de nossos modelos deve disponibilizar a edição das RN através da sua camada de interface com o usuário. Tal característica é disponibilizada na ferramenta Atena por um editor de regras independente. O motivo por não contemplarmos essa característica em nossa solução proposta é que não pretendemos atender o porte completo de uma máquina de regras de negócio (ou *business rules engine*).

No escopo da nossa proposta não atendemos extração automática das regras como é feito na ferramenta Atena. A captação das regras é permitida através do cadastro eletrônico de regras pelo profissional envolvido no projeto da RN; uma pequena parte da tradução das expressões OCL para SQL é feita automaticamente, e grande parte deve ser feita manualmente pelo projetista ou programador. Porém, propomos padrões para escrita da regra em OCL para cada tipo de RN, que facilita a transformação para funções armazenadas em um SGBD (escritas em SQL).

Em [8] discute-se o uso das linguagens OCL, UML e SQL na modelagem de restrições de integridade em BD. Este trabalho cita algumas vantagens dessa combinação de linguagens: UML é uma linguagem amplamente aceita para modelagem OO e é suportada por muitas ferramentas CASE; OCL provê uma linguagem abstrata e precisa para especificar restrições de integridade como invariantes em um modelo OO; os SGBDs Relacionais atuais oferecem diferentes dialetos de SQL para implementar asserções e gatilhos e com o uso de OCL a especificação das restrições de integridade torna-se independente da escolha do SGBD Relacional. Especificações de operações através de pré e pós-condições em OCL são um bom início para geração automática de código SQL para consultas ou gatilhos em BD correspondentes a essas operações. Neste artigo propõe-se um conjunto de padrões que descrevem o mapeamento de expressões em OCL para código em SQL. Este mapeamento consiste basicamente na criação de asserções em SQL.

A proposta do nosso trabalho segue uma sistemática semelhante à de [8], no que concerne o uso da linguagem OCL para modelar RN, especialmente aquelas com conotação restritiva, e da linguagem SQL como forma de implementação das RN em um SGBD Relacional. Porém nosso trabalho apresenta diferenças significativas: para a modelagem da RN, o formato padrão usado é a especificação de operações e para cada tipo de RN determinamos o uso de um estereótipo da OCL; em relação ao mapeamento da regra em OCL para SQL utilizamos, principalmente, a criação de funções armazenadas em SGBD. Também em comum com [8] temos a idéia do padrão Operação, o qual orienta o uso de linguagens procedurais para tradução de

expressões complexas em OCL para SQL como, por exemplo, expressões que utilizam a operação *Iterate*. Logo, instanciamos a idéia desse padrão e a aplicamos para regras de validação e de derivação, visando atender tanto regras simples quanto complexas.

Em [9] é apresentada uma investigação sobre como as regras escritas em OCL podem ser manipuladas em aplicações orientadas a objetos que fazem uso de SGBDs e diferentes mapeamentos objeto-relacionais. O artigo explica o projeto da ferramenta OCL2SQL, que é um gerador de código SQL adaptável para diferentes SGBDs Relacionais. Este gerador faz a interpretação de um modelo de objetos na linguagem UML e do mapeamento objeto-relacional, obtém as regras escritas em OCL deste modelo UML, e gera um script em SQL, incluindo um esquema de BD em DDL, mais definições de visões e gatilhos.

A idéia básica proposta em [9] é utilizar as visões geradas a partir das expressões invariantes em OCL. O resultado da avaliação de uma visão é um conjunto de tuplas inválidas da tabela restringida relacionada aos objetos que violam a RN especificada. Este trabalho define regras de negócio como restrições definidas pelo usuário; e dentro do contexto de RN considera somente um subconjunto de restrições de integridade que são suportadas por SGBDs. Restrições dessa natureza para OCL são as invariantes. A geração de código SQL declarativo (visões e gatilhos) a partir de invariantes OCL, segundo [9], é mais simples do que a geração de código procedural para SGBDs, pois um código declarativo está sujeito à otimização de consultas pelos SGBDs. No entanto, isto só funciona bem para regras simples, mas no caso de regras mais complexas não há alternativa senão o uso de código procedural.

Assim como em [9] também nossa solução visa a flexibilidade para gerar esquemas SQL adaptáveis a diferentes SGBDs Relacionais, permitindo diferentes mapeamentos objeto-relacionais. A principal diferença está na opção de mapeamento OCL para SQL: ao invés de visões e gatilhos, adotamos a criação de funções armazenadas.

O uso de gatilhos não atende adequadamente os propósitos da nossa proposta, especificamente em algumas tarefas de implementação das RN, tais como: obter os parâmetros da RN dinamicamente via aplicação e obter a RN associada com o conceito do negócio dinamicamente (é o caso do comportamento de uma Regra de Exportação combinado com uma Regra de Importação). Neste caso, necessitamos que o controle esteja totalmente a cargo do mecanismo que gerencia as RN em conjunto com a aplicação, evitando quaisquer efeitos indesejáveis provocados por gatilhos (discutidos em [48]).

A proposta de [9] é basicamente a verificação da regra de negócio através da avaliação de uma visão de BD que retorna tuplas inválidas. Quando o BD é atualizado (isto é, quando ocorre alguma transação de inserção, alteração ou remoção de tuplas), imediatamente antes da transação ser efetivada, é executada a verificação das regras através de uma visão que retorna todas as tuplas que violam essas regras.

Já o mecanismo da nossa proposta permite que a aplicação execute a verificação da regra em dois momentos: antes ou depois de uma transação disparada via uma operação da interface da aplicação (esta operação corresponde a algum comportamento essencial do negócio que está associado com a RN em questão).

Em [1] é apresentada a ferramenta Régula que visa atender a modelagem de sistemas

baseada em regras de negócio. Esta ferramenta provê funcionalidades para auxiliar o projetista do software na fase de requisitos, tais como: captura de parte dos requisitos através das regras de negócio (utiliza gabaritos em português estruturado que orientam o projetista na construção da regra); permite consultas a permissões, proibições e obrigações; gera o modelo de informação a partir da definição dos termos do negócio; e gera regras executáveis (tradução da regra escrita em português estruturado para a linguagem *Prolog*). O projeto dessa ferramenta, assim como a nossa proposta, utiliza o modelo proposto pelo *Business Rules Group* [40].

