PPGI PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Universidade Federal do Rio de Janeiro

CARLOS ALESSANDRE
ASSIS DA CUNHA

UMA ABORDAGEM PARA A TRANSFORMAÇÃO DE REGRAS DE NEGÓCIO NA ARQUITETURA DIRIGIDA POR MODELOS

DISSERTAÇÃO DE MESTRADO

Instituto de Matemática

Núcleo de Computação Eletrônica

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
NÚCLEO DE COMPUTAÇÃO ELETRÔNICA

CARLOS ALESSANDRE ASSIS DA CUNHA

UMA ABORDAGEM PARA A TRANSFORMAÇÃO DE REGRAS DE NEGÓCIO NA ARQUITETURA DIRIGIDA POR MODELOS

Rio de Janeiro

2009

Carlos Alessandre Assis da Cunha

UMA ABORDAGEM PARA A TRANSFORMAÇÃO DE REGRAS DE NEGÓCIO NA ARQUITETURA DIRIGIDA POR MODELOS

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Informática do Departamento de Ciência da Computação, Instituto de Matemática, Núcleo de Computação Eletrônica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Informática.

Orientador: Eber Assis Schmitz.

Co-orientador: Alexandre Luis Correa.

Rio de Janeiro

2009

Carlos Alessandre Assis da Cunha

UMA ABORDAGEM PARA A TRANSFORMAÇÃO DE REGRAS DE NEGÓCIO NA ARQUITETURA DIRIGIDA POR MODELOS

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Informática do Departamento de Ciência da Computação, Instituto de Matemática e Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro como parte dos requisitos necessários à obtenção do título de Mestre em Informática.

Aprovada em: 13 de abril de 2009.

_____

Prof. Eber Assis Schmitz, Ph.D. – Orientador – UFRJ/DCC/IM-NCE

_____

Prof. Alexandre Luis Correa, D.Sc. – Co-orientador – UFRJ/DCC/IM-NCE

_____

Prof. Antonio Juarez Sylvio Menezes de Alencar, D.Phil. – UFRJ/DCC/IM-NCE

_____

Prof. Márcio de Oliveira Barros, D.Sc. – UNIRIO

Dedicada aos meus queridos pais.

AGRADECIMENTOS

Aos meus pais, José Carlos e Neusa, pelo carinho, por me proporcionarem a oportunidade de estudar e por sempre estarem por perto.

Aos meus orientadores, Eber Assis Schmitz e Alexandre Luis Correa, por acreditarem na minha capacidade e compartilharem comigo os conhecimentos que possuem. Agradeço os momentos de orientação, as revisões cuidadosas e o incentivo constante.

Aos professores Márcio de Oliveira Barros e Antonio Juarez Alencar por, gentilmente, aceitarem o convite para participar da banca examinadora deste trabalho e pelas contribuições e sugestões dadas.

Aos amigos Gustavo Nardelli e Rosa Figueiredo pelo incentivo, apoio, preocupação e companheirismo.

Ao professor Fernando Manso e aos colegas do curso de mestrado pelas contribuições dadas ao longo das reuniões semanais do grupo de pesquisa.

Aos professores que participaram da avaliação deste trabalho nos diversos seminários do programa de mestrado e no *workshop* de teses e dissertações do IV Simpósio Brasileiro de Sistemas de Informação.

À Petrobras por proporcionar aos seus funcionários a oportunidade de continuar crescendo academicamente.

Por fim, agradeço a todos aqueles que, de alguma forma, contribuíram na elaboração deste trabalho.

RESUMO

CUNHA, Carlos Alessandre Assis. **Uma abordagem para a transformação de regras de negócio na arquitetura dirigida por modelos**. 2009. 162 f. Dissertação (Mestrado em Informática) – Instituto de Matemática, Núcleo de Computação Eletrônica, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2009.

O aumento da importância dos sistemas de informação nas organizações estimula o aparecimento de tecnologias que visam tornar mais fácil, mais rápido e menos dispendioso o complexo processo de desenvolvimento de aplicações. A utilização de modelos na construção de sistemas de informação é uma das maneiras possíveis de se lidar com essa complexidade. No arcabouço conceitual definido pela Arquitetura Dirigida por Modelos (MDA), os modelos são os principais artefatos do processo de desenvolvimento, sendo possível agilizar a geração dos mesmos através do uso de transformações automáticas. Dentre os elementos que podem fazer parte da especificação de um sistema de informação, estão as regras de negócio, que são sentenças que definem ou restringem algum aspecto do negócio. Na MDA, existem diversas abordagens com diferentes níveis de abstração para a representação das regras de negócio. Nesse contexto, o presente trabalho propõe uma abordagem para a transformação automática das regras de negócio do Modelo Independente de Computação (CIM) para o Modelo Independente de Plataforma (PIM) da MDA. Neste trabalho, os padrões SBVR (*Semantics of Business Vocabulary and Business Rules*) e OCL (*Object Constraint Language*) foram utilizados na especificação das regras de negócio nos modelos CIM e PIM, respectivamente. A transformação de modelos definida na abordagem foi construída a partir de mapeamentos entre os elementos do metamodelo da SBVR e do metamodelo da OCL. Por fim, a implementação dessa transformação permitiu sua aplicação prática em um conjunto de regras de negócio extraído da documentação da SBVR.

Palavras-chaves: MDA. SBVR. OCL. Regras de negócio. Transformação de modelos.

ABSTRACT

CUNHA, Carlos Alessandre Assis. **Uma abordagem para a transformação de regras de negócio na arquitetura dirigida por modelos**. 2009. 162 f. Dissertação (Mestrado em Informática) – Instituto de Matemática, Núcleo de Computação Eletrônica, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2009.

The growing importance of information systems in organizations encourages the arising of technologies to facilitate the complex process of software development. Models are an option to deal with this complexity. The Model Driven Architecture (MDA) defines a conceptual framework where models are the main artifacts to develop software systems. Automatic models transformations can improve the model building activity in MDA. Business rules are statements defining or constraining some aspects of the business and are useful in the specification of an information system. There are several approaches with different levels of abstraction to represent business rules in the Model Driven Architecture. In this context, this work proposes an approach to the automatic transformation of business rules from the Computation Independent Model (CIM) to the Platform Independent Model (PIM) of MDA. In this approach, business rules at CIM and PIM levels are represented by SBVR (Semantics of Business Vocabulary and Business Rules) and OCL (Object Constraint Language) specifications, respectively. This model transformation was defined by mappings rules between SBVR and OCL metamodels elements. Finally, a transformation tool was developed and applied over a set of business rules in SBVR.

Keywords: MDA. SBVR. OCL. Business rules. Model transformation.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ADT	ATL *Development Tools*
ATL	*Atlas Transformation Language*
BSBR RFP	*Semantics of Business Rules Request for Proposal*
CIM	*Computation Independent Model*
DDM	Desenvolvimento Dirigido por Modelos
EMF	*Eclipse Modeling Framework*
IDE	*Integrated Development Environment*
KM3	*Kernel Meta Meta Model*
LFSV	*Logical Formulation of Semantics Vocabulary*
MDA	*Model Driven Architecture*
MDD	*Model Driven Development*
MOF	*Meta-Object Facility*
MRV	*Meaning and Representation Vocabulary*
OCL	*Object Constraint Language*
OMG	*Object Management Group*
ORM	*Object Role Modeling*
PIM	*Platform Independent Model*
PSM	*Platform Specific Model*
R2ML	REWERSE *Rule Markup Language*
RFP	*Request for Proposal*
SBVR	*Semantics of Business Vocabulary and Business Rules*
TCS	*Textual Concrete Syntax*
TI	Tecnologia da Informação
UML	*Unified Modeling Language*
VDBR	*Vocabulary for Describing Business Rules*
VDBV	*Vocabulary for Describing Business Vocabularies*
XMI	XML *Metadata Interchange*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Atualmente, as organizações operam em um contexto de mudança contínua e em mercados de alta competitividade. A globalização e a liberalização de diversos setores da economia propiciaram a criação e o desaparecimento de produtos e serviços em escala mundial, aumentando significativamente a disponibilidade dos mesmos para os consumidores e acirrando a disputa entre as organizações para manter e ampliar seu espaço de atuação.

Esse cenário dinâmico amplia a percepção de que a Tecnologia da Informação (TI) pode, de fato, auxiliar os negócios, o que leva as organizações a investimentos cada vez mais expressivos nessa área. A TI, nesse contexto, apresenta-se como a nova infra-estrutura das organizações, tornando viáveis novos produtos e serviços, permitindo a criação de estruturas organizacionais inovadoras, o acesso a novos mercados e a diferenciação na prestação de serviços aos consumidores (ALENCAR; SCHMITZ, 2006).

Assegurar que os investimentos em TI tragam retornos para o negócio e, mais especificamente, que os sistemas de informação desenvolvidos atendam às necessidades organizacionais é um grande desafio para as organizações. Aos especialistas em TI, cabe a complexa tarefa de traduzir essas necessidades em sistemas de informação de qualidade que apóiem efetivamente as operações das organizações e proporcionem um diferencial competitivo (MORGADO, 2007).

1.1 Motivação

O crescente aumento da importância dos sistemas de informação estimula o desenvolvimento de tecnologias que buscam tornar mais fácil, mais rápido e menos dispendioso o processo de desenvolvimento de aplicações. Contudo, apesar dos avanços

trazidos por elas, o desenvolvimento de sistemas de informação continua a ser uma atividade inerentemente complexa (BROOKS, 1987; PRESSMAN, 2006).

Um modelo consiste numa coleção de elementos que descreve, usando uma notação formal ou informal, alguma realidade física, abstrata ou hipotética. O uso de modelos no desenvolvimento de sistemas de informação é uma das maneiras possíveis de se lidar com a complexidade desse processo (MELLOR et al., 2004). Nesse contexto, destaca-se o Desenvolvimento Dirigido por Modelos (DDM), que é uma abordagem para o desenvolvimento de sistemas com enfoque no uso de modelos e de transformações entre esses modelos.

Uma das realizações do DDM é a Arquitetura Dirigida por Modelos (*Model Driven Architecture* – MDA), que consiste em um arcabouço conceitual para o desenvolvimento de sistemas de informação. A idéia central é que o desenvolvimento de um sistema seja feito a partir de transformações sucessivas de modelos em diferentes níveis de abstração, desde um modelo independente de computação, passando por um modelo independente de plataforma, para, finalmente, realizar a transformação em um modelo para uma plataforma de implementação específica.

A especificação da MDA não estabelece um processo de desenvolvimento de sistemas de informação, mas apenas um arcabouço conceitual, e também não define como os modelos CIM[1], PIM[2] e PSM[3] devem ser construídos. Assim, ao definir processos de desenvolvimento de sistemas baseados na MDA, é necessário estabelecer quais metamodelos e modelos farão parte de cada um dos modelos do arcabouço e quais são as transformações necessárias até a entrega do produto final.

---

[1] *Computation Independent Model.*

[2] *Platform Independent Model.*

[3] *Platform Specific Model.*

Dentre os elementos que podem fazer parte dos modelos da MDA, estão as regras de negócio (LINEHAN; FERGUSON, 2005). Regras de negócio são sentenças que definem ou restringem algum aspecto do negócio. Elas podem ser vistas como diretrizes de condução, ação, prática ou procedimento de uma atividade em particular da organização (HAY; HEALY, 2000).

A identificação e modelagem das regras de negócio são passos importantes em direção ao entendimento de como uma organização funciona. Além de uma vantagem competitiva importante, esse conhecimento pode auxiliar no processo de desenvolvimento de sistemas de informação, uma vez que as regras de negócio normalmente precisam ser implementadas nos sistemas que apóiam a organização (ROSCA; GREENSPAN; WILD, 2002).

O interesse no uso das regras de negócio no desenvolvimento de sistemas não é algo recente. Apesar dos avanços já obtidos, algumas questões ainda se apresentam como campo para estudo. Para o presente trabalho, destaca-se a necessidade de integração entre as regras de negócio e os processos de desenvolvimento de sistemas de informação, principalmente no que tange a transformação das regras de negócio entre diferentes níveis de abstração, como ocorre na MDA, por exemplo.

Assim, a motivação para o trabalho é a percepção da necessidade de uma forma estruturada de tratamento das regras de negócio durante o processo de desenvolvimento que use uma abordagem dirigida por modelos, de forma a agilizar a geração dos modelos de regras de negócio ao longo da construção de um sistema de informação.

1.2 Caracterização do Problema

A especificação de processos de desenvolvimento de sistemas utilizando MDA considera como artefatos[4] mais importantes os modelos gerados. Esses modelos passam por transformações que são basicamente um conjunto de regras que descrevem como elementos de um modelo de origem são convertidos em elementos de um modelo de destino. A definição de transformações de modelos e sua automatização permitem agilizar a custosa tarefa de geração de modelos na MDA.

O problema que será foco de estudo deste trabalho é como agilizar a geração de modelos de regras de negócio numa abordagem direcionada a modelos, considerando-se o modelo independente de computação (CIM) e o modelo independente de plataforma (PIM), que são os modelos com maior nível de abstração da MDA.

1.3 Objetivo

O objetivo deste trabalho é apresentar uma abordagem para a transformação de regras de negócio na Arquitetura Dirigida por Modelos. Na transformação, serão consideradas regras de negócio presentes no modelo independente de computação e no modelo independente de plataforma da MDA. Assim, a abordagem visa transformar de maneira estruturada e automática um modelo de regras de negócio no nível CIM em um modelo de regras de negócio no nível PIM. As regras de negócio serão representadas no nível CIM utilizando *Semantics of Business Vocabulary and Rules* (SBVR) (OMG, 2008), padrão do

[4] Um artefato é a especificação de uma informação que é usada ou produzida por um processo de desenvolvimento de *software* ou pela implantação e operação de um sistema (OMG, 2005).

Object Management Group (OMG[5]) para a especificação de regras na linguagem do negócio. No nível PIM, as regras de negócio serão representadas usando *Object Constraint Language* (OCL) (OMG, 2006), linguagem adotada pelo OMG para especificar expressões que agregam informações a modelos orientados a objetos.

Além do aspecto conceitual da abordagem, será desenvolvida uma solução automatizada para a transformação, utilizando ferramentas da plataforma Eclipse[6]. A implementação da transformação baseia-se nos mapeamentos definidos na proposta de sistematização deste trabalho. Assim, a solução de automação tornará possível a geração automática de artefatos importantes para o processo de desenvolvimento de sistemas de informação.

1.4 Contribuições

A principal contribuição do trabalho é permitir a construção, através da abordagem proposta, de um modelo independente de plataforma mais completo e detalhado, provido de informações sobre as regras de negócio que, sem um tratamento sistemático, precisariam ser criadas manualmente durante o processo de desenvolvimento. Isso pode facilitar as transformações subseqüentes que esse modelo, possivelmente, irá sofrer.

Além disso, ao explicitar a forma como a transformação deverá ser feita, através da definição de mapeamentos entre os elementos dos modelos, será possível padronizar e agilizar o processo de transformação das regras de negócio na MDA.

A geração automática de artefatos a partir de modelos com maior nível de abstração constitui uma contribuição adicional, permitindo que os modelos já existentes possam ser

[5] *Object Managment Group* – consórcio para criação e manutenção de padrões para a indústria da computação. Disponível em http://www.omg.org/.

[6] Projeto Eclipse: http://www.eclipse.org.

utilizados de maneira mais efetiva na construção dos sistemas de informação. Com isso, espera-se também aumento na produtividade no processo de desenvolvimento, devido à sistematização e automatização do processo de transformação das regras de negócio.

Finalmente, destaca-se a disponibilidade do conhecimento sobre quais regras de negócio estão sendo consideradas no desenvolvimento dos sistemas de informação, quando a abordagem proposta é utilizada.

1.5 Organização da Dissertação

Esta dissertação está estruturada em seis capítulos. No presente capítulo, apresenta-se uma introdução à pesquisa realizada, considerando-se a motivação para o estudo do tema, a caracterização do problema, o objetivo do trabalho, suas contribuições e organização.

Os capítulos 2 e 3 apresentam o embasamento teórico utilizado neste trabalho. Primeiramente, tratando do Desenvolvimento Dirigido por Modelos, com ênfase na MDA e, em seguida, abordando o estudo das regras de negócio, os padrões para a representação das regras de negócio e sua relação com a MDA.

O capítulo 4 apresenta a proposta de abordagem para a transformação de regras de negócio na MDA. A descrição conceitual da abordagem é apresentada na primeira parte do capítulo, destacando-se a definição dos mapeamentos entre os elementos dos modelos de origem e de destino. Na segunda parte do capítulo, é apresentada a implementação realizada, cujo objetivo foi tornar possível a aplicação prática da abordagem proposta. Um CD com a solução desenvolvida encontra-se na contracapa desta dissertação.

O capítulo 5 apresenta um exemplo de utilização da abordagem. Esse exemplo é baseado em um estudo de caso presente em uma das especificações utilizadas e visa mostrar o uso prático da abordagem, com ênfase na utilização da solução implementada.

Por fim, o capítulo 6 apresenta as conclusões deste trabalho, destacando os resultados alcançados, as limitações encontradas e apontando possíveis melhorias e trabalhos futuros.

Além dos capítulos apresentados, esta dissertação conta com cinco apêndices, a saber: Apêndice A – mostra o metamodelo desenvolvido para a solução que automatiza a abordagem; Apêndice B – descreve a forma de uso da solução de automação; Apêndice C – apresenta a descrição do negócio e o modelo de classes usado no exemplo apresentado no capítulo 5; Apêndice D – apresenta as regras de negócio utilizadas no exemplo de utilização da abordagem e Apêndice E – apresenta os modelos de objetos em OCL dos exemplos de algumas regras de negócio utilizadas no capítulo 5.

2 ARQUITETURA DIRIGIDA POR MODELOS

A Arquitetura Dirigida por Modelos (*Model Driven Architecture* – MDA) é uma iniciativa do OMG para padronizar a realização do Desenvolvimento Dirigido por Modelos. Este capítulo descreve os principais conceitos da MDA necessários para o entendimento da abordagem proposta nesta dissertação. Dentre eles, destacam-se: a estrutura de modelos e metamodelos, as categorias de modelos, as transformações de modelos e as tecnologias e padrões envolvidos na MDA.

2.1 Desenvolvimento Dirigido por Modelos

O desenvolvimento de sistemas de informação é uma tarefa de grande complexidade, que apresenta, ainda hoje, problemas relacionados à qualidade, orçamento e prazos de entrega dos produtos. Diversos fatores contribuem para essa complexidade: a heterogeneidade cada vez maior de plataformas e ambientes computacionais, a necessidade de agilidade no processo de desenvolvimento, o crescimento do tamanho dos projetos e aplicações, as mudanças freqüentes de requisitos, entre outros. Além disso, segundo (PRESSMAN, 2006), sistemas de informação precisam evoluir com o passar do tempo, independente do seu tamanho, da sua complexidade ou do domínio de aplicação para o qual tenham sido construídos.

O uso de modelos no processo de desenvolvimento de sistemas é um dos caminhos possíveis para lidar com a complexidade existente na construção de sistemas de informação (MELLOR et al., 2004). Assim, os modelos passam a ser os elementos principais no desenvolvimento de sistemas, servindo não apenas como documentação do sistema, mas, principalmente, como uma ferramenta para a implementação. Através do uso de modelos no processo de desenvolvimento de sistemas de informação, acredita-se ser possível aumentar a produtividade e o reuso dos artefatos gerados (MACIEL; SILVA; ROSA, 2008).

Modelos permitem descrever uma realidade de maneira mais simplificada, representando nos seus elementos apenas aquilo que é essencial ao entendimento do objeto de estudo. Assim, é possível lidar com a complexidade inerente ao que está sendo modelado, de forma a permitir a melhor visualização, manipulação e raciocínio sobre a realidade estudada (MELLOR et al., 2004).

Os conceitos de modelagem, modelos e transformação de modelos norteiam um conjunto de abordagens de desenvolvimento de sistemas de informação chamado de Desenvolvimento Dirigido por Modelos (DDM) ou *Model Driven Development* (MDD). No DDM, os aspectos essenciais de um sistema são expressos na forma de modelos e as transformações desses modelos são consideradas o centro do desenvolvimento dos sistemas de informação. Além disso, os modelos são utilizados para entender o domínio do problema e o domínio da solução proposta, e a relação entre os modelos torna possível compreender as implicações oriundas das mudanças que neles ocorrem (BEYDEDA; BOOK; GRUHN, 2005).

Nesse contexto, ao habilitar o desenvolvimento num nível de abstração mais alto, com foco na modelagem e permitindo o uso de conceitos próximos ao domínio do problema, as abordagens de desenvolvimento de sistemas de informação baseadas em modelos apresentam diversos benefícios como o aumento na produtividade e a redução no tempo de construção dos sistemas (SENDALL; KOZACZYNSKI, 2003).

2.2 A Arquitetura Dirigida por Modelos

A Arquitetura Dirigida por Modelos (MDA) é um arcabouço conceitual para o desenvolvimento de sistemas de informação baseado nos princípios do Desenvolvimento Dirigido por Modelos. Lançada como especificação pelo OMG em 2001, a MDA define os modelos desse arcabouço, como eles devem ser usados e as relações entre esses modelos.

Além disso, a MDA indica padrões e tecnologias a serem utilizados no processo de desenvolvimento de forma a tornar possível a realização do DDM.

A idéia central que norteia a MDA é promover a separação entre a especificação das funcionalidades do sistema e a especificação da implementação dessas funcionalidades numa plataforma tecnológica específica (MILLER; MUKERJI, 2003).

Abordagens baseadas na MDA permitem o desenvolvimento de sistemas através da criação de modelos, os quais são refinados a partir de sucessivas transformações realizadas durante o processo de desenvolvimento. Assim, o processo de desenvolvimento de sistemas passa a ser direcionado pela atividade de modelagem e os modelos passam a ser o ponto focal do processo (KLEPPE; BAST; WARMER, 2003).

2.2.1 Modelos, Metamodelos e Tecnologias

Um modelo consiste numa coleção de elementos que descreve alguma realidade física, abstrata ou hipotética, usando uma notação formal ou não (MELLOR et al., 2004). Na MDA, modelos são definidos como especificações formais de uma função, estrutura e/ou comportamento de uma aplicação ou sistema, sendo construídos em diferentes níveis de abstração (MILLER; MUKERJI, 2003).

Já um metamodelo descreve uma notação que contém as informações necessárias para a criação de modelos. Ele define a estrutura, semântica e restrições de um grupo específico de modelos que compartilham a mesma sintaxe e semântica. Para (MELLOR et al., 2004), um metamodelo é simplesmente um modelo de uma linguagem de modelagem e os modelos escritos nessa linguagem pertencem a uma família de modelos.

O OMG padronizou uma hierarquia de quatro níveis que permite melhor entendimento da inter-relação existente entre os conceitos de metamodelo, modelos e instâncias de modelos, conforme mostrado na Figura 1.

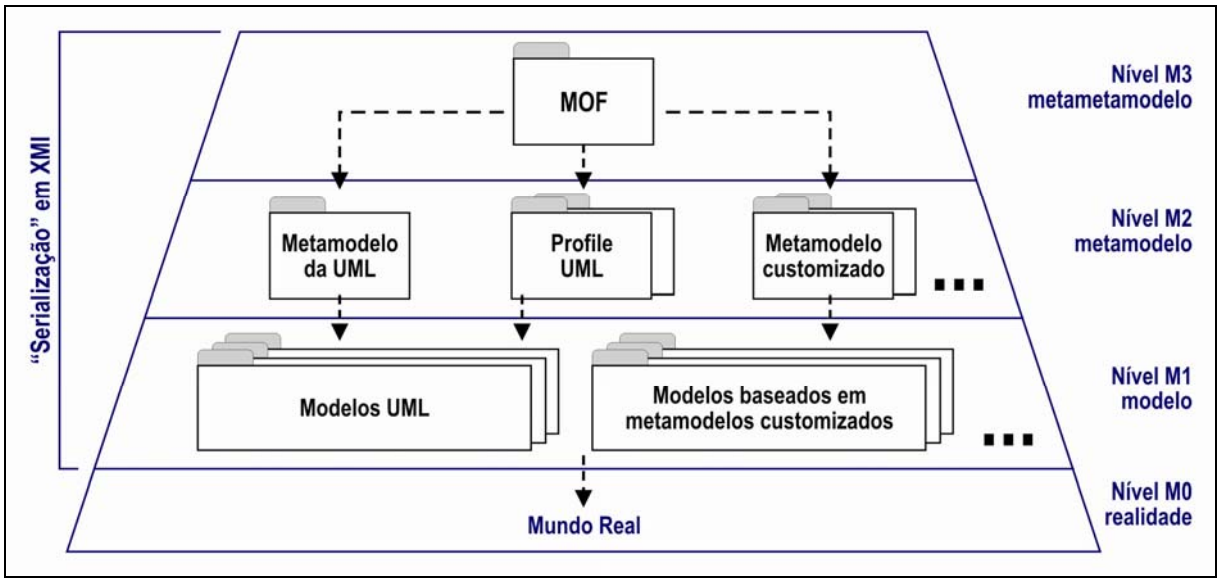

Figura 1 – Inter-relação entre modelos e metamodelos (adaptado de (MILANOVIC, 2007)).

No topo da hierarquia, encontra-se o nível de metametamodelo, M3 ou nível MOF. A motivação para a existência desse nível é definir uma linguagem de modelagem comum para os demais níveis da hierarquia, servindo como base para a construção dos modelos presentes no nível inferior, chamados de metamodelos. Já o nível M2 (nível de metamodelos) engloba todos os metamodelos padrões ou não, definidos em MOF, como, por exemplo, o metamodelo da UML.

Modelos do mundo real, representados por conceitos modelados a partir de um metamodelo do nível M2, pertencem ao nível M1 (nível de modelos). Os conceitos presentes em modelos do nível M1 são categorizações ou classificações das instâncias do nível seguinte, chamado M0 (KLEPPE; BAST; WARMER, 2003).

Finalmente, o nível M0 contém representações de coisas do mundo real. Por exemplo, o conceito *Class* do MOF (no nível M3) pode ser usado para definir outro conceito *Class* da UML, no nível M2. Por sua vez, esse conceito pode ser usado para definir o conceito Aluno no nível M1. Finalmente, o conceito Aluno é uma abstração de algo do mundo real, um aluno real. Uma instância desse conceito está no nível mais baixo da hierarquia, M0 ou nível de instância (MILANOVIC, 2007).

Pode-se dizer que um modelo em um nível superior define os elementos que podem ser utilizados na construção de um modelo em um nível inferior. Por outro lado, um modelo em um nível inferior está em conformidade com um modelo superior a ele na hierarquia. Por exemplo, o metamodelo da UML definido em MOF, encontra-se no nível M2. Assim, esse metamodelo está em conformidade com o metamodelo da MOF no nível M3.

A MDA adota uma série de tecnologias que permitem realizar o desenvolvimento dirigido por modelos. Dentre elas, serão destacadas aquelas usadas na representação e no compartilhamento de modelos no arcabouço. A primeira tecnologia é a própria linguagem de modelagem *Meta-Object Facility* (MOF), usada como padrão para a descrição de metamodelos na MDA.

A segunda tecnologia habilitadora é a *Unified Modeling Language*. UML é uma linguagem de modelagem usada para especificar, visualizar e documentar sistemas, assim como construir modelos de negócio. É a linguagem padrão do OMG para análise e projeto orientado a objetos.

Finalmente, a terceira tecnologia envolvida é XML *Metadata Interchange* (XMI). XMI é usado como padrão para o intercâmbio de metadados na MDA. XMI é definido usando XML, a partir do uso de dois XML *Schema*[7]: XML *Schema* para metamodelos MOF e XML *Schema* para modelos UML. O primeiro define a sintaxe para o compartilhamento de metamodelos descritos em MOF e da própria definição de MOF. Já o segundo esquema é utilizado para o compartilhamento de modelos UML, permitindo que eles sejam importados e exportados por ferramentas de modelagem UML.

[7] Um XML *Schema* especifica um esquema que permite que um processador de XML valide a sintaxe e alguma semântica de um documento XML.

2.2.2 Categorias de Modelos da MDA

Para separar as decisões orientadas ao negócio das decisões específicas de plataforma computacional, a MDA define três modelos[8] com diferentes níveis de abstração: o modelo independente de computação (*Computation Independent Model* – CIM), o modelo independente de plataforma (*Platform Independent Model* – PIM) e o modelo específico de plataforma (*Platform Specific Model* – PSM) (MILLER; MUKERJI, 2003).

Em linhas gerais, na MDA, o ciclo de desenvolvimento de um sistema começa com a especificação de um modelo CIM. Em seguida, um modelo de análise e projeto de sistema independente de plataforma (PIM) é especificado e, posteriormente, um modelo específico (PSM) deve ser definido para a plataforma de desenvolvimento que será adotada (MACIEL; SILVA; ROSA, 2008).

O CIM é um modelo que descreve o sistema no ambiente no qual ele irá operar e deve representar o que se espera que o sistema faça. Um modelo CIM, também conhecido como modelo de negócio, serve como fonte do vocabulário a ser usado nos demais modelos do arcabouço. O modelo de negócio pode funcionar como ponte entre aqueles que dominam o entendimento do negócio e seus requisitos e aqueles responsáveis por construir os artefatos que atenderão às necessidades do negócio (MILLER; MUKERJI, 2003).

O PIM é caracterizado por descrever um sistema sem incluir detalhes específicos de uma plataforma computacional qualquer. Um modelo assim construído pode ser usado como insumo de transformações para plataformas computacionais distintas. Em (MORGADO, 2007), o modelo PIM é chamado de modelo de sistema de informação.

---

[8] Na prática, nota-se que cada um dos modelos da MDA (CIM, PIM e PSM) representa um conjunto de modelos classificados conforme o nível de abstração em que se encontram.

Já o PSM combina a especificação de um PIM a uma plataforma computacional específica. Esse modelo inclui informações sobre detalhes de implementação. A geração de código na MDA é feita a partir do modelo PSM para uma linguagem de programação determinada. Um PIM pode ter diversos modelos PSM associados a ele, um para cada plataforma desejada (MACIEL; SILVA; ROSA, 2008).

2.2.3 Transformação de Modelos

Na MDA, transformações são aplicadas seqüencialmente sobre modelos até a geração de código. Para (KLEPPE; BAST; WARMER, 2003), uma transformação consiste na geração automática de um modelo de destino a partir de um modelo de origem, de acordo com uma definição. Já a especificação da MDA estabelece que transformação de modelo é o processo de converter um modelo em outro modelo de um mesmo sistema (MILLER; MUKERJI, 2003).

Resumidamente, uma transformação consiste na aplicação de um conjunto de regras de transformação. Enquanto a transformação como um todo descreve como um modelo de origem é convertido em um modelo de destino, as regras de transformação aplicadas descrevem como os elementos do modelo de origem são transformados em elementos do modelo de destino, considerando-se os elementos do metamodelo de origem e os elementos do metamodelo de destino.

Considerando-se a inter-relação existente entre modelos e metamodelos, pode-se dizer que o modelo de origem de uma transformação está em conformidade com um metamodelo de origem, da mesma forma como o modelo de destino está em conformidade com o metamodelo de destino. Além disso, as regras de transformação usadas em uma transformação são especificadas em uma linguagem de transformação e, desse modo, estão em conformidade com ela. Uma transformação pode ser aplicada sobre diversos modelos de

origem que estejam em conformidade com o metamodelo de origem considerado nas regras de transformação.

De forma geral, conforme mostrado na Figura 2, uma transformação é aplicada em um modelo de origem e através da execução das regras de transformação definidas, os elementos do modelo de entrada são transformados em elementos do modelo de destino.

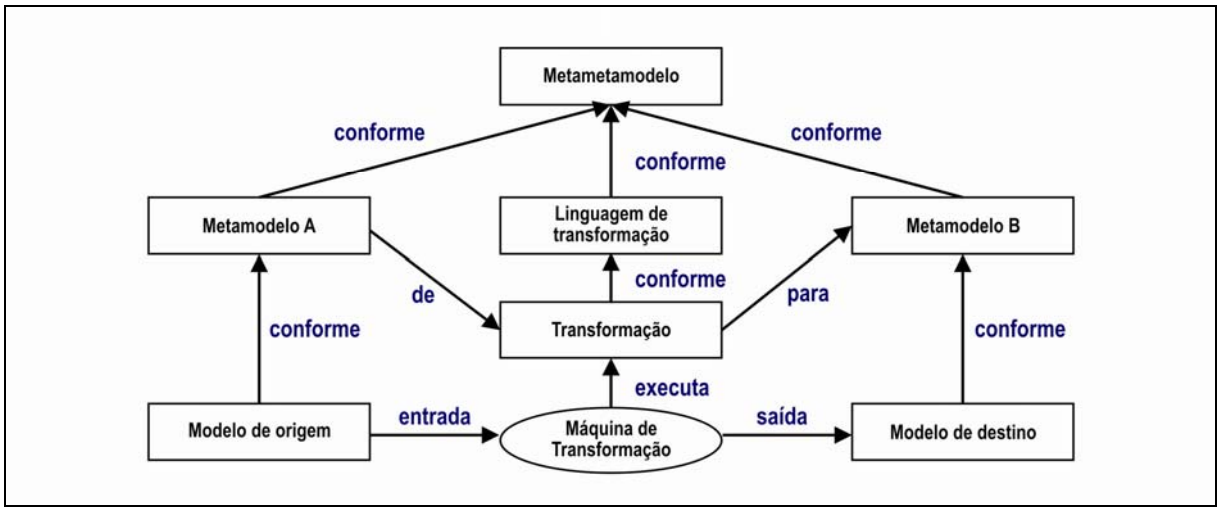

Figura 2 – Transformação de modelos (adaptado de (MILANOVIC, 2007)).

Finalmente, um ponto importante a ser considerado na evolução dos modelos através das transformações é a manutenção da relação de interdependência entre eles, ou seja, mudanças em um modelo deverão se refletir nos modelos que dependem dele, de forma que a consistência entre todos os modelos seja mantida.

2.2.3.1 Classificações para a transformação de modelos

A definição e a aplicação de transformações de modelos são técnicas críticas num processo de desenvolvimento baseado em MDA, permitindo a geração de modelos refinados, novos modelos e até código executável. Sob a ótica das saídas possíveis de uma transformação, as seguintes categorias são estabelecidas (BEYDEDA; BOOK; GRUHN, 2005):

- Transformação modelo-modelo – converte informação de um modelo ou conjunto de modelos para outro modelo ou conjunto de modelos, tipicamente em diferentes níveis de abstração;
- Transformação modelo-texto – converte um elemento de um modelo em um fragmento de texto. As transformações modelo-texto podem ser desenvolvidas para diferentes linguagens, permitindo, por exemplo, a geração de código;
- *Refactoring* – reorganiza um modelo baseado em critérios predefinidos. A saída de um *refactoring* é uma revisão do modelo original. Exemplo: mudar o nome de todas as instâncias de uma determinada classe UML em um modelo.

Na prática, a transformação de modelos pode ser aplicada de diversas formas. Considerando-se a técnica utilizada para a aplicação da transformação, a especificação da MDA define as seguintes abordagens possíveis (BEYDEDA; BOOK; GRUHN, 2005):

- Manual – o modelo de entrada é analisado e os elementos do modelo transformado são criados ou editados. A interpretação da informação e as modificações necessárias são feitas manualmente;
- Perfil Preparado – nesta técnica, um perfil que define regras de transformação é criado previamente. Ao aplicar o perfil a um modelo, o mesmo é transformado seguindo as regras predefinidas;
- Padrões e Marcações – padrões podem ser aplicados em um modelo e resultar na criação de novos elementos no modelo transformado;
- Automático – transformações automáticas aplicam um conjunto de alterações em um ou mais modelos baseado em regras de transformação predefinidas. Esse tipo de transformação exige que o modelo de entrada esteja suficientemente completo, tanto sintaticamente quanto semanticamente e pode requerer que os modelos

sejam marcados com informações específicas para as transformações que serão aplicadas.

2.2.3.2 Linguagens para a transformação de modelos

Grande parte do esforço no desenvolvimento de aplicações usando a MDA concentra-se na construção das transformações. Transformações realizadas manualmente são suscetíveis a erros e, por essa razão, a automatização desse processo é necessária. Através da automatização, é possível reutilizar o conhecimento presente na transformação e aplicá-la repetidas vezes (MACIEL; SILVA; ROSA, 2008).

O suporte à automatização é dado por diversas linguagens e ferramentas disponíveis atualmente. Nesta seção, serão apresentadas algumas classificações para as linguagens usadas para a especificação de transformações. Exemplos e estudos comparativos entre linguagens e ferramentas para a transformação de modelos podem ser encontrados em (MAIA, 2006) e (MILANOVIC, 2007).

Uma linguagem de transformação é declarativa se as definições das transformações escritas nessa linguagem especificam os relacionamentos entre os elementos nos modelos de origem e destino, sem considerar a ordem de execução da transformação. Os relacionamentos podem ser especificados em termos de funções ou regras de inferência. A máquina de transformação executa o algoritmo de transformação para produzir os elementos do modelo de destino.

Já uma linguagem de transformação imperativa especifica de maneira explícita a seqüência de passos que será executada para produzir o resultado esperado. Há também linguagens de transformação híbridas que possuem comandos declarativos e imperativos, usados conforme a necessidade.

Outra classificação para linguagens de transformação utiliza como critério a ordem de aplicação da transformação. Uma linguagem de transformação unidirecional permite que a definição de uma transformação seja aplicada em apenas uma direção: do modelo de origem para o modelo de destino. Linguagens de transformação bidirecionais, por outro lado, permitem a aplicação da transformação em ambas as direções.

Finalmente, as linguagens de transformação podem ser classificadas com base na quantidade de modelos de entrada e saída envolvidos na transformação. O cenário típico é um modelo de entrada gerando um modelo de saída (1 para 1). Além desse caso, três outras situações são possíveis: um modelo de entrada e vários modelos de saída (1 para n), vários modelos de entrada e um modelo de saída (n para 1) e vários modelos de entrada originando vários modelos de saída (m para n) (MILANOVIC, 2007).

2.2.4 Benefícios do Uso da MDA

Na MDA, o processo de desenvolvimento de sistemas de informação é baseado na construção e transformação de modelos em diferentes níveis de abstração. Além disso, o arcabouço determina a separação entre as funcionalidades do sistema e a plataforma computacional escolhida. Nesse contexto, os principais benefícios esperados com a adoção da MDA no processo de desenvolvimento de sistemas, apresentados em (KLEPPE; WARMER, 2003) e (MELLOR et al., 2004), são:

- Maior produtividade – as transformações entre os modelos da MDA, quando automatizadas, precisam ser definidas apenas uma vez e podem ser aplicadas inúmeras vezes para diversos sistemas. Apesar do grande trabalho inicial para a definição das regras de transformação entre CIM, PIM e PSM, o reuso permite a geração automática e mais rápida de artefatos. Além disso, maior esforço é dedicado na construção de modelos que usam conceitos próximos ao domínio do

problema, ampliando o entendimento e permitindo a construção de sistemas que atendam melhor às necessidades dos clientes;

- Maior portabilidade – a criação de modelos independentes de plataforma computacional permite maior portabilidade, na medida em que vários modelos específicos para diferentes plataformas podem ser gerados a partir de um único PIM. Assim, tudo que é especificado no nível independente de plataforma é portável;
- Melhoria na interoperabilidade – a construção de pontes entre as diversas plataformas computacionais dos modelos PSM gerados a partir de um mesmo PIM permite inter-relacionar conceitos de uma plataforma com conceitos de outra plataforma, tornando possível a interoperabilidade. Isso é possível a partir da comparação das transformações de PIM para PSM;
- Facilidades na manutenção e documentação – a relação de interdependência entre os modelos envolvidos no processo de desenvolvimento permite que as informações nos diferentes níveis de abstração estejam sincronizadas. Assim, como os modelos na MDA funcionam como ferramenta para a implementação e documentação do sistema (MACIEL; SILVA; ROSA, 2008), espera-se que as alterações na especificação do sistema realizadas nos modelos permitam a geração automática de código atualizado e mantenham a documentação consistente com a implementação corrente.

