VILNIUS GEDIMINAS TECHNICAL UNIVERSITY

Sergejus SOSUNOVAS

USER DEFINED TEMPLATES FOR THE SPECIFICATION AND TRANSFORMATION OF BUSINESS RULES

Doctoral Dissertation

Technological Sciences, Informatics Engineering (07T)

Vilnius, 2008

Doctoral dissertation was prepared at Vilnius Gediminas technical university in 2004–2008.

Scientific Supervisor

Prof Dr Olegas VASILECAS (Vilnius Gediminas technical university, Technological Sciences, Informatics Engineering – 07T).

http://leidykla.vgtu.lt
VGTU leidyklos TECHNIKA 1547-M mokslo literatūros knyga

ISBN 978-9955-28-353-9

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

Sergejus SOSUNOVAS

VARTOTOJŲ SUDAROMI ŠABLONAI VERSLO TAISYKLĖMS SPECIFIKUOTI IR TRANSFORMUOTI

Daktaro disertacija

Technologijos mokslai, informatikos inžinerija (07T)

Vilnius, 2008

Abstract

The main concept of the current PhD thesis is capturing of business rules through templates and their further propagation to the implementation. The objects of investigation are template specification language and methods of its development. The main aim of the work is the development of specification language of business rules' templates and consequent transformation of business rules using model-driven methods.

The thesis consists of nine chapters including the conclusion.

Chapter 1 contains an introduction to the problem and describes its topicality. The main aims and tasks of the work, its novelty and the methods used, as well as publications of the author and structure of the thesis are presented.

Chapter 2 presents the analysis of publications related to the problems considered in the thesis. General methods for capturing business rules using natural and controlled natural languages are described. This chapter also includes an overview of the business rule dedicated tools. Each tool is analysed from the perspective of easy application and adoption of natural language. Special attention is also given to the ability of the approach to seamlessly propagate business rules to implementation.

Chapter 3 deals with the methods of specification and transformation of domain specific languages within the context of model-driven engineering. The available development methods of the domain specific languages are analysed. The aim is to prepare the background for the development of domain specific business rules templates.

Chapter 4 is the main chapter of the thesis which presents the approach suggested by the author in specifying business rules templates. BRTL approach designated to the specification of templates and the appropriate templates' design technique are described there.

The suggested approach is evaluated through transformation of business rules to semi-formal language. Chapter 5 presents transformation details and design solutions. Transformation is implemented in two steps. First, ORM model and the constraints are transformed to UML/OCL. The second step involves the transformation of a couple of business rules templates to OCL.

Chapter 6 documents the findings of the experiment aimed at determining the extent to which business rules specified by using BRTL can be employed within the model-driven development of the financial reporting systems. The results of the experiment are compared with the data available from four historical projects of the same domain.

The thesis comes to an end with general conclusion about accomplished research and research goals.

Reziumė

Disertacijos pagrindas yra verslo taisyklių įvedimas panaudojant šablonus ir tolimesnis jų naudojimas įgyvendinant informacinę sistemą. Disertacijos tyrimo objektas yra šablonų specifikavimo kalbos ir šablonų kūrimo metodai. Pagrindinis disertacijos tikslas yra išplėtoti verslo taisyklių, pateiktų kontroliuojama natūralia kalba, įvedimo į saugyklą metodus ir pasiūlyti verslo taisyklių šablonų specifikavimo būdą, pritaikomą standartizuotų kalbų transformavimo įrankiuose. Svarbiausias tokiems būdams keliamas reikalavimas yra galimybė dalykinės srities specialistams savarankiškai pritaikyti informacines sistemas prie besikeičiančios verslo aplinkos.

Disertaciją sudaro septyni skyriai.

Pirmajame (įvadiniame) skyriuje nagrinėjamas problemos aktualumas, mokslinis naujumas, darbo tikslai ir uždaviniai, praktinė tyrimų vertė bei aprobavimas tarptautinėse konferencijose ir seminaruose.

Antrasis skyrius skirtas literatūros apžvalgai. Jame pateikta bendra metodų, skirtų aprašyti verslo taisykles naudojant natūralią ir kontroliuojamą natūralią kalbą, apžvalga. Be to, šiame skyriuje aprašyti verslo taisyklių tvarkymo programiniai įrankiai. Kiekvienas įrankis išanalizuotas natūralios kalbos naudojimo aspektu.

Trečiame skyriuje pateikta dalykiniai sričiai skirtų kalbų kūrimo metodų ir šiomis kalbomis užrašytų specifikacijų transformacijų metodų analizė. Yra analizuojami dalykiniai sričiai skirtų kalbų kūrimo metodai. Pagrindinis šio skyriaus uždavinys yra suformuoti pagrindą verslo taisyklių šablonų specifikavimo būdui kurti.

Ketvirtame skyriuje aprašomas siūlomas verslo taisyklių įvedimo naudojant vartotojo sudaromus šablonus būdas. Pateikiama verslo taisyklių šablonų specifikavimo kalbos abstrakti sintaksė, semantika ir konkreti sintaksė. Be to, pristatoma verslo taisyklių šablonų sukūrimo technika.