Divergindo da ferramenta Régula [1], nosso trabalho não propõe gabaritos para escrita da RN em linguagem natural, pois acreditamos que a escrita deve ser feita pelo projetista da RN juntamente com o analista de negócio na forma de declarações que expressam a semântica da RN em uma linguagem compreendida pelo proprietário do negócio. Isto permite ainda adaptar as regras a diferentes idiomas. Essas declarações também podem ser tratadas pela aplicação via algum mecanismo que trate mensagens na camada de interface de usuário, a fim de informar ao usuário da aplicação o texto exato da RN que foi violada.

Em busca de ferramentas CASE para modelagem das RN em OCL, analisamos as ferramentas: *ArgoUML* [44], *Poseidon* [14], *Together Designer* [2], *Dresden ToolKit* [37] e *Odyssey-PSW* [5].

As ferramentas *ArgoUML*, *Poseidon* e *Together Designer* permitem a edição de restrições OCL em um modelo UML e realizam a compilação dessas restrições através de um componente integrado, que é parte do *Dresden ToolKit* [37]. Este *toolkit* contempla um pacote abrangente de soluções para utilizar os recursos da linguagem OCL, incluindo: uma biblioteca de componentes para integração do *toolkit* com outras ferramentas, um compilador de OCL e um gerador de código SQL ou JAVA a partir de restrições OCL dadas em um modelo OO. O trabalho [9] utiliza as facilidades do *Dresden ToolKit*. Citamos aqui duas referências sobre as origens deste *toolkit*: [12, 17].

Maiores detalhes sobre as ferramentas *ArgoUML*, *Poseidon*, *Together Designer* e *Dresden ToolKit* em relação ao suporte dado à OCL (métodos e ferramentas) podem ser consultados em [3], que apresenta um estudo sobre o estado da arte na geração automática de restrições de integridade definidas em um esquema conceitual.

A ferramenta Odyssey-PSW [5] oferece um suporte a elaboração, verificação e validação de modelos UML que contenham restrições especificadas em OCL. A oferta de ferramentas de suporte a OCL ainda é pequena, e grante parte delas se limita à avaliação sintática e à verificação de tipos das restrições especificadas em OCL. Logo, esta ferramenta contribuiu nesse sentido e, além do suporte oferecido às restrições OCL em um modelo UML, também contribui com o gerenciamento de casos de teste que apoiam a validação dessas restrições.

O ponto comum da ferramenta Odyssey-PSW com nosso trabalho é que trata de restrições em OCL e faz validação sintática dessas restrições. Porém, esta validação é feita diretamente no modelo UML, já em nosso trabalho é feita quando a regra é implantada no sistema (nesse momento a regra está registrada no cadastro eletrônico de RN). Esta ferramenta não realiza nenhum mapeamento de restrições OCL para uma linguagem de implementação. Isto também difere do nosso trabalho, o qual permite realizar este mapeamento.

Dentre os trabalhos estudados ([1, 8, 9, 47, 48]) não constatamos um tratamento para RN do tipo derivação, bem como sobre a peculiaridade de RN que são proprietárias de um conceito do negócio que visam validar um outro conceito do negócio; tais trabalhos dedicam-se à modelagem de expressões invariantes da OCL e nenhuma metodologia para tratamento das RN foi explicitamente mencionada. Em nossa proposta, utilizamos outros estereótipos da OCL para modelagem das RN, objetivando definir regras que expressam a validação da consistência das informações do negócio, bem como regras que expressam um conhecimento a partir de outros conhecimentos do negócio. Propomos também uma taxonomia para classificar RN em SI, e uma metodologia para tratamento das RN, a qual pode ser inserida nos processos de desenvolvimento e de manutenção de software de uma organização.

8.3 Contribuições

A principal contribuição desta dissertação é uma proposta completa para tratamento de RN em SI abrangendo especificação, modelagem e implementação. A proposta foi validada em um cenário de um SI complexo (projeto ***SIACorte***, na área de Agronegócios [7]).

A metodologia proposta pode ser utilizada ou instanciada separadamente. Por exemplo: se o projetista fizer a formalização da análise das RN com outro método como, por exemplo, especificações de caso de uso, essas especificações podem ser utilizadas como insumo para as próximas fases da metodologia. Portanto, o projetista deve extrair as RN contidas nessas especificações e, em seguida, poderá utilizar as fases de projeto (Capítulo 6) e de implementação (Capítulo 7) propostas na metodologia.

Esta situação é possível mas não é desejável, pois a metodologia proposta é uma solução integrada, embora seja constituída de partes com interfaces bem definidas. Em outras palavras, a utilização da metodologia como um todo é melhor do que o uso das partes isoladas, pois é uma solução completa para o problema de tratamento de RN em SI. O uso das partes isoladamente é útil quando a organização já possui algum *framework* de implementação de RN.

Como subprodutos do trabalho de elaboração e aplicação de uma nova abordagem para RN em SI, outras contribuições significativas surgiram:

- um estudo esclarecendo e unificando o significado dos termos regras de negócio, requisitos e restrições de integridade no contexto de SI.
  Este estudo é importante justamente para demonstrar que terminologias diferentes são usadas com mesma finalidade semântica em áreas diferentes.
- uma nova taxonomia de RN orientadas a software em SI.
  Classificar as RN é essencial para melhor entendimento e implementação das regras. A taxonomia proposta é uma extensão da taxonomia introduzida em [40]. Partindo desta taxonomia, acrescentamos novas classes que visam facilitar a implementação dessas regras utilizando as tecnologias disponíveis atualmente.
- um estudo e uma validação empírica do uso de paradigma de regras de negócio combinado com as diretrizes da MDA.

Este estudo analisa a possibilidade de implementação das RN como parte separada na construção do software em SI, conforme sugerido em [40, 47, 48].
A validação empírica realizada no contexto do projeto ***SIACorte*** [7] comprova a viabilidade da abordagem proposta neste trabalho.
As diretrizes da MDA orientaram, principalmente, a aplicação da OCL na fase de modelagem das RN, a qual oferece benefícios significativos na compreensão dos modelos de RN do SI.

8.4 Trabalhos Futuros

Este trabalho admite melhorias e extensões. Consideramos que o principal foco para trabalhos futuros seja a manutenção de regras de negócio classificadas e tratadas através das técnicas propostas.