3 REGRAS DE NEGÓCIO

Os negócios são controlados por regras que regulam como eles devem operar e se estruturar, considerando-se aspectos internos e externos à organização. As regras de negócio serão o tema principal deste capítulo, com destaque para a definição, as características e as classificações propostas na literatura. Finalizando o desenvolvimento do capítulo, será apresentada a aplicação das regras de negócio na MDA com o uso de padrões propostos pelo OMG.

3.1 Definição e Características de Regras de Negócio

As regras de negócio são sentenças que definem ou restringem algum aspecto do negócio. Sua intenção é afirmar a estrutura do negócio e controlar ou influenciar o comportamento do negócio (HAY; HEALY, 2000).

Há duas perspectivas sob as quais as regras de negócio podem ser analisadas: a perspectiva do negócio e a perspectiva de sistemas de informação (HAY; HEALY, 2000). Na articulação entre essas duas perspectivas, é preciso considerar as diferentes formas de expressão das regras de negócio e as audiências às quais se destinam (MANSO, 2008).

Na perspectiva do negócio, as regras de negócio possuem uma função descritora e condutora do negócio, sendo consideradas diretivas que visam influenciar ou conduzir o comportamento do negócio. Nesse sentido, o negócio está estruturado e funciona conforme as regras estabelecidas. Nessa perspectiva, a audiência é humana.

Já na perspectiva de sistemas de informação, as regras de negócio são vistas como parte dos requisitos dos sistemas. Assim, precisam ser capturadas, analisadas, projetadas e implementadas. Nessa perspectiva, a audiência é tecnológica.

Dentre as características que as regras de negócio devem possuir, destacam-se (ALENQUER, 2002):

- Regras de negócio são expressões declarativas – declarações são expressões objetivas que visam comunicar a essência da regra da forma mais clara possível. Dentre os sinônimos identificados para o termo declaração, no contexto de regras de negócio, têm-se assertiva, proposição, sentença e frase;
- Regras de negócio não definem processos – a declaração de uma regra de negócio não deve conter informações sobre como a regra deve ser implementada ou executada. Elas devem expressar “o que” deve ser feito, e não “como” deve ser feito (MORGAN, 2002);
- Regras de negócio são independentes de tecnologia – as regras de negócio devem ser definidas sem comprometimento com alguma tecnologia específica. Assim, é possível separar a definição da regra de negócio de alguma implementação em particular;
- Regras de negócio são atômicas – na declaração de uma regra de negócio todas as partes são essenciais e não deve ser possível subdividi-la sem perda de significado.

Além das características citadas, existe uma outra que deve ser levada em consideração quando as regras de negócio são observadas em conjunto: a inter-relação existente entre elas. Sob a ótica dessa característica, problemas como inconsistência, incompatibilidade, redundância e circularidade podem surgir entre as regras de negócio.

3.2 Classificação de Regras de Negócio

Há diferentes enfoques para a classificação das regras de negócio. Algumas classificações priorizam a visão do negócio, enquanto outras procuram dar maior ênfase à maneira como as regras são aplicadas nos sistemas de informação. As classificações de regras de negócio são úteis na descoberta, análise e desenvolvimento das mesmas (HALLE, 2002).

A classificação proposta por (HAY; HEALY, 2000) separa as regras de negócio em termos, fatos, restrições e derivações. Os termos são designações usadas no vocabulário das pessoas do negócio e servem para expressar as regras de negócio, enquanto os fatos representam associações entre termos e expressam características do negócio.

As restrições são limitações impostas ao comportamento do negócio e as derivações são regras de negócio que definem como uma informação pode ser obtida através da transformação de outras informações.

Dentro da proposta de (ERIKSSON; PENKER, 2000) para classificar regras de negócio expressas em *Object Constraint Language* (OCL) (OMG, 2006), destacam-se duas categorias principais: as regras de restrição e as regras de derivação.

As regras de restrição são regras de negócio que restringem a estrutura e o comportamento dos objetos e processos do negócio, ou seja, a maneira como os mesmos se relacionam e mudam de estado. As regras de restrição dividem-se em (i) regras estruturais (definem condições que regulam a estrutura do negócio e devem ser sempre verdadeiras), (ii) regras operacionais (restrições que devem ser verdadeiras antes ou depois da execução de operações) e (iii) regras estímulo / resposta (especificam ações que devem acontecer quando um determinado evento ocorrer).

As regras de derivação são regras de negócio que definem como uma informação pode ser transformada em outra, com base em conclusões sobre a informação. As regras de derivação dividem-se em: (i) regras computacionais (fórmulas matemáticas para calcular

valores, por exemplo) e (ii) regras de inferência (conclusões que podem ser tiradas quando certos fatos são verdadeiros).

Uma visão diferente para a classificação de regras de negócio é apresentada em (OMG, 2008). Nessa abordagem, uma regra é um elemento de orientação que introduz uma obrigação ou uma necessidade. Assim, existem duas categorias principais de regras de negócio: as regras estruturais e as regras operativas.

As regras estruturais especificam como o negócio organiza (isto é, estrutura) os artefatos com os quais ele opera, introduzindo necessidades. Já as regras operativas governam a conduta da atividade do negócio. Em contraste com as regras estruturais, as regras operativas podem ser diretamente violadas pelas pessoas envolvidas no negócio e introduzem obrigações e proibições.

3.3 Regras de Negócio e a Arquitetura Dirigida por Modelos

As regras de negócio, conforme visto, devem possuir expressão declarativa, de forma simples, clara e objetiva. A forma de expressão utilizada no tratamento das regras de negócio é fundamental para que elas possam ser devidamente utilizadas no âmbito organizacional. Nesse sentido, dois pontos importantes devem ser considerados: as regras de negócio devem comunicar a sua mensagem de maneira apropriada ao seu público-alvo (humanos e máquinas) e devem ser manipuláveis computacionalmente (ALENQUER, 2002).

Durante o processo de desenvolvimento de sistemas de informação, as regras de negócio podem ser expressas de diversas formas, usando especificações com maior ou menor grau de formalismo, dependendo da necessidade do público-alvo ao qual se destinam. Isso acontece porque, segundo (ERIKSSON; PENKER, 2000), as regras de negócio são um importante ponto de integração entre o negócio e a tecnologia que dá suporte ao mesmo (os

sistemas de informação). Assim, é necessário que as regras de negócio estejam presentes tanto nos modelos de negócio, quanto nos modelos de sistemas de informação.

A manipulação computacional é importante não apenas para as regras de negócio que serão implementadas em sistemas de informação. O suporte computacional às regras de negócio facilita também as diversas atividades envolvidas no gerenciamento das regras de negócio em uma organização: captura, armazenamento, atualização, validação, supressão, recuperação etc. Além disso, habilita o intercâmbio das regras entre organizações e aplicações, facilita a divulgação aos interessados, permite o tratamento das inconsistências e ambigüidades, entre outras vantagens (ALENQUER, 2002).

Dentre os diversos padrões determinados pelo OMG e utilizados dentro do arcabouço conceitual estabelecido pela MDA, a SBVR (*Semantics of Business Vocabulary and Business Rules*) (OMG, 2008) e a OCL podem ser usados para expressar regras de negócio. A SBVR é o padrão do OMG para a construção de vocabulários e regras de negócio no nível CIM da MDA. Já a OCL é uma linguagem que permite expressar restrições e consultas nos diagramas da UML, ampliando o poder de expressão dos modelos e tornando possível também a modelagem de regras de negócio no nível PIM da MDA.

No nível CIM da MDA, a SBVR permite a expressão de regras de negócio utilizando linguagem natural controlada, adequada para o entendimento pelos especialistas do negócio. Sob essa sintaxe concreta, a SBVR possui um metamodelo que permite a representação e a manipulação computacional das regras de negócio especificadas.

Por sua vez, a OCL é uma linguagem que amplia o poder de expressão dos modelos orientados a objetos. No nível PIM da MDA, a OCL pode complementar modelos de classes em UML, além de ser computacionalmente manipulável. Assim como SBVR, possui uma sintaxe abstrata, representada pelo seu metamodelo, além de ser uma linguagem formal.

Nas próximas seções, serão apresentados maiores detalhes sobre as especificações da SBVR e da OCL, além de informações sobre os metamodelos que serão utilizados na abordagem proposta nesta dissertação.

3.3.1 Semantics of Business Vocabulary and Business Rules

Na busca por um padrão para a definição de regras na terminologia do negócio, o OMG publicou um pedido de propostas (*request for proposal* – RFP) intitulado *Business Semantics of Business Rules Request for Proposal* (BSBR RFP). As respostas a esse RFP deveriam atender a diversos objetivos, dentre os quais se destacava a especificação de uma abordagem que permitisse que pessoas do negócio pudessem definir as políticas e as regras que conduzem as organizações na linguagem própria do negócio, em termos dos conceitos inerentes ao mesmo. Além disso, permitir a captura e a expressão dessas regras de maneira clara, não ambígua e com a possibilidade de tradução para outras formas de representação (OMG, 2003).

As propostas deveriam contemplar os seguintes requisitos: (i) definição de um metamodelo para a especificação das regras de negócio pelos especialistas do negócio, com uma representação em MOF; (ii) definição de um metamodelo para a captura de vocabulários e definições dos termos usados nas regras de negócio; e, finalmente, (iii) uma representação XML para as regras e o vocabulário de negócio baseada em XMI, que permitisse a interoperabilidade entre ferramentas de gerenciamento de regras de negócio.

A proposta escolhida pelo OMG para esse RFP foi a *Semantics of Business Vocabulary and Business Rules* (SBVR). A especificação da SBVR define o vocabulário e as regras para documentação da semântica de vocabulários de negócio e regras de negócio. Além disso, define um esquema XMI para intercâmbio de vocabulários e regras de negócio entre organizações e entre ferramentas computacionais. A semântica da terminologia definida pela

SBVR é independente de domínio, ou seja, pode ser aplicada em diferentes tipos de negócio (OMG, 2008).

3.3.1.1 Uso da SBVR na MDA

SBVR posiciona-se na camada de negócio (modelo CIM) da Arquitetura Dirigida por Modelos. Esse posicionamento tem duas implicações (OMG, 2008):

- O alvo da especificação são apenas as regras de negócio e os vocabulários de negócio. Os demais aspectos da modelagem de negócio também deverão ser desenvolvidos, incluindo os processos de negócio e a estrutura organizacional, contudo outras iniciativas[9] do OMG devem tratar deles;
- Os modelos de negócio, incluindo os modelos baseados em SBVR, descrevem negócios e não os sistemas de informação que dão suporte aos mesmos.

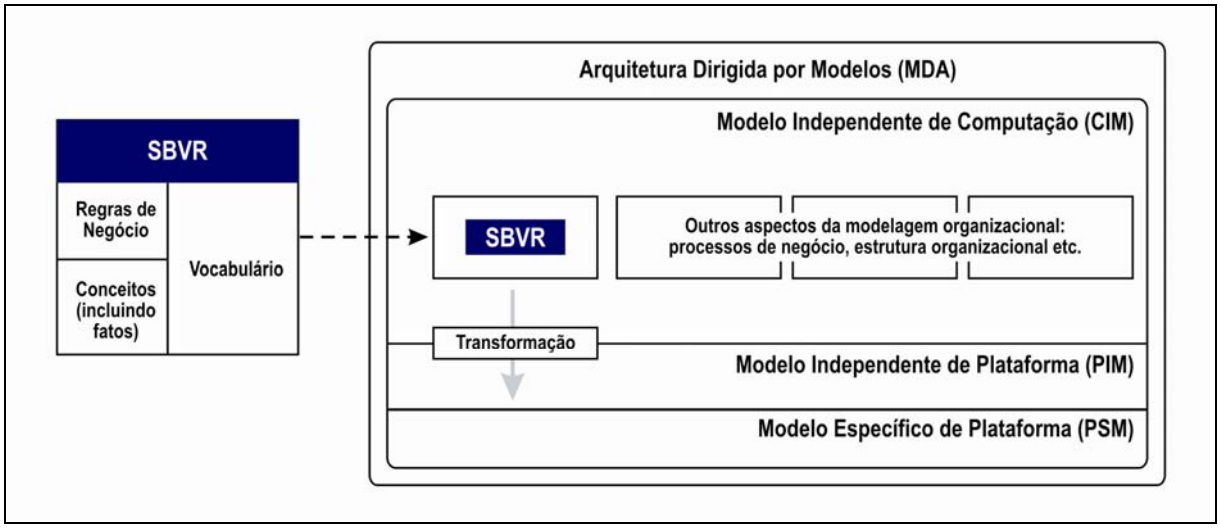

Figura 3 – Posicionamento da SBVR na MDA (OMG, 2008).

[9] Dentre as iniciativas do OMG para modelagem de negócio, destacam-se: *Business Process Definition Metamodel* (BPDM), *Organization Structure Metamodel* (OSM) e *Business Motivation Model* (BMM). Maiores detalhes sobre essas iniciativas podem ser encontrados em: http://www.omg.org/.

Apesar da possibilidade do uso do padrão para capturar terminologia e significado em qualquer nível da MDA, o objetivo da especificação é o uso da SBVR como um veículo para a descrição de negócios. Além disso, há certos aspectos do negócio que são exclusivos do nível CIM, como, por exemplo, regras de negócio que não podem ser automatizadas. Nesse caso, é importante considerar como esses aspectos serão tratados nos demais níveis da MDA.

As transformações de modelos de negócio criados em SBVR em modelos PIM e PSM, além da integração do metamodelo da SBVR com outras iniciativas e padrões da área de sistemas de informação, estão fora do escopo da especificação do padrão SBVR. Apesar disso, algumas diretrizes para transformações e a relação entre SBVR e outras abordagens são brevemente apresentadas nos anexos H e K da especificação (OMG, 2008).

3.3.1.2 Fundamentos e características da SBVR

A especificação da SBVR define um modelo semântico para a construção de vocabulários de negócio e regras de negócio. Ela é organizada em vocabulários descritos usando o próprio padrão SBVR[10].

Os principais vocabulários da especificação descrevem a terminologia necessária para a modelagem dos conceitos do negócio e das regras de negócio. O primeiro deles é o "Vocabulário para Descrição de Vocabulários de Negócio[11]", que trata dos diferentes tipos de conceitos e significados. Esse vocabulário é baseado em padrões ISO de terminologia, a saber: ISO 1087-1 (*Terminology work – Vocabulary – Theory and application*), ISO 704

[10] É importante não confundir os vocabulários definidos na especificação da SBVR com os vocabulários de negócio modelados usando o padrão SBVR, uma vez que os vocabulários definidos na especificação foram modelados usando o próprio padrão.

[11] *Vocabulary for Describing Business Vocabularies* (VDBV).

(*Terminology work – Principles and methods*) e ISO 860 (*Terminology work – Harmonization of concepts and terms*).

O segundo vocabulário é o "Vocabulário para Descrição de Regras de Negócio[12]", que lida com a especificação da semântica das regras de negócio, sendo construído a partir de elementos do primeiro vocabulário. Nesse vocabulário, são definidas as categorias das regras de negócio e das recomendações (*advices*) que podem ser modeladas usando SBVR.

Um vocabulário de negócio define os termos especializados, nomes e fatos que uma organização ou comunidade utiliza ao se expressar sobre esse negócio. Além disso, um vocabulário de negócio pode ser complementado com a documentação das regras de negócio. A construção de um vocabulário de negócio utiliza como base as diretrizes apresentadas nos dois vocabulários da SBVR citados. Assim, permite a documentação da terminologia específica do negócio e das regras de negócio da organização pelos especialistas do negócio de forma otimizada para o entendimento humano e independente de domínio (OMG, 2008).

A especificação define também um vocabulário com conceitos básicos chamado "Vocabulário de Significado e Representação[13]", que dá suporte aos diversos elementos apresentados nos demais vocabulários da SBVR. Além disso, a especificação possui mais um vocabulário que serve para sustentar a fundamentação lógica presente em SBVR, chamado "Vocabulário de Formulação Lógica da Semântica[14]".

A inter-relação entre cinco importantes aspectos da SBVR fornece uma visão integrada do padrão e dos seus vocabulários, conforme mostrado na Figura 4. São eles:

- Comunidade (*Community*) – é a base para um vocabulário de negócio. A principal comunidade, no nível do negócio, é a própria organização para a qual as

[12] *Vocabulary for Describing Business Rules* (VDBR).

[13] *Meaning and Representation Vocabulary* (MRV).

[14] *Logical Formulation of Semantics Vocabulary* (LFSV).

regras de negócio estão sendo estabelecidas e expressas. Além dessa comunidade, é fundamental reconhecer outras comunidades de importância, como a indústria na qual a organização está inserida, autoridades reguladoras, grupos de padronização etc.;

- Conjunto de Significados Compartilhados (*Body of Shared Meanings*) – uma comunidade possui um conjunto de significados compartilhados que compreende os conceitos, fatos e regras de negócio. Para que os significados compartilhados possam ser trocados, discutidos e validados, eles precisam ser expressos. Apesar disso, o que é compartilhado é o significado e não a forma de expressão. Por isso, SBVR separa o significado do negócio de qualquer forma de expressão;
- Formulação Lógica (*Logical Formulation*) – provê uma linguagem formal, abstrata e independente de sintaxe para capturar a semântica de um conjunto de significados compartilhados, dando suporte a múltiplas formas de representação. A formulação lógica suporta duas características essenciais da SBVR. Primeiramente, (i) o mapeamento de um conjunto de significados compartilhados para vocabulários usados pela comunidade e, além disso, (ii) o mapeamento para XMI, que permite o intercâmbio de conceitos, fatos e regras de negócio entre ferramentas compatíveis com SBVR;
- Representação de Negócio (*Business Representation*) – os conceitos e as regras de negócio de um conjunto de significados compartilhados precisam ser representados em um vocabulário aceito pela comunidade que compartilha esses significados. Para isso, SBVR dá suporte à representação de elementos do conjunto de significados compartilhados para linguagens concretas, tais como linguagens naturais, linguagens artificiais (como UML) ou linguagens controladas (como Inglês Estruturado);

- Lógica Formal (*Formal Logic*) – A fundamentação teórica da SBVR, que suporta a formulação lógica e as estruturas do conjunto de significados compartilhados, baseia-se em lógica formal. Mais especificamente, em lógica de predicados de primeira ordem com algumas extensões limitadas em lógica modal, para expressão de formas deônticas (obrigação e permissão) e aléticas (necessidade e possibilidade).

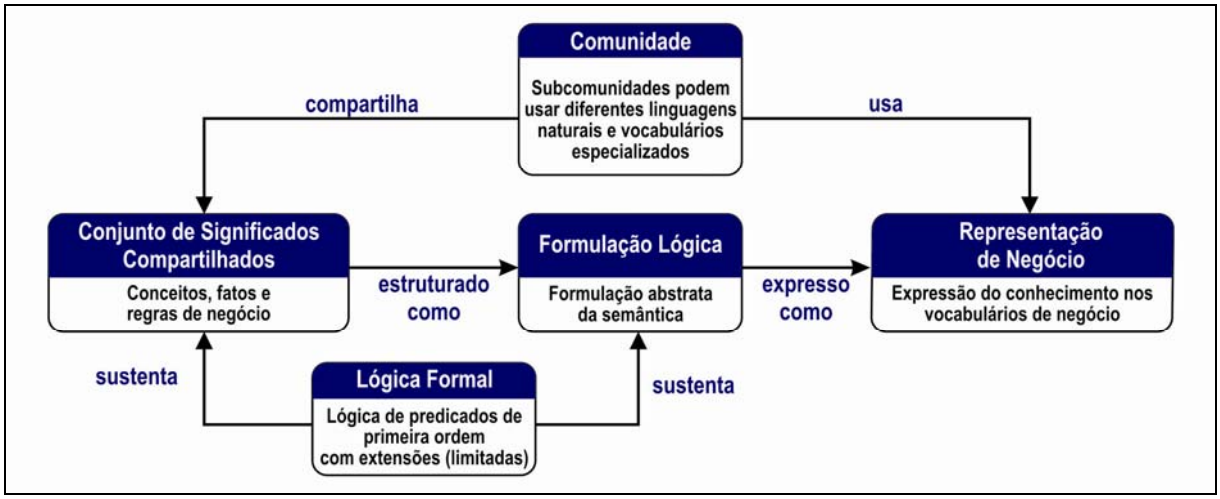

Figura 4 – SBVR em cinco aspectos (OMG, 2008).

3.3.1.3 Inglês estruturado e formulações lógicas da SBVR

A notação sugerida na especificação da SBVR para a representação de vocabulários e regras de negócio é a SBVR *Structured English*. Essa sintaxe concreta utiliza um subconjunto da língua inglesa com um número restrito de palavras e estruturas. O uso da notação não é obrigatório, mas recomendado, uma vez que as expressões definidas na mesma possuem mapeamentos diretos para os conceitos da SBVR.

SBVR *Structured English* define uma padronização de estilos de fonte com significado formal, permitindo diferenciar visualmente os termos, nomes, verbos e palavras-chaves usados. Além disso, define as regras para expressar cada um desses elementos nas sentenças.

Um exemplo de sentença usando a notação é mostrado na Figura 5, onde se pode ver a diferença entre palavras-chaves (em laranja), termos (em verde sublinhado) e verbos (em azul escuro itálico). Além disso, pode-se ver o mapeamento entre as partes da sentença e os conceitos da SBVR.

Figura 5 – Exemplo de uso da notação SBVR *Structured English* (OMG, 2008).

O conjunto de palavras-chaves em SBVR *Structured English* é classificado nas seguintes categorias: quantificações, operações lógicas, operações modais e outras. O Quadro 1 apresenta as principais palavras-chaves e sua classificação. Nele, as letras “n” e “m” representam números inteiros, enquanto as letras “p” e “q” representam expressões de proposições. As palavras-chaves possuem mapeamentos para os elementos do metamodelo da SBVR, conforme descrito no anexo C da especificação (OMG, 2008).

Quadro 1 – Palavras-chaves em SBVR.

Palavras-chaves		Classificação
each	at most *n*	Quantificações
some	exactly one	
at least one	exactly *n*	
at least *n*	at least *n* and at most *m*	
at most one	more than one	
it is not the case that *p*	*q* if *p*	Operações lógicas
p and *q*	*p* if and only if *q*	
p or *q*	not both *p* and *q*	
p or *q* but not both	neither *p* nor *q*	
if *p* then *q*	*p* whether or not *q*	
it is obrigatory that *p*	it is impossible that *p*	Operações modais
it is prohibited that *p*	it is possible that *p*	
it is necessary that *p*	it is permitted that *p*	
the	that	Outras
a, an	who	
another	is of	
a given	what	

Além da forma de expressão provida pelo SBVR *Structured English*, SBVR possui uma maneira de descrever a estrutura do significado das regras e da definição dos conceitos expressas em linguagem natural. Essa forma de expressão da SBVR é chamada de formulação semântica. Há dois tipos de formulações semânticas: as formulações lógicas e as projeções.

As formulações lógicas estruturam o significado das proposições em SBVR e especializam-se em operações lógicas, formulações modais, quantificações e formulações atômicas, entre outras. Um exemplo de formulação lógica para a regra de negócio "*It is necessary that each rental has at most three additional drivers*" é mostrado na Figura 6. Já as projeções estruturam intensões como conjuntos de elementos que satisfazem certas restrições (condições) e são usadas principalmente para formular definições de conceitos (OMG, 2008).

The rule is a proposition meant by an obligation formulation.
. That obligation formulation embeds a universal quantification.
. . The universal quantification introduces a first variable.
. . . The first variable ranges over the concept "rental".
. . The universal quantification scopes over an at-most-n quantification.
. . . The at-most-n quantification has the maximum cardinality 3.
. . . The at-most-n quantification introduces a second variable.
. . . . The second variable ranges over the concept "additional driver".
. . . The at-most-n quantification scopes over an atomic formulation.
. . . . The atomic formulation is based on the fact type "rental has additional driver".
. . . . . The atomic formulation has a first role binding.
. . . . . The first role binding is of the role "rental" of the fact type.
. . . . . The first role binding binds to the first variable.
. . . . . The atomic formulation has a second role binding.
. . . . . The second role binding is of the role "additional driver" of the fact type.
. . . . . The second role binding binds to the second variable.

Figura 6 – Uma formulação lógica em SBVR (OMG, 2008).

A especificação da SBVR provê um vocabulário para descrever essas estruturas semânticas formais do discurso do negócio no "Vocabulário de Formulação Lógica da Semântica". Conforme será visto na seção que descreve o metamodelo da SBVR, existe também uma sintaxe abstrata para a representação das formulações semânticas.

3.3.1.4 Descrição de vocabulários e regras de negócio em SBVR

Conforme visto, um vocabulário de negócio contém os termos especializados, nomes e fatos que uma organização ou comunidade usa ao se expressar sobre determinado negócio. Cada vocabulário de negócio deve representar apenas um conjunto de significados compartilhados. Para sua construção, é usada a terminologia definida no "Vocabulário para Descrição de Vocabulários de Negócio" da especificação da SBVR.

Além desse vocabulário, a especificação define uma padronização para a documentação dos vocabulários de negócio, levando em consideração a identificação da comunidade usuária do vocabulário, a relação com outros vocabulários existentes, o escopo, propósito, idioma e nome do vocabulário. Define também uma padronização para a definição dos verbetes do vocabulário: os conceitos que serão representados, suas definições, relações e sinônimos, as necessidades e possibilidades associadas aos conceitos, além de outras informações relevantes para a construção do vocabulário.

Dentre as vantagens do uso do padrão SBVR para a construção de vocabulários de negócio, destacam-se (OMG, 2008):

- Capacidade de categorização hierárquica de conceitos (definição de taxonomias);
- Definição de sinônimos, abreviações, "veja também" (*see also*) e vocabulários múltiplos para linguagens diferentes;
- Especificação de definições (por intensão ou extensão) formalmente e de forma não ambígua, usando outras definições do próprio vocabulário de negócio;
- Especificação de conexões entre conceitos de interesse da organização;
- Existência de gabaritos para facilitar a captura da semântica dos conceitos e da relação entre eles;
- Capacidade de integração entre vocabulários de negócio distintos.

A descrição das regras de negócio em SBVR é feita no próprio vocabulário de negócio, porém ela ocupa uma seção separada, com um padrão de construção diferenciado. Além da terminologia definida no “Vocabulário para Descrição de Vocabulários de Negócio”, para a especificação das regras de negócio de uma comunidade usando SBVR é necessário o uso dos conceitos definidos no “Vocabulário para a Descrição de Regras de Negócio”.

As entradas de uma seção do vocabulário de negócio que descreve as regras de negócio possuem uma padronização descrita na especificação da SBVR. Na parte inicial da seção, devem ser definidos um nome para o conjunto de regras, a descrição do propósito e escopo, o vocabulário no qual as regras se baseiam, além de outras informações comuns às regras expressas.

Na segunda parte da seção, cada entrada corresponde a um elemento de orientação: uma regra de negócio operativa, uma regra de negócio estrutural ou uma recomendação (permissão ou possibilidade). Além dessa classificação, a sentença que expressa a regra, um nome, uma descrição, as formas sinônimas e um exemplo de uso da regra de negócio também devem ser informados nessa parte da seção.

Um ponto importante a se destacar no processo de construção de regras de negócio em SBVR é a forma como elas são definidas. As regras de negócio no padrão são sempre construídas pela aplicação de necessidades ou obrigações sobre fatos. Dessa maneira, SBVR assume um princípio básico do *Business Rules Approach*[15], que é “as regras de negócio são baseadas nos fatos e os fatos são baseados nos termos”. Nesse contexto, os termos e fatos usados na construção das regras de negócio em SBVR são retirados do próprio vocabulário de negócio no qual as regras de negócio se baseiam.

[15] http://www.businessrulesgroup.org/bra.shtml.

3.3.1.5 Metamodelo da SBVR

A especificação da SBVR define uma sintaxe abstrata para SBVR, representada por um metamodelo descrito em MOF. O metamodelo suporta a representação dos conceitos e das regras de negócio modelados em vocabulários SBVR.

Apesar do metamodelo da SBVR representar os conceitos definidos nos quatro vocabulários que compõem a especificação, serão destacados aqui os elementos do "Vocabulário de Significado e Representação" e do "Vocabulário de Formulação Lógica da Semântica". Esses vocabulários possuem os elementos necessários para a especificação da transformação proposta neste trabalho. A descrição completa do metamodelo da SBVR pode ser encontrada em (OMG, 2008).

Metamodelo do Vocabulário de Significado e Representação

Os conceitos fundamentais que dão embasamento aos demais vocabulários da especificação da SBVR são descritos nessa parte do metamodelo. Além disso, uma importante característica da SBVR, a separação entre significado e expressão, é suportada pelos elementos presentes nesse vocabulário.

Os significados (*meanings*) são os elementos fundamentais da SBVR e dividem-se em conceitos (*concepts*) e proposições (*propositions*), conforme mostrado na Figura 7. Os conceitos são usados para classificar elementos (*noum concepts*) e ações ou estados (*fact types*) (CABOT; PAU; RAVENTÓS, 2009). Já uma proposição é um significado que possui um valor verdadeiro ou falso. Em SBVR, uma regra (elemento *rule*) é um tipo de proposição.

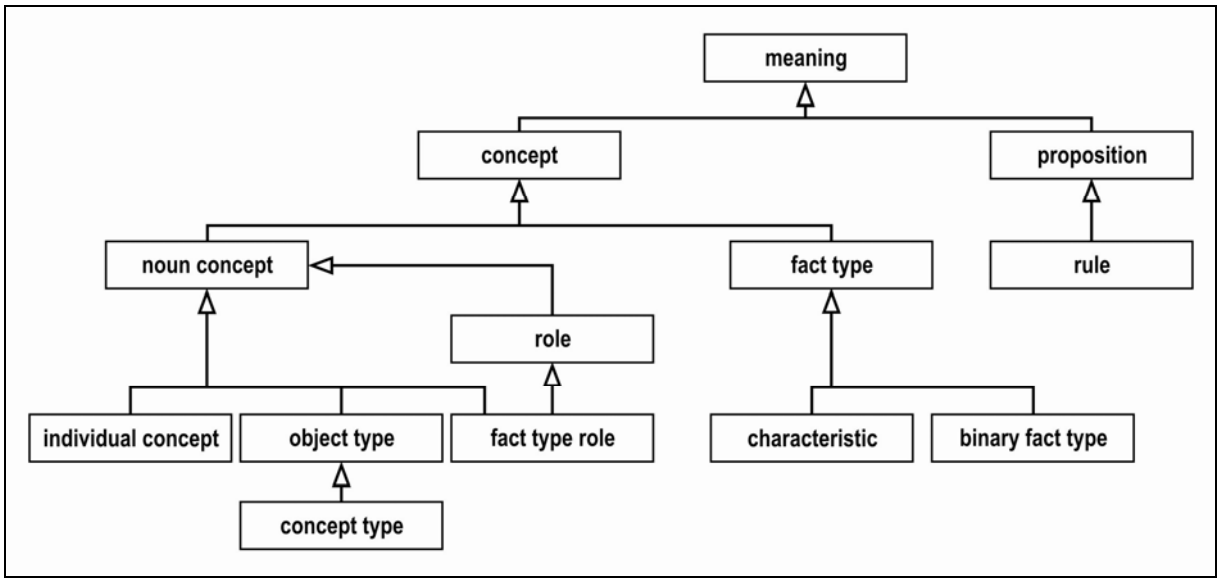

Figura 7 – Significados no metamodelo da SBVR (OMG, 2008).

Além dos significados, existem as expressões (*expressions*), que representam algo que é usado para expressar ou comunicar, independentemente de qualquer interpretação. Uma expressão pode ter diversos significados. As seqüências de caracteres “carro” e “*car*” são exemplos de expressões.

A ligação entre significado e expressão é feita através de diferentes categorias de representação (*representation*). Dependendo do significado que se deseja representar, um tipo específico de representação é utilizado, conforme mostrado na Figura 8. Por exemplo, um conceito do tipo *object type* deve ser expresso com o uso de uma representação do tipo designação (*designation*).

Assim, a representação de um significado em SBVR é feita através do relacionamento entre as instâncias das classes ou especializações de *meaning*, *representation* e *expression* do metamodelo. No caso de um conceito, por exemplo, a relação é entre um elemento *concept*, um elemento *designation* e um elemento *signifier*, que é a expressão de uma designação.

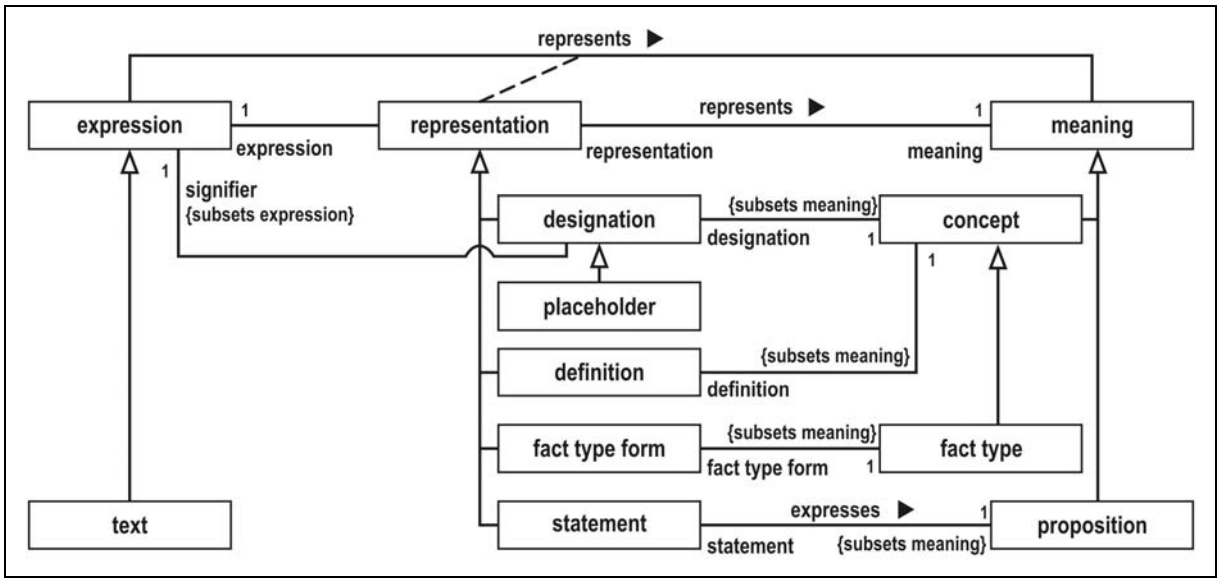

Figura 8 – Significados, expressões e representações no metamodelo da SBVR (OMG, 2008).

Metamodelo do Vocabulário de Formulação Lógica da Semântica

A fundamentação lógica provida por SBVR é suportada pelos elementos do metamodelo que fazem parte do "Vocabulário de Formulação Lógica da Semântica". Esse recorte do metamodelo descreve os elementos necessários para representar a sintaxe abstrata das formulações semânticas da SBVR.

Nessa parte do metamodelo, o principal elemento de interesse são as formulações lógicas. Uma formulação lógica pode ser composta de outras formulações lógicas. Os tipos de formulações lógicas em SBVR são mostrados na Figura 9.

A formulação atômica (*atomic formulation*) é a formulação lógica usada para representar fatos. Possui um *role binding* para cada papel (*role*) presente no fato. Um *role binding* pode referenciar uma variável, uma expressão ou um conceito individual. Já a formulação de instanciação (*instantiation formulation*) é usada para classificar instâncias, através do uso de um conceito e de uma variável que as representa.

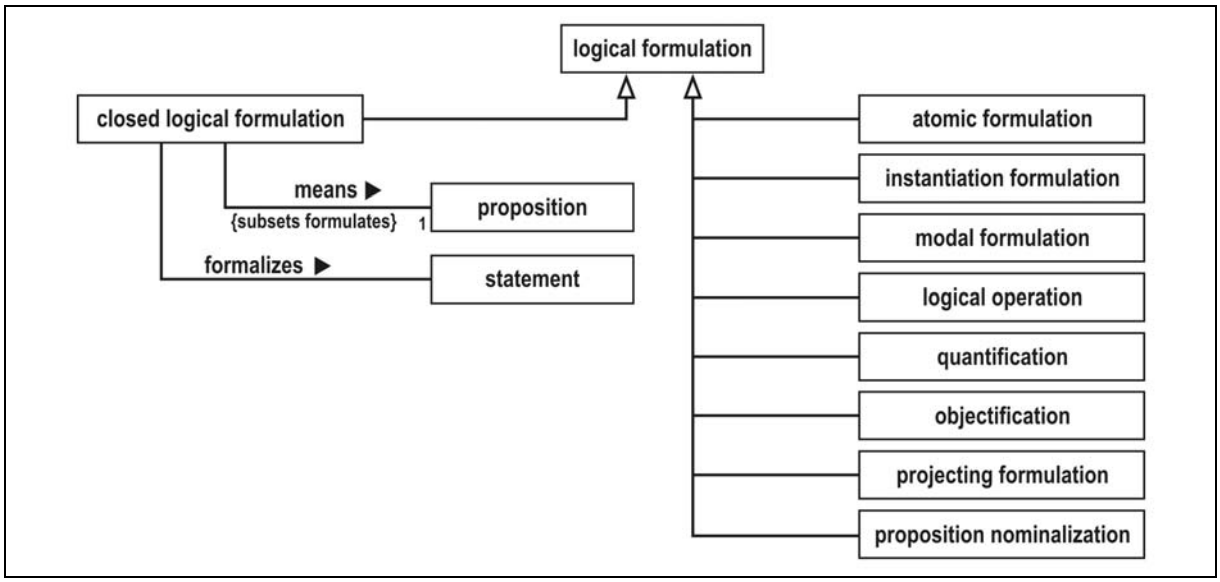

Figura 9 – Classificação das formulações lógicas no metamodelo da SBVR (OMG, 2008).

O suporte à lógica modal é feito no metamodelo da SBVR com o uso do elemento formulação modal (*modal formulation*). Ele é usado para representar uma relação de obrigação (*obligation formulation*), necessidade (*necessity formulation*), permissão (*permissibility formulation*) ou possibilidade (*possibility formulation*) com uma ou mais formulações lógicas.

As operações lógicas (*logical operations*) em SBVR formulam um significado verdadeiro ou falso baseadas em um ou mais operandos lógicos envolvidos na operação. Um operando lógico é também uma formulação lógica. As operações lógicas definidas em SBVR são mostradas na Figura 10.

Finalmente, uma quantificação (*quantification*) é uma formulação lógica que introduz uma variável e possui uma formulação de escopo (*scope formulation*) que deve ser satisfeita por todos ou por alguns referentes da variável da quantificação. As quantificações permitidas em SBVR são mostradas na Figura 11.