Penktame skyriuje yra aprašomas modeliais grindžiamos architektūros, kaip karkaso, naudojimas transformuojant objektų rolių modelius (angl. ORM) į unifikuotos modeliavimo kalbos (angl. UML) modelius papildytus objektų ribojimais (angl. OCL). Yra siūloma formali transformacijos taisyklių specifikacija. Transformavimo rezultatai yra prieinami plačiai naudojamiems įrankiams (pvz., Poseidon for UML, Rational Rose, Elipse UML). Skirtingai nuo jau esamų transformacijų, ši transformacija papildomai apima ne tik ORM struktūrinių elementų transformavimą, bet ir ORM ribojimus.

Penktame skyriuje pateikiami eksperimento, skirto įvertinti siūlomo būdo taikymo praktinėje dalykinėje finansinės atskaitomybės srityje lygį, rezultatai. Eksperimentas yra susijęs su specifikavimu ir vykdomo programinio kodo fragmento įgyvendinimu.

Darbas baigiamas bendromis išvadomis apie atliktą tyrimą ir pasiektus tikslus.

Abbreviations

ACE	Attempto Controlled English
ADBMS	Active Database Management System
AST	Abstract Syntax Tree
BAL	Business Application Language
BR	Business Rule
BRS RuleSpeak	Business Rules Solutions Rule Speak
BRT	Business Rules Template
BRTL	Business Rules Template Language
BS	Business System
CASE	Computer Aided Software Engineering
CIM	Computation Independent Model
DSL	Domain Specific Language
EBNF	Extended Backus-Naur Form
ECA	Event Condition Action
EER	Extended Entity Relationship
EMF	Eclipse Modelling Framework
ER	Entity Relationship
IDE	Interface Development Environment
IS	Information System,
IT	Information Technology
MDA	Model Driven Architecture
MDD	Model Driven Development
MDSD	Model Driven Software Development
MOF	Meta Object Facility
MOLA	Model Transformation Language
NIAM	Natural Language Information Analysis Method
oAW	openArchitectureWare (Tool)
OCL	Object Constraint Language
ODRES	Output Driven Requirements Specification Method
OMG	Object Management Group
OO	Object Oriented
ORM	Object Role Modelling
PIM	Platform Independent Model
PNL	Pseudo Natural Language
PSM	Platform Specific Model
QVT	Query View Transformation
RDF	Resource Description Framework

SBVR	Semantics Of Business Vocabulary And Business Rules
SQL	Structured Query Language
SWRL	Semantic Web Rule Language
TCS	Textual Concrete Syntax
TCSL	Textual Concrete Syntax Specification Language
TEF	Textual Editing Framework
UML	Unified Modelling Language
URML	UML-Based Rule Modelling Language
XDE	Extended Development Environment
XMI	Xml Metadata Interchange
XML	Extensible Markup Language

Table of contents

Introduction

Topicality of the problem

Companies have always had policies and rules to define what should or should not be done. Similarly, business rules have been written down in employee manuals for generations and are currently embedded in many legacy software systems. Today, however, Business Rules have achieved a new status as assets of a company that ought to be explicitly defined and managed.

A business rule is a statement that defines some policy or practice of the business. Business rules, whether implemented by employees or by automated systems, determine that appropriate actions are taken at appropriate times. Changes in company policies or practices are invariably reflected in business rules, and the ability to maintain consistency between policies and business rules used in business processes, IT applications, and employee practices, especially when changes take place, has become a key characteristic of agile companies.

Today's efforts to formalize the capture and management of business rules have created a plethora of formal languages for representing business rules in the form of ontology with axioms, UML/OCL, Z, Contextual Graphs, different kind of logics, etc. While these languages comply with the basic design decisions of the business rules based systems, the human aspects of these languages (learnability, readability, writability) have received little attention. At the same time, it is expected that these languages will be widely used in the future, not

only by machines but also by humans. In this thesis, we argue that this wide adoption can be made possible only by bringing the various formats closer to the end user, somebody who has usually no training in formal methods.

Why templates?

Business rules (BRs) representation as templates is one of the most natural ways to present BR for the business user. Templates have demonstrated their effectiveness in the field of information extraction and specification of ontology axioms. In both cases, they allowed us to hide complex implementation details and simplify knowledge representation for the user.

The main goal of the present research is to make templates available for BR owners that are often unfamiliar with different formal specification techniques. BR template is intended to define the structure of BR of some particular type. There are gaps left in BR templates to be filled in later, when the actual activities on specifying business rules are executed. In addition to automatic processing of BR, another expected advantage of this approach is that users feel comfortable with BR templates as if they were working with natural language statements.

However, it is very difficult to define BR templates acceptable for each enterprise in advance. It becomes especially crucial in the worldwide context when adaptation to different cultures and (as a consequence) languages is a must. Therefore, it is necessary to enable specification of custom user centric BR templates. For custom templates to be still available for automatic transformation they should be built on the basis of well-defined template definition constructs.

Why transformation?