É interessante validar nossa proposta em outros estudos de caso abordando diversos domínios de aplicações a fim de avaliar a generalidade da solução proposta e a qualidade dos seus artefatos em diferentes contextos. O mecanismo proposto foi avaliado quanto às características de um SI específico. Porém, a abordagem pode ser utilizada em outros SI, já que a base da proposta é a definição de regras de negócio para as entidades de negócio, independentemente do código da aplicação específica. Esses conceitos básicos certamente poderão ser portados para outros SI.

Uma melhoria importante a ser feita é em relação à atividade de Implantação da RN: é preciso completar os procedimentos de transformação da RN em OCL para SQL para automatizar toda a transformação. Também é necessário atender outros dialetos de SQL oferecidos por diferentes SGBDs relacionais.

Uma extensão sugerida é definir os procedimentos de transformação da RN em OCL para JAVA e automatizar toda essa transformação, assim como permitir que a regra seja transformada para outras linguagens (PSM).

Dado o modelo conceitual do Modelo de Meta-Objetos, uma melhoria relevante é separar aspectos de apresentação de dados na camada de apresentação (ou seja, dados específicos para visualização na interface de usuário) dos aspectos de representação de dados (ou seja, dados específicos para armazenamento persistente).

Referências Bibliográficas

[1] ARAUJO, B. M; DIAS, F. G; MORGADO, G. P; MARTINS, A. E; SILVEIRA, D. S; MANSO, F. S. P; LIMA, P. M. V; SCHMITZ, E. A. **Uma Ferramenta para a Captura de Requisitos de Software através de Regras de Negócio**. In: XX SIMPóSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE (SBES), p. 37–42, Florianópolis, SC, Outubro 2006.

[2] BORLAND. **Borland Together Designer**. http://www.borland.com/together/designer/index.html.

[3] CABOT, J; TENIENTE, E. **Generación Automática de Restricciones de Integridad: Estado del Arte**. In: II TALLER SOBRE DESARROLLO DE SOFTWARE DIRIGIDO POR MODELOS, MDA Y APLICACIONES (DSDM), p. 48–56, Granada, Espanha, Setembro 2005.

[4] CHIOREAN, D; DEMUTH, B; GOGOLLA, M; WARMER, J. **OCL for (Meta-)Models in Multiple Application Domains**. Relatório Técnico ISSN 1430-211X, Technische Universität Dresden, Dresden, Alemanha, Setembro 2006.

[5] CORREA, A. L; WERNER, C. M. L. **Odyssey-PSW: Uma Ferramenta de Apoio à Verificação e Validação de Especificações de Restrições OCL**. In: XX SIMPóSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE (SBES), p. 61–66, Florianópolis, SC, Outubro 2006.

[6] DATE, C. J. **What, not How - The Business Rules Approach to Application Development**. Addison-Wesley, 2000.

[7] DE OLIVEIRA, J. L. **Tecnologia da Informação para Otimização da Produção de Gado de Corte.** Relatório Técnico Final 505198/2004-5, Projeto de Pesquisa e Desenvolvimento - CNPq, Goiânia, Goiás, Abril 2007.

[8] DEMUTH, B; HUSSMANN, H. **Using UML/OCL Constraints for Relational Database Design**. In: INTERNATIONAL CONFERENCE ON THE UNIFIED MODELING LANGUAGE, p. 598–613, Fort Collins, Colorado, USA, Outubro 1999.

[9] DEMUTH, B; HUSSMANN, H; LOECHER, S. **OCL as a Specification Language for Business Rules in Database Applications**. In: INTERNATIONAL CONFERENCE ON THE UNIFIED MODELING LANGUAGE, p. 104–117, Toronto, Ontario, Canada, Outubro 2001.

[10] ELMASRI, R; NAVATHE, S. B. **Sistemas de Banco de Dados**. Addison Wesley, 2005. 4ª edição.

[11] FAYAD, M. E; JOHNSON, R. E; SCHMIDT, D. C. **Building Application Frameworks: Object-Oriented Foundations of Framework Design**. Wiley, 1999. 1ª edição.

[12] FINGER, F. **Design and Implementation of a Modular OCL Compiler**. PhD thesis, Dresden University of Tecnology, Dresden, Alemanha, 2000.

[13] GAGNON, E. M. **An Object-Oriented Compiler Framework**. Master's thesis, School of Computer Science, McGill University, Montreal, Quebec, Canada, Março 1998.

[14] GENTLEWARE. **Poseidon for UML**. http://www.gentleware.com.

[15] GUILARDUCCI, G; DE OLIVEIRA, J. L. **Uma Proposta para Tratamento de Restrições de Integridade Semânticas em Sistemas de Informação**. In: Anais do Encontro de Tecnologia e Informática (ETI), Goiânia, Goiás, Setembro 2006. Instituto de Informática Universidade Federal de Goiás. Resumo.

[16] GUILARDUCCI, G; DE OLIVEIRA, J. L. **Uma Proposta para Tratamento de Restrições de Integridade Semânticas em Sistemas de Informação**. In: Anais do III Congresso de Pesquisa, Ensino e Extensão (CONPEEX) / III Seminário de Pós-Graduação, Goiânia, Goiás, Outubro 2006. Universidade Federal de Goiás. Resumo expandido.

[17] HUSSMANN, H; DEMUTH, B; FINGER, F. **Modular Architecture for a Toolset Supporting OCL**. In: INTERNATIONAL CONFERENCE ON THE UNIFIED MODELING LANGUAGE, p. 51–69, York, UK, Outubro 2000.

[18] ISO/IEC. **ISO/IEC 9126-1, Software engineering — Product Quality — Part 1: Quality Model**, 2001.

[19] ISO/IEC. **ISO/IEC 12207, Software Life Cycle Processes**, 2002.

[20] JACKSON, M. **Software Requirements & Specifications a lexicon of practice, principles and prejudices**. Addison-Wesley Professional, Agosto 1995. 1ª edição.

[21] KLEPPE, A; WARMER, J; BAST, W. **MDA Explained: The Model Driven Architecture: Practice and Promise**. Addison Wesley, 2003. 1ª edição.

[22] KOTONYA, G; SOMMERVILLE, I. **Requirements Engineering Processes and Techniques**. John Wiley & Sons Ltd., 1998.

[23] LAENDER, A; JR., C. A. D; BORGES, K. A. V. **Restrições de Integridade em Bancos de Dados Geográficos**. In: III WORKSHOP BRASILEIRO DE GEOINFORMáTICA (GEOINFO), Rio de Janeiro, RJ, Outubro 2001.