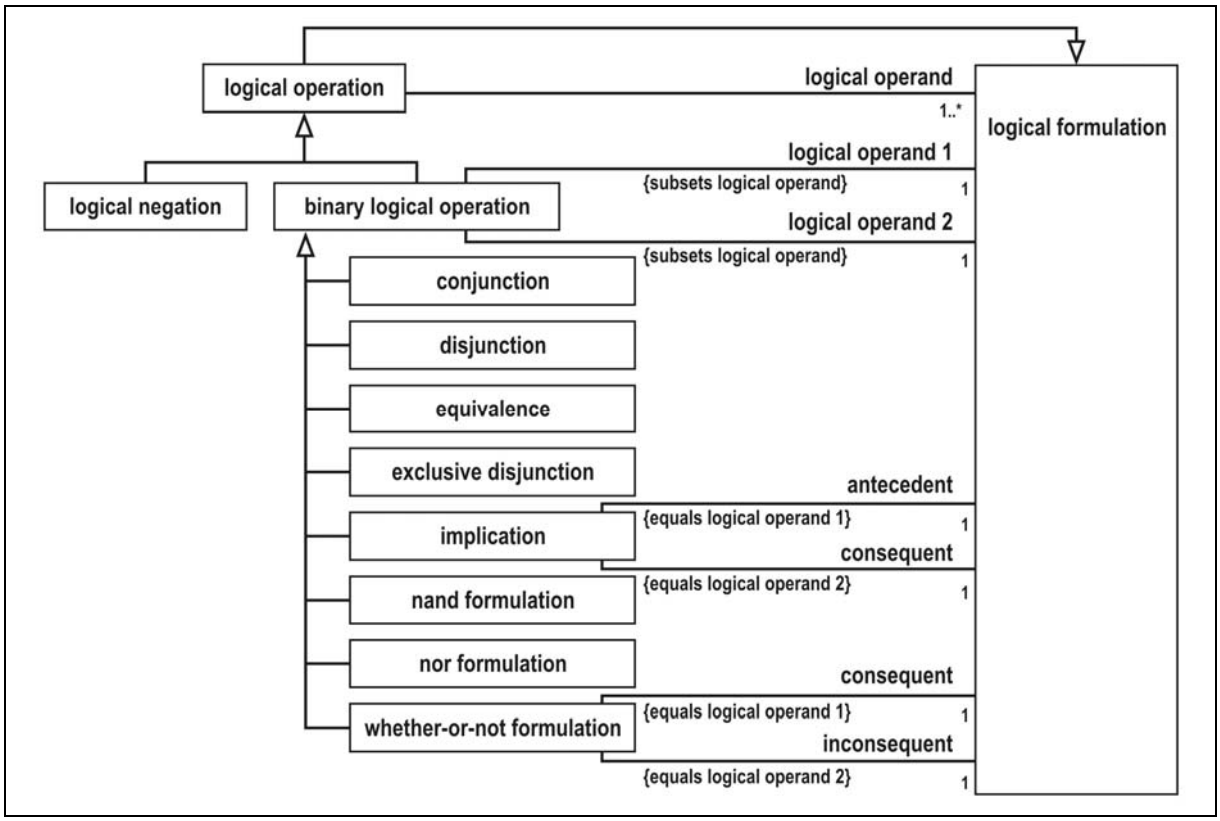

Figura 10 – Operações lógicas no metamodelo da SBVR (OMG, 2008).

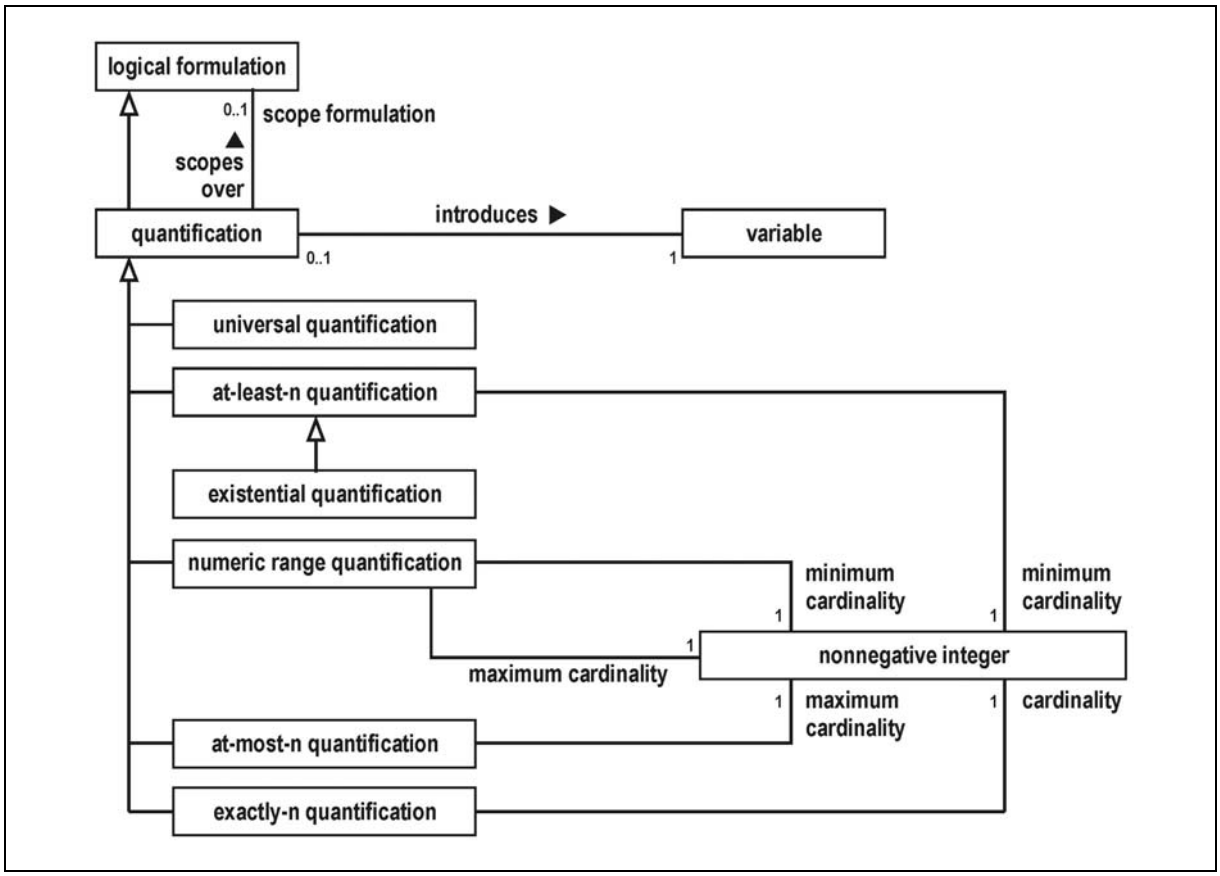

Figura 11 – Quantificações no metamodelo da SBVR (OMG, 2008).

Algumas formulações lógicas são recursivas, ou seja, podem conter outras formulações lógicas. A estrutura hierárquica da descrição textual de uma formulação lógica (representada pela indentação do conjunto de proposições da formulação), em alguns casos, evidencia essa característica. Por exemplo, uma formulação lógica em um nível pode aplicar uma operação modal de necessidade sobre uma ou mais formulações lógicas em um nível inferior.

Já a relação entre a descrição textual de uma formulação lógica e sua representação na sintaxe abstrata da SBVR é feita considerando-se cada uma das proposições do conjunto. Assim, uma proposição pode descrever uma instância de uma classe do metamodelo, o valor de um dos seus atributos ou a relação entre duas classes instanciadas previamente (CABOT; PAU; RAVENTÓS, 2009).

3.3.2 Object Constraint Language

A linguagem padrão adotada pelo OMG para especificar expressões que agregam informações a modelos orientados a objetos é a *Object Constraint Language*. Neste trabalho, a OCL será usada como linguagem para representação das regras de negócio no modelo independente de plataforma (PIM). As próximas seções apresentam maiores detalhes sobre a linguagem OCL, destacando sua importância para a MDA, sua utilização nos diagramas da UML, suas principais características e o seu metamodelo.

3.3.2.1 Uso da OCL na UML e na MDA

Reforçando a relação existente entre os diversos padrões estabelecidos pelo OMG, a OCL apresenta-se como um importante elemento habilitador da MDA ao permitir a construção de diagramas UML mais precisos (KLEPPE; WARMER, 2003).

Na construção de modelos usando uma linguagem de modelagem com notação gráfica como a UML, existem informações importantes para a especificação de um sistema que não podem ser expressas apenas com a utilização de diagramas. A OCL insere-se nessa lacuna, permitindo a construção de expressões que complementam os diagramas da UML.

Expressões escritas em uma linguagem precisa como OCL oferecem alguns benefícios. Primeiramente, essas expressões possuem uma interpretação única e sem ambigüidade, o que torna os modelos mais completos e detalhados. Além disso, elas podem ser verificadas por ferramentas automáticas, a fim de garantir que estão corretas e consistentes com outros elementos do modelo.

Para a construção de um modelo mais preciso, que agregue as vantagens do uso de uma linguagem gráfica com o poder de expressão de uma linguagem textual, é preciso utilizar UML e OCL juntas. Sem as expressões OCL, não será possível representar tudo aquilo que é necessário especificar no modelo. Já sem os diagramas da UML, as expressões farão referência a elementos que não existem, uma vez que não é possível representar classes e associações usando OCL (KLEPPE; WARMER, 2003).

Existem diversas formas de aplicação das expressões OCL nos diagramas da UML, dentre as quais se destacam (OMG, 2006): invariantes para diagramas de classes, pré-condições e pós-condições para operações de uma classe; definição de valores iniciais e valores derivados para atributos de uma classe e condições de guarda em diagramas de estados. Além disso, a representação de regras de negócio na UML pode ser feita com o uso de expressões em OCL (ERIKSSON; PENKER, 2000).

A essência da MDA é o uso de modelos como base para o processo de desenvolvimento de sistemas. Nesse contexto, deseja-se que os modelos sejam precisos, sólidos, consistentes e coerentes. Ao aumentar o poder de expressão dos diagramas modelados, a OCL provê a MDA com modelos melhor especificados e com mais informações

agregadas, tornando possível também a construção de transformações de modelos mais eficientes (KLEPPE; WARMER, 2003).

3.3.2.2 Conceitos e características da OCL

A OCL é uma linguagem que permite a definição de informações que aumentam o poder de expressão dos elementos gráficos da UML. Nos diagramas da UML, a OCL é usada para especificar restrições e realizar consultas sobre os elementos de um modelo. OCL possui uma sólida fundamentação matemática e lógica, cujo formalismo assegura, principalmente no contexto da MDA, a construção de modelos precisos que podem ser transformados de maneira automática.

A linguagem OCL consiste de expressões, envolvendo operadores e operandos, que retornam um valor. Toda expressão OCL possui um tipo e está relacionada a um contexto, que é a parte do modelo na qual a expressão deverá ser avaliada. A definição do contexto pode ser feita graficamente no modelo UML ou através da palavra reservada *context*. A Figura 12 mostra um modelo UML simples e uma expressão OCL associada à classe Conta.

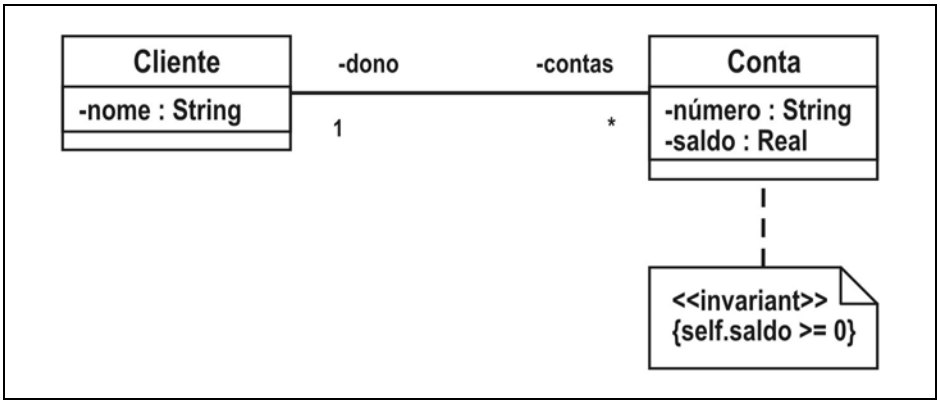

Figura 12 – Exemplo de modelo UML com uma expressão OCL.

Uma característica importante da OCL é ser uma linguagem tipada, tornando possível a avaliação das expressões durante a modelagem. Cada tipo da linguagem define um conjunto de operações que podem ser aplicadas a um valor desse tipo. Por exemplo, o

operador “+” (soma) só pode ser aplicado em operandos do tipo *Integer* (um número inteiro) ou *Real* (um número real).

Além disso, cabe destacar também que a OCL é uma linguagem declarativa. Uma expressão em OCL apenas diz o que deve ser feito, mas não como. Essa característica permite que um modelo em UML com expressões OCL seja completamente independente de plataforma, uma vez que não há restrições no modelo sobre como as expressões deverão ser implementadas ou qual plataforma computacional será utilizada (KLEPPE; WARMER, 2003).

A definição de invariantes sobre um modelo de classes é uma das formas de aplicação da OCL nos diagramas da UML. Invariantes são restrições estruturais estáticas definidas no contexto de um tipo. Uma invariante é definida como um expressão booleana que deve ser verdadeira para todas as instâncias do tipo ao qual a invariante se aplica (CORREA; WERNER; BARROS, 2007).

Na Figura 13, a invariante apresentada define que todas as instâncias do tipo Conta tenham o valor do atributo saldo maior ou igual a zero. Uma vez que a invariante foi definida no contexto da classe Conta, a expressão *self.saldo >= 0* deve ser verdadeira para todas as instâncias da classe. A palavra reservada *self* refere-se a cada instância do tipo do contexto usado na avaliação da invariante.

```
context Conta
inv: self.saldo >= 0
```

Figura 13 – Exemplo de uma invariante em OCL.

Na construção de expressões mais complexas, um conceito importante é a navegação pelas associações definidas no modelo de classes. Uma expressão de navegação resulta em uma coleção de elementos. A OCL define operações específicas que podem ser aplicadas sobre coleções de elementos. Por exemplo, na expressão OCL mostrada na Figura 14,

self.contas resulta no conjunto de instâncias da classe Conta associadas a uma instância da classe Cliente (representada por *self*). A operação *size* retorna o número de elementos do conjunto e é aplicada sobre a coleção através do uso do operador "->". Assim, a expressão mostrada retorna o número de contas associadas a um cliente.

```
context Cliente
def: QtdeContas: Integer = self.contas -> size()
```

Figura 14 – Exemplo de navegação em OCL.

3.3.2.3 Metamodelo da OCL

A especificação da OCL (OMG, 2006) define uma sintaxe abstrata para a linguagem, representada pelo metamodelo que será descrito nesta seção. O metamodelo da OCL é definido usando a linguagem de modelagem MOF e reusa algumas classes do metamodelo da UML.

O metamodelo da OCL é dividido em dois pacotes: o pacote *Types* e o pacote *Expressions*. Além desses pacotes, uma extensão ao metamodelo da OCL foi proposta por (MILANOVIC, 2007), representada pelo pacote *EnhancedOCL*. Esse pacote não faz parte da especificação padrão da OCL e foi acrescentado com o objetivo de permitir a representação de elementos específicos da sintaxe concreta da linguagem.

Pacote Types

Como a OCL é uma linguagem tipada, toda expressão deve possuir um tipo que é declarado explicitamente ou derivado de maneira estática. A avaliação de uma expressão produz um valor de um tipo. A Figura 15 mostra os principais elementos do pacote *Types*.

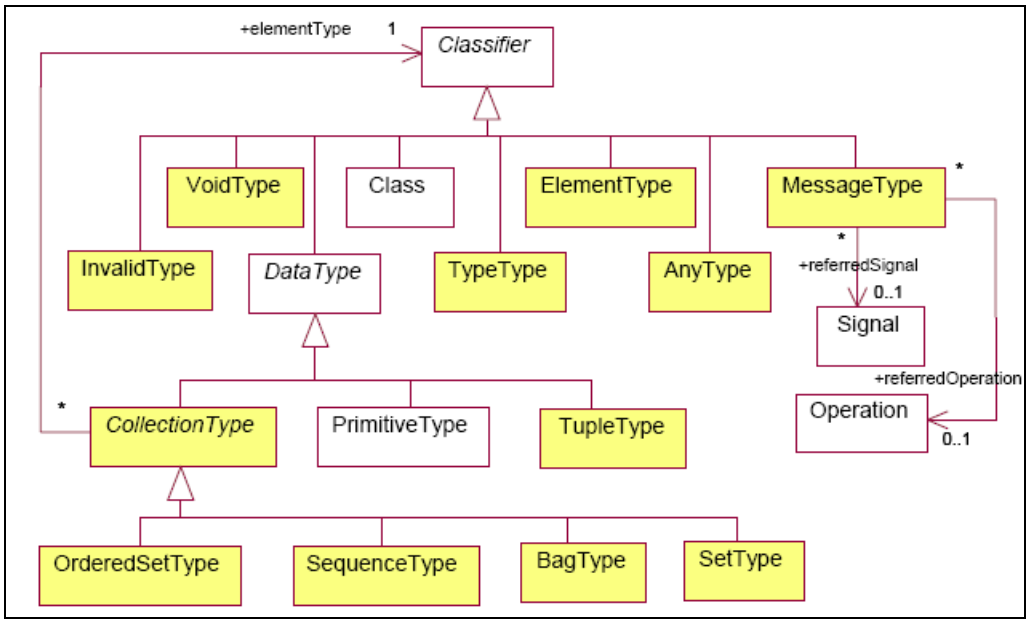

Figura 15 - Os tipos no metamodelo da OCL (OMG, 2006).

O tipo básico representado no metamodelo é o tipo *Classifier* da UML e inclui todos os subtipos de *Classifier*. Dentre os tipos específicos da OCL mostrados no metamodelo, destacam-se (OMG, 2006):

- *CollectionType* – classe abstrata que descreve uma lista de elementos de um tipo particular. Possui quatro subclasses concretas: *OrderedSetType*, *SequenceType*, *BagType* e *SetType*. A associação *elementType* indica o tipo dos elementos que fazem parte da coleção;
- *TupleType* – permite combinar diversos tipos em um único tipo agregado. As partes de um tipo *TupleType* são descritas pelos seus atributos e cada um deles possui um nome e um tipo;
- *AnyType* – um tipo especial que representa o supertipo de todos os outros tipos, com exceção dos tipos de coleção. *OclAny* é o nome da instância única desse tipo e possui diversas operações predefinidas válidas para os demais tipos da OCL (exceto coleções);

- *VoidType* – representa um tipo que está em conformidade com todos os tipos. Possui uma instância única, chamada *OclUndefined*.

Pacote Expressions

Nesse pacote, estão definidas as estruturas que uma expressão OCL pode conter. Os principais elementos dessa parte do metamodelo são mostrados na Figura 16. A principal classe nesse modelo é *OclExpression*, que representa a superclasse abstrata de todas as outras expressões apresentadas no metamodelo.

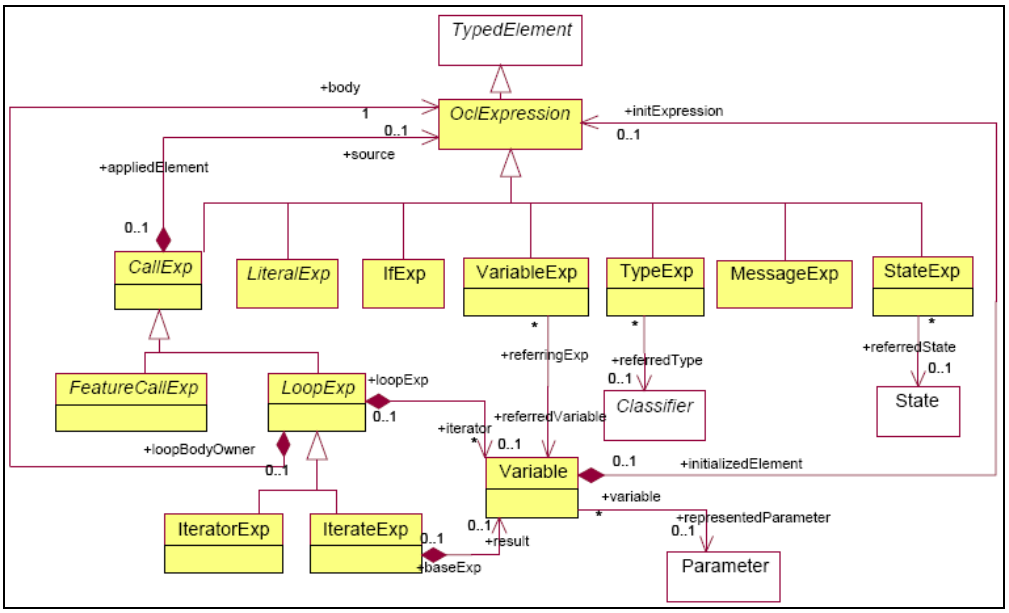

Figura 16 – A estrutura básica das expressões no metamodelo da OCL (OMG, 2006).

Dentre os elementos da estrutura básica do pacote *Expression*, destacam-se (OMG, 2006):

- *OclExpression* – representa uma expressão que pode ser avaliada. Sempre possui um tipo, que pode ser determinado pela análise da expressão e do seu contexto;
- *CallExp* – classe abstrata que representa uma expressão que se refere à uma propriedade ou à uma expressão de iteração predefinida para uma coleção;

- *FeatureCallExp* – generalização das expressões que se referem à uma propriedade definida para um *Classifier* em um modelo UML;
- *VariableExp* – expressão que consiste na referência à uma variável, como, por exemplo, a variável *self* ;
- *LoopExp* – expressão que representa uma iteração sobre os elementos de uma coleção. Associa-se a uma outra expressão (através da propriedade *body*) que é avaliada para cada elemento presente na coleção. Além disso, possui uma variável de iteração que representa cada elemento na coleção durante a iteração;
- *IteratorExp* – expressão do tipo *LoopExp* que representa as operações OCL predefinidas sobre coleções que usam uma variável de iteração: *select*, *collect*, *reject*, *forAll*, *one*, *exists* etc.
- *LiteralExp* – expressão sem argumentos que produz um valor. Geralmente, o valor resultante é idêntico ao símbolo da expressão (atributo *symbol*). Dentre as especializações possíveis, estão *BooleanLiteralExp*, *IntegerLiteralExp* e *StringLiteralExp*, representando, respectivamente, os tipos OCL predefinidos *Boolean*, *Integer* e *String*;
- *TypeExp* – expressão usada para referenciar um metatipo em uma expressão. Usada especialmente nas operações *oclIsKindOf*, *oclIsTypeOf* e *oclAsType* como referência para os tipos avaliados nas chamadas a essas operações.

Além dos elementos mostrados, é importante destacar as especializações da classe *FeatureCallExp*, mostradas na Figura 17. Um elemento *FeatureCallExp* pode se referenciar a qualquer um dos subtipos de *Feature* definidos no metamodelo da UML. Os principais subtipos de *FeatureCallExp* são:

- *PropertyCallExp* – representa a referência a *Attribute* de um *Classifier*, definido em um modelo UML. Sua avaliação resulta no valor do atributo referenciado;

- *OperationCallExp* – constitui uma referência a uma operação definida em um *Classifier* de um modelo UML. Se a operação referenciada possuir parâmetros, então a expressão irá possuir uma lista de argumentos. Essa lista possui a mesma quantidade de elementos dos parâmetros da operação. Além disso, os tipos dos elementos da lista coincidem com os tipos dos parâmetros da operação.

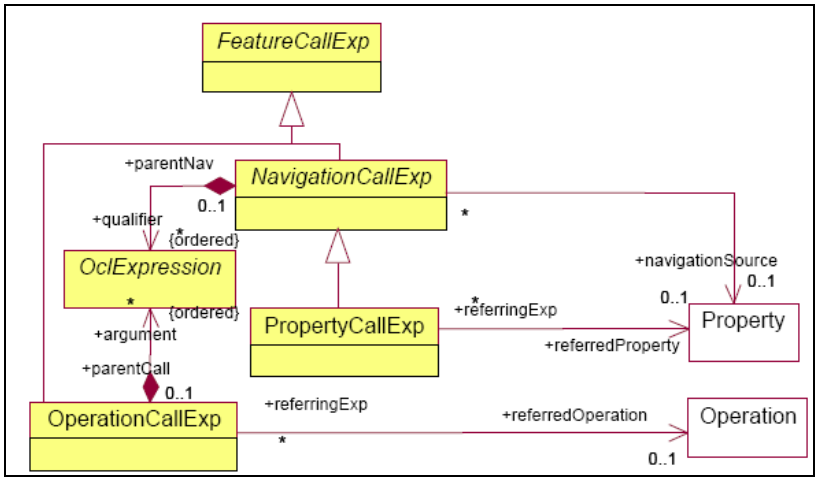

Figura 17 – Especializações de *FeatureCallExp* no metamodelo da OCL (OMG, 2006).

Pacote EnhancedOCL

A especificação padrão do metamodelo da OCL não possui suporte para representar expressões invariantes em OCL. Para tornar possível essa representação, será utilizada uma extensão ao metamodelo padrão da OCL, composta pelo pacote *EnhancedOCL* (MILANOVIC et al., 2007). As invariantes serão utilizadas nesta dissertação para a representação das regras de negócio no modelo independente de plataforma (PIM).

O pacote *EnhancedOCL* possui a classe *Invariant*, como uma subclasse da classe *OclModuleElement*, conforme mostrado na Figura 18. A classe *OclModuleElement* é o principal elemento do pacote e representa uma generalização das seguintes classes:

- *Invariant* – representa uma invariante aplicada em uma classe de um modelo UML;

- *DefOclModuleElement* – usada para representar operações e propriedades;
- *DeriveOclModuleElement* – representa as regras de derivação em OCL.

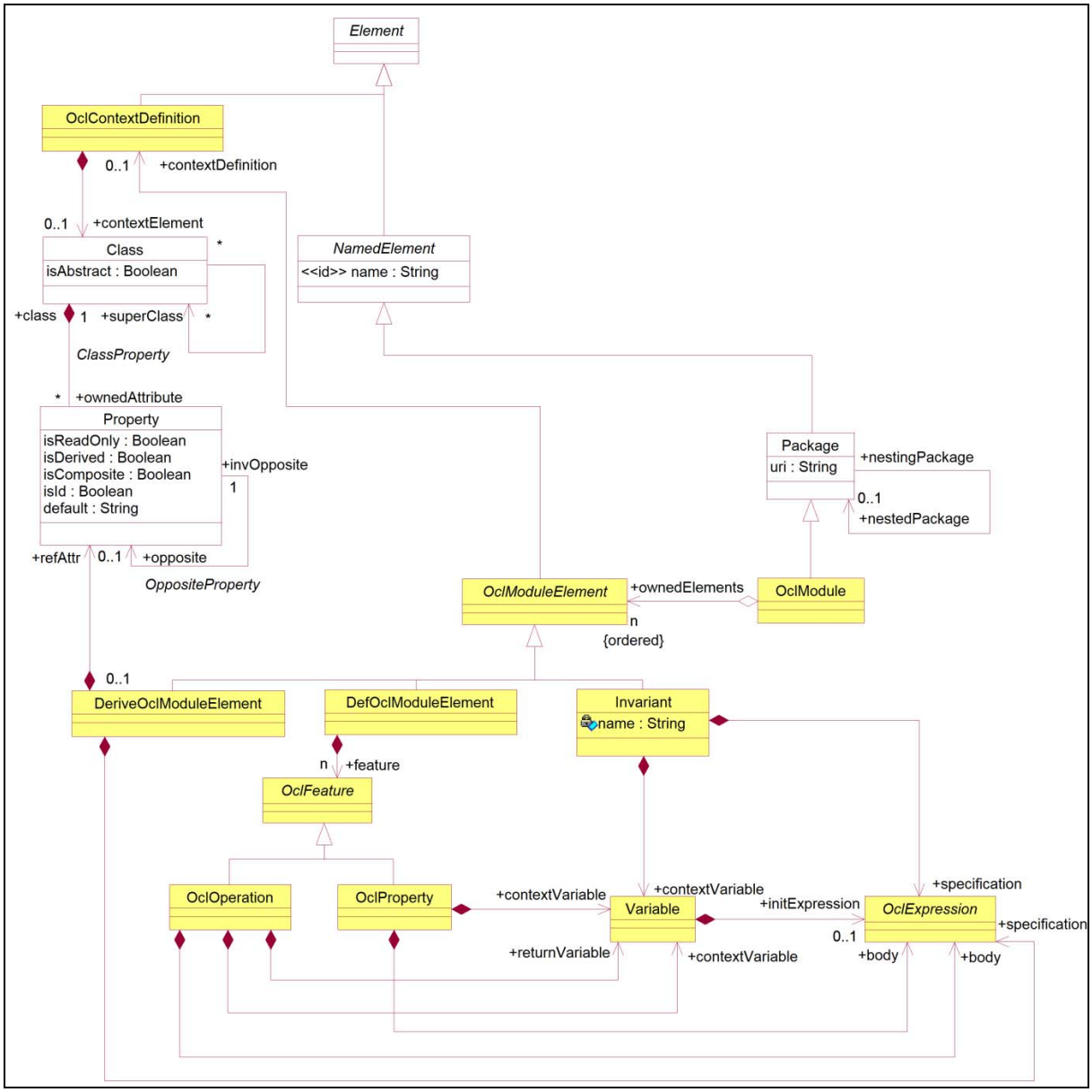

Figura 18 – Elementos do pacote *EnhancedOCL* (MILANOVIC et al., 2007).

3.3.3 Trabalhos Relacionados

Os trabalhos que buscam relacionar o tratamento das regras de negócio aos diversos níveis de abstração propostos na MDA ganharam novos desafios com o lançamento da SBVR. A integração dessa especificação com os demais padrões propostos pelo OMG, além do uso da mesma em conjunto com abordagens e metamodelos já existentes, são apresentados como questões ainda em discussão, conforme descrito em (OMG, 2008).

Dentre os trabalhos recentes que procuram relacionar o padrão SBVR e a MDA, destaca-se (PAU; CABOT, 2008). Nesse trabalho, é apresentada uma proposta para a transformação automática de um esquema conceitual composto por um modelo de classes UML e por regras de negócio expressas em OCL em um vocabulário em SBVR *Structured English* e formulações semânticas. Nessa abordagem, a transformação de PIM para CIM realizada visa facilitar a validação do esquema conceitual junto aos usuários durante o processo de desenvolvimento de um sistema de informação.

A paráfrase de expressões em OCL usando linguagem natural também é tratada por outros dois trabalhos. Diferentemente da proposta anterior, baseada em transformação de modelos, essas abordagens utilizam gramáticas para realizar a paráfrase das expressões. A primeira delas, apresentada em (BURKE; JOHANNISSON, 2005), descreve um sistema que traduz automaticamente especificações formais de *software* para linguagem natural. O estudo de caso apresentado no trabalho mostra a aplicação do sistema na tradução de expressões em OCL para o Inglês. Já (ZIMBRÃO et al., 2002) apresenta uma ferramenta que gera um texto explicativo em linguagem natural (Português) de uma expressão OCL, com o objetivo de facilitar o trabalho de validação e manutenção de sistemas de informação.

Na linha de transformações de CIM para PIM na MDA, (RAJ; PRABHAKAR; HENDRYX, 2008) propõe uma metodologia para converter um modelo de negócio descrito em SBVR em um conjunto de diagramas UML: diagrama de atividades, diagrama de

seqüências e diagrama de classes. Algumas diretrizes usadas nas transformações descritas nesse trabalho foram retiradas da própria especificação da SBVR, que apresenta uma abordagem simplificada para converter conceitos e fatos em SBVR para um modelo de classes em UML.

Seguindo essa linha de pesquisa, (SOSUNOVAS; VASILECAS, 2007) apresenta uma abordagem para a transformação de modelos ORM (*Object Role Modeling*), usados para especificar esquemas conceituais, em diagramas de classes UML e expressões OCL. Na implementação da transformação proposta, foi utilizada a linguagem de transformação *Atlas Transformation Language* (ATL).

Por fim, alguns trabalhos apresentam padrões para a representação de regras de negócio com o intuito de facilitar o intercâmbio das regras entre sistemas e habilitar transformações entre diferentes níveis de abstração e linguagens de modelagem. Nessa direção, (WAGNER; GIURCA; LUKICHEV, 2006) define uma linguagem de intercâmbio para regras de negócio, chamada REWERSE *Rule Markup Language* (R2ML). Em (MILANOVIC, 2007), o metamodelo da R2ML é usado como ponte para a transformação bidirecional de regras de negócio entre as linguagens *Semantic Web Rule Language* (SWRL) e OCL.

Os trabalhos relacionados mostrados apresentam diversas abordagens para o tratamento das regras de negócio, considerando-se diferentes níveis de abstração. Em linhas gerais, os problemas abordados envolvem a geração de novos modelos a partir de modelos já existentes. De forma mais específica, os trabalhos tratam de como aproveitar os modelos de regras de negócio já especificados na construção de novos modelos de regras de negócio, adequados a outras necessidades. Em alguns casos, são apresentadas transformações no sentido proposto pela MDA, ou seja, do CIM para o PSM, passando pelo PIM. Já outras

abordagens procuram, a partir de um modelo mais específico, construir um modelo com maior nível de abstração, propondo transformações do PIM para o CIM.

A lacuna identificada nos trabalhos apresentados encontra-se justamente na definição de uma transformação de regras de negócio do modelo CIM para o modelo PIM (conforme definido na MDA), utilizando-se as tecnologias, técnicas e padrões adotados no arcabouço. É nesse contexto que está inserida a abordagem de transformação de regras de negócio que será mostrada no próximo capítulo. Na proposta, a transformação de modelos é usada para definir como gerar regras de negócio especificadas em OCL, a partir de regras de negócio expressas como formulações lógicas em SBVR, numa transformação de CIM para PIM na MDA.

4 ABORDAGEM PARA A TRANSFORMAÇÃO DE REGRAS DE NEGÓCIO NA MDA

O objetivo deste capítulo é apresentar uma abordagem para a transformação das regras de negócio do modelo independente de computação para o modelo independente de plataforma na Arquitetura Dirigida por Modelos. Para isso, será apresentada uma visão geral da abordagem proposta e descritos os tipos de regras de negócio, os metamodelos e os mapeamentos envolvidos na transformação. Em seguida, o detalhamento da implementação da abordagem será apresentado, destacando-se o ambiente utilizado, os artefatos criados e o uso da solução de automação desenvolvida.

4.1 Visão Geral da Abordagem

Um processo de desenvolvimento de sistemas é um conjunto de atividades e resultados associados que geram um produto de *software* (SOMMERVILLE, 2003). Embora a especificação da MDA não defina um processo, os conceitos e as tecnologias envolvidos no arcabouço podem ser aplicados tanto em processos de desenvolvimento já existentes, conforme mostrado em (BROWN; CONALLEN, 2005), quanto na definição de novos processos de desenvolvimento de sistemas, como mostrado, por exemplo, em (MACIEL; SILVA; MASCARENHAS, 2006) e (MORGADO, 2007).

Naturalmente, um processo de desenvolvimento que use a MDA deverá possuir atividades que envolvam a criação e a transformação de modelos. Uma vez que a especificação da MDA não estabelece um processo, mas apenas um arcabouço conceitual, e também não indica como os modelos CIM, PIM e PSM devem ser construídos, um passo fundamental na definição de processos de desenvolvimento de sistemas usando MDA é a

escolha de quais metamodelos e modelos farão parte de cada um dos modelos do arcabouço e as definições das transformações entre esses modelos (MACIEL; SILVA; ROSA, 2008).

O entendimento de como a MDA se relaciona com o processo de desenvolvimento é importante para que se possa inserir a abordagem proposta dentro do contexto de construção de sistemas de informação. O objetivo deste trabalho não é propor um processo MDA. Contudo, será mostrado, em linhas gerais, como seria a inserção da abordagem em um processo de desenvolvimento de sistemas de informação, com foco nas atividades de especificação.

A modelagem de um vocabulário de negócio contendo os conceitos, fatos e regras de negócio necessários para a construção do sistema de informação é uma atividade que pode fazer parte das etapas iniciais do processo de especificação. Dentro da MDA, esse modelo encontra-se no nível CIM, uma vez que ele é específico do negócio e independente de computação.

Há diversas formas possíveis de modelar um vocabulário de negócio, inclusive usando linguagem natural. Na abordagem proposta, será considerada a construção do vocabulário de negócio utilizando SBVR *Structured English*, que é a notação recomendada na especificação do padrão SBVR. Além disso, as regras de negócio do vocabulário serão representadas também através de formulações lógicas em SBVR e formalizadas na sintaxe abstrata da SBVR. Conforme visto no capítulo anterior, a SBVR é o padrão adotado pelo OMG para a modelagem de vocabulários e regras de negócio no modelo independente de computação (CIM) da MDA.

Outra atividade que pode ser incorporada em um processo de desenvolvimento de sistemas de informação é a construção do modelo conceitual[16], importante artefato da etapa de

[16] Há divergências quanto ao nome dado a esse modelo na literatura: modelo de domínio, esquema conceitual, modelo conceitual de informação etc. Em linhas gerais, um modelo conceitual deve incluir uma representação

análise. O modelo conceitual ilustra os conceitos importantes do domínio do problema, suas associações e atributos. Um exemplo de linguagem bastante utilizada para a modelagem conceitual é a UML (OMG, 2005).

Normalmente, o diagrama UML utilizado para representar modelos conceituais é o diagrama de classes. Ele descreve os tipos de objetos presentes no sistema e os vários tipos de relacionamentos estáticos existentes entre eles. Uma diagrama de classes permite também representar as propriedades e operações de uma classe e as restrições aplicadas às conexões entre os objetos (FOWLER, 2005).

Considerando-se o contexto de um processo de desenvolvimento baseado em MDA, pode-se definir uma transformação que leva automaticamente conceitos definidos no vocabulário de negócio para um modelo conceitual preliminar. Um exemplo de abordagem para a transformação de termos e fatos em um esquema conceitual pode ser encontrado em (DIAS et al., 2006). A própria especificação da SBVR (OMG, 2008) define algumas diretrizes para a transformação de vocabulários de negócio em modelos de classes. Essas diretrizes foram refinadas em (RAJ; PRABHAKAR; HENDRYX, 2008).

A abordagem de transformação proposta pressupõe a existência de um modelo conceitual em UML, aderente aos conceitos e fatos definidos no vocabulário de negócio SBVR. Na abordagem, esse modelo estará no nível PIM da MDA.

Além dos conceitos e fatos usados na criação do modelo conceitual, o conjunto de regras de negócio contido no vocabulário SBVR pode fazer parte do PIM. Na abordagem proposta, isso será feito com o uso da OCL, linguagem que permite aumentar a expressividade dos modelos da UML, tornando-os mais precisos e detalhados.

---

formal do conhecimento de um domínio que um sistema de informação precisa ter para executar suas funções (OLIVÉ, 2007).

Dentro do contexto apresentado, o objetivo do trabalho é propor uma transformação automática baseada em modelos com o intuito de levar regras de negócio modeladas usando formulações lógicas em SBVR do modelo CIM para expressões OCL aplicáveis sobre um modelo conceitual UML preexistente, no modelo PIM da MDA. A principal motivação para a automatização da transformação é agilizar o processo de criação de modelos de regras de negócio na Arquitetura Dirigida por Modelos.

A abordagem de transformação proposta está limitada ao tratamento das regras de restrição estruturais e das regras de inferência, conforme a classificação definida em (ERIKSSON; PENKER, 2000). Uma classificação adicional foi proposta e para cada uma das categorias estabelecidas foram definidos mapeamentos que tornam possível a transformação de SBVR para OCL. As formulações lógicas precisam estar classificadas previamente em uma das categorias para que os mapeamentos adequados possam ser aplicados.

Além disso, para limitar o escopo da abordagem proposta, o tratamento dos modais (deônticos e aléticos) não será realizado. Por essa razão, para padronizar a representação das regras e facilitar a implementação da abordagem, todas as regras de negócio transformadas deverão ser previamente modeladas como necessidades. Essa estratégia é semelhante à utilizada por (PAU; CABOT, 2008).

4.2 Definição dos Modelos Usados na Abordagem

Na MDA, a transformação de modelos é definida como o processo de converter um modelo em outro modelo de um mesmo sistema (MILLER; MUKERJI, 2003). Como a abordagem proposta é baseada em uma transformação do tipo modelo-modelo, é preciso inicialmente definir os modelos que serão usados no processo.

Nesse contexto, como o objetivo é transformar regras de negócio de SBVR para OCL, os modelos de entrada e de saída da transformação serão modelos em conformidade com a especificação da SBVR e da OCL, respectivamente.

Para garantir que os modelos estejam em conformidade com as especificações citadas é necessário que os mesmos estejam em conformidade com os metamodelos definidos para a SBVR e para a OCL, ou seja, suas sintaxes abstratas. O metamodelo da SBVR é apresentado em (OMG, 2008), enquanto o metamodelo da OCL é descrito em (OMG, 2006).