Enterprise transition to the modern information processing and management requires the establishment of a new project and execution of the appropriate phases like business analysis and requirement elicitation. The most part of the enterprises have already executed a number of business analyses and requirement specification phases in the previous projects. Hence, the new project could use the available business models and requirements' specifications. Nevertheless, the existing models were created with the old technology in mind and could not be very useful. Hence, the existence of technological independent business oriented models will simplify enterprise adaptation to the future technologies.

The problem of reuse of models is addressed by the model-driven architecture (MDA) developed by object management group (OMG). MDA is based on several layers of models and transformation of models of platform-independent layer to the models of platform-specific layer. At the last layer, applications are generated. Model transformation is defined by transformation rules of transformation specification. These transformation rules refer to the

meta-models of the models being used.

A modern enterprise system consists of business, information and software systems. It is usual to create models of business systems, elicit business needs, specify information system (IS) requirements, and, then, using IS needs, to specify software systems' requirements. At each of the above-mentioned steps, some kinds of models should be created. Transformation between these models is usually executed manually by the appropriate IT staff. It is feasible to map the models of these systems to the corresponding MDA levels and automate transformation.

BRs make the main part of all these systems. BRs in the context of business systems are defined as directives, intended to influence or guide business behaviour, in support of business policy that has been formulated in response to an opportunity, threat, strength, or weakness. However, in the IS context BR is a statement that defines or constrains some aspect of the business. Usually, BRs are implemented in the software code. It is possible to claim that automatic transformation of BRs from one system to another will facilitate system development.

In the presented thesis, we advocate the idea of using transformable business rules templates to capture business rules. The ability to create templates is an important feature, because they can help enforce the consistent deployment of rules across different business scenarios, applications, projects and business units. They also provide the basis for creating rule maintenance applications, which only allow end users to modify or create rules within a strict set of constraints appropriate to satisfying different user requirements, application functionality, and security concerns.

In the main part of the thesis, we offer the approach for developing custom business rules templates and capturing business rules, using these templates. Then we discuss how business rules, captured using the suggested approach, can be transformed to different target languages, hence, ensuring seamless and consistent implementation, using de-facto standard transformation languages.

Problem statement

Business rules transformation can be presented schematically as shown in Figure I.1. The schema is organized around various kinds of repositories. Each repository is intended to store models from one particular MDA level. Business rules repository, fact model, and business process model are aimed at the CIM. PIM and PSM artefact repositories store the appropriate MDA models. Each repository is accompanied with repository artefact editors. In the case of business rules, this editor is business rules definer block. PIM and PSM repositories are supported using PIM and PSM artefact editors.

The human figures on the scheme correspond to the actual distribution of the roles in the business rules implementation project. Business analyst is responsible for the authoring of business rules, making changes and reviewing. System analyst is responsible for the correct "translation" of user needs to the specification, using some formal or semi-formal specification language. Designer creates system architecture and reviews it in order that the architecture to be compliant with specification. Developer is responsible for the development of code templates and patterns that implement this architecture. All these roles create the basis for transformation specification.

As a result of their activity, a set of interlevel transformational mapping rules are created that all together are called transformation specifications. Transformation specifications are stored in transformation rules repositories. These transformation specifications govern the way in which business rules from CIM through PIM and PSM are implemented as a code or are loaded to the business rules engine. MDA transformation engines do the job of transforming models from one level to another. During the last transformation step the code is generated.

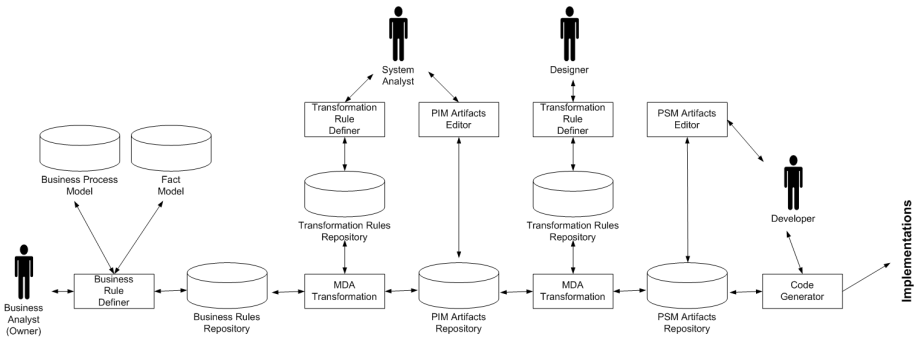

Figure I.1. *Business rules transformation schematically*

However, all these transformations are possible only when the rules are presented in the format understandable to business analyst, and (which is most important) in the form available for transformation. Therefore, a special focus in the work is given to the business rules definer block, which matches the suggested approach of using custom transformable business rules templates as a means to capture business rules. Business rules templates do not introduce any new logic to the business rules. They actually really target the user interface block. The forms and creation principles of BRs templates correspond to MDA requirements, making them transformable by using standardized tools.

The research objects

The thesis research objects are:

1. Business rules authoring using user defined business rules templates.
2. Transformation approaches of business rules.

The aim of the work

The work aims to investigate authoring of business rules using the controlled natural language and to propose the specification approach of business rules templates suitable for the model-driven transformation.