[24] LEFFINGWELL, D; WIDRIG, D. **Managing Software Requirements - A Unified Approach**. Addison-Wesley, 2000.

[25] MAIA, N. E. N; BLOIS, A. P. T. B; WERNER, C. M. **Odyssey-MDA: Uma Abordagem para a Transformação de Modelos de Componentes**. In: WORKSHOP DE DESENVOLVIMENTO BASEADO EM COMPONENTES (WDBC), volume 1, p. 89–96, Juiz de Fora, MG, 2005.

[26] MILLER, J; MUKERJI, J. **MDA Guide Version 1.0.1**. Object Management Group, Inc., Junho 2003.

[27] MORGAN, T. **Business Rules and Information Systems: Aligning IT with Business Goals**. Addison Wesley, 2001.

[28] Object Management Group, Inc. **OCL 2.0 Specification**, Junho 2005.

[29] O´BRIEN, J. A. **Sistemas de Informação e as Decições Gerenciais na Era da Internet**. Editora Saraiva, 2004. 2ª edição.

[30] PRESSMAN, R. S. **Software Engineering - A Practitioner´s Approach**. Mc Graw Hill, 2005. 6ª edição.

[31] REITERT, R. **On Integrity Constraints**. In: on Database Systems (TODS), A. T, editor, II CONFERENCE ON THEORETICAL ASPECTS OF REASONING ABOUT KNOWLEDGE, p. 97–111, San Francisco, CA, USA, Março 1988.

[32] ROSS, R. G. **Expressing Business Rules**. SIGMOD Rec., 29(2):515–516, 2000.

[33] SILBERSCHATZ, A; KORTH, H. F; SUDARSHAN, S. **Sistema de Banco de Dados**. Campus, 2006. 5ª edição.

[34] SOFTWARE ENGINEERING INSTITUTE. **CMMI for Development, Version 1.2**. Technical Report CMU/SEI-2006-TR-008, Carnegie Mellon University, Pittsburgh, PA, Agosto 2006.

[35] Software Engineering Standards Committee of the IEEE Computer Society. **IEEE Standard 1233, IEEE Guide for Developing System Requirements Specifications**, Dezembro 1998.

[36] Software Engineering Standards Committee of the IEEE Computer Society. **IEEE Standard 830, IEEE Recommended Practice for Software Requirements Specifications**, Junho 1998.

[37] SOFTWARE TECHNOLOGY GROUP, TECHNISCHE UNIVERSITÄT DRESDEN. **Dresden OCL Toolkit**. http://http://dresden-ocl.sourceforge.net/index.html.

[38] SOMMERVILLE, I. **Engenharia de Software**. Addison Wesley, 2003. 6ª edição.

[39] SOMMERVILLE, I; SAWYER, P. **Requirements Engineering: A good practice guide**. John Wiley & Sons Ltd., 1997.

[40] THE BUSINESS RULES GROUP. **GUIDE Business Rules Project: Final Report - Defining Business Rules - What Are They Really?** http://www.businessrulesgroup.org/first_paper/br01c0.htm, Julho 2000.

[41] THE BUSINESS RULES GROUP. **Business Rules Manifesto - The Principles of Rule Independence**. http://www.businessrulesgroup.org, Novembro 2003.

[42] THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. **PostgreSQL**. http://www.postgresql.org/.

[43] THE WORLD WIDE WEB CONSORTIUM (W3C) . **Extensible Markup Language (XML)**. http://www.w3.org/XML/.

[44] TIGRIS.ORG. **ArgoUML**. http://argouml.tigris.org.

[45] WAHLER, M; KOEHLER, J; BRUCKER, A. D. **Model-Driven Constraint Engineering**. In: WORKSHOP ON OCL FOR META-MODELS IN MULTIPLE APPLICATION DOMAINS AT THE UML/MODELS CONFERENCE, p. 111–125, Genova, Itália, Outubro 2006.

[46] WARMER, J; KLEPPE, A. **The Object Constraint Language: Getting Your Models Reading for MDA**. Addison Wesley, 2003. 2ª edição.

[47] ZIMBRÃO, G; DE ALMEIDA, V. T; SOUZA, J. M; SULAIMAN, A; NETO, F. P. **ATENAS: Um Sistema Gerenciador de Regras de Negócio**. In: XV SIMPóSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE (SBES), p. 338–343, Rio de Janeiro, RJ, Outubro 2001.

[48] ZIMBRÃO, G; MIRANDA, R; SOUZA, J. M; ESTOLANO, M. H; NETO, F. P. **Enforcement of Business Rules in Relational Databases Using Constraints**. In: XVIII SIMPóSIO BRASILEIRO DE BANCO DE DADOS (SBBD), p. 129–141, Manaus, AM, Outubro 2003.

[49] ZSIFKOV, N; CAMPEANU, R. **Business Rules Domains and Business Rules Modeling**. In: INTERNATIONAL SYMPOSIUM ON INFORMATION AND COMMUNICATION TECHNOLOGIES (ISICT), p. 172–177, Las Vegas, Nevada, Junho 2004. Trinity College Dublin.

APÊNDICE A

Glossário

A Tabela A.1 apresenta o significado dos termos utilizados neste trabalho.

Tabela A.1: *Glossário*

Termo	Descrição
ABNT/NBR	Associação Brasileira de Normas Técnicas/Norma Brasileira.
Agronegócio	Sistema constituído de cadeias produtivas compostas de fornecedores de insumos e serviços, produção agropecuária, indústria de processamento e transformação, agentes de distribuição e comercialização, tendo como objetivo comum suprir o consumidor de produtos de origem agropecuária e florestal.
AST	*Abstract Syntax Tree*.
BD	Banco de Dados ou Bancos de Dados.
CASE	*Computer-Aided Software Engineering*.
CMMI	*Capability Maturity Model Integration*.
CNPq	Conselho Nacional de Desenvolvimento Científico e Tecnológico.
CORBA	*Common Object Request Broker Architeture*.
DDL	*Data Definition Language*.
Design	Projeto arquitetural detalhado de um software.
DFD	Diagrama de Fluxo de Dados.
DML	*Data Manipulation Language*.
DTD	*Document Type Definition*.
DTE	Diagrama de Transição de Estados.
ERP	*Enterprise Resource Planning*.
GUI	*Graphic User Interface*.
IA	Inteligência Artificial.
IEC	*International Electrotechnical Commission*.
IEEE	*Institute of Electrical and Electronics Engineers*.
ISO	*International Organization for Standardization*.
Java Beans	Modelo de componente independente de plataforma escrito na linguagem de programação Java.