O metamodelo da SBVR é dividido em cinco partes, contudo na abordagem foram utilizados apenas os elementos do metamodelo pertencentes ao “Vocabulário de Significado e Representação” e ao “Vocabulário de Formulação Lógica da Semântica”. Esse recorte do metamodelo contém todos os elementos necessários para representar as formulações lógicas das regras de negócio usando a sintaxe abstrata da SBVR.

Já o metamodelo de destino da transformação será o metamodelo da sintaxe abstrata da OCL conforme descrito na sua especificação, acrescido das classes do pacote *Enhanced* OCL, proposto em (MILANOVIC et al., 2007).

4.2.1 Construção do Modelo de Entrada

As formulações lógicas em SBVR permitem descrever a estrutura do significado das regras de negócio. Uma regra de negócio descrita em SBVR *Structured English* pode ser especificada utilizando formulações lógicas. Conforme definido no padrão, as regras de negócio devem ser expressas nos vocabulários de negócio usando SBVR *Structured English*, enquanto que as formulações lógicas são utilizadas para representar o significado das regras de negócio. A especificação do padrão SBVR define alguns mapeamentos entre as palavras-chaves do SBVR *Structured English* e os elementos que constituem a formulação lógica.

Na construção dos modelos de entrada para a transformação da abordagem, pressupõe-se que as regras de negócio estejam especificadas como formulações lógicas em SBVR. Nos exemplos de regras de negócio apresentados neste trabalho, as formulações lógicas foram construídas manualmente a partir da expressão em SBVR *Structured English* da regra de negócio e da sua classificação nas categorias propostas na abordagem.

Os modelos de entrada da transformação são instâncias do metamodelo da SBVR. Sua construção é baseada em mapeamentos entre a formulação lógica SBVR da regra de negócio e os elementos do metamodelo da SBVR. No Quadro 2, são apresentados alguns desses mapeamentos.

Quadro 2 – Parte dos mapeamentos para a criação dos modelos de entrada.

Formulação Lógica	Elementos SBVR
The rule is a proposition meant by a necessity formulation.	*Rule* *NecessityFormulation*
The necessity formulation embeds a universal quantification.	*UniversalQuantification*
The universal quantification introduces a variable.	*Variable*
The variable ranges over the concept…	*Concept*
The variable is restricted by an atomic formulation.	*AtomicFormulation*
The atomic formulation is based on the fact type…	*FactType*
The … role is bound to the variable.	*RoleBinding*
The universal quantification scopes over an atomic formulation.	*AtomicFormulation*
The … quantification has the cardinality …	*Integer*
The universal quantification scopes over an instantiation formulation.	*InstantiationFormulation*
The universal quantification scopes over an implication.	*Implication*

A construção do modelo de objetos SBVR para a formulação lógica apresentada na Figura 19, seguindo os mapeamentos mostrados, resulta na criação dos elementos *Rule*, *NecessityFormulation*, *UniversalQuantification*, *Variable* e *Concept*.

The rule is a proposition meant by a necessity formulation.
. The necessity formulation embeds a first universal quantification.
. . The first universal quantification introduces a first variable.
. . . The first variable ranges over the concept “A”.
. . The first universal quantification scopes over...

Figura 19 – Parte de uma formulação lógica.

As associações entre os elementos do modelo seguem as associações definidas no metamodelo da SBVR, garantindo assim a conformidade com a sintaxe abstrata da especificação. A Figura 20 mostra o modelo de objetos resultante para a formulação lógica apresentada.

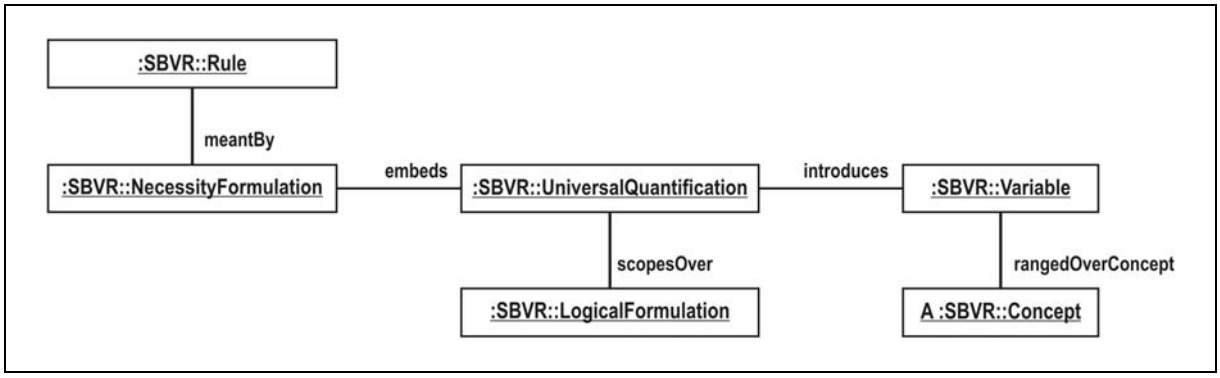

Figura 20 – Modelo de objetos da formulação lógica.

Na transformação proposta, as formulações lógicas representadas através do metamodelo da SBVR estruturam o significado das regras de negócio expressas em SBVR *Structured English*. Na especificação da SBVR, não é descrito como as formulações lógicas das regras de negócio devem ser construídas, apesar das mesmas serem apresentadas como um dos grandes diferenciais do padrão.

Neste trabalho, foi proposta uma classificação para as regras de negócio. Notou-se que as formulações lógicas das regras de negócio em uma mesma categoria são semelhantes, o que pode indicar a existência de um padrão entre essas formulações. Contudo, uma definição mais aprofundada da relação entre as regras de negócio em SBVR *Structured*

English e sua representação em formulações lógicas não está inserida no escopo da abordagem proposta.

4.3 Classificação das Regras de Negócio para a Transformação

As categorias de regras de negócio definidas na abordagem baseiam-se na observação dos exemplos de regras disponíveis na especificação da SBVR (OMG, 2008) e na classificação definida em (ERIKSSON; PENKER, 2000), que divide as regras de negócio em regras de restrição e regras de derivação.

A transformação modelo-modelo aqui apresentada é classificada também como automática, ou seja, ela aplica um conjunto de mudanças em um modelo baseadas em regras de transformação predefinidas. Esse tipo de transformação exige que o modelo de entrada esteja suficientemente completo (sintaticamente e semanticamente) e pode requerer que os modelos sejam marcados com informações específicas para as regras de transformação que serão aplicadas.

A completude pode ser assegurada através da classificação dos modelos de entrada. Na abordagem proposta, um modelo de entrada especifica, na sintaxe abstrata da SBVR, a formulação lógica de uma regra de negócio. Além disso, para que esse modelo de entrada seja válido, ele deve ser classificado em uma das categorias propostas. A indicação da categoria da regra de negócio é fundamental para que as regras de transformação adequadas possam ser aplicadas. Na abordagem, a escolha da categoria é feita pelo responsável pela modelagem da regra de negócio em SBVR e explicitamente indicada no modelo de entrada.

Já a marcação com informações específicas é feita através da identificação dos tipos dos elementos, ou seja, a qual classe do metamodelo um elemento de um modelo de entrada pertence. Essa identificação é fundamental para que os mapeamentos propostos na abordagem possam ser aplicados nos elementos dos modelos de entrada.

Automatizar uma transformação significa dizer como elementos de entrada transformam-se em elementos de saída, a partir de mapeamentos predeterminados. A criação de categorias e a classificação dos modelos de entrada facilitam a automatização do processo de transformação, uma vez que, a partir da correta classificação das regras de negócio, é possível aplicar as regras de transformação adequadas e padronizar a transformação para cada categoria.

A classificação proposta nesta abordagem não engloba todas as regras de negócio que podem ser representadas usando SBVR *Structured English*, contudo será possível notar que ela incorpora um número significativo de casos no exemplo de utilização da abordagem, apresentado no próximo capítulo. Porém, cabe ressaltar que novas categorias de regras de negócio podem ser incorporadas à abordagem, ampliando a cobertura da classificação apresentada.

Nas seções seguintes, serão apresentadas as categorias de regras de negócio definidas na abordagem.

4.3.1 Restrição simples

O caso mais trivial de regra de negócio é a restrição simples. Nesse tipo de regra, as propriedades de um conceito possuem restrições quanto aos valores que elas podem assumir. A restrição pode se dar de três formas: por um valor literal, por outra propriedade do conceito ou por uma propriedade de outro conceito.

Exemplos:

> É necessário que a duração de um aluguel seja de, no máximo, 90 dias.
>
> É necessário que a data de início do aluguel seja posterior à data de reserva do aluguel.

É necessário que a data de expiração da carteira de motorista do locatário de um aluguel seja posterior à data prevista de retorno do aluguel.

4.3.2 Restrição de cardinalidade

A restrição de cardinalidade ocorre quando existe um tipo de fato associativo entre dois conceitos e há uma limitação na quantidade de instâncias que podem participar da associação.

Exemplo: É necessário que cada aluguel tenha exatamente uma agência de retorno.

4.3.3 Restrição de tipo

Essa restrição pressupõe a existência de um tipo de fato associativo entre dois conceitos e que um desses conceitos possui categorias. A restrição de tipo se aplica, nesse caso, indicando qual categoria do conceito pode participar da associação.

Exemplo: É necessário que o locatário de um aluguel por pontos seja um membro do clube de fidelidade[17].

4.3.4 Restrição simples sobre elementos de uma associação

Pode ser considerada uma variação da restrição simples. Nesse caso, a restrição se aplica a uma propriedade de todas as instâncias que participam de uma associação entre dois conceitos. Além disso, o fato associativo entre os conceitos possui cardinalidade maior que um em uma das pontas da associação.

Exemplo: É necessário que cada motorista de um aluguel seja qualificado[18].

---

[17] “Membro do clube de fidelidade” é uma categoria de locatário.

4.3.5 Derivação simples

As derivações são regras do tipo “se-então”, modeladas em SBVR com o uso de implicações. Possuem um antecedente e um conseqüente, cuja relação é a seguinte: o significado do conseqüente deve ser verdadeiro, sempre que o significado do antecedente for verdadeiro.

A derivação simples engloba as regras de negócio que possuem restrições simples tanto no antecedente, quanto no conseqüente.

Exemplo: Se o país da agência de retirada não é o país de registro do carro alugado, então é necessário que o país da agência de retorno seja o país de registro do carro alugado.

4.3.6 Derivação com cardinalidade mínima

A derivação com cardinalidade mínima compreende as regras de negócio que possuem restrição simples no antecedente e um tipo de fato associativo entre dois conceitos no conseqüente. Nesse caso, a cardinalidade mínima da associação é um.

Exemplo: Se a data de retorno do aluguel é posterior à data de tolerância do aluguel, então é necessário que incida uma multa por atraso sobre o aluguel.

4.3.7 Derivação com restrição simples sobre elementos de uma associação

A derivação com restrição simples sobre elementos de uma associação compreende as regras de negócio que possuem restrição simples no antecedente e uma restrição aplicada a

---

[18] Um aluguel possui um ou mais motoristas.

uma propriedade de todas as instâncias que participam de uma associação entre dois conceitos no conseqüente.

Exemplo: Se um aluguel está aberto, então cada motorista do aluguel não é um motorista barrado.

4.4 Mapeamentos da Transformação

A definição dos mapeamentos para as categorias propostas na abordagem de transformação foi feita a partir da análise de exemplos de regras de negócio disponíveis na especificação da SBVR. Assim, para cada uma das categorias, foram selecionados alguns exemplos em SBVR *Structured English*. As formulações lógicas em SBVR e os modelos de objetos para essas formulações foram construídos manualmente. A seguir, para cada regra de exemplo foi especificada uma sugestão de expressão OCL correspondente e também, de forma manual, foram construídos os modelos de objetos representando essas expressões OCL na sintaxe abstrata da linguagem.

A partir da comparação entre os modelos de objetos SBVR e OCL construídos, foi possível estabelecer os mapeamentos entre os elementos dos metamodelos da SBVR e da OCL para cada uma das categorias de regras de negócio e, assim, definir as regras de transformação para a transformação proposta na abordagem. Cabe ressaltar que a tradução em OCL das regras de negócio foi avaliada também por um especialista na linguagem.

Nesta seção, os mapeamentos entre os metamodelos de origem e de destino são apresentados agrupados conforme a categoria de regra de negócio ao qual eles se aplicam. Para cada categoria, são apresentados um exemplo de regra de negócio em SBVR *Structured English*, a formulação lógica correspondente ao exemplo e a expressão OCL que representa a regra de negócio modelada em SBVR. Além disso, o conjunto de mapeamentos necessários para transformar um modelo de objetos em conformidade com a sintaxe abstrata da SBVR em

um modelo de objetos em conformidade com a sintaxe abstrata da OCL é mostrado para cada uma das categorias da abordagem.

4.4.1 Mapeamentos Comuns

Há um conjunto de mapeamentos que é comum a todas as categorias de regras de negócio consideradas na abordagem. O objetivo desses mapeamentos é realizar a transformação dos elementos SBVR que originarão os elementos da OCL que compõem a parte inicial da expressão OCL invariante, mostrada na Figura 21.

```
context <nome da classe>
inv <nome da invariante>:
```

Figura 21 – Parte inicial de uma invariante.

As formulações lógicas de todas as regras de negócio da abordagem possuem uma parte comum, apresentada na Figura 22. É essa parte da formulação lógica que dá origem aos elementos de entrada necessários para o conjunto de mapeamentos aqui apresentados.

The rule is a proposition meant by a necessity formulation.
. The necessity formulation embeds a first universal quantification.
. . The first universal quantification introduces a first variable.
. . . The first variable ranges over the concept “A”.
. . The first universal quantification scopes over...

Figura 22 – Formulação lógica comum às regras de negócio na abordagem.

Os elementos do metamodelo da OCL criados nessa parte da transformação pertencem ao pacote *Enhanced* OCL. Em linhas gerais, um elemento SBVR *Rule*, que representa a regra a ser transformada, é mapeado para um elemento *OCLModule*, enquanto um elemento *NecessityFormulation* é transformado em um elemento *Invariant*. Os elementos *UniversalQuantification* e *Variable* são usados para definir a variável *self* (usada nos demais mapeamentos) e a classe de contexto da invariante (elemento *Class*). A associação

contextDefinition do elemento *Invariant* aponta para o elemento *Class* criado, enquanto a associação *specification* aponta para um dos elementos criados nos próximos mapeamentos e que corresponde à especificação da invariante.

Os mapeamentos definidos na transformação serão apresentados em quadros com três colunas que mostram as classes do metamodelo da SBVR e as classes e propriedades do metamodelo da OCL. Nos quadros, é possível ver quais são os elementos do metamodelo da OCL que deverão ser criados a partir de um elemento do metamodelo da SBVR.

Na primeira coluna de cada quadro, são mostradas as classes do metamodelo da SBVR usadas na transformação. Na segunda coluna, são mostradas as classes do metamodelo da OCL que deverão ser instanciadas na transformação. Na terceira coluna, é mostrado como as propriedades (atributos e associações) das classes do metamodelo da OCL são iniciados. Os atributos (mostrados em itálico) são preenchidos com valores literais. Já as associações (mostradas em sublinhado) apontam para as classes do metamodelo da OCL instanciadas na transformação. Os elementos dos metamodelos da SBVR e da OCL podem ser vistos, respectivamente, nas figuras da seção 3.3.1.5 e da seção 3.3.2.3.

Quadro 3 – Mapeamentos para a criação dos elementos da invariante.

SBVR		OCL		Propriedades
Rule	(A1)	OCLModule	(A1.1)	ownedElements ← A2.1
NecessityFormulation	(A2)	Invariant	(A2.1)	contextDefinition ← A3.1 specification ← B1.1, C1.1, D1.1, E1.1, F1.1, G1.1 ou H1.1
UniversalQuantification	(A3)	ContextDefinition	(A3.1)	contextElement ← A4.2
Variable	(A4)	Variable	(A4.1)	*name* ← "self"
		Class[19]	(A4.2)	*name* ← nome do conceito apontado pela variável SBVR

[19] O elemento *Class* pertence ao metamodelo da UML.

O conjunto completo dos mapeamentos usados para a criação da parte inicial da invariante é mostrado no Quadro 3, no qual se pode ver os elementos de origem SBVR, os elementos de destino OCL criados e como as propriedades dos elementos de destino são iniciadas.

4.4.2 Mapeamentos para Regras de Restrição

Os mapeamentos para a transformação das regras de restrição são apresentados nas próximas seções, seguindo a classificação de regras de negócio proposta na abordagem.

4.4.2.1 Restrição simples

Em um modelo de classes UML, os conceitos podem ser representados como classes e propriedades. Conforme visto, a restrição simples corresponde a restrições sobre os valores possíveis que uma característica de um conceito pode ter. Essas características podem ser transformadas em propriedades no modelo de classes. A abordagem propõe a transformação da restrição sobre os valores da característica em uma expressão invariante OCL que restrinja os valores possíveis da propriedade de uma classe.

Um exemplo de formulação lógica para a restrição simples, quando há comparação entre duas propriedades de um mesmo conceito, é mostrado na Figura 23. A regra de negócio representada é "*It is necessary that the pick-up branch of a round trip rental is the return branch of the round trip rental*".

The rule is a proposition meant by a necessity formulation.
. The necessity formulation embeds a first universal quantification.
. . The first universal quantification introduces a first variable.
. . . The first variable ranges over the concept “round trip rental”.
. . The first universal quantification scopes over a second universal quantification.
. . . The second universal quantification introduces a second variable.
. . . . The second variable ranges over the concept “pick-up branch”.
. . . . The second variable is restricted by an atomic formulation.
. . . . . The atomic formulation is based on the fact type “round trip rental has pick-up branch”.
. . . . . The “round trip rental” role is bound to the first variable.
. . . . . The “pick-up branch” role is bound to the second variable.
. . . The second universal quantification scopes over a third universal quantification.
. . . . The third universal quantification introduces a third variable.
. . . . . The third variable ranges over the concept “return branch”.
. . . . . The third variable is restricted by an atomic formulation.
. . . . . The atomic formulation is based on the fact type “round trip rental has return branch”.
. . . . . The “round trip rental” role is bound to the first variable.
. . . . . The “return branch”role is bound to the third variable.
. . . . The third universal quantification scopes over an atomic formulation.
. . . . . The atomic formulation is based on the fact type “thing1 is thing2”.
. . . . . The “thing1” role is bound to the second variable.
. . . . . The “thing2” role is bound to the third variable.

Figura 23 – Uma formulação lógica para a restrição simples.

A formulação lógica da restrição simples é transformada em uma invariante OCL aplicada sobre uma classe que corresponde ao seu contexto. As variações possíveis são comparar uma propriedade da classe com uma outra propriedade ou com um literal.

Para a formulação lógica apresentada, a expressão OCL gerada pela aplicação da transformação é mostrada na Figura 24. Essa invariante tem como contexto a classe *RoundTripRental* e se aplica ao modelo de classes descrito no apêndice C. Nesse exemplo, os conceitos *pickUpBranch* e *returnBranch* presentes na formulação lógica são representados na expressão OCL pelas propriedades da classe *RoundTripRental* comparadas pelo operador “=” na invariante.

```
context RoundTripRental inv:
self.pickUpBranch = self.returnBranch
```

Figura 24 – Exemplo de expressão OCL para a restrição simples.

Para realizar as comparações de valores em SBVR são utilizadas formulações atômicas, envolvendo fatos associativos, variáveis e conceitos. Não há um padrão para os tipos de fatos que podem ser usados nas comparações, contudo foi estabelecido na abordagem um conjunto de fatos válidos e os seus respectivos mapeamentos para a OCL, conforme mostrado no Quadro 4.

Quadro 4 – Comparações possíveis em SBVR e mapeamentos para OCL.

SBVR	OCL
a is less than b	a < b
a is equal to b a equals b a is b	a = b
a is greater than b	a > b
a is at least b a is greater than or equal to a is not smaller than b	a >= b
a is at most b a is less than or equal to a is not greater than b	a <= b
a is not equal to a not equals b a is not b	a <> b
(date) a is before (date) b	a.isBefore(b)[20]
(date) a is after (date) b	a.isAfter(b) [21]

Nos mapeamentos da restrição simples, o elemento mais importante a ser transformado é *AtomicFormulation*. Para representar a restrição na sintaxe abstrata da OCL, esse elemento será transformado num elemento *OperationCallExp*, cujo nome identifica a operação que será realizada. A definição desse nome é feita a partir da análise do tipo de fato

[20] A operação *isBefore* compara duas datas e retorna verdadeiro se a primeira data for anterior à segunda data.

[21] A operação *isAfter* compara duas datas e retorna verdadeiro se a primeira data for posterior à segunda data.

apontado pela formulação atômica, seguindo os mapeamentos das comparações mostrados no Quadro 4. Os demais elementos criados estão relacionados aos operandos envolvidos na operação: uma propriedade da classe de contexto (*PropertyCallExp*) e uma outra propriedade ou um literal (*StringLiteralExp* ou *IntegerLiteralExp*). Esses elementos são criados a partir dos elementos apontados pelas associações *roleBinding* da formulação atômica, conforme mostrado no Quadro 5.

Quadro 5 – Mapeamentos para a restrição simples.

SBVR	OCL	Propriedades
AtomicFormulation (B1)	OperationCallExp (B1.1)	*name* ← "<=", ">=", "=", "<>", "<" ou ">" source ← B2.1 argument ← B3.1 ou B4.1
Variable (B2)	PropertyCallExp (B2.1)	*name* ← nome do conceito apontado pela variável SBVR source ← B2.2
	VariableExp (B2.2)	referredVariable ← A4.1 ("self")
IndividualConcept ou Integer (B3)	StringLiteralExp ou IntegerLiteralExp (B3.1)	*stringSymbol* ← nome do conceito ou *integerSymbol* ← valor do inteiro
Variable (B4)	PropertyCallExp (B4.1)	*name* ← nome do conceito apontado pela variável SBVR source ← B4.2
	VariableExp (B4.2)	referredVariable ← A4.1 ("self")

4.4.2.2 Restrição de cardinalidade

As regras de restrição de cardinalidade aplicam-se quando há uma limitação no número de instâncias que podem participar de uma associação. No modelo de classes, a restrição de cardinalidade aplica-se quando há uma associação entre duas classes e a regra restringe a quantidade de objetos participantes do relacionamento.

Um exemplo da estrutura da formulação lógica para a restrição de cardinalidade, usando a regra "*It is necessary that each rental has at most four drivers*", é mostrado na Figura 25.

The rule is a proposition meant by a necessity formulation.
. The necessity formulation embeds a first universal quantification.
. . The first universal quantification introduces a first variable.
. . . The first variable ranges over the concept "rental".
. . The first universal quantification scopes over an at-most-n quantification.
. . . The at-most-n quantification has the cardinality 4.
. . . The at-most-n quantification introduces a second variable.
. . . . The second variable ranges over the concept "driver".
. . . The at-most-n quantification scopes over an atomic formulation.
. . . . The atomic formulation is based on the fact type "rental has driver".
. . . . . The role "rental" is bound to the first variable.
. . . . . The role "driver" is bound to the second variable.

Figura 25 – Uma formulação lógica para a restrição de cardinalidade.

A expressão OCL gerada para a restrição de cardinalidade consiste em uma invariante aplicada sobre uma associação da classe de contexto. A operação sobre coleções *size* (retorna o número de elementos numa coleção) é aplicada sobre a associação e seu resultado comparado a um valor numérico. A expressão OCL representando a regra de negócio apresentada é mostrada na Figura 26.

```
context Rental inv:
self.driver -> size() <= 4
```

Figura 26 – Exemplo de expressão OCL para a restrição de cardinalidade.

O principal elemento transformado nos mapeamentos para a restrição de cardinalidade (Quadro 6) é um quantificador, representado por um dos seguintes elementos do metamodelo da SBVR: *AtMostNQuantification*, *ExactlyNQuantification* e *AtLeastNQuantification*. A transformação desse elemento origina um *OperationCallExp*, cujo nome da operação será dado conforme o tipo do quantificador aplicado. Além disso, são criados os elementos *OperationCalExp* com nome *size* e *IntegerLiteralExp*, representando,

respectivamente, a operação sobre coleções *size* e o valor numérico que será usado na comparação. A associação ao qual se aplica a operação *size* é definida pelos elementos *PropertyCallExp* e *VariableExp*, criados a partir do elemento SBVR *Variable*.

Quadro 6 – Mapeamentos para a restrição de cardinalidade.

SBVR	OCL	Propriedade
AtMostNQuantification, ExactlyNQuantification ou AtLeastNQuantification (C1)	OperationCallExp (C1.1)	*name* ← "<=", "=" ou ">=" argument ← C1.2 source ← C1.3
	IntegerLiteralExp (C1.2)	*integerSymbol* ← valor da cardinalidade do quantificador SBVR
	OperationCallExp (C1.3)	*name* ← "size" source ← C2.1
Variable (C2)	PropertyCallExp (C2.1)	*name* ← nome do conceito apontado pela variável SBVR source ← C2.2
	VariableExp (C2.2)	referredVariable ← A4.1 ("self")

4.4.2.3 Restrição de tipo

A aplicação da restrição de tipo em um modelo de classes pressupõe a existência de duas classes que possuem uma associação, sendo que uma delas possui subtipos. A restrição de tipo aplica-se limitando os subtipos da classe que podem participar da associação.

A formulação lógica para a restrição de tipo deve necessariamente possuir um elemento *instantiation formulation*, com referências a um conceito e a um elemento de ligação (*bindable target*). A formulação de instanciação associa as instâncias de um conceito a uma das suas categorias, indicando qual categoria pode participar da associação introduzida na formulação atômica presente na formulação lógica.

Na Figura 27, é mostrada a formulação lógica da regra de negócio "*It is necessary that the renter of each points rental is a club member*". A expressão OCL resultante da transformação para essa formulação lógica é apresentada na Figura 28.

The rule is a proposition meant by a necessity formulation.
. The necessity formulation embeds a first universal quantification.
. . The first universal quantification introduces a first variable.
. . . The first variable ranges over the concept "points rental".
. . The first universal quantification scopes over a second universal quantification.
. . . The second universal quantification introduces a second variable.
. . . . The second variable ranges over the concept "renter".
. . . . The second variable is restricted by an atomic formulation.
. . . . . The atomic formulation is based on the fact type "points rental has renter".
. . . . . The "points rental" role is bound to the first variable.
. . . . . The "renter" role is bound to the second variable.
. . . The second universal quantification scopes over an instantiation formulation.
. . . . The instantiation formulation considers the concept "club member".
. . . . The instantiation formulation binds to the second variable.

Figura 27 – Uma formulação lógica para a restrição de tipo.

Em OCL, a representação da restrição de tipo será feita através de uma expressão com uso da operação *oclIsKindOf*. Ela resulta verdadeiro somente se o tipo do objeto é idêntico ao tipo ou a qualquer subtipo do argumento da operação.

```
context PointsRental inv:
self.renter.oclIsKindOf(ClubMember)
```

Figura 28 – Exemplo de expressão OCL para a restrição de tipo.

No conjunto de mapeamentos aplicado na restrição de tipo (Quadro 7), o principal elemento envolvido é *InstantiationFormulation*. Esse elemento dá origem ao elemento OCL *OperationCallExp* que irá representar a operação *oclIsKindOf* da invariante. Além disso, são criados elementos para a propriedade ao qual a operação é aplicada (*PropertyCallExp* e *VariableExp*) e para o tipo usado como parâmetro da operação (*TypeExp* e *Class*).

Quadro 7 – Mapeamentos para a restrição de tipo.

SBVR		OCL		Propriedade
InstantiationFormulation	(D1)	OperationCallExp	(D1.1)	*name* ← "oclIsKindOf" source ← D2.1 argument ← D3.1
Variable	(D2)	PropertyCallExp	(D2.1)	argument ← D2.2
		VariableExp	(D2.2)	referredVariable ← A4.1 ("self")
Concept	(D3)	TypeExp	(D3.1)	referredType ← D3.2
		Class[22]	(D3.2)	*name* ← nome do conceito SBVR

4.4.2.4 Restrição simples sobre elementos de uma associação

Além da restrição de quantidade de objetos participantes de uma associação, pode existir uma restrição quanto aos valores das propriedades desses objetos. Esse caso é uma variante da restrição simples aplicada aos objetos de um conjunto.

A formulação lógica para uma regra de negócio classificada nessa categoria é idêntica à formulação lógica usada para expressar regras de negócio do tipo restrição simples. A diferença entre elas está na cardinalidade da associação entre os conceitos envolvidos. Na restrição simples, o conceito A só pode se relacionar com no máximo uma instância de um conceito B. Já na restrição sobre os elementos de um conjunto, a associação entre os conceitos A e B deve ser maior que um.

Um exemplo de formulação lógica para a regra de negócio "*It is necessary that each driver of a rental is qualified*" é mostrado na Figura 29.

[22] O elemento *Class* pertence ao metamodelo da UML.

The rule is a proposition meant by a necessity formulation.
. The necessity formulation embeds a first universal quantification.
. . The first universal quantification introduces a first variable.
. . . The first variable ranges over the concept “rental”.
. . The first universal quantification scopes over a second universal quantification.
. . . The second universal quantification introduces a second variable.
. . . . The second variable ranges over the concept “driver”.
. . . . The second variable is restricted by an atomic formulation.
. . . . . The atomic formulation is based on the fact type “rental has driver”.
. . . . . The “rental” role is bound to the first variable.
. . . . . The “driver” role is bound to the second variable.
. . . The second universal quantification scopes over an atomic formulation
. . . . The atomic formulation is based on the fact type “driver is qualified”
. . . . . The “driver” role is bound to the second variable.

Figura 29 – Uma formulação lógica para a restrição simples sobre elementos de uma associação.

A expressão OCL usada para representar esse tipo de regra de negócio na abordagem utiliza a operação *forAll*. A operação é aplicada sobre o conjunto de objetos relacionados a uma associação da classe de contexto. Para cada um dos elementos do conjunto, uma propriedade é testada e esse teste deve ser verdadeiro para todos os elementos do conjunto.

A Figura 30, correspondente a formulação lógica mostrada, apresenta um exemplo, em que o atributo booleano *isQualified* é testado para todos os elementos do conjunto retornado pela navegação por *self.driver*.

```
context Rental inv:
self.driver -> forAll (x |x.isQualified)
```

Figura 30 – Exemplo de expressão OCL para a restrição simples sobre elementos de uma associação.

Na sintaxe abstrata da OCL, a operação *forAll* é expressa através de um elemento *IteratorExp*. Esse elemento é criado a partir de um elemento SBVR *UniversalQuantification*. A propriedade *body* do elemento *IteratorExp* aponta para o elemento *OperationCallExp* criado a partir da formulação atômica (*AtomicFormulation*). Os mapeamentos das operações usadas na formulação atômica são os mesmos definidos para a restrição simples (Quadro 4).

É necessária também a criação de uma variável para iteração sobre os elementos da coleção (elemento OCL *Variable* com nome "x"), um elemento para representar a propriedade ao qual é aplicada a operação *forAll* e um elemento para representar a propriedade a ser testada, ambos do tipo *PropertyCallExp*.

Finalmente, o valor que será usado como argumento do elemento *OperationCallExp* permite duas possibilidades: um literal (*StringLiteralExp* ou *IntegerLiteralExp*) ou uma outra propriedade (representada por outro elemento *PropertyCallExp*).

Quadro 8 – Mapeamentos para a restrição simples sobre elementos de uma associação.

SBVR	OCL	Propriedade
UniversalQuantification (E1)	IteratorExp (E1.1)	*name* ← "forAll" source ← E2.1 body ← E3.1 iterator ← E4.2
Variable (E2)	PropertyCallExp (E2.1)	*name* ← nome do conceito apontado pela variável SBVR source ← E2.2
	VariableExp (E2.2)	referredVariable ← A4.1 ("self")
AtomicFormulation (E3)	OperationCallExp (E3.1)	*name* ← "<=", ">=", "=", "<>", "<" ou ">" source ← E4.1 argument ← E5.1 ou E6.1
Variable (E4)	PropertyCallExp (E4.1)	*name* ← nome do conceito apontado pela variável SBVR source ← E4.3
	Variable (E4.2)	*name* ← "x"
	VariableExp (E4.3)	referredVariable ← E4.2
IndividualConcept ou Integer (E5)	StringLiteralExp ou IntegerLiteralExp (E5.1)	*stringSymbol* ← nome do conceito ou *integerSymbol* ← valor do inteiro
Variable (E6)	PropertyCallExp (E6.1)	*name* ← nome do conceito apontado pela variável SBVR source ← E6.2
	VariableExp (E6.2)	referredVariable ← A4.1 ("self")

Uma variação possível dos mapeamentos mostrados no Quadro 8 ocorre quando o tipo de fato da formulação atômica é uma característica (tipo de fato unário). Nesse caso, o elemento *AtomicFormulation* é transformado em um *PropertyCallExp* e em um *VariableExp*, não sendo necessária a criação do elemento *OperationCallExp*.

4.4.3 Mapeamentos para Regras de Derivação

Os mapeamentos para a transformação das regras de derivação são apresentados nas próximas seções, seguindo a classificação de regras de negócio proposta na abordagem.

4.4.3.1 Derivação simples

As regras de negócio do tipo derivação simples apresentam uma restrição simples no antecedente e uma outra restrição simples no conseqüente e são representadas em SBVR com o uso da operação lógica de implicação (*implication*) em uma formulação lógica.

Uma formulação lógica que representa uma regra de derivação simples em SBVR é mostrada na Figura 31, na qual se pode notar o uso do elemento *implication* e a definição do antecedente e conseqüente da implicação. A regra de negócio especificada nessa formulação lógica é a seguinte: "*It is necessary that if an advance rental is assigned then the store branch of the rental car of the advance rental is the pick-up branch of the advance rental*".

A representação de regras de negócio de derivação em OCL é feita, normalmente, com o uso da operação *implies*, que possui um antecedente e um conseqüente como operandos. No caso da derivação simples, o resultado da operação *implies* é verdadeiro se a restrição sobre uma propriedade de uma classe (expressa no antecedente) for verdadeira e uma outra restrição (expressa no conseqüente) também for verdadeira.

The rule is a proposition meant by a necessity formulation.
. The necessity formulation embeds a first universal quantification.
. . The first universal quantification introduces a first variable.
. . . The first variable ranges over the concept “advance rental”.
. . The first universal quantification scopes over an implication.
. . . The antecedent of the implication is an atomic formulation
. . . . The atomic formulation is based on the fact type “advance rental is assigned”
. . . . . The “advance rental” role is bound to the first variable.
. . . The consequent of the implication is a second universal quantification.
. . . . The second universal quantification introduces a second variable.
. . . . . The second variable ranges over the concept “rental car”.
. . . . . The second variable is restricted by an atomic formulation.
. . . . . The atomic formulation is based on the fact type “advance rental has rental car”.
. . . . . The “advance rental” role is bound to the first variable.
. . . . . The “rental car” role is bound to the second variable.
. . . . The second universal quantification scopes over a third universal quantification.
. . . . . The third universal quantification introduces a third variable.
. . . . . The third variable ranges over the concept “store branch”.
. . . . . The third variable is restricted by an atomic formulation.
. . . . . The atomic formulation is based on the fact type “rental car has store branch”.
. . . . . The “rental car” role is bound to the second variable.
. . . . . The “store branch” role is bound to the third variable.
. . . . . The third universal quantification scopes over a fourth universal quantification.
. . . . . The fourth universal quantification introduces a fourth variable.
. . . . . The fourth variable ranges over the concept “pick-up branch”
. . . . . The fourth variable is restricted by an atomic formulation.
. . . . . The atomic formulation is based on the fact type “advance rental has pick-up branch”.
. . . . . The “advance rental” role is bound to the first variable.
. . . . . The “pick-up branch” role is bound to the fourth variable.
. . . . . The fourth universal quantification scopes over an atomic formulation.
. . . . . The atomic formulation is based on the fact type “thing1 is thing2”.
. . . . . The “thing1” role is bound to the second variable.
. . . . . The “thing2” role is bound to the fourth variable.

Figura 31 – Uma formulação lógica para a derivação simples.

A Figura 32 apresenta a regra de derivação em OCL com a operação *implies* para a formulação lógica da regra de negócio mostrada.

```
context AdvanceRental inv:
self.isAssigned implies
self.rentalCar.storeBranch = self.pickUpBranch
```

Figura 32 – Exemplo de expressão OCL para a derivação simples.

Na sintaxe abstrata da OCL, o principal elemento de uma expressão OCL que possui a operação *implies* é o elemento *OperationCallExp*. Para representar a implicação, a

propriedade *source* do elemento deve apontar para o elemento que representa o antecedente da expressão, enquanto o elemento *argument* deve apontar para o elemento que representa o conseqüente da expressão.

Os elementos de origem que representam o antecedente e conseqüente da implicação são facilmente identificados a partir do elemento SBVR *Implication*, uma vez que esse elemento possui uma propriedade *antecedent* e uma propriedade *consequent*. Os elementos apontados por essas associações são restrições simples e, assim, os mapeamentos usados para a criação dos elementos (mostrados no Quadro 9) são idênticos àqueles apresentados na categoria de regras de restrição simples.

Quadro 9 – Mapeamentos para a derivação simples.

SBVR	OCL	Propriedade
Implication (F1)	OperationCallExp (F1.1)	*name* ← "implies" source ← F2.1 argument ← F6.1
AtomicFormulation (F2)	OperationCallExp (F2.1)	*name* ← "<=", ">=", "=", "<>", "<" ou ">" source ← F3.1 argument ← F4.1 ou F5.1
Variable (F3)	PropertyCallExp (F3.1)	*name* ← nome do conceito apontado pela variável SBVR source ← F3.2
	VariableExp (F3.2)	referredVariable ← A4.1 ("self")
IndividualConcept ou Integer (F4)	StringLiteralExp ou IntegerLiteralExp (F4.1)	*stringSymbol* ← nome do conceito ou *integerSymbol* ← valor do inteiro
Variable (F5)	PropertyCallExp (F5.1)	*name* ← nome do conceito apontado pela variável SBVR source ← F5.2
	VariableExp (F5.2)	referredVariable ← A4.1 ("self")

AtomicFormulation	(F6)	OperationCallExp	(F6.1)	*name* ← “<=”, “>=”, “=”, “<>”, “<” ou “>” source ← F7.1 argument ← F8.1 ou F9.1
Variable	(F7)	PropertyCallExp	(F7.1)	*name* ← nome do conceito apontado pela variável SBVR source ← F7.2
		VariableExp	(F7.2)	referredVariable ← A4.1 (“self”)
IndividualConcept ou Integer	(F8)	StringLiteralExp ou IntegerLiteralExp	(F8.1)	*stringSymbol* ← nome do conceito ou *integerSymbol* ← valor do inteiro
Variable	(F9)	PropertyCallExp	(F9.1)	*name* ← nome do conceito apontado pela variável SBVR source ← F9.2
		VariableExp	(F9.2)	referredVariable ← A4.1 (“self”)

4.4.3.2 Derivação com cardinalidade mínima

Na derivação com cardinalidade mínima, o antecedente da implicação possui uma restrição simples e o conseqüente engloba um tipo de fato associativo. O conseqüente será verdadeiro se houver pelo menos uma instância que satisfaça à associação.

A formulação lógica dessa categoria possui antecedente semelhante à categoria de derivação simples. Já o conseqüente da derivação com cardinalidade mínima possui um quantificador existencial e um tipo de fato associativo a ser avaliado. Na Figura 33, é mostrada a formulação lógica para a regra de negócio “*It is necessary that if the drop-off location of a rental is not the return branch of the rental then the rental incurs a location penalty charge*”.