The main tasks of the work

The following tasks have to be implemented to reach the main goal:

1. The methods to capture business rules using templates and controlled natural language should be considered.
2. The requirements for the transformable representation of business rules should be analysed.
3. An approach to capturing business rules should be offered in compliance with the results of the analysis performed.
4. An experiment has to be performed to check the suitability of the proposed approach for transformation of business rules to UML/OCL.
5. An experiment has to be performed to determine if the proposed approach is viable within a practical implementation domain. The results should be compared with the data available from the historical projects.

Methodology of research

1. The comparative research and library research methods were used while analysing modelling languages of the existing business rules,

the methods of business rules implementation in software systems and the methods of template-based representation of knowledge.

2. The results of analysis were summarised and the approach was expounded using the methods of the research generalisation and logical induction.
3. The proposed approach was implemented using the constructive research method.
4. An experiment was performed using the experimental research method.

Scientific novelty

In the present thesis, a fresh look at business rules representation and business rules engineering is provided by discussing how business rules can be entered using templates, and which templates constructs of business rules are important from the transformational point of view.

The following new results are presented in the thesis:

1. A new business rules specification approach BRTL is offered. This approach has the following distinctive features:
    a. BRTL approach employs an idea of user-defined templates. It enables users to develop their own business rules templates or modify the existing ones for the specification of business rules.
    b. It suggests the integration of business rules with fact modelling approach ORM. As a result, business rules templates (and business rules) are integrated using one fact model.
    c. Business rules specified by BRTL are transformable when MDA tools are used. Business rules specified by the approach suggested in the present work can be transformed to the formal languages as well as to the programming languages by applying a standard set of MDA tools from different vendors.
2. A formal transformation definition of business rules specified through BRTL to UML/OCL, with the emphasis placed on practical implementation of UML in widely used modelling tools is suggested. The proposed transformation extends the existing approaches by specifying the transformation of ORM constraints.

3. The experiment results based on BRTL were compared with the data available from the historic software development projects.

The author‘s participation in the scientific projects

During the preparation of doctoral thesis the author participated in the following international research projects:

1. The community programme’s LEONARDO DA VINCI project „Improving Skills, Competencies, and Professional Qualifications in the area of Network Information Security for IT Managers and Staff in the Public Sector” ISCAN (contract No EL/05/B/P/PP-148210]) Participation time 2005.12–2008.01. VGTU project manager: prof. dr. O. Vasilecas.
2. VeTIS project – Business rules solutions for the development of information systems. The VeTIS project was initiated aiming to improve the quality of business model-based development of information systems and the quality of information systems themselves by providing a novel business rules specification method and engineering solutions of this method. Participation time 2007.01–2009.12. VGTU project manager: prof. dr. O. Vasilecas.

Defended propositions

1. Templates-based business rules specification approach BRTL.
2. Specification of model-based BRTL/ORM transformation to the UML/OCL.
3. Evaluation of BRTL approach supplemented with a case study from the financial reporting domain.

Approbation of the results

The results of the thesis were presented at 16 Lithuanian and international conferences. Fifteen scientific papers were published on the topic of dissertation. [1A], [2A], [3A], [4A], [5A], [6A] in the reviewed scientific periodical publications and [7A], [8A], [9A], [10A], [11A], [12A], [13A], [14A], [15A] in the other editions.

The results of the dissertation were presented at the following scientific conferences:

- 10th International Conference on Business Information Systems BIS 2007, Poznan, Poland.
- Conference "Information technologies", 2004, 2006, 2007 and 2008, Kaunas, Lithuania.
- International Conference on Information Systems Development 2005 (ISD'2005), Karlstad, Sweden.
- 18th International Conference on Systems for Automation of Engineering and Research, 2004, Varna, Bulgaria.
- International Conference Information Technologies for Business 2005, Kaunas, Lithuania.
- 13th international conference on information systems development, ISD 2004, Vilnius, Lithuania.
- International Conference Baltic DB&IS 2006, Vilnius, Lithuania.
- International Conference Baltic DB&IS 2004, Riga, Latvia.
- International Conference Baltic DB&IS 2008, Tallinn, Estonia.
- International Conference CompSysTech'05, 2005, Varna, Bulgaria.
- Junior Scientists Conference „Lithuania without science – Lithuania without future" 2003, 2004 and 2005, Vilnius, Lithuania.

1

The related works

In this chapter, the existing works in the application area of the controlled natural language and other related methods as front-end for business rules are discussed. In Section 1.1., a general overview of the related work is presented, while in Section 1.2., the work that is more relevant to our investigation methods, i.e. approaches that use templates and target business rules (or close concepts) is described in detail. These approaches either propose authoring business rules in natural language, or describe verbalization of business rules in controlled English. Only few approaches propose transformable solutions for the representation of business rules, where the approach is used for both seamless authoring and transformation.

1.1. Overview of related work

Currently, there is no unique business rule (BR) definition. Different approaches use their own assumptions to define a BR and use it for different purposes. For example, in [99], BRs are:

…statements of goals, policies, or constraints on an enterprise's way of doing business.