continua na próxima página

Tabela A.1: *Glossário (continuação)*

Termo	Descrição
MDA	*Model-Driven Architecture*.
MER	Modelo Entidade-Relacionamento.
MMO	Modelo de Meta-Objetos.
MOF	*Meta Object Facility*.
OCL	*Object Constraint Language*.
ODBC	*Open Data Base Connectivity*.
OMG	*Object Management Group*.
OO	Orientação a Objetos ou Orientado a Objetos.
ORM	*Object Role Modeling*.
PIM	*Plataform Independent Model*.
POO	Paradigma de Orientação a Objetos.
PSM	*Plataform Specific Model*.
SGBD	Sistema Gerenciador de Bancos de Dados.
SiaCorte	Software que compõe um SI na área de agronegócios, especificamente no domínio de pecuária de corte, dentro do contexto de um projeto de pesquisa apoiado pelo CNPq conforme publicado em [7].
SQL	*Structured Query Language*.
RD	Regra de Derivação ou Regras de Derivação.
RE	RN de Exportação. RN que é exportada para outro conceito do negócio.
RI	Restrição de Integridade ou Restrições de Integridade.
RImp	Regra de Importação ou Regras de Importação. RN cujas declarações expressam como importar uma RN que é propriedade de outro conceito do negócio.
RN	Regra de Negócio ou Regras de Negócio.
RV	Regra de Validação ou Regras de Validação.
SI	Sistema de Informação ou Sistemas de Informação.
TI	Tecnologia da Informação.
UML	*Unified Modeling Language*.
XML	*Extensible Markup Language*.

APÊNDICE B

Dicionário do Modelo de Meta-Objetos

Apresentamos o dicionário conceitual do Modelo de Meta-Objetos conforme proposto em [7] e mostrado na Figura 4.2.

Tabela B.1: *Descrição das classes do MMO (adaptado de [7]).*

Classe	Descrição
Atributo	Caracteriza uma propriedade particular de uma entidade do negócio. Exemplo: "a entidade Propriedade Rural pode ser caraterizada pelo nome e endereço". Vide atributos na Tabela B.2.
Composto	Classifica um Atributo de uma entidade do negócio quando ele é composto por outros atributos. Exemplo: "uma instância de Atributo Composto denominado Endereço é composta pelos atributos: rua, número e bairro".
Domínio Enumerado	Caracteriza um conjunto de valores ou domínio de valores válidos para os Atributos de uma entidade do negócio. Exemplo: "o domínio para o atributo Peso da entidade Animal é Inteiro". Vide atributos na Tabela B.3.
Enumerado	Caracteriza um Atributo de uma entidade do negócio que contém valores que são obtidos de um conjunto de informações pré-estabelecidas. Exemplo: "um atributo Cor de Pelo da entidade Animal pode conter os seguintes valores pré-estabelecidos: preta, branca e marrom".
Objeto Externo	Classifica um Atributo que é um valor ou um dado externo ao software do negócio. Exemplo: "um arquivo de texto, uma imagem, um áudio ou vídeo".
Objeto Interno	Classifica um Atributo que contém a referência para uma entidade do negócio. Exemplo: "uma Propriedade Rural possui um atributo que é uma referência para entidade Proprietário Rural".
Simples	Classifica um Atributo quando ele não é composto por outros atributos. Exemplo: "nome de fantasia da entidade Propriedade Rural".
TipoEnt	É uma entidade do negócio que compartilha propriedades comuns. Exemplo: "Propriedade Rural e Animal". Vide atributos na Tabela B.4.
Valor Enumerado	Caracteriza os elementos ou valores permitidos por um Domínio Enumerado. Exemplo: "os valores permitidos para o Domínio Enumerado Cor são: preta, branca e marrom". Vide atributos na Tabela B.5.

Tabela B.2: *Descrição dos atributos da classe Atributo do MMO (adaptado de [7]).*

Atributo	Descrição
Mnemônico	É a identificação única de um atributo de uma entidade do negócio.
Tipo Atributo	Informa a classificação ou tipo do atributo.
Tipo Domínio	Informa o tipo do domínio (ou tipo de dado) do atributo.
Tamanho	É o tamanho máximo do valor do atributo (ou quantidade de caracteres).
Eh Editável	Define se o atributo é ou não editável na interface gráfica da aplicação.
Cardinalidade Mínima	É a quantidade mínima de valores que o atributo pode ter.
Cardinalidade Máxima	É a quantidade máxima de valores que o atributo pode ter.
Menor Valor	É o menor valor possível para o atributo.
Maior Valor	É o maior valor possível para o atributo.
Regra	Regra de integridade semântica a qual este atributo estará sujeito.
Seqüencial do Atributo	É a posição do atributo na interface gráfica.
Eh Atributo de Ligacao	Define se o atributo é ou não de ligação. Torna o atributo visível quando a sua entidade for referenciada por outra entidade, por exemplo: sejam os tipos de entidade: Programa de Melhoramento e Pessoa Física; em um programa de melhoramento um técnico é uma pessoa física. O elo de ligação entre essas duas entidades pode ser definido através do atributo "nome da pessoa física e cpf", e ambos são definidos como atributos de ligação na entidade Pessoa Física.
Tipo Subject	Define o tipo de *Subject* do atributo. Este conceito faz parte do padrão de projeto "*Observer*" para ser utilizado na interface gráfica."
Tipo Observer	Define o tipo de *Observer* do atributo. Este conceito faz parte do padrão de projeto "*Observer*" para ser utilizado na interface gráfica."
Extensão	Descreve o tipo de extensão de um arquivo externo. Utilizado para Atributo Objeto Externo.
Aplicativo	Descreve o tipo de aplicativo que executa o arquivo externo. Utilizado para Atributo Objeto Externo.
Plataforma	Descreve a plataforma operacional onde o aplicativo é executado. Utilizado para Atributo Objeto Externo.
Valor Default	Valor padrão para o atributo. É usado para apresentação de dados.
Máscara	Descrece a máscara de edição para o valor do atributo.
PtBr	Descreve a tradução do mnemônico do atributo para o idioma Português (Brasil). É usado para apresentação de dados.