The rule is a proposition meant by a necessity formulation.
. The necessity formulation embeds a first universal quantification.
. . The first universal quantification introduces a first variable.
. . . The first variable ranges over the concept “rental”.
. . The first universal quantification scopes over an implication.
. . . The antecedent of the implication is a second universal quantification.
. . . . The second universal quantification introduces a second variable.
. . . . . The second variable ranges over the concept “drop-off location”.
. . . . . The second variable is restricted by an atomic formulation.
. . . . . The atomic formulation is based on the fact type “rental has drop-off location”.
. . . . . The “rental” role is bound to the first variable.
. . . . . The “drop-off location” role is bound to the second variable.
. . . . The second universal quantification scopes over a third universal quantification.
. . . . . The third universal quantification introduces a third variable.
. . . . . The third variable ranges over the concept “return branch”.
. . . . . The third variable is restricted by an atomic formulation.
. . . . . The atomic formulation is based on the fact type “rental has return branch”.
. . . . . The “rental” role is bound to the first variable.
. . . . . The “return branch” role is bound to the third variable.
. . . . . The third universal quantification scopes over an atomic formulation.
. . . . . The atomic formulation is based on the fact type “thing1 is not thing2”.
. . . . . The “thing1” role is bound to the second variable.
. . . . . The “thing2” role is bound to the third variable.
. . . The consequent of the implication is an existential quantification.
. . . . The existential quantification introduces a fourth variable.
. . . . . The variable ranges over the concept “location penalty charge”.
. . . . The existential quantification scopes over an atomic formulation.
. . . . . The atomic formulation is based on the fact type “rental incurs location penalty charge”.
. . . . . The “rental” role is bound to the first variable.
. . . . . The “location penalty charge” role is bound to the fourth variable.

Figura 33 – Uma formulação lógica para a derivação com cardinalidade mínima.

O significado desejado para essa categoria no modelo de classes é que se a restrição sobre uma propriedade de uma classe (expressa no antecedente) for verdadeira, então deve existir pelo menos uma ocorrência de uma classe associada à ocorrência de outra classe. A modelagem dessa categoria com OCL é feita com a operação *implies*. A expressão do antecedente é uma restrição simples e o conseqüente é modelado usando a operação *notEmpty*, que retorna verdadeiro se a coleção possuir um ou mais elementos. A Figura 34 mostra um exemplo de expressão OCL para essa categoria, baseado na formulação lógica anteriormente apresentada.

```
context Rental inv:
self.dropOffLocation<> self.returnBranch implies
self.locationPenaltyCharge -> notEmpty()
```

Figura 34 – Exemplo de expressão OCL para a derivação com cardinalidade mínima.

Os mapeamentos dos elementos do antecedente da implicação na derivação com cardinalidade mínima são semelhantes à derivação simples. No conseqüente, modelado em SBVR com o uso do elemento *ExistentialQuantification*, os elementos na sintaxe abstrata da OCL são *OperationCallExp* (representando a operação *notEmpty*), *PropertyCallExp* (usado para indicar a propriedade na qual será aplicada a operação *notEmpty*) e *VariableExp* (usado como referência para a variável *self*). O Quadro 10 apresenta o conjunto dos mapeamentos definidos para a derivação com cardinalidade mínima.

Quadro 10 – Mapeamentos para a derivação com cardinalidade mínima[23].

SBVR	OCL	Propriedade
ExistentialQuantification (G6)	OperationCallExp (G6.1)	*name* ← **"**notEmpty**"** source ← G7.1
Variable (G7)	PropertyCallExp (G7.1)	*name* ← nome do conceito apontado pela variável SBVR source ← G7.2
	VariableExp (G7.2)	referredVariable ← A4.1 ("self")

4.4.3.3 Derivação com restrição simples sobre elementos de uma associação

A derivação com restrição simples sobre elementos de uma associação é uma variação da derivação simples. Nessa categoria, o conseqüente possui uma restrição simples que é aplicada sobre uma propriedade de todos os elementos que participam de uma associação. Essa associação deve possuir cardinalidade maior que um.

[23] Os mapeamentos G1-G5 são análogos aos mapeamentos F1-F5.

A formulação lógica para essa derivação é semelhante à apresentada na derivação simples. Na Figura 35, a formulação lógica da regra de negócio "*It is necessary that if a rental is open then each driver of the rental is not a barred driver*" apresenta no seu conseqüente uma formulação atômica com tipo de fato unário (característica).

The rule is a proposition meant by a necessity formulation.
. The necessity formulation embeds a first universal quantification.
. . The first universal quantification introduces a first variable.
. . . The first variable ranges over the concept "rental".
. . The first universal quantification scopes over an implication.
. . . The antecedent of the implication is an atomic formulation
. . . . The atomic formulation is based on the fact type "rental is open"
. . . . . The "rental" role is bound to the first variable.
. . . The consequent of the implication is a second universal quantification.
. . . . The second universal quantification introduces a second variable.
. . . . . The second variable ranges over the concept "driver".
. . . . . The second variable is restricted by an atomic formulation.
. . . . . The atomic formulation is based on the fact type "rental has driver".
. . . . . The "rental" role is bound to the first variable.
. . . . . The "driver" role is bound to the second variable.
. . . . The second universal quantification scopes over a logical negation.
. . . . . The logical operand of the logical negation is an atomic formulation.
. . . . . The atomic formulation is based on the fact type "driver is barred"
. . . . . The "driver" role is bound to the second variable.

Figura 35 – Uma formulação lógica para a derivação com restrição simples sobre elementos de uma associação.

A expressão OCL, gerada pela transformação da formulação lógica mostrada, utiliza a operação *forAll* no conseqüente de uma operação *implies*, conforme ilustrado na Figura 36.

```
context Rental inv:
self.isOpen implies
self.driver -> forAll (x | not x.isBarred)
```

Figura 36 – Exemplo de expressão OCL para a derivação com restrição simples sobre elementos de uma associação.

O conjunto de mapeamentos usado para a transformação do antecedente é idêntico ao da derivação simples. Já os mapeamentos para os elementos do conseqüente são semelhantes

aos mapeamentos definidos na categoria de restrição simples sobre elementos de uma associação, conforme mostrado no Quadro 11.

Quadro 11 – Mapeamentos para a derivação com restrição simples sobre elementos de uma associação [24].

SBVR		OCL		Propriedade
UniversalQuantification	(H6)	IteratorExp	(H6.1)	*name* ← "forAll" source ← H7.1 body ← H8.1 iterator ← H9.2
Variable	(H7)	PropertyCallExp	(H7.1)	*name* ← nome do conceito apontado pela variável SBVR source ← H7.2
		VariableExp	(H7.2)	referredVariable ← A4.1 ("self")
AtomicFormulation	(H8)	OperationCallExp	(H8.1)	*name* ← "<=", ">=", "=", "<>", "<" ou ">" source ← H9.1 argument ← H10.1 ou H11.1
Variable	(H9)	PropertyCallExp	(H9.1)	*name* ← nome do conceito apontado pela variável SBVR source ← H9.3
		Variable	(H9.2)	*name* ← "x"
		VariableExp	(H9.3)	referredVariable ← H9.2
IndividualConcept ou Integer	(H10)	StringLiteralExp ou IntegerLiteralExp	(H10.1)	*stringSymbol* ← nome do conceito ou *integerSymbol* ← valor do inteiro
Variable	(H11)	PropertyCallExp	(H11.1)	*name* ← nome do conceito apontado pela variável SBVR source ← H11.2
		VariableExp	(H11.2)	referredVariable ← A4.1 ("self")

[24] Os mapeamentos H1-H5 são análogos aos mapeamentos F1-F5.

4.5 Implementação da Abordagem

Com o intuito de apoiar a utilização prática da abordagem proposta para a transformação de regras de negócio, foi desenvolvida uma solução que automatiza os mapeamentos apresentados. Para essa implementação, foi utilizada como linguagem de transformação de modelos a *Atlas Transformation Language* (ATL). Nas próximas seções, serão mostrados maiores detalhes sobre a solução construída.

4.5.1 Ambiente da Implementação

Dentre as várias linguagens e ferramentas disponíveis para a transformação de modelos, a escolhida para a implementação da abordagem foi a *Atlas Transformation Language* (ATL). Uma grande comunidade de usuários, um ambiente de desenvolvimento estável e integrado, diversos exemplos e projetos completos disponíveis, além de suporte para vários tipos de modelos e metamodelos, foram alguns dos argumentos que motivaram o uso da linguagem ATL.

A ATL é uma linguagem de transformação híbrida (declarativa e imperativa), cujos tipos de dados e expressões declarativas baseiam-se na OCL. O ambiente utilizado para o desenvolvimento na linguagem, chamado de ATL *Development Tools* (ADT), é integrado à plataforma de ferramentas do projeto Eclipse. Cabe ressaltar que, desde 2007, ATL é reconhecida como solução padrão para a implementação de transformações de modelos dentro do projeto *Model to Model*[25] (M2M) do Eclipse.

Para a criação dos metamodelos e modelos necessários para a transformação desenvolvida foi utilizado o *Eclipse Modeling Framework* (EMF). EMF consiste em um arcabouço de modelagem conceitual e desenvolvimento de ferramentas baseado em modelos

[25] Projeto *Model to Model* (M2M): http://www.eclipse.org/m2m/.

estruturados. Em EMF, os modelos podem ser usados tanto como documentação de um sistema, quanto como entrada para a geração de partes de uma aplicação ou até de uma aplicação completa. Além disso, EMF pode também ser usado como *model handler* para ferramentas de transformação de modelos, como ATL, por exemplo.

O metamodelo usado para representar os modelos em EMF é chamado Ecore (Figura 37). O Ecore é também um modelo EMF e, desse modo, Ecore é o próprio metamodelo de Ecore (BUDINSKY et al., 2003). A criação de modelos em EMF pode ser feita através da edição direta do arquivo Ecore, usando o editor gráfico disponível no ambiente. Além disso, é possível criar modelos em UML usando uma ferramenta de modelagem, como, por exemplo, IBM *Rational Rose*, e, em seguida, importá-los em um projeto EMF. Automaticamente, o modelo de classes criado é convertido em um modelo Ecore correspondente, que é armazenado em um arquivo no formato XMI (Ecore XMI).

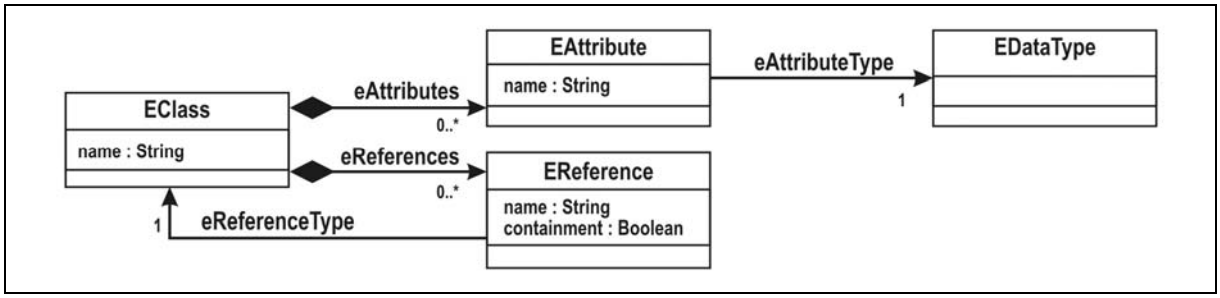

Figura 37 – Representação simplificada do modelo Ecore (BUDINSKY et al., 2003).

Além da criação de modelos, EMF permite gerar um editor gráfico para um modelo Ecore, permitindo que instâncias desse modelo possam ser facilmente criadas. Um breve tutorial sobre a criação de modelos e instâncias de modelos usando EMF pode ser encontrado em (IBM, 2008).

Uma funcionalidade necessária na solução desenvolvida é a representação da regra de negócio transformada na sintaxe concreta da OCL. A transformação do modelo OCL representando a regra de negócio para sua expressão em sintaxe textual é realizada com a

utilização de *Textual Concrete Sintax* (TCS), integrado à IDE de desenvolvimento ATL. TCS representa uma linguagem específica de domínio para a definição de sintaxes concretas no Desenvolvimento Dirigido por Modelos. A TCS pode ser usada tanto para gerar a árvore sintática abstrata a partir do texto, quanto para gerar texto a partir da árvore sintática abstrata. No último caso, para cada elemento do metamodelo deve ser definido em TCS sua descrição em sintaxe textual, usando os construtos da linguagem (MILANOVIC, 2007).

A execução de uma transformação ATL pode ser feita através do assistente disponível na própria IDE ou através de um arquivo de configuração *Ant*[26]. *Apache Ant* é uma ferramenta em Java para automatizar tarefas de *build* que utiliza arquivos de configuração escritos em XML. A extensão da ferramenta que permite executar comandos específicos para a transformação de modelos é chamada de AM3 *Ant tasks*[27].

Uma das vantagens de se utilizar um arquivo de configuração é a possibilidade de executar conjuntamente a transformação de modelos e as tarefas anteriores necessárias para a correta execução do programa, como a carga dos modelos e metamodelos, definindo as mesmas em um único arquivo XML. O arquivo de configuração criado é executado automaticamente dentro do ambiente Eclipse. Outra vantagem é a possibilidade de programar a execução da transformação de modelo para texto dentro do mesmo arquivo, permitindo que o processo completo de transformação seja realizado de uma vez.

4.5.2 Descrição da Implementação

A primeira etapa da implementação de uma transformação usando ATL é a criação de um projeto ATL. O ambiente de desenvolvimento provê um assistente para automatizar

[26] *The Apache Ant project*: http://ant.apache.org/index.html.

[27] AM3 *Ant Tasks*: http://wiki.eclipse.org/index.php/AM3_Ant_Tasks.

essa etapa, sendo necessário apenas fornecer o nome desejado para o projeto. Uma prática recomendável é a criação de uma estrutura de pastas para separar os artefatos que serão usados no projeto.

4.5.2.1 Implementação dos metamodelos

Após a criação do projeto ATL, a próxima etapa é a construção dos arquivos que correspondem ao metamodelo de origem e ao metamodelo de destino usados na transformação. No caso da abordagem proposta, o metamodelo de origem é o metamodelo da SBVR e o metamodelo de destino é o metamodelo da OCL.

Para modelar as classes e relacionamentos do metamodelo da SBVR foi utilizado o IBM *Rational Rose*. Os mapeamentos utilizam basicamente elementos do "Vocabulário de Formulação Lógica da Semântica" e do "Vocabulário de Significado e Representação". Por isso, não houve necessidade de representar todo o metamodelo da SBVR, mas apenas os principais elementos dos dois vocabulários citados. Algumas modificações no metamodelo original foram necessárias e serão explicadas posteriormente. O arquivo mdl gerado no *Rational Rose* foi importado dentro do ambiente Eclipse, através de um assistente, e um modelo Ecore correspondente ao metamodelo foi gerado automaticamente.

Cabe ressaltar que se encontra em andamento uma iniciativa SBVR dentro do projeto *Model Development Tools*[28] (MDT) do Eclipse. O principal objetivo dessa iniciativa é prover metamodelos Ecore para a especificação da SBVR. Nas discussões dos participantes do projeto, é consenso a necessidade de criação de dois metamodelos para a especificação. O primeiro metamodelo proposto, chamado SBVR *Exchange Metamodel*, será usado para a

[28] Projeto *Model Development Tools* (MDT): http://www.eclipse.org/modeling/mdt/.

representação em XMI de vocabulários SBVR, em conformidade com o formato SBVR para a troca de documentos.

Já a segunda proposta de metamodelo, chamado de SBVR *Tools Metamodel*, será otimizada para o uso por aplicações computacionais, incluindo ferramentas de transformação (ECLIPSE, 2008). Até o momento não há uma versão completa dos metamodelos disponível para uso e, por isso, foi necessário implementá-lo. A estratégia de modelagem usada no metamodelo da solução proposta assemelha-se à do metamodelo SBVR *Tools Metamodel*.

Para o metamodelo da OCL, foi utilizado um modelo Ecore disponível em um dos cenários do ATL *Zoo*[29] (um repositório de transformações em ATL). O metamodelo da OCL disponível inclui o pacote *Enhanced* OCL, necessário para a representação das expressões OCL do tipo invariante na transformação. Além disso, um arquivo TCS disponível para esse metamodelo também foi aproveitado.

Devido a restrições impostas pela TCS, o metamodelo da OCL utilizado possui algumas diferenças em relação à especificação do OMG para a OCL. Foram criados os elementos *OperatorCallExp* (para operações do tipo “=”, “not”, “or”, “implies”, “>”, “<” etc.), *CollectionOperationCallExp* (para operações sobre coleções: “notEmpty”, “isEmpty”, “size” etc.) e *Iterator* (para variáveis de iteração).

Além disso, foi criado um pacote chamado *EnhancedOCL*, para permitir suporte a elementos específicos da sintaxe concreta da OCL. No pacote, destacam-se o elemento *Invariant*, usado na abordagem para representar as invariantes, além de um elemento para a definição do contexto das expressões OCL (*OclContextDefinition*) e um módulo para agrupar as invariantes (*OclModule*). O elemento *Class* pertencente ao metamodelo da UML também foi incluído no metamodelo usado (MILANOVIC, 2007).

---

[29] ATL *Zoo*: http://www.eclipse.org/m2m/atl/atlTransformations/.

O Quadro 12 apresenta um resumo dos principais artefatos utilizados na solução desenvolvida. Foram criados três modelos Ecore para os metamodelos da SBVR, OCL e TCS. Além disso, o programa de transformação ATL foi implementado em um arquivo único, incorporando os mapeamentos propostos nas categorias de regras de negócio definidas na abordagem.

Quadro 12 – Resumo dos artefatos usados na solução.

Nome do artefato	Tipo	Descrição
SBVR.mdl	Arquivo MDL	Metamodelo da SBVR em formato Rational Rose.
SBVR.ecore	Arquivo XMI	Metamodelo da SBVR em formato Ecore.
OCL.ecore	Arquivo XMI	Metamodelo da OCL em formato Ecore.
SBVR2OCL.atl	Arquivo ATL	Código fonte da transformação.
SBVR2OCL-All.xml	Arquivo de configuração *Ant* (XML)	Arquivo de configuração para automatizar a execução da transformação.
OCL.tcs	Arquivo TCS	Sintaxe concreta textual do metamodelo da OCL.
TCS.ecore	Arquivo XMI	Metamodelo da TCS em formato Ecore

4.5.2.2 Transformação de modelo para modelo

Um programa de transformação ATL ou módulo ATL é composto por regras que definem como os elementos do modelo de origem são tratados e navegados a fim de se criar e iniciar novos elementos no modelo de destino. Na transformação da abordagem, os modelos de origem devem estar em conformidade com o metamodelo da SBVR, enquanto os modelos de destino gerados devem estar em conformidade com o metamodelo da OCL, para posteriormente serem transformados em expressões OCL em formato texto.

Um módulo ATL é formado pelos seguintes elementos (ATL, 2006):

- Uma seção de cabeçalho que define atributos relativos ao módulo;
- Uma seção de importação opcional que permite reusar outras bibliotecas ATL;

- Uma seção de *helpers* (análogos a métodos em linguagem Java, por exemplo);
- Uma seção de regras que definem como os modelos de saída são gerados a partir dos modelos de entrada.

Os principais tipos de regras de transformação permitidas na linguagem ATL são:

- Regras do tipo *matched* – regras declarativas que são automaticamente chamadas durante a execução do módulo de transformação ATL. Nesse caso, cada elemento do modelo de origem considerado em um regra do tipo *matched* é automaticamente transformado em um ou mais elementos do modelo de destino;
- Regras do tipo *lazy* – regras declarativas que devem ser explicitamente chamadas no módulo de transformação. Cada vez que uma regra desse tipo é chamada, novos elementos de destino são criados, independente do elemento de origem;
- Regras do tipo *unique lazy* – regras declarativas que geram sempre os mesmos elementos de destino quando chamadas para certo elemento de origem;
- Regras do tipo *called* – regras imperativas que devem ser explicitamente chamadas no módulo de transformação.

Recomenda-se que a implementação de transformações em ATL utilize preferencialmente regras declarativas, sendo as regras imperativas reservadas para exceções que não podem ser tratadas com outros tipos de regras (ATL, 2006).

Na implementação realizada, foi desenvolvido um único programa de transformação que engloba todas as categorias de regras de negócio tratadas pela abordagem. Os mapeamentos apresentados serviram como base para a construção do módulo ATL e

permitiram a definição dos mapeamentos entre os elementos do metamodelo da SBVR e do metamodelo da OCL implementados, conforme mostrado no Quadro 13.

Quadro 13 – Parte dos mapeamentos entre os elementos dos metamodelos.

Metamodelo da SBVR	Metamodelo da OCL (adaptado)
AtLeastNQuantification	OperatorCallExp (“>=”)
AtMostNQuantification	OperatorCallExp (“<=”)
AtomicFormulation	OperatorCallExp
AtomicFormulation	PropertyCallExp
Concept	Class
Concept	Variable
Conjunction	OperatorCallExp (“and”)
Disjunction	OperatorCallExp (“or”)
ExactlyNQuantification	OperatorCallExp (“=”)
ExistentialQuantification	CollectionOperationCallExp (“notEmpty”)
Implication	OperatorCallExp (“implies”)
IndividualConcept	StringLiteralExp
InstantiationFormulation	OperationCallExp (“oclIsKindOf”)
Integer	IntegerLiteralExp
NecessityFormulation	Invariant
Rule	OclModule
UniversalQuantification	IterarorExp (“forAll”)
Variable	Iterator
Variable	PropertyCallExp

A decisão de construir um programa único permitiu reusar diversas regras de mapeamento que são comuns a categorias de regras diferentes. Contudo, isso acarretou a necessidade de maior uso de regras do tipo *lazy*, uma vez que elementos de origem iguais podem ser transformados em elementos de destino diferentes nos diversos mapeamentos. Em um total de 24 regras, 11 são do tipo *lazy*, conforme mostrado no Quadro 14.

Quadro 14 – Resumo dos *helpers* e regras da transformação ATL.

Transformação	Matched rules	Lazy rules	Unique lazy rules	Helpers
SBVR2OCL.atl	9	11	4	19

O metamodelo da SBVR usado precisou ser alterado para facilitar a implementação da transformação em um programa único. A principal mudança realizada consistiu na criação de uma enumeração, chamada *RuleCategory*, contendo as categorias que classificam as regras de negócio tratadas pela abordagem. Assim, cada elemento do tipo *Rule* passa a se relacionar com um elemento do tipo *Category*, ou seja, a regra de negócio modelada passa a ter a categoria ao qual ela pertence, permitindo que durante a execução do programa seja possível identificar o tipo da regra de negócio que está sendo transformada.

Além disso, duas outras alterações foram realizadas. Uma classe *Package* foi criada para agrupar elementos do tipo *Thing*, ou seja, qualquer elemento do metamodelo. Ao modelar uma regra, a mesma deve ser criada dentro de um *Package*. Todos os demais elementos necessários para representar a regra também são armazenados dentro do *Package*, visando facilitar a instanciação e reuso dos objetos durante a criação dos modelos de entrada.

Já a associação *roleBinding* existente entre os elementos *AtomicFormulation* e *RoleBinding* foi transformada em duas associações nomeadas como *firstRoleBinding* e *secondRoleBinding*. A primeira associação passou a corresponder ao primeiro *role binding* da formulação atômica, enquanto a segunda associação passou a corresponder ao segundo *role binding*. A motivação para essa separação foi a necessidade constante de se identificar a ordem dos *role bindings* das formulações atômicas na implementação.

O cenário da transformação entre um modelo na sintaxe abstrata da SBVR em um modelo na sintaxe abstrata da OCL é mostrado na Figura 38. Em linhas gerais, o módulo implementado, armazenado no arquivo “SBVR2OCl.atl”, transforma formulações lógicas em SBVR em invariantes OCL, baseados em mapeamentos entre os elementos dos dois metamodelos. Pode-se dizer que a transformação é definida no nível de metamodelos e executada no nível de modelos.

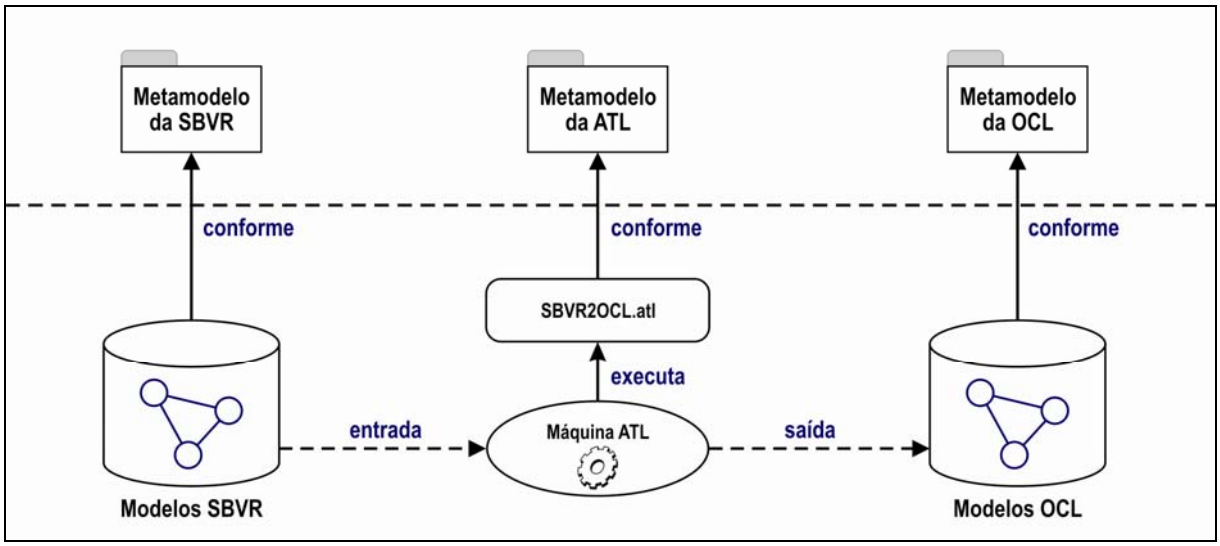

Figura 38 – O cenário da transformação entre os modelos SBVR e os modelos OCL.

As chamadas às regras ATL, durante a execução do módulo, dependem do tipo de regra de negócio que está sendo transformada. Contudo, há duas regras do tipo *matched* que são sempre executadas, independente do tipo da regra de negócio a ser transformada. Elas implementam o conjunto de mapeamentos comuns mostrado no início deste capítulo.

A primeira regra é a regra *Rule2OclModule* (Figura 39) que transforma um elemento do tipo *Rule* em um elemento *OCLModule* e a segunda é a regra *NecessityFormulation2Invariant* que transforma um elemento do tipo *NecessityFormulation* nos elementos *Invariant* e *OclContextDefinition*. Já o elemento *Class*, que representa o contexto da invariante, é criado nessa mesma regra, a partir de um elemento do tipo *Concept*.

```
-- Rule 'Rule2OclModule'
-- Create OclModule from SBVR Rule

rule Rule2OclModule {
       from
              i: SBVR!Rule (
                     i.oclIsTypeOf(SBVR!Rule)
                 )
       to
              o: OCL!OclModule (
                     ownedElements <-    i.closedLogicalFormulation->asSequence()->
                                         first().isThing
                 )
        }
```

Figura 39 – Regra ATL que transforma um elemento *Rule* em um elemento *OCLModule*.

Na Figura 40, é mostrado um exemplo de regra ATL do tipo *unique lazy rule*, que transforma um elemento *IndividualConcept* do metamodelo da SBVR em um elemento *StringLiteralExp* do metamodelo da SBVR.

```
-- Unique lazy rule 'IndividualConcept2StringLiteralExp'
-- Create OCL StringLiteralExp from IndividualConcept element

lazy rule IndividualConcept2StringLiteralExp {
    from
           i: SBVR!IndividualConcept

    to o: OCL!StringLiteralExp (
                    name <- i.designation -> first().signifier.value,
                    stringSymbol <- i.designation -> first().signifier.value
        )
    }
```

Figura 40 – Regra ATL que transforma um elemento *IndividualConcept* em um elemento *StringLiteralExp*.

4.5.2.3 Transformação de modelo para texto

A transformação modelo-modelo descrita é apenas o primeiro passo para se obter expressões OCL a partir de regras de negócio e tem como resultado um modelo OCL em conformidade com o metamodelo da OCL. Porém, é necessário transformar esse modelo em uma representação textual que possa ser entendida por pessoas e usada em ferramentas de validação.

Para realizar essa tarefa é preciso definir uma sintaxe concreta para OCL, utilizando o ambiente da *Textual Concrete Sintaxe* (TCS), integrado ao ADT. Inicialmente, o metamodelo da OCL é especificado usando a linguagem *Kernel Meta Meta Model*[30] (KM3) e a sintaxe concreta da OCL é expressa em TCS (denotada como “modelo OCL_TCS” na Figura 41). A ponte entre a representação abstrata e a representação textual da OCL é feita através de um extrator EBNF, implementado como uma classe Java no próprio ambiente TCS.

[30] KM3 – linguagem para definir metamodelos (como MOF e Ecore) com sintaxe semelhante à linguagem Java.

No momento da execução do extrator EBNF, ele utiliza como entrada o modelo a ser transformado, o metamodelo da OCL em KM3 e a descrição da sintaxe concreta textual da OCL em TCS, a fim de transformar o modelo abstrato em um texto, representando a sintaxe concreta OCL da regra transformada (“Código OCL”), conforme mostrado na Figura 41.

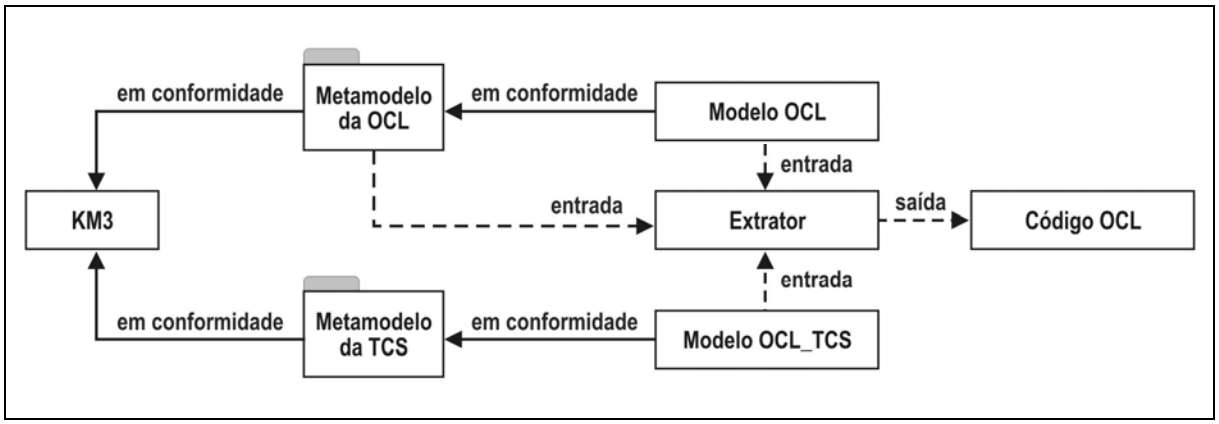

Figura 41 – Uso da TCS para extração da sintaxe concreta (adaptado de (MILANOVIC, 2007)).

Conforme visto anteriormente, o ambiente da TCS é integrado à IDE de desenvolvimento ATL. Para executar a transformação de modelo para texto mostrada, foi necessário utilizar um conjunto de arquivos disponível no cenário “R2ML to OCL[31]” do repositório de transformações ATL da comunidade Eclipse (ATL *Zoo*). Desse cenário, foram aproveitadas as implementações do metamodelo da OCL (incluindo o pacote *EnhancedOCL*) em Ecore e KM3, a especificação em TCS da sintaxe concreta da OCL e a implementação em Ecore e XMI do metamodelo da TCS. Esses 5 arquivos foram incorporados à solução desenvolvida e tornaram possível executar a transformação dos modelos de objetos OCL em expressões OCL na sintaxe concreta textual.

[31] Cenário “R2ML to OCL”: http://www.eclipse.org/m2m/atl/atlTransformations/#R2ML2OCL.

4.5.3 Descrição do Uso da Solução

Esta seção apresenta as principais etapas da execução da solução de transformação desenvolvida. Todas essas etapas são realizadas no próprio ambiente Eclipse, uma vez que a solução foi criada como um projeto ATL nesse ambiente. A estrutura de pastas do projeto da solução é mostrada no Quadro 15.

Quadro 15 – Estrutura de pastas da solução.

Pasta	Descrição
SBVR2OCL	Pasta raiz
SBVR2OCL\metamodels	Metamodelos
SBVR2OCL\metamodels\OCL	
SBVR2OCL\metamodels\SBVR	
SBVR2OCL\metamodels\TCS	
SBVR2OCL\models	Arquivos de entrada e saída
SBVR2OCL\models\templates	Templates das categorias de regras
SBVR2OCL\transformations	Transformação
SBVR2OCL\tcs	Arquivo TCS
SBVR2OCL\examples	Arquivos do exemplo de uso da abordagem

O processo de execução da solução é ilustrado na Figura 42, na qual são mostradas as duas principais etapas do mesmo: “Modelar regra de negócio” e a execução do arquivo de configuração “SBVR2OCL-All.xml”. No apêndice B, é disponibilizado o manual de uso da solução.

A primeira etapa (“Modelar regra de negócio”) corresponde à criação de um modelo de objetos representando a formulação lógica da regra de negócio a ser transformada na sintaxe abstrata da SBVR (ou seja, no metamodelo da SBVR). Esse modelo pode ser feito utilizando o editor gráfico gerado pelo EMF ou através da própria solução, desde que se utilize um dos *templates* disponíveis. Como a solução desenvolvida não faz verificação quanto à correção do modelo de entrada, o uso dos *templates* visa minimizar problemas na

execução da transformação, oriundos de modelos mal formados, além de auxiliar no processo de modelagem da regra de negócio. O modelo de objetos criado, salvo no formato Ecore XMI, deve ser nomeado como “sample-sbvr.ecore” e armazenado dentro da pasta “models” da solução.

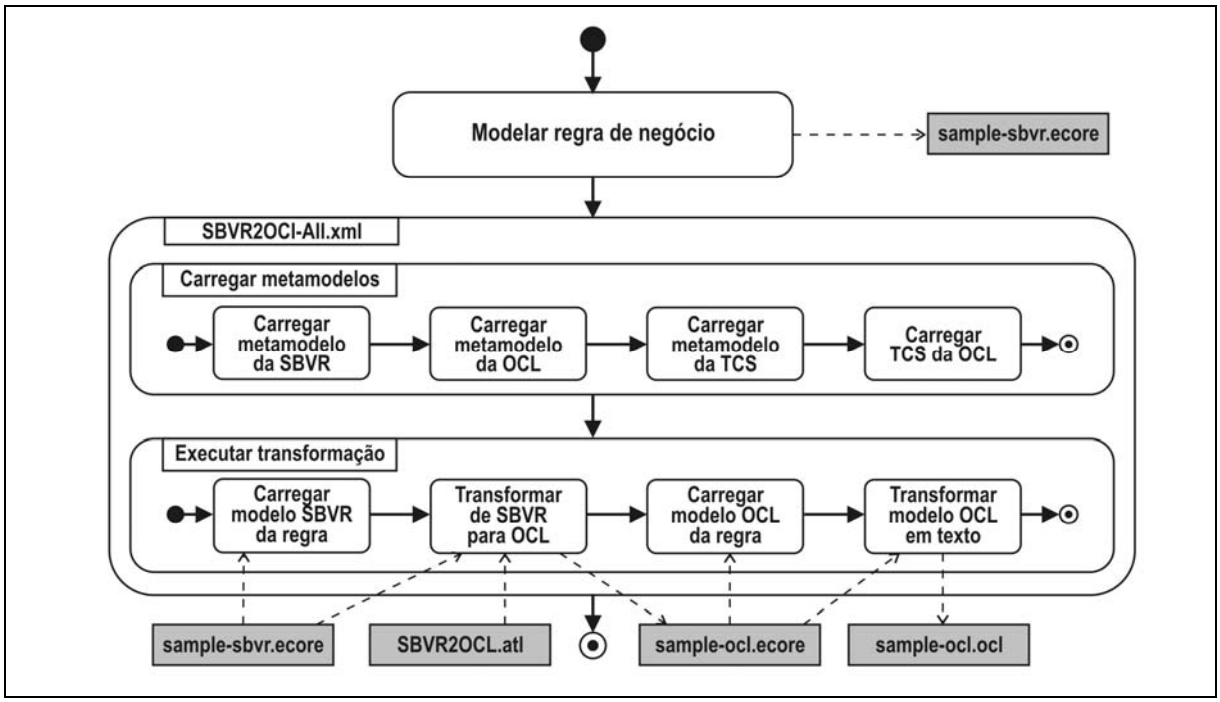

Figura 42 – Diagrama do processo de transformação da solução.

A segunda etapa compreende a execução do arquivo “SBVR2OCL-All.xml”. Esse é um arquivo de configuração *Ant* construído com o objetivo de automatizar os demais passos necessários para realizar a transformação do modelo de entrada.

Primeiramente, na etapa “Carregar metamodelos”, os metamodelos necessários para a transformação (SBVR, OCL e TCS) são automaticamente carregados nos seus respectivos *model handlers*: EMF ou MDR (*Metadata Repository*). Além dos 3 metamodelos, essa etapa executa também a carga do arquivo TCS contendo a sintaxe concreta do metamodelo da OCL. Com todos os arquivos devidamente carregados, é possível iniciar a transformação do modelo de entrada.

Na etapa “Executar transformação”, é feito o carregamento do modelo SBVR da regra de negócio (arquivo “sample-sbvr.ecore”) no *model handler*. Em seguida, o módulo ATL (“SBVR2OCL.atl”) é executado (“Transformar de SBVR para OCL”), transformando o modelo de entrada em um modelo de saída que é armazenado no arquivo intermediário “sample-ocl.ecore”, no formato Ecore XMI. Esse arquivo contém o modelo OCL da regra de negócio após a execução das regras de transformação do módulo ATL, sendo utilizado como entrada da etapa “Carregar modelo OCL da regra”. Nessa etapa, o arquivo “sample-ocl.ecore” é carregado no *model handler* e fica disponível para a transformação de modelo para texto com o uso da TCS.

Na última etapa da execução da solução, “Transformar modelo OCL em texto”, a expressão OCL representada na sintaxe abstrata é transformada em um invariante OCL na sintaxe concreta da linguagem. Nessa etapa, um arquivo texto de saída é gerado e armazenado na pasta “models” com o nome “sample-ocl.ocl”, finalizando o processo de transformação.

5 EXEMPLO DE UTILIZAÇÃO DA ABORDAGEM

No capítulo anterior, foi apresentada uma abordagem para a transformação das regras de negócio na Arquitetura Dirigida por Modelos. Para mostrar uma aplicação da abordagem proposta, será apresentado neste capítulo um exemplo de utilização, baseado em uma adaptação do estudo de caso *EU-Rent Example*, presente no anexo E da especificação da SBVR (OMG, 2008). Além disso, será mostrada a metodologia utilizada na aplicação da abordagem de transformação de regras de negócio, os resultados obtidos com o exemplo e a forma de avaliação dos mesmos, além da discussão sobre a aplicação da abordagem e os resultados alcançados.

5.1 Metodologia Utilizada

A especificação da SBVR apresenta um estudo de caso com o intuito de mostrar a aplicação prática do padrão. O estudo de caso modela o negócio (fictício) de uma empresa de aluguel de carros, chamada EU-Rent. O exemplo apresentado na documentação, chamado *EU-Rent Example*, possui uma breve descrição em linguagem natural do negócio da EU-Rent. Além disso, apresenta dois vocabulários SBVR. O primeiro deles, chamado *The EU-Rent Vocabulary Business Context*, descreve o contexto do negócio da empresa, envolvendo as comunidades de interesse, os vocabulários utilizados e os conjuntos de significados compartilhados.