In [94], they are defined as:

statements about how the business is done, i.e. about guidelines and restrictions with respect to states and processes in an organization.

Krammer considers them as "programmatic implementations of the policies

and practices of a business organization" [119] whilst Halle states that:

depending on whom you ask, business rules may encompass some or all relationship verbs, mathematical calculations, inference rules, step-by-step instructions, database constraints, business goals and policies, and business definitions.[5].

The SBVR [120] follows a common-sense definition of 'business rule':

rule that is under business jurisdiction

'Under business jurisdiction' is taken to mean that the business can enact, revise and discontinue BR as it sees fit. If a rule is not under business jurisdiction in that sense, then it is not a BR. For example, the 'law' of gravity is obviously not a BR. Neither are the 'rules' of mathematics.

The more fundamental question in defining 'BR' is the meaning of 'rule'. For the context of BR, rules serve as criteria for making decisions [120].

There are two fundamental categories of Rules:

- Structural Rule (necessities): These are rules about how the business chooses to organize (i.e., 'structure') the things it deals with. A structural rule (like terms, facts, integrity constraints) is intended as a definitional criterion. It is can be captured by the domain model.
- Operative Rules or dynamic constraints (like in [5]) (obligations): These are rules that govern the conduct of business activity. In contrast to Structural Rules, Operative Rules are ones that can be directly violated by people involved in the affairs of the business.

From a conceptual perspective there are approaches that consider BR as an integral part of the modelling and analysis of systems' requirements. An early effort in this direction was the RUBRIC project [96], [97] parts of which were integrated into the information engineering [98] method.

Business Rule-Oriented Conceptual Modelling (BROCOM) introduced a metamodel that formalizes BR in conceptual modelling [94], [95]. In BROCOM, a BR is composed of three components namely event that triggers BR, condition that should be satisfied before an action, and action that describes the task to be done. Moreover, rules are organized according to a rich meta-model, and can be retrieved based on a number of different criteria. As far as methodological guidance is concerned, Herbst proposes the development of various models which are helpful during the analysis phase, but the process of creating and using them is not clearly defined. The transition from analysis to design and implementation has not been addressed by this approach.

The DSS approach [99], [100], [101] focuses on the analysis phase of IS development by supporting the rationale behind the establishment of rules. DSS adopts the ECA (event-condition-action) paradigm for structuring rule expressions and also links these expressions to the entities of an underlying enterprise model. The absence of a formal rule language confines the use of DSS on modelling tasks.

Ross proposed the functional categories of BR i.e. rejectors, projectors, and producers [2]. He also provided a set of rule sentence templates for specifying and capturing BR. The BRS approach is formal, in accordance with the underlying data models of an organization, offers sufficient methodological guidance, and allows management of rule expressions based on a very detailed meta-model. It is also one of the few methods that adopt a graphical notation for expressing rules. Regarding the development process, BRS introduces a BR methodology called BRS ProteusTM methodology that defines a number of steps for both business and system modelling. BRS also provides the BRS RuleTrackTM, an automated tool for recording and organizing BR. However the method mainly oriented towards specification of BR on business level. It does not deal with transformation issues.

The Business Rules Group (BRG), formerly known as the GUIDE Business Rule Project [102], investigated an appropriate formalization for the analysis and expression of BR [103]. This approach identifies terms and facts in natural language rule statements, and consequently, it offers a high level of expressiveness. The meta-model it provides for describing the relations between these terms and facts is very detailed. Therefore, rule models are (a) highly manageable and (b) formal and fully consistent with the information models of a specific organization.

[21], [5A] analyses the ways BR can be entered using semiformal OCL language [22]. OCL is originally developed to specify business constraints in the IBM insurance division. Because of the semantic of OCL is described using meta-model this language can be used in MDA transformations. However OCL is very complex language and it rarely is understood by non technical business owners. The authors of [32] describe the usage of the OCL for the specification of BR in database applications.

The paper [41] present the framework for business IS development which makes use of the conceptual graphs as a conceptual modelling language and employs active databases triggering mechanism for the rules enforcement. The major components of the proposed framework are the following:

- Conceptual model of the BR and business domain
- Active databases for BR implementation
- Trigger generation component
- Business rules repository

The authors of [104] propose a methodology that helps business people and developers to keep BR at the business level inline with the rules that are implemented at the system level. In contrast to several existing approaches that primarily focus on BR in the scope of an application, presented methodology addresses the entire IS of an organisation. The paper also describes requirements for a tool support that would be appropriate to support the methodology.

Obvious that in general case high detail level of the BR model and variety of

templates is treated as a benefit [36]. There are possible scenarios when, BR templates sets are developed for the particular purpose and depending on that purpose BR types are defined. For example, paper [36] demonstrates the application of six BR templates for the generation of full scale UML class model.

It is worth mentioned an experiment of modelling BR in TEMPORA method [37] which demonstrated that communication with owners of BR can be significantly improved presenting BR informally, using natural language or templates.