continua na próxima página

Tabela B.2: *Descrição dos atributos da classe Atributo do MMO (adaptado de [7]) (continuação).*

Atributo	Descrição
EsEs	Descreve a tradução do mnemônico do atributo para o idioma Espanhol (Espanha). É usado para apresentação de dados.
EnUs	Descreve a tradução do mnemônico do atributo para o idioma Inglês (EUA). É usado para apresentação de dados.
FrFr	Descreve a tradução do mnemônico do atributo para o idioma Francês (França). É usado para apresentação de dados.
AjudaPtBr	Descreve o texto de ajuda do atributo para o idioma Português (Brasil). É usado para apresentação de dados.
AjudaEsEs	Descreve o texto de ajuda do atributo para o idioma Espanhol (Espanha). É usado para apresentação de dados.
AjudaEnUs	Descreve o texto de ajuda do atributo para o idioma Inglês (EUA). É usado para apresentação de dados.
AjudaFrFr	Descreve o texto de ajuda do atributo para o idioma Francês (França). É usado para apresentação de dados.
Eh Único	Define se o atributo é único.
Eh Parte de Chave	Define se o atributo é parte de uma identificação única de atributo.
Filtro	Descreve em SQL uma condição utilizada como um filtro de seleção das instâncias selecionáveis de um Atributo Objeto Interno.
Eh Definido pelo Sistema	Define se o atributo é definido pelo sistema ou pelo usuário.
Classe Executável	Descreve o nome de uma classe JAVA (sem a extensão .java) que executa alguma tarefa.

Tabela B.3: *Descrição dos atributos da classe Domínio Enumerado do MMO (adaptado de [7]).*

Atributo	Descrição
Nome	Descreve o nome único e semântico de um Domínio Enumerado.

Tabela B.4: *Descrição dos atributos da classe TipoEnt do MMO (adaptado de [7]).*

Atributo	Descrição
Mnemônico	É a identificação única de uma entidade do negócio.
Status	Determina a situação de uma entidade do negócio: em modelagem, etc.

Tabela B.5: *Descrição dos atributos da classe Valor Enumerado do MMO (adaptado de [7]).*

Atributo	Descrição
Mnemônico	Descreve o nome único e semântico de um Valor Enumerado.
Nome_PtBr	Descreve a tradução do mnemônico de um Valor Enumerado para o idioma Português (Brasil).
Nome_EsEs	Descreve a tradução do mnemônico de um Valor Enumerado para o idioma Espanhol (Espanha).
Nome_EnUs	Descreve a tradução do mnemônico de um Valor Enumerado para o idioma Inglês (EUA).
Nome_FrFr	Descreve a tradução do mnemônico de um Valor Enumerado para o idioma Francês (França).

APÊNDICE C

Dicionário do Modelo Conceitual para Registro das RN

Apresentamos o dicionário do modelo conceitual para registro das RN mostrado na Figura 5.1.

Tabela C.1: *Descrição das classes do Modelo Conceitual para registro das RN.*

Classe	Descrição
AC	Classifica um comportamento de alteração de um conceito do negócio.
Aspecto de Tempo	Define um aspecto de tempo diretamente associado com a RN.
Atrasado	Classifica um aspecto de tempo o qual informa que a RN é avaliada depois do comportamento do negócio associado à RN em questão.
Comportamento do Negócio	Define um comportamento de um conceito do negócio.
Conceito do Negócio	Define uma informação do negócio, que pode ser expressa por um termo (uma palavra ou uma frase) ou por um fato do negócio (conforme definido na Seção 2.3). Neste modelo conceitual é representado por entidades ou atributos provenientes do modelo do negócio e que originam regras de negócio. O atributo "identificação" é a propriedade onde a identificação lógica e única de um conceito do negócio é representado.
EC	Caracteriza um comportamento de exclusão de um conceito do negócio.
Espanhol	Caracteriza a declaração em linguagem natural da RN no idioma Espanhol (Espanha).
EnumStatusRegra	Valores para status (ou situação) da RN: Especificação (RN no nível de análise); Modelagem (RN na camada conceitual do nível de projeto); Implantação (RN na camada interna do nível de projeto, isto é, a RN está pronta para uso).
Francês	Caracteriza a declaração em linguagem natural da RN no idioma Francês (França).

continua na próxima página

Tabela C.1: *Descrição das classes do Modelo Conceitual para registro das RN (continuação).*

Classe	Descrição
IC	Caracteriza um comportamento de inclusão de um conceito do negócio.
Idioma	Define a declaração em linguagem natural da RN nos principais idiomas. O atributo "texto" é a propriedade onde a declaração em linguagem natural é representada. A associação da classe Idioma com a classe Regra de Negócio apresenta uma restrição: uma RN deve ser escrita, minimamente, no idioma Português.
Imediato	Caracteriza um aspecto de tempo o qual informa que a RN é avaliada antes do comportamento do negócio associado à RN em questão.
Inglês	Caracteriza a declaração em linguagem natural da RN no idioma Inglês (EUA).
JAVA	Caracteriza uma forma de implementação dependente de plataforma para uma RN; neste caso, é a linguagem de programação JAVA.
Linguagem de BD	Define uma forma de implementação dependente de plataforma para uma RN; neste caso, a plataforma alvo é o conjunto de linguagens de BD.
Linguagem de Programação	Define uma forma de implementação dependente de plataforma para uma RN; neste caso, a plataforma alvo é o conjunto de linguagens de programação.
OCL	Caracteriza uma forma de implementação independente de plataforma para uma RN; neste caso, é a linguagem OCL.
Português	Caracteriza a declaração em linguagem natural da RN no idioma Português (Brasil).
Regra de Derivação	Identifica (ou classifica) uma RN que é uma RN do tipo Derivação.
Regra de Negócio	Define as regras de negócio de um conceito do negócio. O atributo "identificação" é a propriedade onde a identificação lógica e única da RN é representada. O atributo "status" indica a situação atual da RN, ou seja, em qual fase da metodologia a RN se enquadra.
Regra de Importação	Identifica (ou classifica) uma regra de importação.
Regra de Validação	Identifica (ou classifica) uma RN que é uma RN do tipo Assertiva Estrutural ou Assertiva de Ação.
Regra PIM	Define uma forma de implementação da RN em um modelo ou linguagem independente de plataforma computacional. O atributo "texto" é a propriedade onde a declaração em linguagem independente de plataforma é representada.

continua na próxima página

Tabela C.1: *Descrição das classes do Modelo Conceitual para registro das RN (continuação).*

Classe	Descrição
Regra PSM	Define uma forma de implementação da RN em um modelo ou linguagem dependente de plataforma computacional. O atributo "texto" é a propriedade onde a declaração em linguagem dependente de plataforma é representada.
SQL_PG	Caracteriza uma forma de implementação dependente de plataforma para uma RN; neste caso, é a linguagem de BD SQL no dialeto no SGBD PostgreSQL.