O segundo vocabulário, chamado *EU-Rent English Vocabulary*, descreve a terminologia e as regras de negócio que compõem o negócio do exemplo de maneira detalhada. Esse vocabulário apresenta um grande número de conceitos e fatos, especificados textualmente (conforme definido na SBVR) e representados graficamente com o uso de diagramas UML.

O vocabulário SBVR *EU-Rent English Vocabulary* serviu de base para a construção do exemplo de utilização aqui apresentado. A descrição adaptada do negócio da EU-Rent encontra-se disponível no apêndice C. A partir dessa adaptação e da análise dos diagramas UML e dos verbetes do vocabulário do estudo de caso da especificação, foi possível construir um diagrama de classes para o exemplo proposto, representando seu modelo conceitual. Conforme visto, a abordagem de transformação pressupõe a existência de um modelo de classes preliminar, cuja forma de construção não está no escopo da proposta desta dissertação. No exemplo de utilização da abordagem, esse modelo será usado para facilitar o entendimento das expressões OCL resultantes da transformação e para a validação da sintaxe das mesmas com o uso de uma ferramenta. O diagrama de classes construído para o exemplo também se encontra disponível no apêndice C.

Além do vocabulário de negócio, o estudo de caso do anexo E da SBVR apresenta um conjunto de elementos de orientação (regras de negócio e recomendações). Esse conjunto foi modelado usando como base os conceitos e fatos descritos no *EU-Rent English Vocabulary* e é composto por 29 regras de negócio (estruturais e operativas) e 5 recomendações (permissões e possibilidades).

Inicialmente, as regras de negócio do conjunto foram remodeladas como necessidades. Além disso, algumas regras de negócio precisaram ser reescritas ou divididas em duas, devido às simplificações realizadas no estudo de caso do anexo E da especificação da SBVR. Nessa etapa, as regras de negócio que apresentavam eventos e aquelas não contempladas pelas categorias da abordagem foram retiradas do grupo. Assim, do conjunto original de 29 regras de negócio, foi possível aplicar a abordagem em 19 delas.

As recomendações (permissões e possibilidades) não foram incluídas no exemplo apresentado, uma vez que a transformação delas em expressões OCL não representaria efetivamente restrições no modelo de classes. Um exemplo é a recomendação que diz que é

permitido que a agência de retirada de um aluguel não seja a agência de retorno do aluguel. Nesse caso, não há uma restrição para as agências que podem participar da associação, mas apenas uma informação de que elas podem ser diferentes.

Para complementar o conjunto das regras de negócio usadas no exemplo, foram incluídas outras regras presentes na documentação da SBVR e que não faziam parte do conjunto de regras de negócio do estudo de caso da especificação. Assim como as demais regras de negócio, elas foram remodeladas como necessidades e algumas delas precisaram ser reescritas.

No conjunto inicial de 19 regras tratáveis pela abordagem, foram incorporadas mais 18, totalizando, assim, 37 regras de negócio. Devido às simplificações realizadas no estudo de caso original, duas regras de negócio do conjunto original foram divididas. Assim, no total, 39 regras de negócio foram utilizadas no exemplo de aplicação da abordagem deste trabalho.

Após a definição e adaptação das regras de negócio a serem usadas no exemplo, foi necessário classificá-las seguindo a categorização proposta. A classificação correta da regra de negócio é fundamental para que os mapeamentos adequados sejam aplicados durante o processo de transformação, uma vez que eles são definidos de acordo com a categoria da regra em questão. No Quadro 16, pode-se ver a distribuição das regras de negócio entre as categorias existentes na abordagem. Já o apêndice D apresenta as regras de negócio (expressas em SBVR *Structured English*) utilizadas no exemplo e devidamente classificadas.

Quadro 16 – Classificação das regras de negócio do exemplo.

Categoria	Quantidade
Restrição simples	22
Restrição de cardinalidade	5
Restrição de tipo	1
Restrição simples sobre elementos de uma associação	2
Derivação simples	2
Derivação com cardinalidade mínima	6
Derivação com restrição simples sobre elementos de uma associação	1

Para cada uma das categorias propostas na abordagem foram criados *templates* (no formato Ecore XMI) para facilitar o trabalho de especificação das regras de negócio. Assim, com a utilização dos *templates*, foi possível construir os modelos de objetos (em conformidade com o metamodelo da SBVR) que representam as formulações lógicas em SBVR das regras de negócio do exemplo. A transformação das regras de negócio em SBVR *Structured English* para formulações lógicas em SBVR foi realizada manualmente e não foi tratada pela abordagem proposta.

Os modelos de objetos criados foram salvos no formato Ecore XMI, para posteriormente serem utilizados como entrada para o programa de transformação ATL. No apêndice B, há uma seção explicando como é feita a construção dos modelos de entrada usando a solução desenvolvida. Nesse ponto, a etapa inicial do processo de utilização da solução (“Modelar regra de negócio”) encontra-se concluída.

A execução do módulo ATL foi realizada individualmente para cada um dos modelos de objetos de entrada, gerando 39 arquivos textos contendo as expressões OCL resultantes da aplicação da transformação sobre as formulações lógicas das regras de negócio em SBVR. Como resultado intermediário obtido, 39 arquivos (no formato Ecore XMI) contendo os modelos de objetos OCL (em conformidade com o metamodelo da OCL) das regras de negócio foram gerados.

Para complementar o exemplo de aplicação da abordagem, foi realizada uma avaliação da correção da sintaxe das expressões OCL geradas pela solução. O objetivo dessa avaliação foi saber se as expressões OCL estavam de acordo com a sintaxe concreta da linguagem OCL e, além disso, se a sintaxe das expressões era válida, considerando-se o diagrama de classes do exemplo utilizado (mostrado no apêndice C).

A avaliação foi realizada na ferramenta de modelagem *Papyrus* UML2 *Modeler*[32]. Baseada na plataforma Eclipse, ela permite a modelagem dos diversos diagramas estabelecidos pela UML, além de contar com um construtor e avaliador de expressões OCL. Inicialmente, o diagrama de classes do exemplo foi construído na ferramenta *Papyrus*. Em seguida, as 39 invariantes foram incorporadas na ferramenta como restrições no modelo de classes. Para isso, a classe de contexto da invariante era selecionada no modelo e uma nova restrição era criada no *Papyrus*. Por fim, cada uma das invariantes especificadas teve sua sintaxe avaliada na ferramenta de modelagem.

A Figura 43 apresenta o módulo de construção e avaliação de restrições do *Papyrus*, destacando algumas restrições da classe *Rental* (mostradas no painel esquerdo) e a avaliação de uma das restrições (no painel direito). Pode-se notar que não há indicação da classe de contexto ou do nome da invariante na expressão avaliada. A identificação do contexto da restrição é feita através da classe selecionada no modelo de classes ou através do cabeçalho do módulo de construção e avaliação de restrições da ferramenta. Já o nome da restrição é exibido no painel esquerdo do módulo, conforme mostrado na figura.

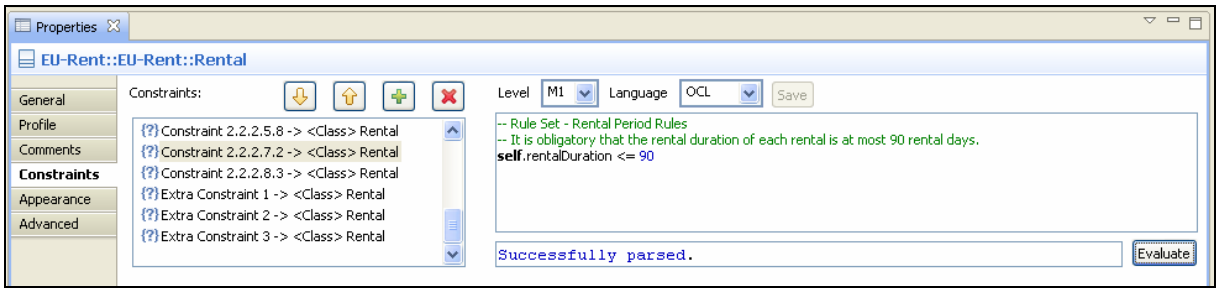

Figura 43 – Módulo de restrições na ferramenta *Papyrus*.

[32] *Papyrus* UML2 *Modeler* – http://www.papyrusuml.org/.

5.2 Resultados Obtidos

Após a etapa de classificação e modelagem das regras de negócio, foi possível utilizar a solução que implementa a abordagem de transformação. Como há um programa único para executar todos os mapeamentos das categorias de regras de negócio, é preciso que a classificação das mesmas seja representada no modelo de objetos da regra em SBVR. Por essa razão, cada regra de negócio possui uma propriedade chamada *Category*, que indica qual é a sua categoria.

Nas próximas seções, serão descritos os passos que foram necessários para a aplicação da transformação nas regras de negócio do exemplo, além dos resultados obtidos e a avaliação realizada.

5.2.1 Aplicação da Transformação

A solução de transformação implementada para a abordagem foi aplicada individualmente para cada uma das 39 regras de negócio do exemplo. Para isso, foi necessário executar o arquivo de configuração *Ant* "SBVR2OCL-All.xml", que automatiza todos os passos necessários para realizar a transformação dos modelos de objetos de entrada. Os resultados obtidos com a execução da solução são um arquivo Ecore XMI contendo o modelo de objetos de saída da regra de negócio (conforme o metamodelo da OCL) e um arquivo texto com a expressão OCL resultante da transformação, conforme mostrado no apêndice B. As regras de negócio em OCL geradas no exemplo aplicam-se ao modelo de classes apresentado no apêndice C.

Para a classificação das regras de negócio proposta na abordagem, serão apresentados, a seguir, por categoria, um exemplo detalhado e os resultados alcançados com a aplicação da transformação.

5.2.1.1 Restrição simples

A categoria “restrição simples” apresenta o maior número de regras de negócio no exemplo, totalizando 22. Desse total, 6 regras compreendem a comparação de uma propriedade com um valor e 16 regras compreendem a comparação de uma propriedade com outra propriedade.

Dentre as regras de negócio de restrição simples por valor, será mostrada aqui a aplicação da abordagem sobre a regra “*It is necessary that the age of the driver is at least 21 years*”, cujo modelo de objetos em EMF é mostrado na Figura 44. A representação gráfica em árvore do modelo de objetos em Ecore XMI é disponibilizada pelo próprio EMF. Na figura, pode-se notar a presença do elemento *Package*, utilizado para agrupar os demais elementos SBVR da regra de negócio.

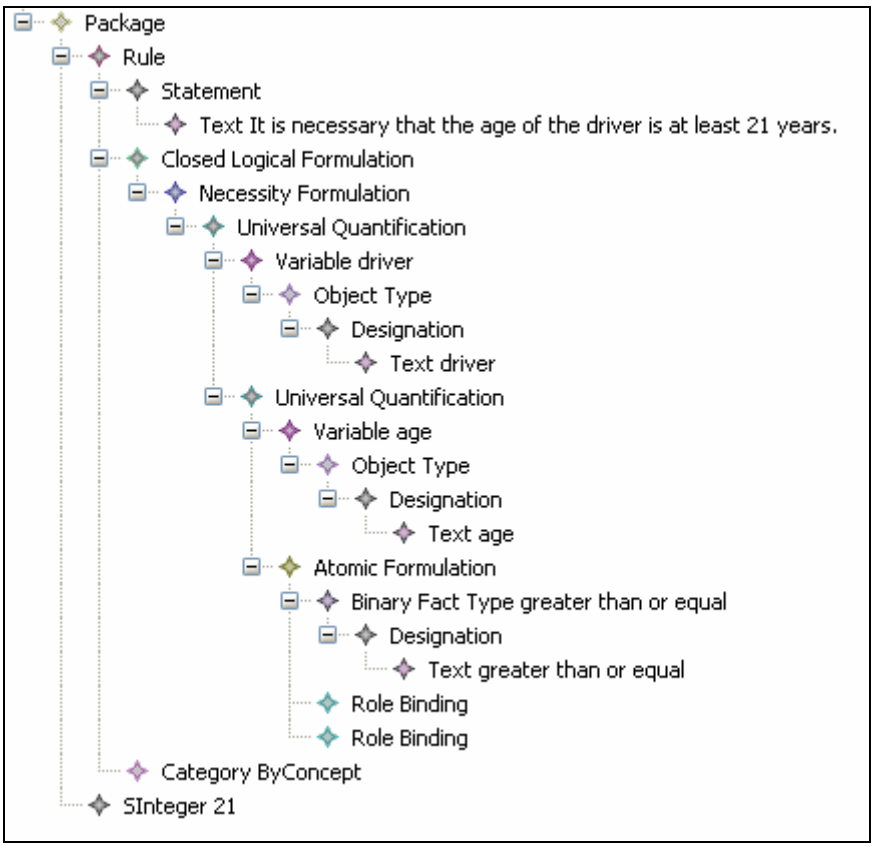

Figura 44 – Um modelo de objetos para a restrição simples.

A expressão OCL resultante da transformação do modelo de objetos do exemplo (Figura 44) é mostrada na Figura 45. Essa expressão é obtida a partir da execução da solução de automação implementada, conforme descrito no capítulo anterior.

```
context Driver inv invName:
self.age >= 21
```

Figura 45 – Expressão OCL resultante da transformação.

Os resultados alcançados para todas as regras de negócio da categoria "restrição simples" são mostrados no Quadro 17, que apresenta as regras de negócio expressas em SBVR *Structured English* e as representações correspondentes como invariantes em OCL. As regras de negócio em OCL mostradas aplicam-se ao modelo de classes do apêndice C.

Quadro 17 – Resultados da transformação para a restrição simples.

Regra em SBVR	Regra em OCL
It is necessary that the pick-up branch of a round trip rental is the return branch of the round trip rental.	`context RoundTripRental inv invName: self.pickUpBranch = self.returnBranch`
It is necessary that the local area of the pick-up branch of a local one way rental is the local area of the return branch of the local one way rental.	`context LocalOneWayRental inv invName: self.pickUpBranch.localArea = self.returnBranch.localArea`
It is necessary that the local area of the pick-up branch of an in country one way rental is different from the local area of the return branch of the in country one way rental.	`context InCountryOneWayRental inv invName: self.pickUpBranch.localArea <> self.returnBranch.localArea`
It is necessary that the operating country of the pick-up branch of an in country rental is the operating country of the return branch of the in country rental.	`context InCountryRental inv invName: self.pickUpBranch.operatingCountry = self.returnBranch.operatingCountry`
It is necessary that the operating country of the pick-up branch of an international rental is not the operating country of the return branch of the international rental.	`context InternationalRental inv invName: self.pickUpBranch.operatingCountry <> self.returnBranch.operatingCountry`

It is necessary that the operating country of the transfer pick-up branch of an international return is not the operating country of the transfer drop-off branch of the international return.	`context InternationalReturn inv invName: self.transferPickupBranch.operatingCountry <> self.transferDropoffBranch.operatingCountry`
It is necessary that the local area of the transfer pick-up branch of a local car transfer is the local area of the transfer drop-off branch of the local car transfer.	`context LocalCarTransfer inv invName: self.transferPickupBranch.localArea = self.transferDropoffBranch.localArea`
It is necessary that the operating country of the transfer pick-up branch of an in-country car transfer is the operating country of the transfer drop-off branch of the in-country car transfer.	`context InCountryCarTransfer inv invName: self.transferPickupBranch.operatingCountry = self.transferDropoffBranch.operatingCountry`
It is necessary that the operating country of the pick-up branch of each international outward rental is the country of registration of the rental car of the rental.	`context InternationalOutwardRental inv invName: self.pickUpBranch.operatingCountry = self.rentalCar.countryOfRegistration`
It is necessary that the scheduled pick-up date time of each advance rental is after the booking date time of the advance rental.	`context AdvanceRental inv invName: self.scheduledPickUpDateTime.isAfter(self.bookingDateTime)`
It is necessary that the operating country of the return branch of each international inward rental is the country of registration of the rental car of the rental.	`context InternationalInwardRental inv invName: self.returnBranch.operatingCountry =self.rentalCar.countryOfRegistration`
It is necessary that the operating country of the transfer drop-off branch of an international return is the country of registration of the transferred car of the international return.	`context InternationalReturn inv invName: self.transferDropoffBranch.operatingCountry = self.transferredCar.countryOfRegistration`
It is necessary that the booking date time of a points rental is before the scheduled pick-up date time of the rental.	`context PointsRental inv invName: self.bookingDateTime.isBefore(self.scheduledPickUpDateTime)`
It is necessary that the business currency of the actual rental charge of each rental is equal to the business currency of the pick-up branch of the rental.	`context Rental nv invName: self.actualRentalCharge.businessCurrency = self.pickUpBranch.businessCurrency`
It is necessary that the service reading of each rental car is at most 5500 miles.	`context RentalCar inv invName: self.serviceReading <= 5500`
It is necessary that the rental duration of each rental is at most 90 rental days.	`context Rental inv invName: self.rentalDuration <= 90`

It is necessary that the age of the driver is at least 21 years.	`context Driver inv invName: self.age >= 21`
It is necessary that the transfer pick-up branch of a car transfer is not the transfer drop-off branch of the car transfer.	`context CarTransfer inv invName: self.transferPickupBranch <> self.transferDropoffBranch`
It is necessary that the business currency of the estimated rental charge of each rental is equal to the business currency of the pick-up branch of the rental.	`context Rental inv invName: self.estimatedRentalCharge.businessCurrency = self.pickUpBranch.businessCurrency`
It is necessary that the name of the fuel type is Gasoline, LPG or Eletricity[33].	`context FuelType inv invName: self.name = "Gasoline" or self.name = "LPG" or self.name = "Eletricity"`
It is necessary that the name of the currency is Euro, GBP or USD.	`context Currency inv invName: self.name = "Euro" or self.name = "GBP" or self.name = "USD"`
It is necessary that the fuel level of a rental car is full, 1/2 or empty.	`context RentalCar inv invName: self.fuelLevel = "full" or self.fuelLevel = "1/2" or self.fuelLevel = "empty"`

5.2.1.2 Restrição de cardinalidade

No exemplo proposto, 5 regras de negócio são classificadas como restrições de cardinalidade. Como exemplo da aplicação da abordagem nessa categoria, a regra "*It is necessary that each rental has at most four drivers*" será utilizada. A modelagem dessa regra de negócio, usando o metamodelo da SBVR, resultou no modelo de objetos mostrado na Figura 46. Nela, pode-se ver o elemento *Category*, com valor *Cardinality*, indicando que a regra de negócio modelada é uma restrição de cardinalidade.

[33] Uma alternativa para a representação de regras de negócio que definem um conjunto limitado de valores válidos para um atributo é a especificação de uma enumeração em UML contendo esses valores. O mesmo se aplica para as duas regras seguintes do quadro.

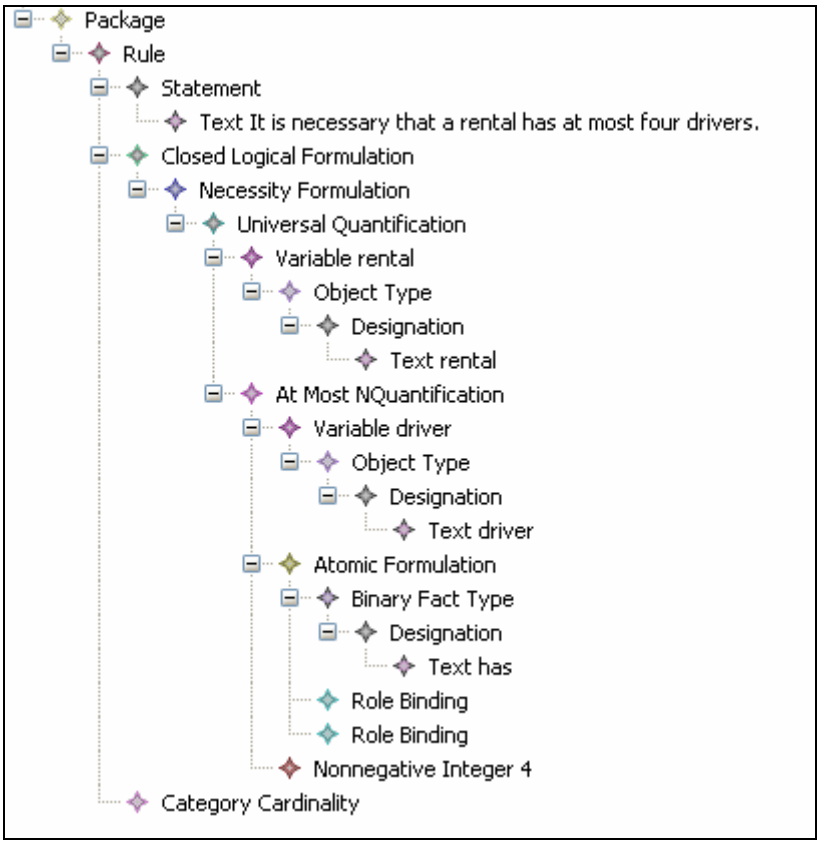

Figura 46 – Um modelo de objetos para a restrição de cardinalidade.

Após a execução do arquivo de configuração que automatiza a transformação, obteve-se como resultado a expressão OCL mostrada na Figura 47.

```
context Rental inv invName:
self.driver -> size() <= 4
```

Figura 47 – Expressão OCL resultante da transformação.

É importante destacar que a forma usual de representar restrições de cardinalidade em modelos de classes é através da utilização de multiplicidades nas associações entre as classes. Normalmente, a representação de restrições de cardinalidade em OCL (com o uso da operação *size*) é feita apenas em casos nos quais a multiplicidade irá variar sob certas condições. Nesse caso, a restrição é especificada como o conseqüente de uma regra de derivação em uma operação *implies*.

A aplicação da abordagem sobre as regras de negócio da categoria “restrição de cardinalidade” resultou no conjunto de expressões OCL mostrado no Quadro 18.

Quadro 18 – Resultados da transformação para a restrição de cardinalidade.

Regra em SBVR	Regra em OCL
It is necessary that each rental has at most four drivers.	`context Rental inv invName: self.driver -> size() <= 4`
It is necessary that each car model has at least one fuel type.	`context RentalCar inv invName: self.fuelType -> size() >= 1`
It is necessary that each rental has exactly one requested car group.	`context Rental inv invName: self.requestedCarGroup ->size() = 1`
It is necessary that each rental has exactly one return branch.	`context Rental inv invName: self.returnBranch -> size() = 1`
It is necessary that each rental car is owned by exactly one store branch	`context RentalCar inv invName: self.storeBranch -> size() = 1`

5.2.1.3 Restrição de tipo

Dentre as regras de negócio consideradas no exemplo, apenas uma delas foi classificada na categoria “restrição de tipo”. A expressão representando essa regra de negócio na linguagem OCL é obtida após a execução da transformação sobre o seu modelo de objetos. A expressão OCL resultante utiliza a operação *oclIsKindOf*, conforme mostrado no Quadro 19.

Quadro 19 – Resultado da transformação para a restrição de tipo.

Regra em SBVR	Regra em OCL
It is necessary that the renter of each points rental is a club member.	`context PointsRental inv invName: self.renter.oclIsKindOf(clubMember)`

O modelo de objetos para a regra de negócio “*It is necessary that the renter of each points rental is a club member*”, classificada nessa categoria, é mostrado na Figura 48.

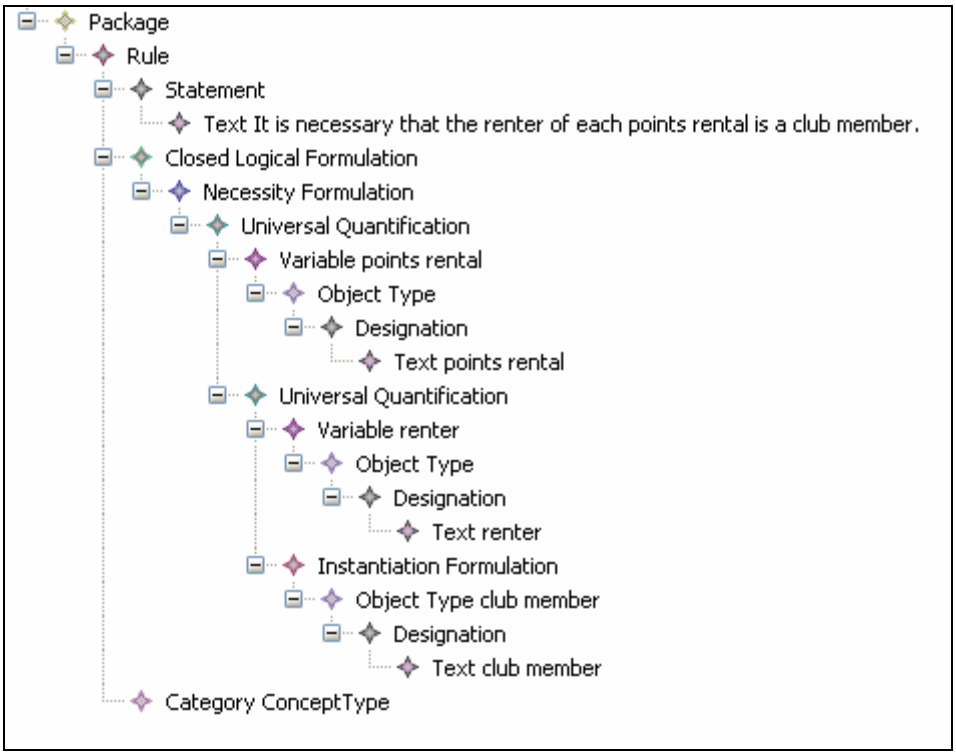

Figura 48 – Um modelo de objetos para a restrição de tipo.

5.2.1.4 Restrição simples sobre elementos de uma associação

Duas regras de negócio do exemplo foram classificadas na categoria "restrição simples sobre elementos de uma associação". A aplicação da abordagem sobre as regras dessa categoria resultou nas invariantes OCL apresentadas no Quadro 20.

Quadro 20 – Resultados da transformação para a restrição simples sobre elementos de uma associação.

Regra em SBVR	Regra em OCL
It is necessary that the license expiry date of each driver of a rental is after the scheduled return date time of the rental.	`context Rental inv invName: self.driver -> forAll (x | x.licenseExpiryDate.isAfter(self.scheduledReturnDateTime))`
It is necessary that each driver of a rental is qualified.	`context Rental inv invName: self.driver -> forAll (x |x.isQualified)`

Como exemplo, o modelo de objetos para a regra de negócio "*It is necessary that each driver of a rental is qualified*" é mostrado na Figura 49.

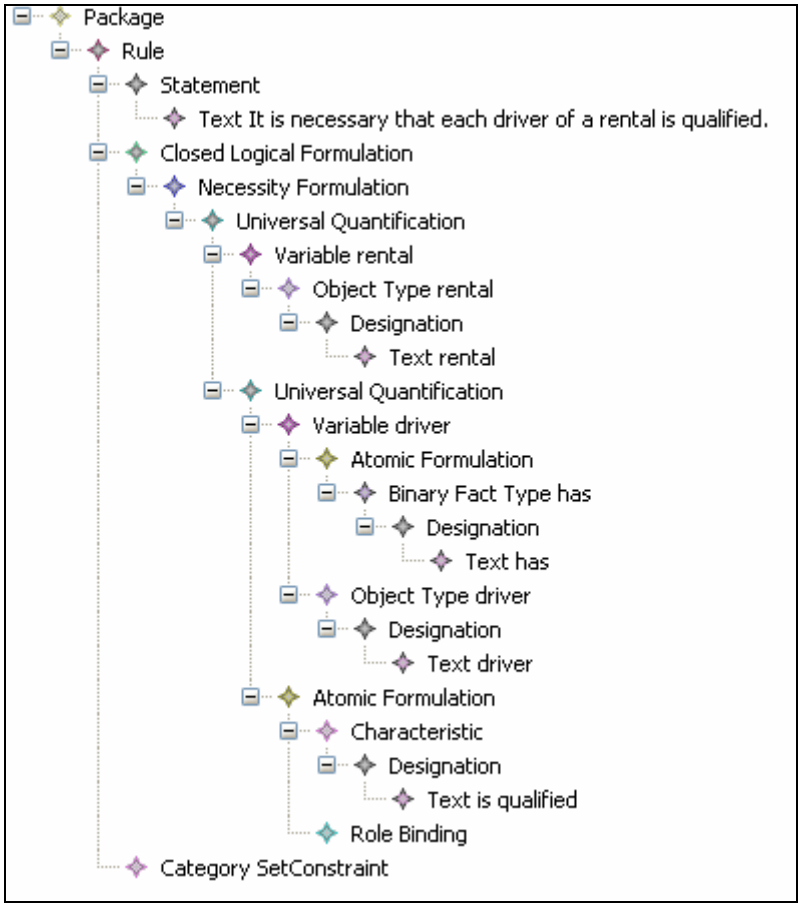

Figura 49 – Um modelo de objetos para a restrição simples sobre elementos de uma associação.

Após a transformação, a expressão OCL obtida com a aplicação dos mapeamentos definidos para a categoria na regra de exemplo, é mostrada na Figura 50.

```
context Rental inv invName:
self.driver -> forAll (x |x.isQualified)
```

Figura 50 – Expressão OCL resultante da transformação.

5.2.1.5 Derivação simples

Duas regras de negócio do conjunto foram classificadas como derivações simples. Para exemplificar a aplicação da abordagem nessa categoria, será apresentado, na Figura 51, o modelo de objetos para a regra "*It is necessary that if an advance rental is assigned then the store branch of the rental car of the advance rental is the pick-up branch of the advance rental*".

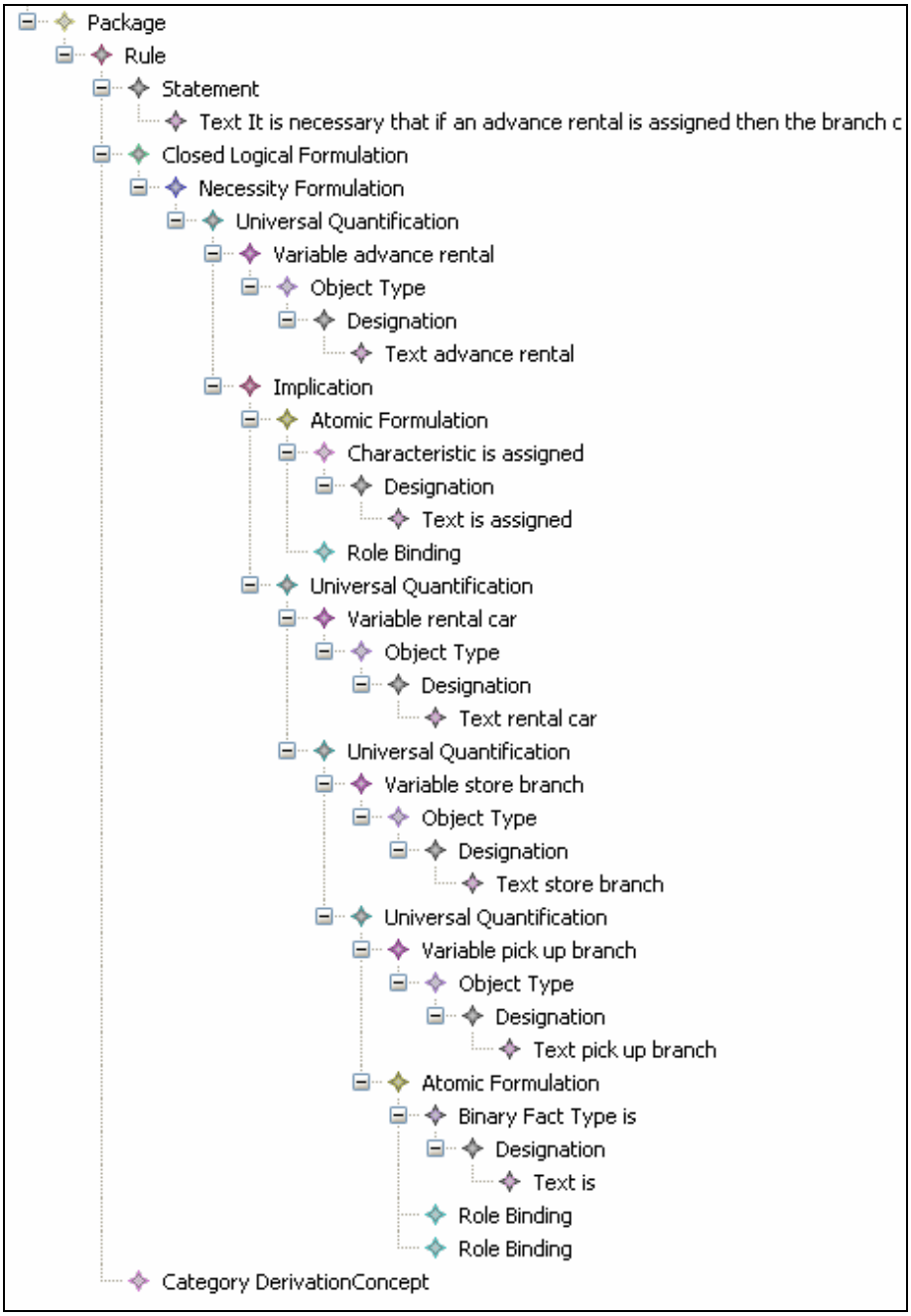

Figura 51 – Um modelo de objetos para a derivação simples.

Na abordagem, a transformação da regra de derivação resulta numa invariante que contém a operação *implies*. A utilização dessa operação é mostrada na Figura 52, que apresenta a expressão OCL resultante da transformação da regra de negócio do exemplo.

```
context AdvanceRental inv invName:
self.isAssigned implies
self.rentalCar.storeBranch = self.pickUpBranch
```

Figura 52 – Expressão OCL resultante da transformação.

A aplicação da abordagem sobre as regras de negócio da categoria "derivação simples" resultou nas expressões OCL apresentadas no Quadro 21.

Quadro 21 – Resultados da transformação para a derivação simples.

Regra em SBVR	Regra em OCL

It is necessary that if an advance rental is assigned then the store branch of the rental car of the advance rental is the pick-up branch of the advance rental.	`context AdvanceRental inv invName:`
	`self.isAssigned implies`
	`self.rentalCar.storeBranch =`
	`self.pickUpBranch`

It is necessary that if the operating country of the pick-up branch of a rental is not the country of registration of the rental car of the rental then the operating country of the return branch of the rental is the country of registration of the rental car.	`context Rental inv invName:`
	`self.pickUpBranch.operatingCountry`
	`<>`
	`self.rentalCar.countryOfRegistration`
	`implies`
	`self.returnBranch.operatingCountry =`
	`self.rentalCar.countryOfRegistration`

5.2.1.6 Derivação com cardinalidade mínima

A categoria de regra de derivação que apresentou o maior número de regras de negócio foi a "derivação com cardinalidade mínima". Nessa categoria, foram classificadas 6 regras de negócio do exemplo mostrado.

Para exemplificar a transformação das regras de derivação com cardinalidade mínima, será utilizada como exemplo a regra de negócio "*It is necessary that if a rental car is*

in need of service then the rental car has a scheduled service". Sua representação na sintaxe abstrata da SBVR é mostrada na Figura 53.

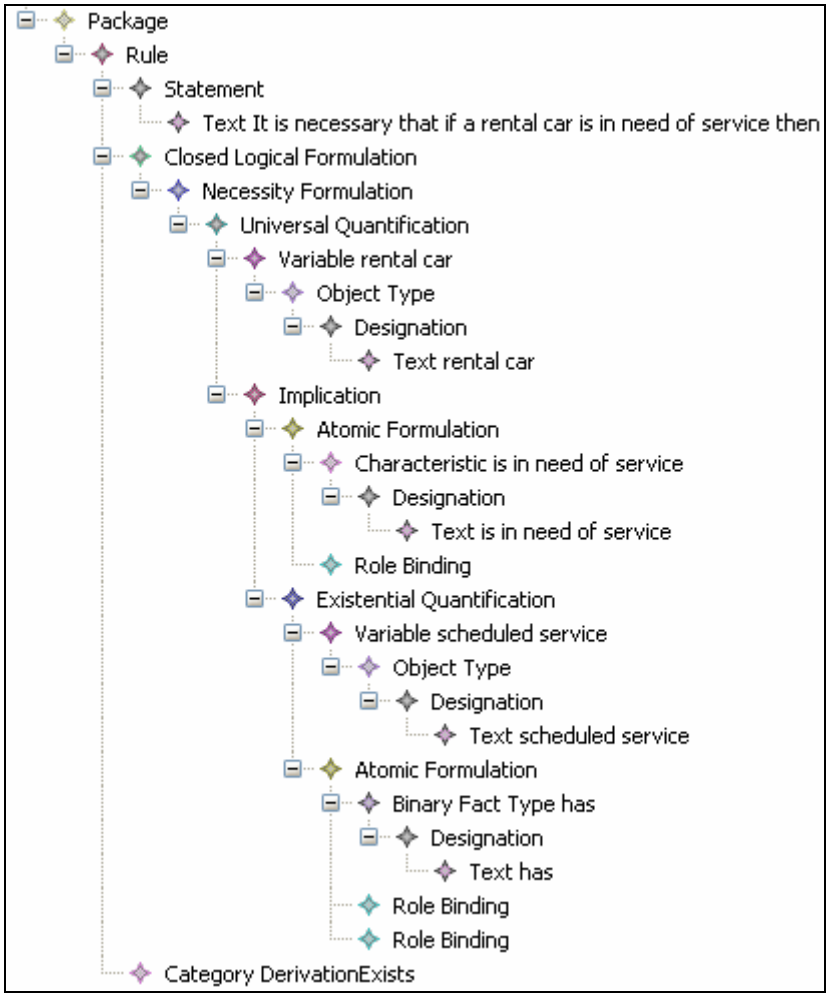

Figura 53 – Um modelo de objetos para a derivação com cardinalidade mínima.

Para as regras de negócio dessa categoria, a expressão OCL resultante utiliza a operação sobre coleções *notEmpty*. A Figura 54 mostra o uso da operação no resultado obtido para o exemplo citado.

```
context RentalCar inv inv invName:
self.isInNeedOfService implies
self.scheduledService -> notEmpty()
```

Figura 54 – Expressão OCL resultante da transformação.

Os resultados obtidos para as 6 regras de negócio classificadas na categoria "derivação com cardinalidade mínima" são mostrados no Quadro 22.

Quadro 22 – Resultados da transformação para a derivação com cardinalidade mínima.

Regra em SBVR	Regra em OCL
It is necessary that if a rental car is in need of service then the rental car has a scheduled service.	`context RentalCar inv invName: self.isInNeedOfService implies self.scheduledService -> notEmpty()`
It is necessary that if a rental is open then an estimated rental charge is provisionally charged to a credit card.	`context Rental inv invName: self.isOpen implies self.estimatedRentalCharge.creditCard -> notEmpty()`
It is necessary that if the actual return date/time of a rental is after the grace period of the rental then the rental incurs a late return charge.	`context Rental inv invName: self.actualReturnDateTime.isAfter(self.gracePeriod) implies self.lateReturnCharge -> notEmpty()`
It is necessary that if the drop-off location of a rental is not the return branch of the rental then the rental incurs a location penalty charge.	`context Rental inv invName: self.dropOffLocation <> self.returnBranch implies self.locationPenaltyCharge -> notEmpty()`
It is necessary that if the rental car of an open rental is in need of service then the rental incurs a car exchange during rental.	`context Rental inv invName: self.isOpen and self.rentalCar.isInNeedOfService implies self.carExchangeDuringRental -> notEmpty()`
It is necessary that if the rental car of an open rental is in need of repair then the rental incurs a car exchange during rental.	`context Rental inv invName: self.isOpen and self.rentalCar.isInNeedOfRepair implies self.carExchangeDuringRental -> notEmpty()`

5.2.1.7 Derivação com restrição simples sobre elementos de uma associação

Dentre as regras de negócio consideradas no exemplo, apenas uma delas foi classificada na categoria "derivação com restrição simples sobre elementos de uma associação". O modelo de objetos SBVR para a regra de negócio "*It is necessary that if a rental is open then each driver of the rental is not a barred driver*" é mostrado na Figura 55.