The Semantics for Business Vocabulary and BR (SBVR) [3] was released in 2005 by the Object Modelling Group (OMG) as the industry Standard for business semantics. However, the lack of an integrated ontology limits the reasoning ability of SBVR. Furthermore at the moment of writing this thesis there was no implementation of SBVR. SBVR distinctive features are summarized in [167]:

- Models are expressed in a fragment of a natural language
- Distinguishes between an expression of a business term, fact, or rule in natural language and the meaning of such an expression.
- Includes metaconcepts vocabulary, speech community, and semantic community, which enable the specification of multiple vocabularies of shared meaning.
- Logical formulation includes deontic modal logic, which enables logical formulations about duty and obligation, i.e., rules.
- Logical formulation includes a restricted higher-order logic, which can capture the expression of multiple meta-levels in the same model – a common occurrence in natural language.
- Fact is a primitive concept in SBVR, a proposition that is taken as true.
- A fact type in SBVR is a logical predicate whose variables (fact type roles) are bound to concepts included in the vocabulary.
- The above mix of terms and fact types is normally and naturally used to write rules in SBVR as grammatically correct English sentences, though possibly somewhat stilted by the chosen fragment of English used for expression.
- Is compatible with the Model Driven Architecture™ (MDA) of the OMG.
- In transforming a SBVR model to UML, terms generally correspond to UML classes and fact types to UML associations.

In general case it is possible to state that during requirements analysis phase BR should be captured using natural language templates, but during IS development phase using formal languages. It is possible to notice distribution of opinions between authors proposing specification of terms and facts using natural language templates and authors proposing to use data and fact models. It is

possible to make assumption that more perspective is the second approach as it is close to the IS development logic.

BR specified using natural language templates, transformation to formal representation techniques are not well investigated. Majority of authors propose to specify BR using generalized templates and as a consequence simplify the information systems development process. Another proposal is to specify BR using formal languages or precise templates, however in this case BR owner's involvement into the specification of BR is limited [38].

From the transformation point of view, the best is to have BR specified using predefined rule templates, which are based on the main types of BR [20]. BR templates are close to the natural language statements and at the same time, the tool allowing transformation of business rule template can be developed easily.

1.1.1. Tool support

Corticon [9] Extended decision table format directly supports users in graphically defining and modelling rules and rule sets. This includes creation of decision tables as well as scorecards type rules. Corticon also supports natural language expressions of rules within a section of Studio, and uses this representation as reference documentation and as source for messages posted during the course of rule set execution. Central to Corticon's model-driven architecture is the generation of executables from the models created and maintained in Studio. These executables are deployed, without modification, on the Corticon Server.

Blaize Advisor [10] Blaze Advisor uses a rules management approach that is based on a combined repository and OO programming language (i.e., Structured Rule Language) that is designed to make writing and reading BR as English-like as possible. Rules can be written using English words and grammar such as "If customer's average balance is more than.", or by using the mathematical symbols and object model "dot notation" familiar to programmers. An extensive selection of rule language keywords is provided, and rules can take advantage of regular expressions and powerful pattern constructs to dramatically reduce the number of rules required. The Blaze Advisor Structured Rule Language (SRL) is an OO language designed to make writing and reading BR as English-like as possible. It shares many features of common programming languages, and is intended for use by programmers to create the entities, control the execution flow, and perform the operations required by the (rule) decision making service.

Blaze Advisor supports the creation of reusable rules management templates that can incorporate entire rule services with multiple steps, functions, and rule sets. Templates help enforce the consistent deployment of rule changes as well as provide the foundation for rule maintenance applications— allowing rules to be modified or created within a strict set of constraints appropriate to satisfy

different users or tasks. Templates are exposed through a Web interface so that end users can update the values, choosing from a list or range of values that is set by the developers.

ILOG [11] Business analysts and other non-technical users can use familiar business terms to create rules using the Business Application Language (BAL). It allows business analysts and other non-technical users to use familiar business terms to create rules, instead of a programming language. (With BAL, business rule artefacts are comprised of combinations of modifiable building blocks (objects), which represent vocabulary elements, rule set parameters and variables, constructs and operators, etc.). IT programmers can author rules using Java-or XML-like syntax using the ILOG Rule Language (IRL) ILOG JRules includes tools for creating and editing templates for BR, decision tables, and decision trees. JRules' template capabilities include: Creating simple, form-like BR suitable for untrained users; Creating many BR with the same form; Restricting the type of BR that end users can write or modify.

Resolution iR [12] does not employ traditional natural-language style rules (i.e., IF-Then…) Rather, the iR Manager GUI features a grid-based interface that makes it easy for the subject matter expert to capture and maintain business logic without any technical expertise. Rule templates are a mechanism to simplify the creation of BR and enable existing rule sets to be easily duplicated. Resolution's rule grid approach enables business analysts to model BR without the need for custom-designed rule templates. The underlying, open XML schema of Resolution's rule definitions allows third parties to add their own templates if needed. The Decision Package is created via the Core using a set of base methods supplied as standard Velocity code templates. The packaging step, performed at design-time, takes each data element in the model and creates a corresponding runtime Java object for integration. In addition, the rule grids defined are converted into directed graphs, combined, and reduced to remove irrelevant combination data.