APÊNDICE D

DTD que valida os Códigos E.1 e F.1

Código D.1 DTD que valida os Códigos E.1 e F.1 (adaptado de [37]).

```
1
2  <?xml version="1.0" encoding="UTF-8"?>
3
4  <!-- Baseado no arquivo MappingRules.dtd de Dresden ToolKit. -->
5
6  <!ELEMENT catalog       (description?, pattern*)>
7  <!ATTLIST catalog       name CDATA #REQUIRED>
8  <!ELEMENT name          (#PCDATA)>
9  <!ELEMENT description   (#PCDATA)>
10 <!ELEMENT pattern       (template)*>
11 <!ATTLIST pattern       rule ID #REQUIRED>
12 <!ELEMENT template      (li)+>
13 <!ATTLIST template      spec CDATA #REQUIRED
14                         rem  CDATA #IMPLIED>
15 <!ELEMENT li            (#PCDATA | param)*>
16 <!ATTLIST li            connector (true|false) "false">
17 <!ELEMENT param         EMPTY>
18 <!ATTLIST param         name CDATA #REQUIRED>
```

APÊNDICE E

Padrões de OCL para SQL - Parte 1

Código E.1 Padrões de OCL para SQL - parte 1 (adaptado de [37]).

```
<?xml version="1.0"?>
<!DOCTYPE catalog SYSTEM "Mapeamento.dtd">
<catalog name="OCL2SQL92_PostgreSQL">
<description>
  Descreve o mapeamento de padrões OCL para SQL. Este catálogo foi criado
  baseado no arquivo OCL2SQL4SQL92.xml que faz parte do Dresden ToolKit.
</description>

<pattern rule="corpo_regra">
  <template spec="funcao" rem="FUNCAO PADRAO">
   <li>CREATE OR REPLACE FUNCTION <param name="nomeFuncao"/> </li>
   <li>(<param name="listaArgumentos"/>) RETURNS VARCHAR AS $$ </li>
   <li>DECLARE</li>
   <li> resultado VARCHAR; </li>
   <li> <param name="variaveis"/> </li>
   <li>BEGIN</li>
   <li> resultado := ''; </li>
   <li> <param name="expressoes"/> </li>
   <li> RETURN resultado; </li>
   <li>END; $$ LANGUAGE plpgsql; </li>
  </template>

  <template spec="derivacao" rem="DERIVACAO PADRAO">
   <li>CREATE OR REPLACE FUNCTION <param name="nomeFuncao"/> </li>
   <li>(<param name="args"/>) RETURNS <param name="tipoRetorno"/> AS $$ </li>
   <li>DECLARE</li>
   <li> <param name="variaveis"/> </li>
   <li>BEGIN</li>
   <li> <param name="expressoes"/> </li>
   <li>END; $$ LANGUAGE plpgsql; </li>
  </template>
</pattern>
</catalog>
```

APÊNDICE F

Padrões de OCL para SQL - Parte 2

Código F.1 Padrões de OCL para SQL - parte 2 (adaptado de [37]).

```
<?xml version="1.0"?>
<!DOCTYPE catalog SYSTEM "Mapeamento.dtd">
<catalog name="OCL2SQL92_PostgreSQL">
<description>
  Descreve o mapeamento de padrões OCL para SQL. Este catálogo foi criado
  baseado no arquivo OCL2SQL4SQL92.xml que faz parte do Dresden ToolKit.
</description>

<pattern rule="tipos_basicos">
  <template spec="string" rem="TIPO BASICO ALFANUMERICO">
        <li>VARCHAR</li>
  </template>
  <template spec="integer" rem="TIPO BASICO INTEIRO">
        <li>INTEGER</li>
  </template>
  <template spec="boolean" rem="TIPO BASICO LÓGICO">
        <li>BOOL</li>
  </template>
  <template spec="real" rem="TIPO BASICO DECIMAL">
        <li>DECIMAL(15,2)</li>
  </template>
  <template spec="data" rem="TIPO BASICO DATA">
        <li>DATE</li>
  </template>
</pattern>
</catalog>
```

APÊNDICE G

Dicionário dos Modelos de Implementação

Apresentamos o dicionário resumido [1] dos componentes representados nos modelos de implementação para tratamento de RN mostrados nas Figuras 7.4 e 7.5. Os tipos de componentes aqui representados são: classes, atributo de classe (ACl), método de classe (MCl), método construtor (MCon), enumerado (Enum), pacote e arquivo (por exemplo: XML e DTD).

Tabela G.1: *Descrição dos componentes que implementam o tratamento de RN.*

Componente	Tipo	Descrição
CodeAgent	Classe.	Responsável pela escolha de modelos de código especificados em um arquivo XML. Este pacote pertence ao *Dresden ToolKit* [37].
criarAST	MCl ObjetoRegras.	Realiza a criação da AST.
destruirAST	MCl ObjetoRegras.	Remove da memória a AST criada.
gerarCodigoSQL	MCl ObjetoRegras.	Realizar o mapeamento da regra em OCL para linguagem alvo (SQL-DML, JAVA, etc).
gerenciarRegras	MCl ObjetoRegras.	Gerenciar o mecanismo de execução dos tipos de RN.
implantarRegra	MCl ObjetoRegras.	Realiza o mecanismo de implantação de uma RN na aplicação.
Mapeamento	DTD.	Utilizado para interpretação do arquivo XML que contém o mapeamento dos padrões OCL para SQL. Baseado no arquivo *MappingRules.dtd* de *Dresden ToolKit* [37].
ObjetoRegra	MCon ObjetoRegra.	Construtor com parâmetros para criação de um objeto ObjetoRegra.
ObjetoRegra	MCon ObjetoRegra.	Construtor padrão para criação de um objeto ObjetoRegra.
ObjetoRegras	Classe.	Responsável pelo tratamento das RN.

continua na próxima página

[1] Suprimimos a assinatura completa do métodos.