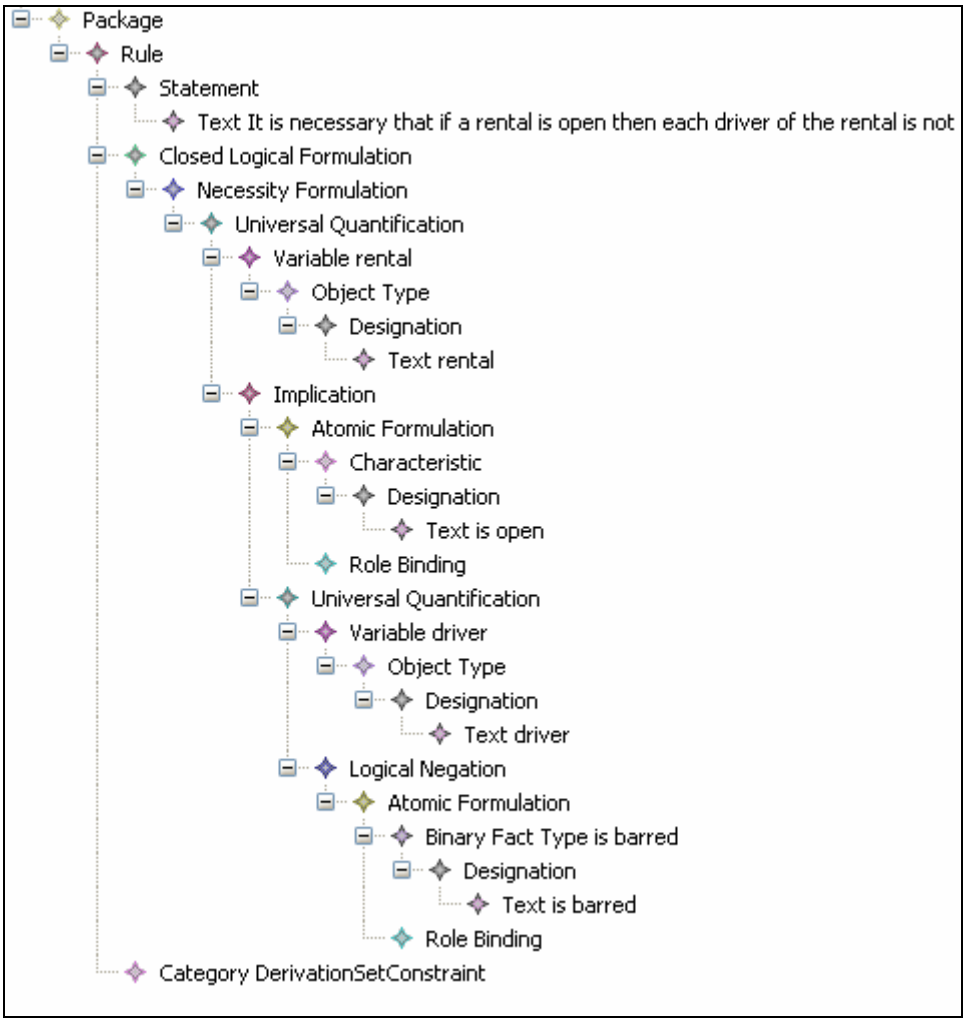

Figura 55 – Um modelo de objetos para a derivação com restrição simples sobre elementos de uma associação.

A expressão OCL resultante da transformação da regra de negócio nessa categoria é mostrada no Quadro 23.

Quadro 23 – Resultado da transformação para a derivação com restrição simples sobre elementos de uma associação.

Regra em SBVR	Regra em OCL
It is necessary that if a rental is open then each driver of the rental is not a barred driver.	`context Rental inv invName: self.isOpen implies self.driver -> forAll (x | not x.isBarred)`

5.2.2 Avaliação dos Resultados

A avaliação dos resultados obtidos com o uso da solução limitou-se a verificação da sintaxe das expressões OCL geradas. Para isso, foi necessária a inclusão das expressões na ferramenta *Papyrus*, tendo como base o modelo de classes construído previamente. Cada expressão foi associada a sua classe de contexto dentro do módulo de restrições OCL do *Papyrus.*

Uma particularidade da transformação de modelo para texto ocasionou problemas na avaliação das expressões que possuíam valores literais com cadeias de caracteres. Nesse caso, as cadeias de caracteres eram representadas entre aspas duplas (“”), enquanto a ferramenta *Papyrus* necessita que as mesmas estejam entre aspas simples (‘’). Apenas 3 regras de negócio encontravam-se nessa situação e precisaram ser ajustadas manualmente. Após esses ajustes, todas as expressões OCL puderam ser avaliadas no *Papyrus* e apresentaram sintaxe válida, considerando-se o modelo de classes criado para o exemplo proposto.

Cabe ressaltar que, além da avaliação da sintaxe das expressões realizada, a tradução proposta para as categorias de regras de negócio da abordagem foi analisada previamente por um especialista na linguagem OCL. Para isso, foram apresentados para cada categoria, exemplos de regras de negócio em SBVR e uma sugestão de tradução para a sintaxe concreta da OCL. A partir da avaliação pelo especialista da tradução sugerida, foi possível definir os mapeamentos da transformação para cada categoria e aplicá-los para todas as regras de negócio classificadas.

Finalmente, a formalização dos mapeamentos entre os elementos dos metamodelos da SBVR e da OCL, utilizando a linguagem ATL como parte do ferramental disponibilizado pela abordagem, foi fundamental para que os mapeamentos propostos para cada categoria pudessem ser testados de forma automática. Além disso, a definição da transformação em um

módulo ATL assegura a correção sintática dos padrões de mapeamentos e a conformidade dos modelos de saída com o metamodelo de destino (na abordagem, OCL).

5.3 Discussão sobre a Aplicação da Abordagem

A definição de mapeamentos para a transformação de modelos permite sistematizar o processo de geração de artefatos na MDA. Nesse contexto, padrões para as transformações são definidos e podem ser aplicados inúmeras vezes para modelos de entrada com características semelhantes. No exemplo de uso apresentado, foi possível, através da classificação das regras de negócio, aplicar os mapeamentos definidos previamente para as categorias estabelecidas. Assim, os resultados esperados pela transformação são conhecidos, uma vez que a definição da transformação é preestabelecida. Essa definição pode ser ajustada e reaplicada conforme a necessidade, permitindo maior agilidade na geração dos artefatos.

Soma-se a essa vantagem, a possibilidade de automatizar os mapeamentos, garantindo assim a correção sintática sob o formalismo de uma linguagem de transformação, como a ATL. As verificações de correção semântica não fizeram parte do escopo deste trabalho, contudo a observação dos resultados obtidos pela abordagem e a comparação com transformações manuais realizadas por especialistas são parâmetros que podem ser considerados na avaliação semântica das transformações de modelos. Além disso, o ganho de produtividade oriundo da automatização pode ser notado com o uso da solução desenvolvida para a abordagem proposta. Após sua execução, automaticamente obtém-se uma expressão OCL representando a regra de negócio em SBVR que foi disponibilizada como entrada para a solução.

A aplicação da abordagem permite também que se tenha, lado a lado, a regra de negócio em SBVR *Structured English* e a sua representação correspondente na linguagem OCL (no nível de abstração adequado aos envolvidos na construção de sistemas de

informação). Assim, a paráfrase das regras de negócio obtida com a abordagem pode tornar mais fácil a validação do modelo conceitual do sistema de informação sendo construído, conforme defendido por (PAU; CABOT, 2008).

Apesar das categorias propostas apresentarem uma cobertura parcial sobre o conjunto de regras de negócio disponíveis no estudo de caso da especificação da SBVR, a maioria delas puderam ser tratadas pela abordagem. Dentre as classificações de regras de negócio presentes na literatura, categorias de restrição e de derivação são constantemente citadas e, por essa razão, optou-se em destacá-las na proposta de transformação. A extensão da abordagem com a inclusão de novas categorias de regras de negócio e a definições de outros mapeamentos é factível e desejável, a fim de se obter a ampliação da cobertura alcançada.

O conjunto de expressões OCL geradas pela aplicação da abordagem complementa o modelo de classes do exemplo com invariantes, tornando-o mais expressivo, completo e preparado para novas transformações, dentro do contexto da MDA. Além disso, essas expressões fornecem uma representação única e sem ambigüidade para as regras de negócio, permitindo sua verificação por ferramentas automáticas, a fim de garantir sua consistência e correção em relação aos elementos do modelo de classes, conforme exemplificado com a utilização do *Papyrus*.

Exemplos de informações que não poderiam ser expressas apenas usando os elementos gráficos da UML são as 22 restrições simples do conjunto de regras de negócio utilizado. Essa categoria representou aproximadamente 55% do total de regras de negócio do exemplo mostrado. A aplicação da transformação resultou em expressões OCL que apresentam, em linhas gerais, comparações de propriedades das classes do modelo em UML com literais ou com outras propriedades, restringindo os valores que essas propriedades podem ter. Em modelos de classes, uma alternativa possível para a representação de tais

restrições é com uso da OCL, uma vez que não é possível representá-las usando somente a UML. Assim, no modelo de classes do exemplo, foi possível estabelecer através da utilização de expressões OCL, o significado das especializações das classes do modelo (através da definição de restrições que permitiram diferenciar especializações de uma classe) e os conjuntos de valores de domínio de uma propriedade (ou seja, seus valores válidos).

Finalmente, pode-se dizer que a abordagem de transformação de regras de negócio proposta estabelece uma ponte entre duas especificações importantes do OMG. A primeira delas é a OCL, já estabelecida e consolidada, e a segunda, mais recente, é a SBVR, que se apresenta como uma peça importante na definição de modelos independentes de computação que possam ser utilizados efetivamente na MDA. Além disso, ao propor uma transformação entre os níveis CIM e PIM do arcabouço, contribui com o conhecimento atualmente disponível nessa área, uma vez que transformações entre esses dois níveis são menos citadas na literatura sobre o assunto.

6 CONCLUSÃO

O desenvolvimento de sistemas de informação é uma tarefa de grande complexidade. Plataformas e ambientes computacionais heterogêneos, demanda por maior agilidade no processo de desenvolvimento, sistemas de informação cada vez maiores e com requisitos mudando freqüentemente são alguns dos fatores que contribuem para essa complexidade.

O uso de modelos na construção de sistemas de informação permite lidar com essa complexidade e apresenta-se como um importante aliado da área de sistemas. Nesse contexto, a MDA insere-se estabelecendo um arcabouço conceitual baseado em modelos que permite a realização do Desenvolvimento Dirigido por Modelos. Com a MDA, o processo de desenvolvimento de sistemas de informação passa a ser direcionado pela atividade de modelagem e os modelos passam a ser os principais elementos do processo.

O entendimento do negócio a fim de se construir sistemas de informação que atendam às necessidades dos usuários pode ser facilitado com a identificação e modelagem das regras de negócio. Em um contexto de desenvolvimento de sistemas de informação baseado nos princípios da MDA, as diversas formas de expressão das regras de negócio, as audiências às quais os modelos de regras de negócio se destinam e os diferentes níveis de abstração definidos no arcabouço devem ser considerados na modelagem das regras de negócio.

Considerando a inserção dos modelos de regras de negócio na MDA, a presente dissertação apresentou uma abordagem para a transformação de regras de negócio entre o modelo independente de computação e o modelo independente de plataforma na Arquitetura Dirigida por Modelos.

6.1 Contribuições

Existem diversas abordagens para o tratamento das regras de negócio na Arquitetura Dirigida por Modelos. Conforme discutido neste trabalho, o problema de geração de novos modelos de regras de negócio a partir de modelos já especificados permeia várias das propostas apresentadas. Nesse sentido, a construção de transformações que permitam o reuso e a geração automática de modelos é um dos caminhos disponíveis para o tratamento desse problema.

A abordagem proposta nesta dissertação utilizou a transformação de modelos para converter regras de negócio especificadas em SBVR em regras de negócio especificadas em OCL, numa transformação de CIM para PIM. Conforme mostrado anteriormente, a transformação de regras de negócio entre esses dois modelos da MDA, usando as técnicas e padrões estabelecidos no arcabouço, foi a lacuna identificada nos trabalhos relacionados a esta pesquisa. O problema estudado aqui foi como agilizar a geração de modelos de regras de negócio numa abordagem direcionada a modelos, considerando-se o modelo independente de computação e o modelo independente de plataforma da MDA.

Na abordagem de solução apresentada, foram definidos mapeamentos entre os elementos do metamodelo da SBVR e da OCL, a partir de uma proposta de classificação para as regras de negócio. Além disso, para apoiar a utilização prática da abordagem e automatizar os mapeamentos propostos foi desenvolvida uma solução de automação utilizando a linguagem de transformação ATL e outras ferramentas da plataforma Eclipse.

Uma das principais contribuições da transformação foi estabelecer um modelo independente de plataforma mais detalhado e completo. As expressões OCL geradas pela solução implementada tornam o modelo de classes mais expressivo. As regras de negócio são representadas no PIM de maneira única e sem ambigüidade, podendo ser verificadas por ferramentas automáticas.

Além disso, o tratamento sistemático das regras de negócio estabelecido na abordagem padroniza a forma como elas devem ser especificadas durante o desenvolvimento de um sistema de informação. Soma-se a isso a agilidade na geração dos modelos de regras de negócio no PIM, uma vez que os mapeamentos necessários para a transformação já estão previamente determinados.

Por fim, destaca-se o conhecimento explícito sobre quais regras de negócio estão sendo consideradas no desenvolvimento dos sistemas de informação. Como a abordagem mantém a relação existente entre a regra de negócio no CIM e no PIM, validações e refinamentos da especificação dos sistemas podem ser facilitadas com o uso dessa informação.

6.2 Limitações e Trabalhos Futuros

O uso da abordagem no exemplo apresentado serviu para mostrar a utilização prática da proposta. Apesar de não ser suficiente para a validação efetiva do trabalho, esse exemplo permitiu a análise da transformação e serviu também para a identificação de algumas limitações da abordagem e da solução desenvolvida.

Dentre as limitações identificadas, a cobertura das categorias de regras de negócio propostas na abordagem poderia ser ampliada. Além disso, a estratégia de modelar todas as regras de negócio como necessidades interfere na semântica das regras de negócio definidas pelo especialista do negócio.

Melhorias na solução certamente facilitarão o uso da abordagem. Dentre as características que podem ser desenvolvidas destacam-se: suporte à modelagem de regras de negócio usando termos e fatos já especificados, criação automática das formulações lógicas das regras de negócio, visualização do modelo de classes e das regras de negócio em OCL associadas ao modelo, entre outras.

No exemplo de utilização mostrado, questões relacionadas à performance da solução implementada e ao tempo para a especificação e transformação das regras de negócio não foram tratadas. Um estudo específico pode ser realizado para a avaliação dessas questões e do aumento de produtividade obtido com a aplicação da abordagem em um processo de desenvolvimento de sistemas de informação.

Finalmente, considerando-se as limitações apresentadas e as perspectivas de novas pesquisas, apontam-se como trabalhos futuros:

- Especificação da transformação das regras de negócio descritas em SBVR *Structured English* em formulações lógicas;
- Especificação da transformação das formulações lógicas da sintaxe concreta para o metamodelo da SBVR;
- Definição de novas categorias de regras de negócio, ampliando a classificação estabelecida no trabalho;
- Ampliação da transformação proposta, estabelecendo novas linguagens de destino, além da OCL;
- Integração da abordagem proposta com processos de desenvolvimento de sistemas de informações baseados na MDA;
- Verificação semântica da transformação de modelos proposta.

REFERÊNCIAS BIBLIOGRÁFICAS

ALENCAR, A. J. ; SCHMITZ, E. A. **Análise de investimentos em projetos de tecnologia da informação**. Rio de Janeiro: Expert Books, 2006.

ALENQUER, P. L. **Regras de negócio para análise em ambientes OLAP**. 2002. Dissertação (Mestrado em Informática) – Instituto de Matemática, Núcleo de Computação Eletrônica, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2002.

ATL. Atlas transformation language: ATL user manual. Version 0.7. Nantes: ATLAS Group, 2006. Disponível em: <http://www.eclipse.org/m2m/atl/doc/ATL_User_Manual%5Bv0.7%5D.pdf>. Acesso em: set. 2008.

BEYDEDA, S. ; BOOK, M. ; GRUHN, V. **Model-driven software development**. Berlin: Springer, 2005.

BROOKS, F. P. No silver bullet: essence and accidents of software engineering. **Computer**, New York: v.20, n.4, p.10-19, Abr.1987.

BROWN, A. ; CONALLEN, J. **An introduction to model-driven architecture**. Armonk: IBM, 2005. Disponível em: <http://www.ibm.com/developerworks/rational/library/may05/brown/>. Acesso em: jan. 2009.

BUDINSKY, F. et al. **Eclipse modeling framework:** a developer's guide. Boston: Addison Wesley, 2003.

BURKE, D. A. ; JOHANNISSON, K. Translating formal software specifications to natural language. In: BLACHE, P. et al. (Ed). **Logical Aspects of Computational Linguistics**: 5th International Conference … Berlin: Springer, 2005. (LNAI/LNCS, 3492).

CABOT, J .; PAU, R. ; RAVENTÓS, R. From UML/OCL to SBVR specifications: a challenging transformation. **Information Systems**, New York, In press, accepted manuscript, 2009, available online: 24 Jan. 2009.

CORREA, A. ; WERNER, C. ; BARROS, M. Enhancing the understandability of OCL specifications. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 21., 2007, João Pessoa. **Anais ....** João Pessoa: SBC**,** 2007.

DIAS, F. et al. Uma abordagem para a transformação automática do modelo de negócio em modelo de requisitos. In: WORKSHOP ON REQUIREMENTS ENGINEERING, 9., 2006, Rio de Janeiro. **Anais ...** Rio de Janeiro: PUC-RJ, 2006.

ECLIPSE. **MDT-SBVR 0.7 project plan**. Disponível em: <http://wiki.eclipse.org/MDT-SBVR_0.7_Project_Plan>. Acesso em: set. 2008.

ERIKSSON, H. ; PENKER, M. **Business modeling with UML:** business patterns at work. New York: Wiley Computer Publishing, 2000.

FOWLER, M. **UML essencial**. Porto Alegre: Bookman, 2005.

HALLE, B. V. **Business rules applied**. New York: John Wiley & Sons, 2002.

HAY, D. ; HEALY, K. A. **GUIDE business rules project report**. [S.I.]: Business Rules Group, 2000.

IBM. **Generating an EMF model**. Disponível em: <http://publib.boulder.ibm.com/infocenter/rtnlhelp/v6r0m0/index.jsp?topic=/org.eclipse.emf.doc/tutorials/clibmod/clibmod.html>. Acesso em: abr. 2008.

KLEPPE, A. G. ; BAST, W. ; WARMER, J. B. **MDA explained:** the model driven architecture – practice and promise. Boston: Addison-Wesley Professional, 2003.

KLEPPE, A. G. ; WARMER, J. B. **The object constraint language: getting your models ready for MDA**. Boston: Addison-Wesley Professional, 2003.

LINEHAN, M. H. ; FERGUSON, D. F. Business rule standards – interoperability and portability. In: W3C WORKSHOP ON RULE LANGUAGES FOR INTEROPERABILITY, 2005, Washington. **Eletronic Proceedings…** Washington: W3C, 2005.

MACIEL, R. S. P. ; SILVA, B. C. D. ; MASCARENHAS, L. A. An edoc-based approach for specific middleware services development. In: WORKSHOP ON MODEL-BASED DEVELOPMENT OF COMPUTER BASED SYSTEMS, 4., 2006, Postdam, **Proceedings…** Los Alamitos: IEEE Press, 2006.

MACIEL, R. S. P. ; SILVA, B. C. D. ; ROSA, N. S. Desenvolvimento dirigido por modelos: especificando processos através da MDA. In: SIMPÓSIO BRASILEIRO DE SISTEMAS DE INFORMAÇÃO, 4., 2008, Rio de Janeiro. **Anais ...** Rio de Janeiro: SBC, 2008.

MAIA, N. E. N. **ODYSSEY-MDA**: uma abordagem para a transformação de modelos. 2006. Dissertação (Mestrado em Engenharia de Sistemas e Computação) – Programa de Engenharia de Sistema e Computação, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2006.

MANSO, F. **Regras de negócio**. Rio de Janeiro: UFRJ, 2008. Disponível em: <http://equipe.nce.ufrj.br/eber/DSIBRN/ppt%5CAula2.ppt>. Acesso em: set.2008.

MELLOR, S. J. et al. **MDA distilled:** principles of model driven architecture. Boston: Addison-Wesley Professional, 2004.

MILANOVIC, M. **Modeling rules on the semantic web**. Thesis (Master) – Faculty of Organizational Sciences, University of Belgrade, Belgrade, 2007.

MILANOVIC, M.; et al. Sharing OCL constraints by using web rules. In: INTERNATIONAL CONFERENCE ON MODEL DRIVEN ENGINEERING LANGUAGES AND SYSTEMS, 10., 2007, Nashville. WORKSHOP ON MODELLING SYSTEMS WITH OCL 2007, Nashville. **Eletronic Proceedings …** Nashville: ACM/IEEE, 2007.

MILLER, J. ; MUKERJI, J. **MDA guide version 1.0.1**. Needham: Object Management Group, 2003.

MORGADO, G. P. **RAPDIS:** Um processo e um ambiente MDA para o desenvolvimento de sistemas de informação. 2007. Dissertação (Mestrado em Informática) – Instituto de Matemática, Núcleo de Computação Eletrônica, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2007.

MORGAN, T. **Business rules and information systems:** Aligning IT with business goals. Boston: Addison-Wesley Professional, 2002.

OLIVÉ, A. **Conceptual modeling of information systems**. Berlin: Springer, 2007.

OBJECT MANAGEMENT GROUP. **Business semantics of business rules request for proposal**. Needham, 2003. 50 p.

_____. **Unified modeling language:** version 2.0. Needham, 2005. 218 p.

_____. **Object constraint language:** version 2.0. Needham, 2006. 232 p.

_____. **Semantics of business vocabulary and business rules (SBVR),** v. 1.0. Needham, 2008. 434 p.

PAU, R. ; CABOT, J. Paraphrasing OCL expressions with SBVR. In: INTERNATIONAL CONFERENCE ON APPLICATIONS OF NATURAL LANGUAGE TO INFORMATION SYSTEMS, 13., 2008, London, **Proceedings …** Berlin: Springer, 2008.

PRESSMAN, R. S. **Engenharia de software**. São Paulo: McGraw-Hill, 2006.

RAJ, A. ; PRABHAKAR, T. V. ; HENDRYX, S. Transformation of SBVR business design to UML models. In: INDIA SOFTWARE ENGINEERING CONFERENCE, 1., 2008, Hyderabad. **Eletronic Proceedings …** New York: ACM, 2008.

ROSCA, D. ; GREENSPAN, S. ; WILD, C. Enterprise modeling and decision-support for automating the business rules lifecycle. **Automated Software Engineering**, Amsterdam, v. 9, n. 4, p. 361-404, Oct. 2002.

SENDALL, S. ; KOZACZYNSKI, W. Model transformation: the heart and soul of model-driven software development. **IEEE Software**, Los Alamitos, v.20, n.5, p.42-45, set./out. 2003.

SOMMERVILLE, I. **Engenharia de software**. São Paulo: Addison Wesley, 2003.

SOSUNOVAS, S. ; VASILECAS, O. Tool-Supported method for the extraction of OCL from ORM models. In: ABRAMOVICZ, W. (Ed.). **Business Information Systems**. Berlin: Springer, 2007. p.449-463.

WAGNER, G. ; GIURCA, A. ; LUKICHEV, S. A usable interchange format for rich syntax rules integrating OCL, RuleML and SWRL. In: INTERNATIONAL CONFERENCE ON WORLD WIDE WEB, 15., 2006, Edinburgh, **Proceedings …** New York: ACM, 2006.

ZIMBRÃO, G. et al. FalaOCL: uma ferramenta para parafrasear OCL. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, 16., 2002, Gramado. **Anais eletrônicos...** Gramado: SBC, 2002.

APÊNDICE A – METAMODELO DA SBVR USADO NA SOLUÇÃO

Neste apêndice, é apresentado o modelo de classes do metamodelo da SBVR criado no IBM *Rational Rose*. Esse modelo foi utilizado como entrada para o EMF, a fim de se obter uma versão em Ecore do metamodelo da SBVR para uso na implementação da transformação.

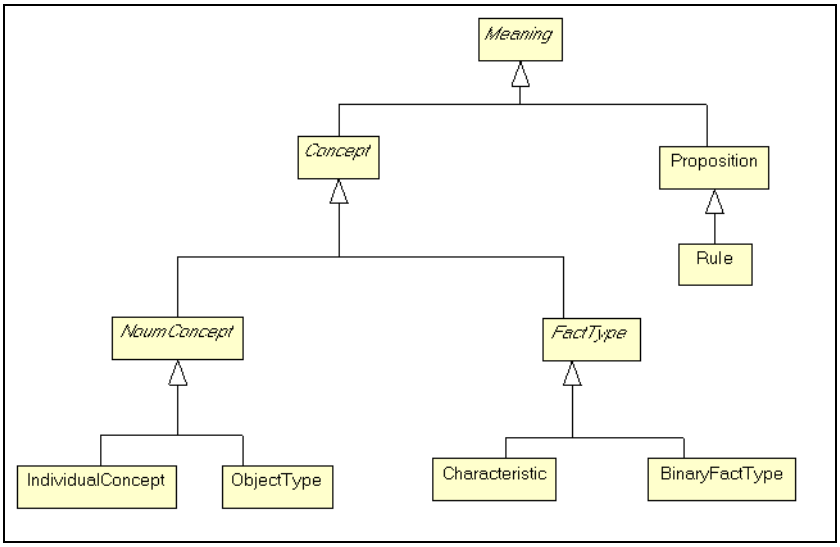

Figura 56 – Significados no metamodelo.

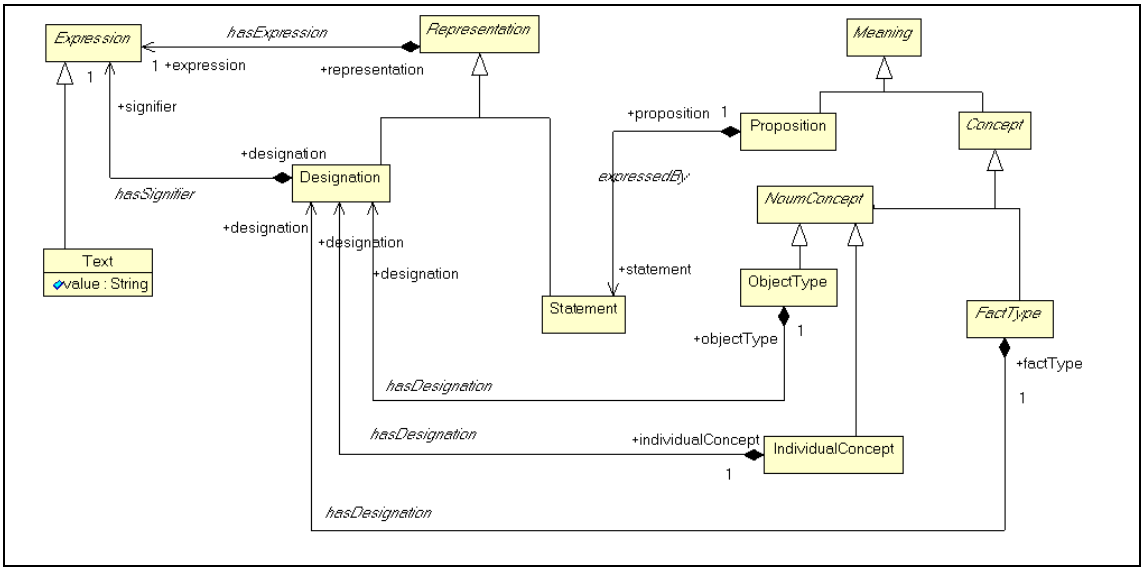

Figura 57 – Significados, expressões e representações no metamodelo.

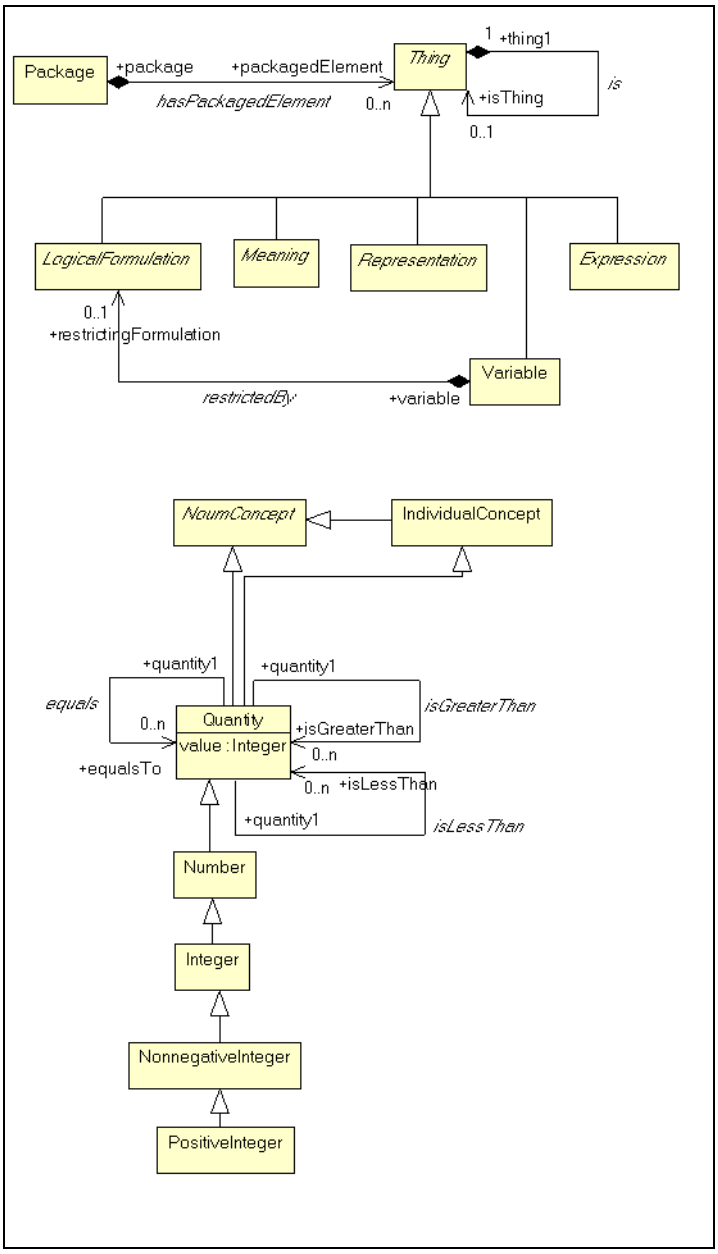

Figura 58 – Conceitos elementares no metamodelo.

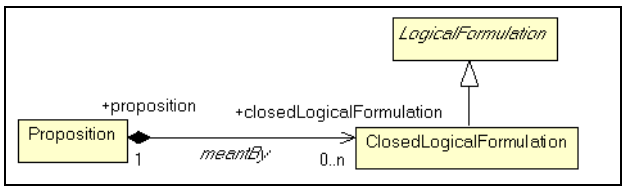

Figura 59 – Formulações lógicas no metamodelo.

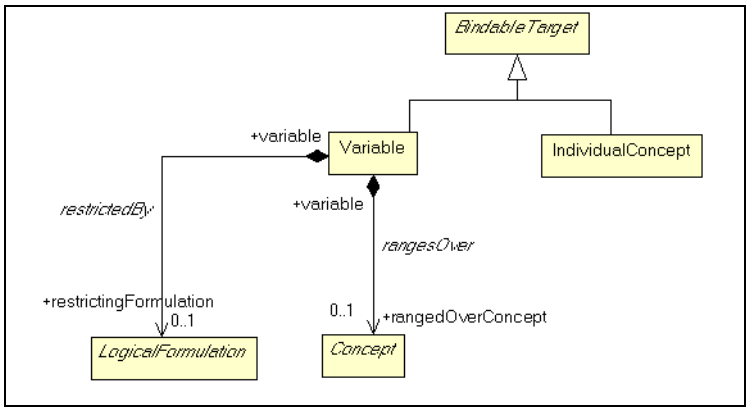

Figura 60 – Variáveis e *binds* no metamodelo.

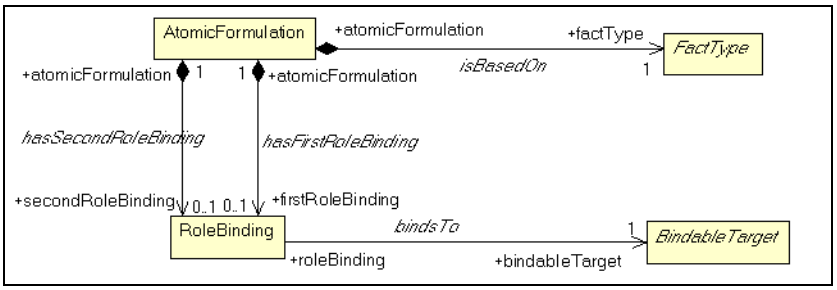

Figura 61 – Formulações atômicas no metamodelo.

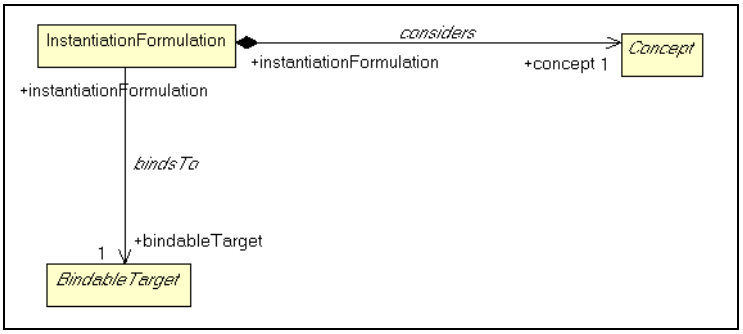

Figura 62 – Formulações de instanciação no metamodelo.

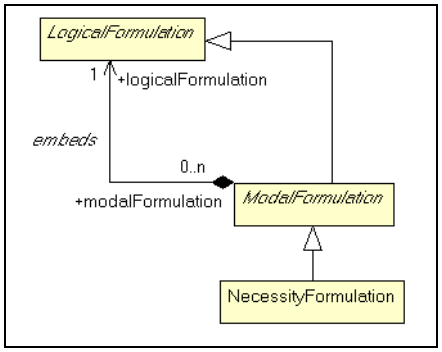

Figura 63 – Formulações modais no metamodelo.

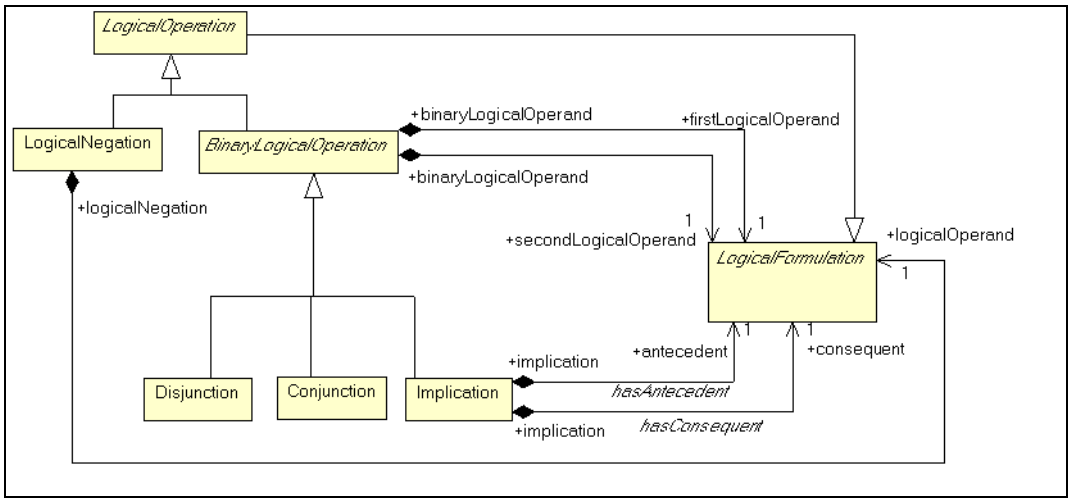

Figura 64 – Operações lógicas no metamodelo.

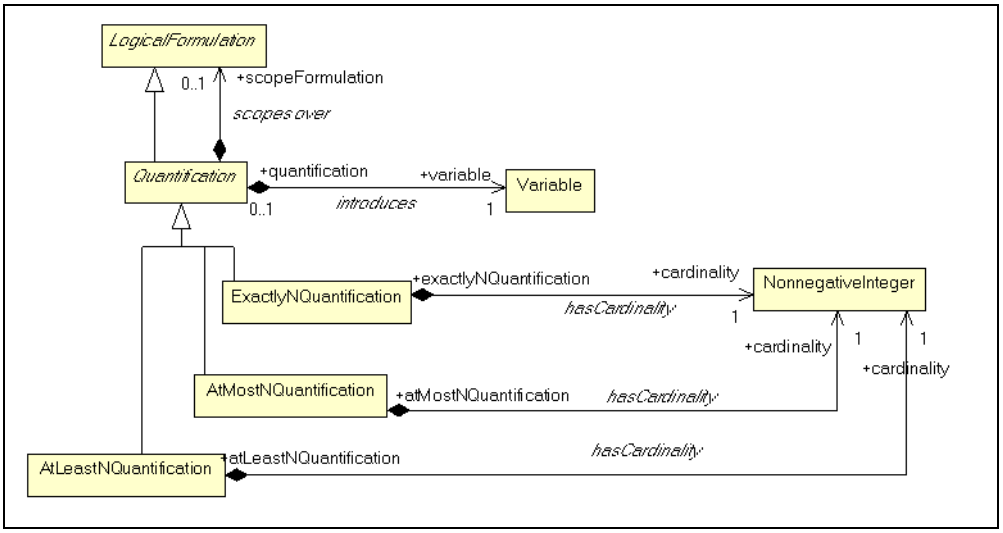

Figura 65 – Quantificações no metamodelo.

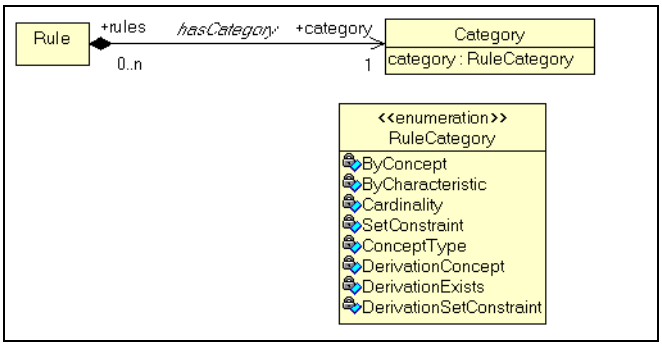

Figura 66 – Regras de negócio e categorias no metamodelo.

APÊNDICE B – MANUAL DE USO DA SOLUÇÃO

Este apêndice apresenta a descrição do uso da solução desenvolvida para a aplicação da abordagem. Para executá-la, é necessária a instalação do Eclipse na versão 3.2.1, além dos *plugins* da linguagem ATL e do EMF. Em http://www.sciences.univ-nantes.fr/lina/atl/www/atldemo/ATLBundle_20070217.zip, encontra-se disponível um arquivo em formato zip com uma versão do Eclipse e de todos os *plugins* necessários para a execução de módulos ATL. Esse pacote foi utilizado no desenvolvimento da solução de transformação deste trabalho.

B.1 Preparação do Modelo de Entrada para a Transformação

Na estrutura de diretórios da solução, há uma pasta chamada “models/templates”, na qual se encontram 7 arquivos Ecore XMI. Cada arquivo corresponde a um *template* de uma categoria de regra de negócio da abordagem e deve ser usado na modelagem da regra a ser transformada pela solução.