German company Visual Rules [13] provides a tool for visual modelling of rules in block-schema like style, which may cover some types of BR.

The Protege tool [14] provides facilities for ontology and rules modelling. In particular, it supports modelling in RDF and OWL as well as modeling of SWRL rules. In conjunction with reasoning engine, the tool can be used for consistency check of ontology and serialization to the rule markup. Protege is not a visual tool and requires a significant knowledge of ontology modelling. Moreover it is doubtful that it can be easily adopted in enterprises, which already use UML technologies for software engineering.

Leap SE [39] is a case tool for the requirements specification management and transformation to logical system models. The tool enables unambiguous specification of requirements, using special structured templates. The user can not modify templates. The templates used in the tool are very close to the BR

templates. The tool generates SQL code which can be used developing data bases.

The KeY tool [121], [122] implements OCL language. This tool is worth mentioned because of the number of BR that can be specified using OCL. The tool implements the most often used templates of OCL in natural language. Therefore non-technical user can create constraints using only natural language.

The tool Strelka [17], [15] is an implementation of UML extended with BR which is called URML. This tool proposes an interesting solution for the inclusion of derivation, production, reaction rules into UML. The tool implements visual modelling of BR which is close to the Ross notation.

1.1.2. Templates

The research on which we are report here is inspired by the success of knowledge representation using templates. Templates have demonstrated its effectiveness in the field of information extraction [23], [24] and specification of ontology axioms [25]. In both cases they allowed to hide complex implementation details and simplify knowledge representation for the user.

Many domain experts participate in knowledge acquisition, often without the collaboration of knowledge engineers [25]. Domain experts enter information about classes and properties of concepts through a convenient interface. Unfortunately, to specify additional relational information, they encounter an axiom-editing environment that has remained free-text based. The act of conceptualising a thought in a symbolic representation is often extremely difficult for a domain expert. For example, one may not understand why representing the simple constraint “every employee has a unique ID” in an axiom in first-order logic requires the equivalent English translation of “for every two employees both of whom have IDs, if the two employees are not the same, their IDs cannot be identical.”

The authors of [25] have established the purpose that is very close to the one formulated by this thesis: To achieve truly meaningful transfer of knowledge we must attempt to reduce the barrier between a user and a knowledge acquisition system introduced in the axiom-acquisition phase. Differently from [25] our main target is acquisition of BR.

The process for generating generic axiom templates from the actual axioms in the ontology consisted of the following steps:

1. Identification of axioms that followed exactly the same pattern. They were identical except for the names of specific variables and frames.

2. Generalisation similar patterns into templates. A template accounted for minor variations among patterns. For example, two patterns “A contains B” and “A does not contain B” give rise to one template “A contains/does not contain B.”

3. Derivation of generic properties for categorizing the templates.

Hobbs addresses the problem of template design as an instance of the problem of knowledge representation [123]. In particular, it is the problem of representing essential facts about situations in a way that can mediate between texts that describe those situations and a variety of applications that involve reasoning about them. Furthermore the Hobbs describes what slots to include in the template, and what restrictions to place on their potential fillers.

The problem of automatic creation of domain templates for the information extraction is analysed in [124]. The authors propose a novel methodology for corpus analysis based on cross-examination of several document collections representing different instances of the same domain.

1.1.3. Controlled natural languages

Basic English which was developed as a universally accessible language in 1920 can be treated as a one of the first appearance of the controlled language. Only later, industry (e.g., the European Association of Aerospace Manufacturers, Boeing and General Motors) began to realize the benefits of controlled languages: documentation that is more readable, consistent and (machine) translatable. Only relatively recently have controlled languages begun to focus primarily on computer processability. Here we overview the works in controlled language area related to our research.

Semantics of Business Vocabulary and Business Rules (SBVR) includes a Meaning and Representation Vocabulary that is similar to RDF and can be mapped to OWL in addition to a Structured English notation [114]. A limitation, however, is its lack of support for anaphoric references.

Common Logic Controlled English (CLCE) CLCE [115] is syntactically similar to but slightly less natural than ACE. While it includes ontology for sets, sequences and integers, CLCE does not handle plurals.

Controlled English to Logic Translation (CELT) CELT [116] was originally inspired by ACE, but its lexicon is imported from WordNet (including default word senses) and mapped to the Suggested Upper Merged Ontology (SUMO). The intent of CELT is to simplify ontology-based knowledge representation. It also does not support plurals.

Processable ENGlish (PENG) Like CLCE and CELT, PENG is also quite similar to ACE though lacking plurals. Unlike them, however, work has been done on relating PENG to the existing SemanticWeb standards of OWL and RDF (e.g., [117]). Also of interest is ECOLE [118], a look-ahead text editor that guides the author of PENG texts on-the-fly with syntactic hints, meaning he or she need not learn the rules explicitly.