Tabela G.1: *Descrição dos componentes que implementam o tratamento de RN (continuação).*

Componente	Tipo	Descrição
obterMsgErroRegra	MCl Objeto-Regra.	Retornar a descrição correta da RN em linguagem natural de acordo com o idioma cadastrado.
ObterPartesDaRegra	Classe interna.	Responsável por obter o conteúdo de algumas partes da RN em OCL.
obterRegra	MCl Objeto-Regra.	Obter todas partes de uma RN.
obterRegraIdent	MCl Regra.	Retorna o conteúdo do atributo regraIdent.
obterRegraTecnica	MCl Regra.	Retorna o conteúdo do atributo regraTecnica.
obterRegraOper	MCl Regra.	Retorna o conteúdo do atributo regraOper.
obterRegraTempoOper	MCl Regra.	Retorna o conteúdo do atributo regraTempoOper.
obterRegraTipo	MCl Regra.	Retorna o conteúdo do atributo regraTipo.
obterStatusRegra	MCl Regra.	Retorna o conteúdo do atributo regraStatus.
obterValorDerivado	MCl Objeto-Regra.	Retornar o conteúdo de um atributo gerado após execução de uma regra de derivação.
ocl.parser	Pacote.	Pacote de classes geradas por ***SableCC*** publicado em [13].
OCL2SQL92_PostgreSQL	XML.	Mapeamento de padrões OCL para SQL.
Regra	Classe.	Implementação de uma Regra de Negócio.
Regra	MCon Regra.	Construtor para criação de um objeto Regra.
regraIdent	ACl Regra.	Identificador lógico da RN.
regraOper	ACl Regra.	Indica as operações e/ou eventos para execução da RN.
regraTecnica	ACl Regra.	Descreve a RN em linguagem técnica e formal (OCL).
regraTempoOper	ACl Regra.	Indica o momento que as operações e/ou eventos executarão a RN.
regraTipo	ACl Regra.	Indica o tipo da RN.
Sablecc.jar	Pacote.	Gerador de *parser* OO [13].
StatusRegra	Enum.	Enumerado que contém as opções de status da RN. Os enumerados são: Especificação (RN no nível de análise); Modelagem (RN na camada conceitual do nível de projeto); Implantação (RN na camada interna do nível de projeto, isto é, a RN está pronta para uso).

continua na próxima página

Tabela G.1: *Descrição dos componentes que implementam o tratamento de RN (continuação).*

Componente	Tipo	Descrição
TipoOperacaoRegra	Enum.	Enumerado que contém as operações que executam uma RN. Os tipos de operações correspondem às principais operações realizadas pela GUI. Os enumerados são: ALTERACAO (Regra executada na operação Alteração), EXCLUSAO (Regra executada na operação Exclusão), INCLUSAO (Regra executada na operação Inclusão), ALTERACAO_EXCLUSAO (Regra executada nas operações Alteração e Exclusão), ALTERACAO_INCLUSAO (Regra executada nas operações Alteração e Inclusão), INCLUSAO_EXCLUSAO (Regra executada nas operações Inclusão e Exclusão), ALTERACAO_INCLUSAO_EXCLUSAO (Regra executada nas operações Alteração, Inclusão e Exclusão), NENHUMA (Usada juntamente com Regra de Importação, pois não corresponde exatamente a uma operação da GUI) **Restrição:** um objeto tipo Regra possui uma restrição no número de operações permitidas. Esta restrição estabelece um conjunto de combinações pré-determinadas; assim sendo temos, um objeto Regra pode ter somente uma das operações: ALTERACAO ou EXCLUSAO ou INCLUSAO ou ALTERACAO_EXCLUSAO ou INCLUSAO_EXCLUSAO ou ALTERACAO_INCLUSAO_EXCLUSAO ou NENHUMA; ou um objeto Regra pode ter a combinação das operações: ALTERACAO + EXCLUSAO + INCLUSAO, ou ALTERACAO_EXCLUSAO + INCLUSAO, ou INCLUSAO_EXCLUSAO + ALTERACAO.
TipoRegra	Enum.	Enumerado que contém os tipos de RN. Os tipos são: DERIVACAO (Regra de Derivação), VALIDACAO (Regra de Validação), EXPORTACAO_DERIVACAO (Regra de Exportação dedicada para uma Regra de Derivação), EXPORTACAO_VALIDACAO (Regra de Exportação dedicada para uma Regra de Validação), IMPORTACAO_DERIVACAO (Regra de Importação dedicada para busca dinâmica de uma Regra de Derivação), IMPORTACAO_VALIDACAO (Regra de Importação dedicada para busca dinâmica de uma Regra de Validação).

continua na próxima página

Tabela G.1: *Descrição dos componentes que implementam o tratamento de RN (continuação).*

Componente	Tipo	Descrição
TipoTempoOperacaoRegra	Enum.	Enumerado que contém os tipos que especificam o momento que as operações executam uma RN. Esses tipos são: ANTES (Regra executada antes de uma dada operação da GUI), DEPOIS (Regra executada depois de uma dada operação da GUI).
tratarConteudo	MCl Objeto-Regras.	Tratar o conteúdo de um parâmetro da RN obtido através da GUI da aplicação.
tratarExecucaoRegra	MCl Objeto-Regras.	Realizar procedimentos para execução da RN.
tratarNomeParamOperRegra	MCl Objeto-Regras.	Retorna um vetor contendo a identificação lógica de um atributo e seu contexto, ou seja, a entidade do negócio que possui o respectivo atributo.
tratarObtencaoPartesDaRegra	MCl Objeto-Regras.	Realizar procedimentos para tratar a obtenção de partes da regra (regra técnica em OCL).
validarRISem	MCl Objeto-Regras.	Realiza a validação sintática da RN, ou seja, se os tipos definidos no modelo de objetos do negócio que estão descritos na regra em OCL constam nos metadados da aplicação.
verificarCompOperacoes	MCl Objeto-Regras.	Verificar a compatibilidade do tipo da operação da GUI com o tipo de operação da RN.
xerces.jar	Pacote.	Pacote de classes utilitárias usadas pela classe *CodeAgent*.