Após selecionar o *template* desejado, é necessário adequá-lo à regra de negócio sendo representada, conforme mostrado na Figura 67. Na parte inferior, pode-se ver a área de edição de propriedades dos elementos do modelo de objetos mostrado na árvore exibida na parte superior da figura.

Após a edição do arquivo de *template*, ele deve ser salvo com o nome “sample-sbvr.ecore” na pasta “models” da solução. O arquivo assim preparado está pronto para ser usado como entrada do programa de transformação ATL e a etapa “Modelar regra de negócio” do processo de transformação da solução encontra-se, então, concluída.

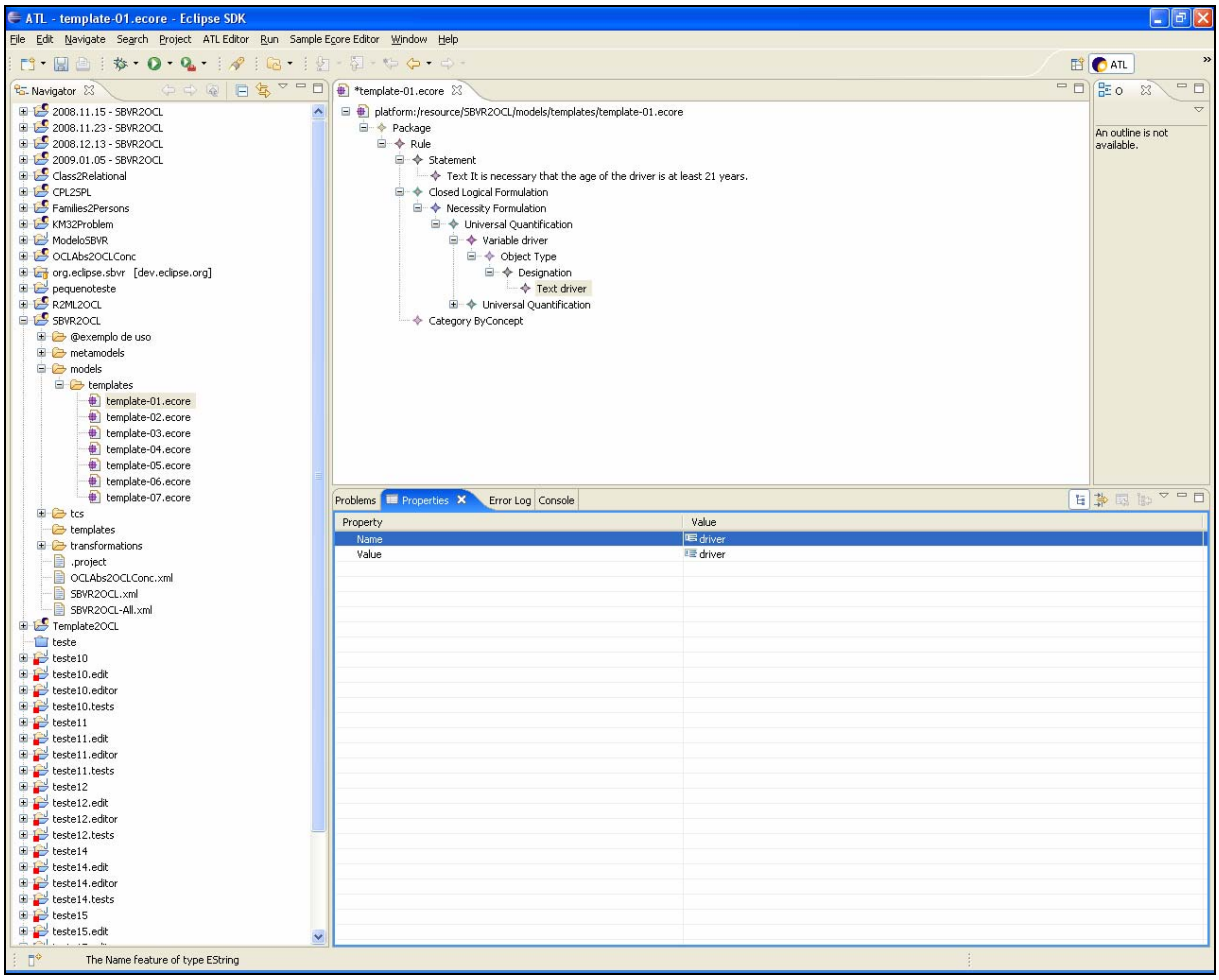

Figura 67 – Edição do arquivo de entrada para a transformação.

B.2 Execução da Transformação

A segunda etapa do processo de transformação compreende a execução do arquivo "SBVR2OCL-All.xml". Esse é um arquivo de configuração *Ant* construído com o objetivo de automatizar os demais passos necessários para realizar a transformação do modelo de entrada.

Após salvar o arquivo "sample-sbvr.ecore" na pasta "models", basta selecionar o arquivo de configuração "SBVR2OCL-All.xml" com o botão direito do mouse e clicar no comando *Run As* → *Ant Build*, conforme mostrado na Figura 68. Assim, automaticamente, as etapas "Carregar metamodelos" e "Executar transformação" do processo são completadas.

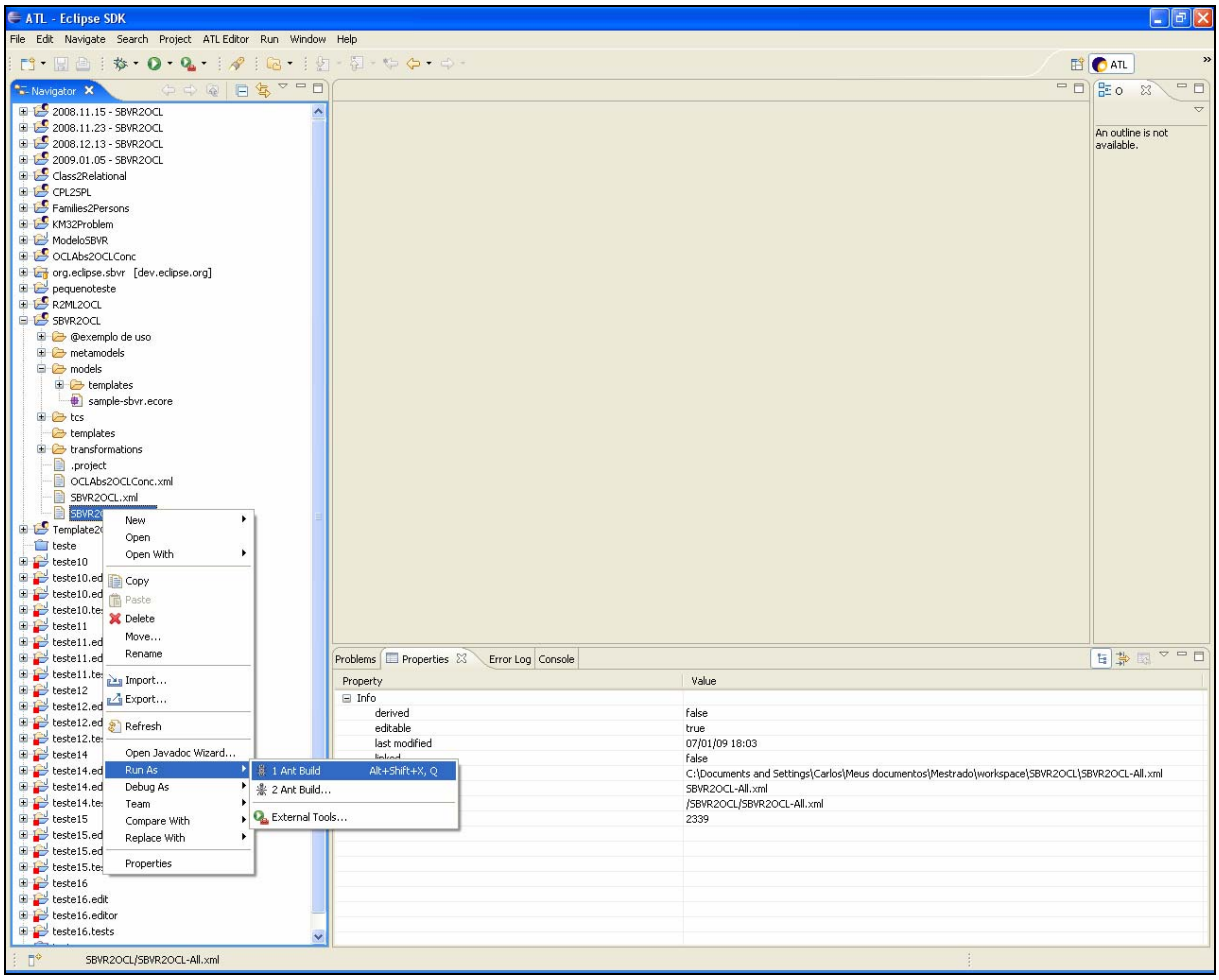

Figura 68 – Execução da transformação com o uso do arquivo de configuração.

Após a execução da transformação, são gerados dois arquivos de saída na pasta "models": "sample-ocl.ecore" e "sample-ocl.ocl". O primeiro deles contém o modelo de objetos da regra de negócio transformada em conformidade com o metamodelo da OCL e o segundo contém a expressão OCL gerada na sintaxe concreta da linguagem. Na Figura 69, podem ser vistos os dois arquivos gerados (no painel esquerdo) e o resultado da execução da transformação na *console* da ADT (na parte inferior da figura).

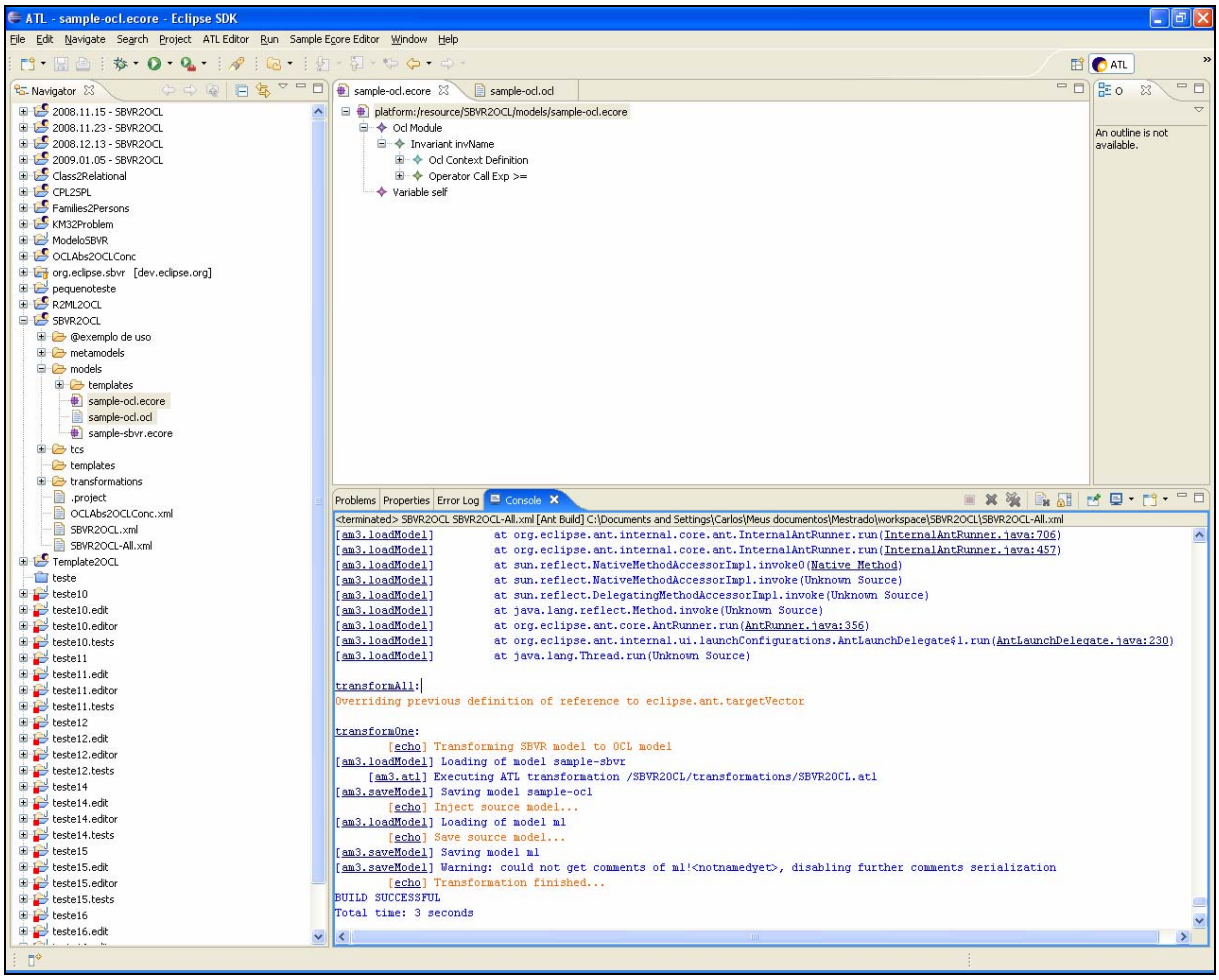

Figura 69 – Arquivos de saída da transformação e *console* da solução.

APÊNDICE C – DESCRIÇÃO DO NEGÓCIO DO EXEMPLO DE UTILIZAÇÃO DA ABORDAGEM

O escopo do negócio usado no exemplo proposto compreende uma empresa de aluguel de carros, chamada EU-Rent. Esse exemplo é razoavelmente difundido em publicações na área de regras de negócio[34], tendo sido completamente revisto para ser incluído na especificação da SBVR.

O vocabulário *EU-Rent English Vocabulary and Rules*, presente no anexo E da especificação da SBVR, serviu como base para a descrição do exemplo utilizado nesta dissertação. Inicialmente, será mostrado, neste apêndice, um panorama geral do negócio de aluguel de carros da EU-Rent e, em seguida, o modelo de classes construído a partir da adaptação e da observação dos diagramas UML e dos verbetes que compõem o vocabulário utilizado.

C.1 Panorama Geral do Negócio do Exemplo

A EU-Rent é uma empresa de aluguel de carros com agências em diversos países. A EU-Rent estabelece regionais (*local areas*) nas cidades dos países onde opera. Cada regional possui diversas agências, que são responsáveis pelos aluguéis dos carros.

Um carro pode ser alugado por um cliente através de uma reserva (*advance rental*) ou de um aluguel imediato (*walk-in rental*). No aluguel, o cliente informa o grupo do carro desejado, as datas de início e término do aluguel, a agência na qual o carro será retirado e na qual será entregue. Vários modelos de carros, organizados em grupos, estão disponíveis para aluguel. Os modelos dos carros de um mesmo grupo possuem o mesmo valor de aluguel. Cada modelo de carro possui um ou mais tipos de combustível.

[34] Em (HAY; HEALY, 2000), pode ser encontrada uma versão mais antiga do exemplo da EU-Rent..

Os carros para aluguel pertencem a uma regional e ficam armazenados em uma agência. Um carro pode ter uma revisão agendada, dependendo do valor do seu marcador de revisão (*service reading*). Os motoristas autorizados pelo cliente de um aluguel a dirigir o carro alugado podem ser qualificados ou barrados. Por padrão, o cliente do aluguel é um motorista autorizado. A empresa EU-Rent possui um clube de fidelidade. Os clientes que fazem parte do clube são classificados como membros e, após acumularem certo número de pontos, estão aptos a solicitar aluguéis de carros com esses pontos.

A EU-Rent opera com aluguéis nacionais e internacionais. O critério para a classificação dos aluguéis nacionais é baseado na agência de retirada e na agência de retorno do aluguel. No aluguel com “ida e volta” (*round-trip rental*), a agência de retirada e a agência de retorno são as mesmas. O aluguel local sem retorno (*local one-way rental*) é um aluguel nacional entre agências de uma mesma regional. Já o aluguel sem retorno no país (*in-country one-way rental*) ocorre entre agências de regionais distintas no mesmo país.

A classificação dos aluguéis internacionais considera o país de registro do carro alugado e os países da agência de retirada e da agência de retorno do aluguel. No aluguel internacional com carro estrangeiro (*international inward rental*), o país da agência de retorno é o mesmo país de registro do carro alugado. Já no aluguel internacional com carro nacional (*international outward rental*), o país da agência de retirada é o mesmo país de registro do carro alugado.

O valor de um aluguel pode ser calculado em dinheiro ou em pontos (no caso de um aluguel por pontos) e é baseado no tempo de duração do aluguel. Há um valor estimado no início do aluguel que é temporariamente provisionado no cartão de crédito do cliente. Ao término do aluguel, o valor real do aluguel é cobrado do cliente. O valor devido é calculado na moeda local do país no qual a agência de retirada se localiza. Caso o cliente deseje, esse valor pode ser convertido para outra moeda.

A operação de transferência de carros entre agências é uma atividade bastante comum, uma vez que os clientes podem devolver o carro alugado em qualquer agência da EU-Rent. Ela ocorre em uma data determinada e envolve uma agência de retirada, uma agência de entrega e um carro. As transferências podem ocorrer entre agências de um mesmo país ou de países diferentes, sendo esse o critério para a classificação das operações.

Um carro alugado pode ser devolvido em qualquer local. Caso o local de devolução não seja a agência da EU-Rent acordada com o cliente, é aplicada uma multa por entrega em local indevido. Além disso, quando o carro é devolvido após o prazo previsto (considerando-se o período de tolerância), uma multa por atraso na entrega é aplicada.

Eventualmente, o carro alugado precisa ser trocado durante um aluguel. Essa situação ocorre quando o carro alugado apresenta um problema operacional ou necessita de manutenção durante o período do aluguel. Além disso, as experiências ruins sofridas pelos clientes durante um aluguel (danos ao carro alugado, estacionamento proibido, multas por excesso de velocidade etc.) são registradas.

C.2 Modelo de Classes do Exemplo

O diagrama de classes apresentado nessa seção mostra as classes, atributos e associações do exemplo usado para a aplicação da abordagem de transformação proposta. Esse modelo de classes foi criado manualmente, baseado no panorama geral apresentado anteriormente, além dos diagramas e verbetes disponíveis no vocabulário *EU-Rent English Vocabulary and Rules* (anexo E da especificação da SBVR). As expressões OCL resultantes da aplicação da abordagem nas regras de negócio do exemplo são válidas para o diagrama de classes mostrado aqui.

Para facilitar a visualização, o diagrama de classes foi dividido em três partes, mostradas nas figuras a seguir. A Figura 70 representa o conceito de aluguel e as suas diversas

categorizações. Além disso, os detalhes relativos aos carros disponíveis para aluguel, aos clientes e às agências da EU-Rent são representados no modelo de classes mostrado.

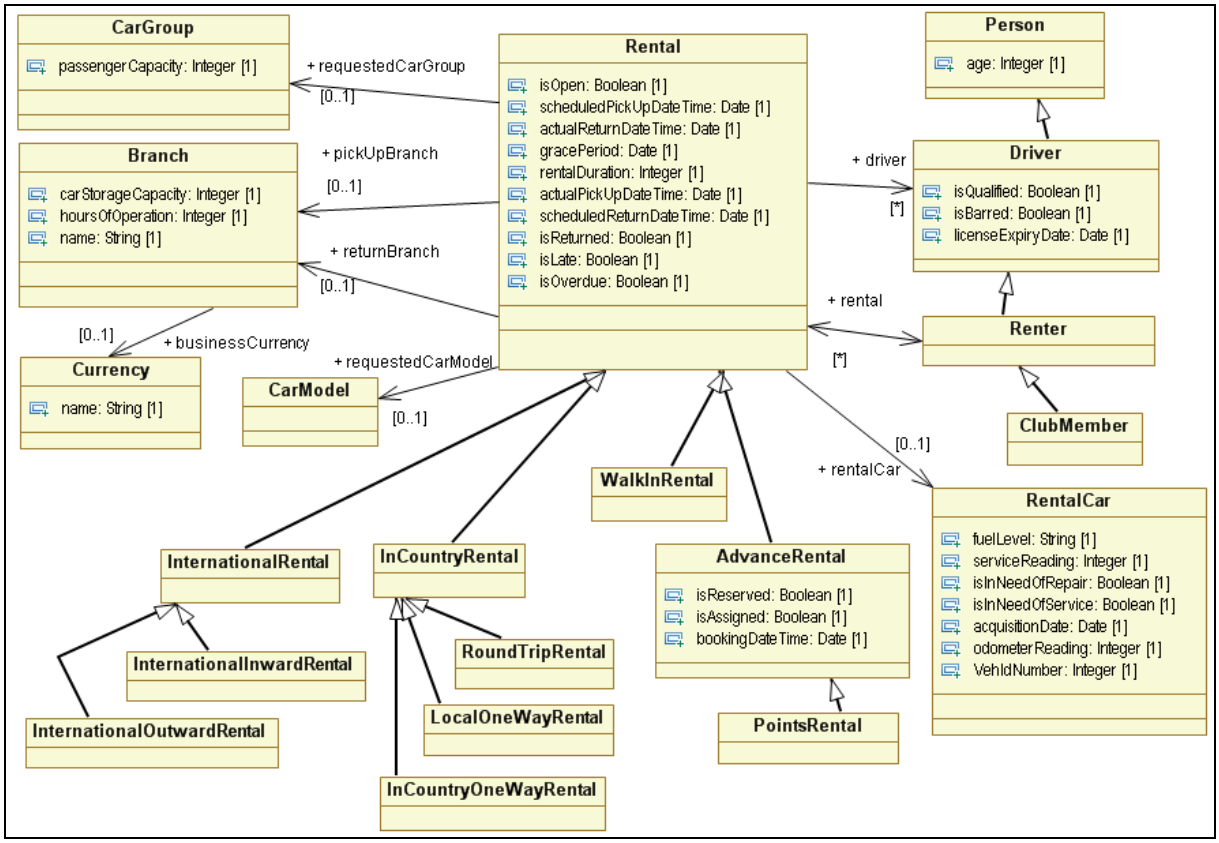

Figura 70 – Primeira parte do modelo de classes do exemplo.

A Figura 71 mostra as diferentes categorias de transferência de carro, a organização física da EU-Rent e os conceitos relativos aos carros disponíveis para aluguel (modelos, grupos, tipos de combustível, serviços de manutenção etc.).

Finalmente, na Figura 72, o foco são os conceitos envolvidos no valor pago pelo cliente em um aluguel e nos locais onde o cliente pode devolver o carro alugado. São mostrados também as categorias de multas que podem ser aplicadas, as experiências ruins e as trocas de carros que podem ocorrer durante um aluguel.

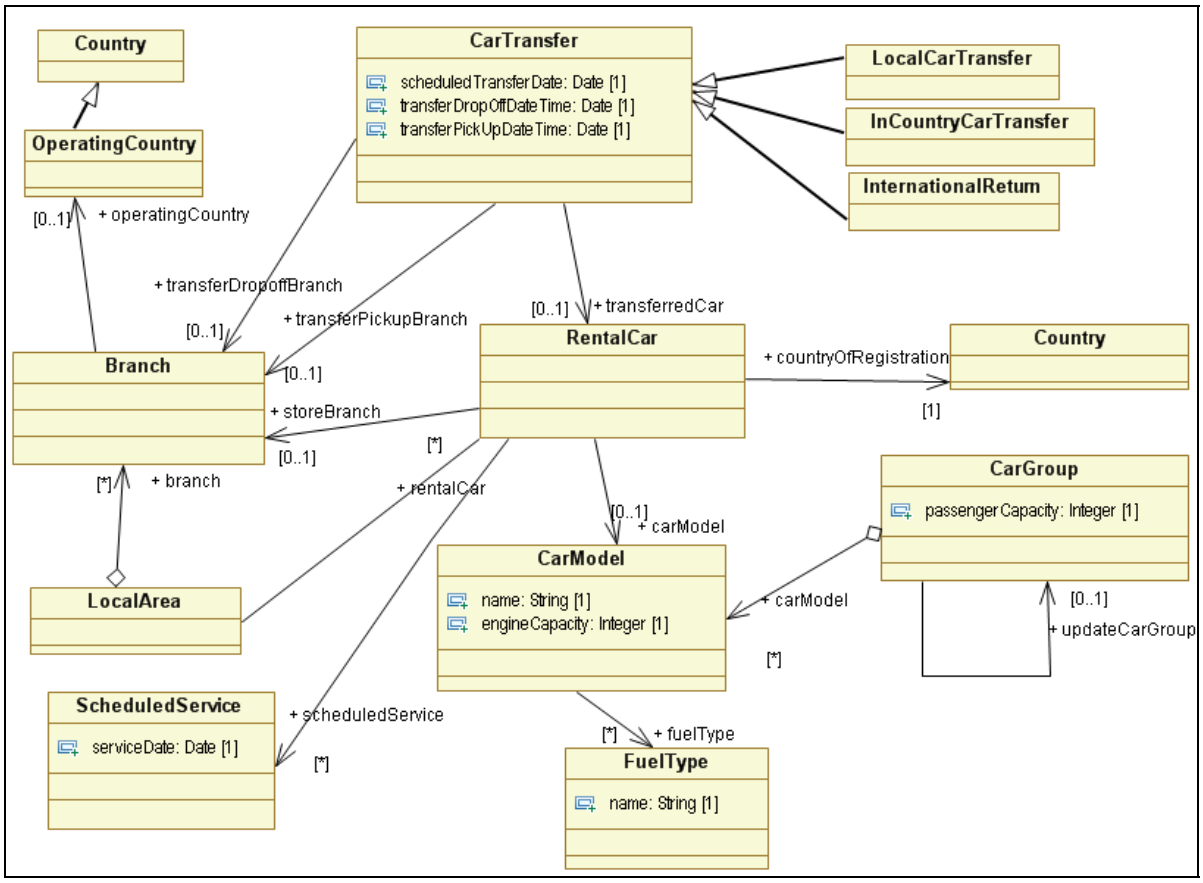

Figura 71 – Segunda parte do modelo de classes do exemplo.

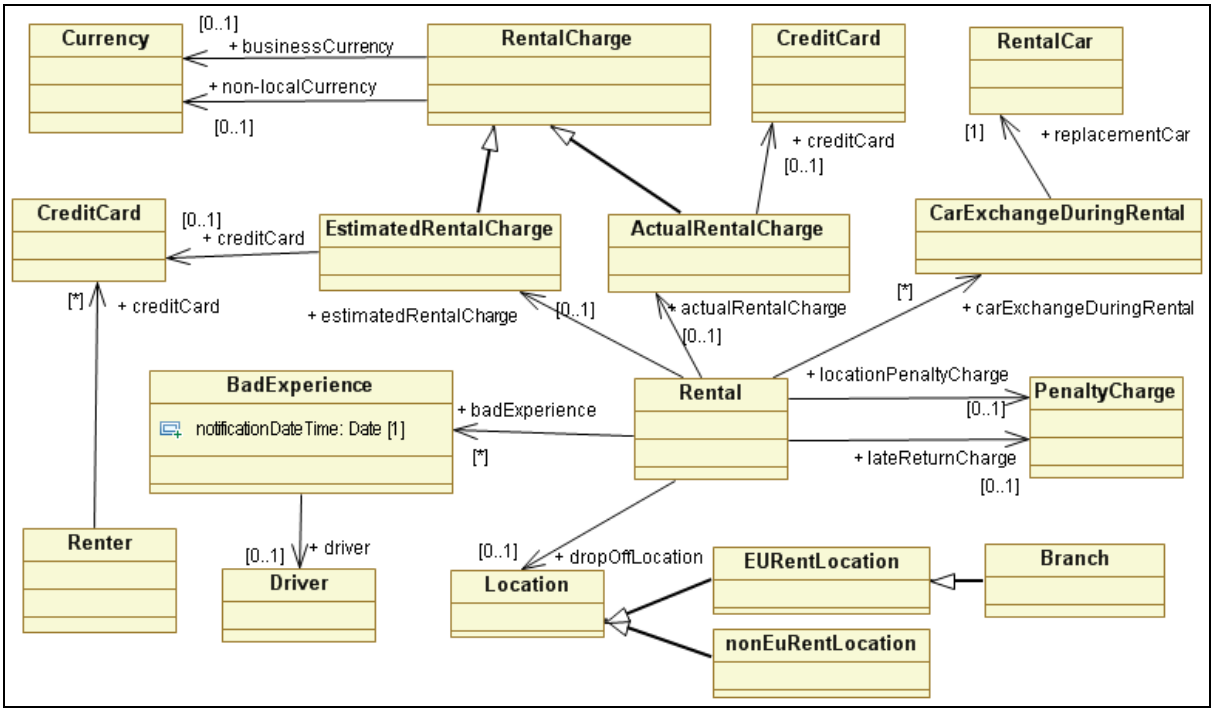

Figura 72 – Terceira parte do modelo de classes do exemplo.

APÊNDICE D – REGRAS DE NEGÓCIO DO EXEMPLO DE UTILIZAÇÃO DA ABORDAGEM

As regras de negócio utilizadas no exemplo mostrado no capítulo 5 são apresentadas nos quadros a seguir, separadas conforme as categorias definidas na abordagem proposta.

Quadro 24 – Regras de negócio com restrição simples.

Regras de Negócio em SBVR
It is necessary that the pick-up branch of a round trip rental is the return branch of the round trip rental.
It is necessary that the local area of the pick-up branch of a local one way rental is the local area of the return branch of the local one way rental.
It is necessary that the local area of the pick-up branch of an in country one way rental is different from the local area of the return branch of the in country one way rental.
It is necessary that the operating country of the pick-up branch of an in country rental is the operating country of the return branch of the in country rental.
It is necessary that the operating country of the pick-up branch of an international rental is not the operating country of the return branch of the international rental.
It is necessary that the operating country of the transfer pick-up branch of an international return is not the operating country of the transfer drop-off branch of the international return.
It is necessary that the local area of the transfer pick-up branch of a local car transfer is the local area of the transfer drop-off branch of the local car transfer.
It is necessary that the operating country of the transfer pick-up branch of an in-country car transfer is the operating country of the transfer drop-off branch of the in-country car transfer.
It is necessary that the operating country of the pick-up branch of each international outward rental is the country of registration of the rental car of the rental.
It is necessary that the scheduled pick-up date time of each advance rental is after the booking date time of the advance rental.
It is necessary that the operating country of the return branch of each international inward rental is the country of registration of the rental car of the rental.
It is necessary that the operating country of the transfer drop-off branch of an international return is the country of registration of the transferred car of the international return.
It is necessary that the booking date time of a points rental is before the scheduled pick-up date time of the rental.
It is necessary that the business currency of the actual rental charge of each rental is equal to the business currency of the pick-up branch of the rental.
It is necessary that the fuel level of a rental car is full, 1/2 or empty.
It is necessary that the service reading of each rental car is at most 5500 miles.
It is necessary that the rental duration of each rental is at most 90 rental days.
It is necessary that the age of the driver is at least 21 years.
It is necessary that the name of the fuel type is Gasoline, LPG or Eletricity.
It is necessary that the name of the currency is Euro, GBP or USD.
It is necessary that the transfer pick-up branch of a car transfer is not the transfer drop-off branch of the car transfer.
It is necessary that the business currency of the estimated rental charge of each rental is equal to the business currency of the pick-up branch of the rental.

Quadro 25 – Regras de negócio com restrição de cardinalidade.

Regras de Negócio em SBVR
It is necessary that each rental has at most four drivers.
It is necessary that each car model has at least one fuel type.
It is necessary that each rental has exactly one requested car group.
It is necessary that each rental has exactly one return branch.
It is necessary that each rental car is owned by exactly one store branch.

Quadro 26 – Regra de negócio com restrição de tipo

Regra de Negócio em SBVR
It is necessary that the renter of each points rental is a club member.

Quadro 27 – Regras de negócio com restrição simples sobre elementos de uma associação.

Regras de Negócio em SBVR
It is necessary that the license expiry date of each driver of a rental is after the scheduled return date time of the rental.
It is necessary that each driver of a rental is qualified.

Quadro 28 – Regras de negócio com derivação simples.

Regras de Negócio em SBVR
It is necessary that if an advance rental is assigned then the store branch of the rental car of the advance rental is the pick-up branch of the advance rental.
It is necessary that if the operating country of the pick-up branch of a rental is not the country of registration of the rental car of the rental then the operating country of the return branch of the rental is the country of registration of the rental car.

Quadro 29 – Regras de negócio com derivação com cardinalidade mínima.

Regras de Negócio em SBVR
It is necessary that if a rental car is in need of service then the rental car has a scheduled service.
It is necessary that if a rental is open then an estimated rental charge is provisionally charged to a credit card.
It is necessary that if the actual return date/time of a rental is after the grace period of the rental then the rental incurs a late return charge.
It is necessary that if the drop-off location of a rental is not the return branch of the rental then the rental incurs a location penalty charge.
It is necessary that if the rental car of an open rental is in need of service then the rental incurs a car exchange during rental.
It is necessary that if the rental car of an open rental is in need of repair then the rental incurs a car exchange during rental.

Quadro 30 – Regra de negócio com derivação com restrição simples sobre elementos de uma associação.

Regra de Negócio em SBVR
It is necessary that if a rental is open then each driver of the rental is not a barred driver.

APÊNDICE E – MODELOS DE OBJETOS EM OCL PARA ALGUMAS REGRAS DE NEGÓCIO DO EXEMPLO DE UTILIZAÇÃO DA ABORDAGEM

Serão apresentados a seguir, os modelos de objetos de saída dos exemplos de algumas regras de negócio mostradas no exemplo de utilização do capítulo 5. As figuras abaixo ilustram, respectivamente, as representações na sintaxe abstrata da OCL (considerando-se as adaptações no metamodelo necessárias para a implementação da transformação usando ATL e TCS) das seguintes regras de negócio:

- "*It is necessary that the age of the driver is at least 21 years*";
- "*It is necessary that each rental has at most four drivers*";
- "*It is necessary that the renter of each points rental is a club member*";
- "*It is necessary that each driver of a rental is qualified*";
- "*It is necessary that if an advance rental is assigned then the store branch of the rental car of the advance rental is the pick-up branch of the advance rental*";
- "*It is necessary that if a rental car is in need of service then the rental car has a scheduled service*";
- "*It is necessary that if a rental is open then each driver of the rental is not a barred driver*".

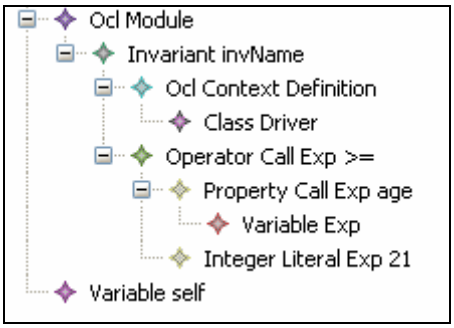

Figura 73 – Modelo de objetos OCL para a restrição simples.

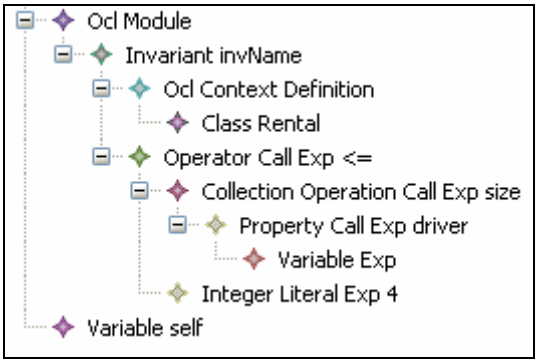

Figura 74 – Modelo de objetos OCL para a restrição de cardinalidade.

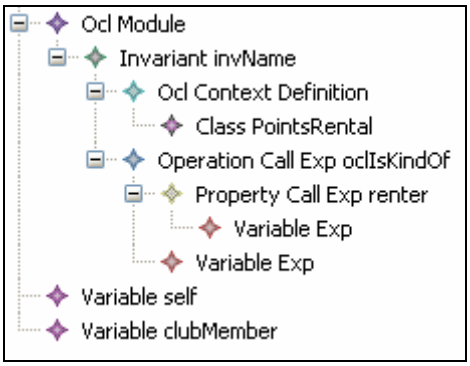

Figura 75 – Modelo de objetos OCL para a restrição de tipo.

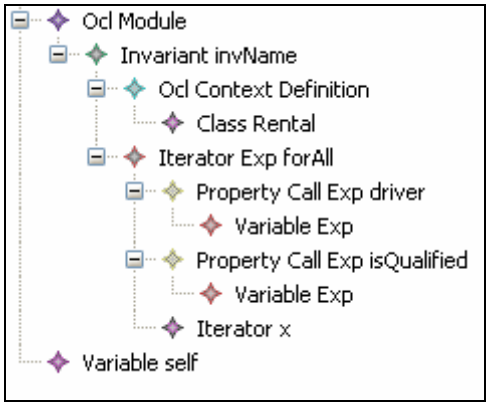

Figura 76 – Modelo de objetos OCL para a restrição simples sobre elementos de uma associação.

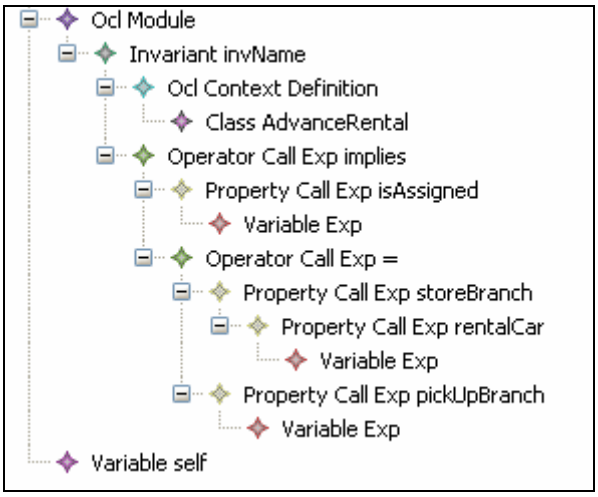

Figura 77 – Modelo de objetos OCL para a derivação simples.

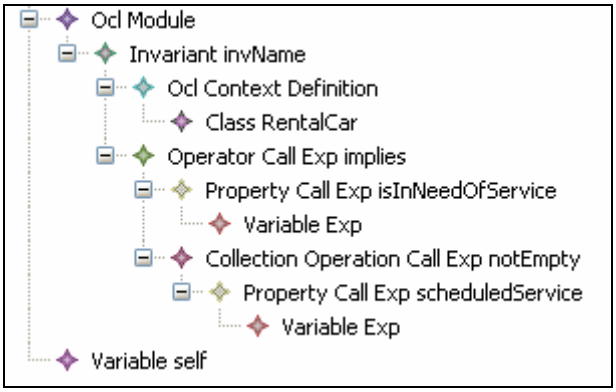

Figura 78 – Modelo de objetos OCL para a derivação com cardinalidade mínima.

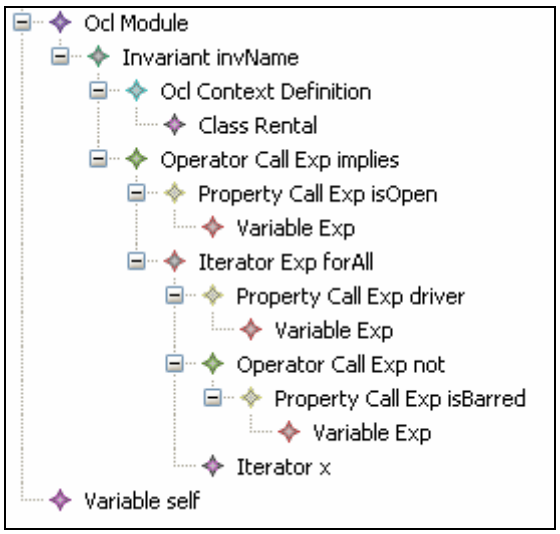

Figura 79 – Modelo de objetos OCL para a derivação com restrição simples sobre elementos de uma associação.

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

CCMN - Bloco C - Cidade Universitária - Ilha do Fundão
Rio de Janeiro - RJ CEP: 21941-916
www.ppgi.ufrj.br