TRANSLATOR is a free tool available as a Java Web Start application designed to allow anyone, even non-experts, to write facts and rules in formal

representation for use on the Semantic Web [112]. This is accomplished by automatically translating natural language sentences written in Attempto Controlled English into the Rule Markup Language, using the Attempto Parsing Engine Web service as a backend. The translator can equally effectively deal with facts and BR. However its primary focus is derivation rules (also known as inference or deductive rules). Another common type is reaction rules, also known as ECA rules because they consist of an event, a condition and an action. Since ACE does not currently support modality (in this case, necessity), integrity constraints are not easily expressed as consequence are not supported by translator.

In paper [146] the authors present the progress of the natural language usage as the programming paradigm for information extraction in distributed database environments. Personal assistants form an environment where distributed knowledge is explored with the JMining interlingua language to support communication between the mobile agents, natural language queries and the mobile agents working environment servers. The Aglets framework is used to build mobile agents and test conceptual designs for information gathering. The implementation of the prototypes using the aglet framework shows that even with the state of the art natural language technologies the applications development is achievable only on the narrow domain and with the small interlingua language design.

1.1.4. Policies

In very close relation to BR research is policies research area. Policies are a means to dynamically regulate the behaviour of system components without changing code and without requiring the consent or cooperation of the components being governed [105], [106]. By changing policies, a system can be continuously adjusted to accommodate variations in externally imposed constraints and environmental conditions. Therefore in the context of BR research it is worthwhile reviewing policies specification semantic languages.

Policies can be specified in many different ways and multiple approaches have been proposed in different application domains [107]. There are, however, some general requirements that any policy representation should satisfy regardless of its field of applicability: expressiveness to handle the wide range of policy requirements arising in the system being managed, simplicity to ease the policy definition tasks for administrators with different degrees of expertise, enforceability to ensure a mapping of policy specifications into implementable policies for various platforms, scalability to ensure adequate performance, and analyzability to allow reasoning about policies. The challenge is to achieve a suitable balance among the objectives of expressiveness, computational tractability, and ease of use.

KAoS [111] is a collection of services and tools that allow for the specification, management, conflict resolution, and enforcement of deontic-logic-based policies within domains describing organizations of human, agent, and other computational actors. KAoS uses ontology concepts encoded in OWL to build policies. The KAoS Policy Service distinguishes between authorization policies and obligation policies. The applicability of the policy is defined by a set of conditions or situations whose definition can contain components specifying required history, state and currently undertaken action. Policy enforcement requires the ability to monitor and intercept actions, and allow or disallow them based on a given set of policies. While the rest of the KAoS architecture is generic across different platforms, enforcement mechanisms are necessarily specific to the way the platform works.

Rei [110] is a policy framework that integrates support for policy specification, analysis and reasoning. Its deontic-logic-based policy language allows users to express and represent the concepts of rights, prohibitions, obligations, and dispensations. In addition, Rei permits users to specify policies that are defined as rules associating an entity of a managed domain with its set of rights, prohibitions, obligations, and dispensations. The Rei framework provides a policy engine that reasons about the policy specifications. The engine accepts policy specification in both the Rei language and in RDF-S [108], consistent with the Rei ontology. The Rei framework does not provide an enforcement model. In fact, the policy engine has not been designed to enforce the policies but only to reason about them and reply to queries.

Semantic Web Rule Language (SWRL) [109] is based on a combination of the OWL DL and OWL Lite sublanguages of the OWL with the Unary/Binary Datalog RuleML sublanguages. SWRL extends the OWL abstract syntax to include a high-level abstract syntax for Horn-like rules.

1.2. Detailed look on some related works

1.2.1. T. Morgan business rules patterns

Tony Morgan [4] proposes to use three BR levels of expressions:

- Informal. This provides colloquial natural-language statements within a limited range of patterns. For example:

  A credit account customer must be at least 18 years old
- Technical. This combines structured data references, operators, and constrained natural language. For example:

  CreditAccount

  self.customer.age >= 18

Table 1.1 Morgan's *pattern elements*

Element	Description
<fact-list>	A list of <fact> items.
<det>	The determiner for the subject; from the following, the one that makes the best business sense in the statement: A, An, The, Each, Every (or nothing).
<m>, <n>	Numeric parameters.
<enum-list>	A list of enumerated values. An open enumeration indicates that the list may be modified in the light of future requirements; for example, a list of status values for an object as currently known. A closed enumeration indicates that changes to the list are not anticipated; for example, days of the week. The distinction is helpful in later implementation;
<classification>	A definition of a term in the fact model. This typically defines either the value of an attribute, perhaps called "state" or something similar, or a subset of the objects in an existing class.
<algorithm>	A definition of the technique to be used to derive the value of a result; normally expressed using combinations of variable terms identifiable in the fact model together with available constants.
<fact>	A relationship between terms identifiable in the fact model, together with defined constants. The relationship may be qualified by other descriptive elements in order to specify the applicability of the rule precisely.
<result>	Any value, not necessarily numeric, that has some business meaning. The result is often, but does not have to be, the value of an attribute of a business object.
<subject>	A recognizable business entity, such as a business object visible in the fact model, a role name, or a property of an object. The entity may be qualified by other descriptive elements, such as its existence in a particular state, in order to specify the applicability of the rule with enough precision.
<characteristic>	The business behaviour that must take place or a relationship that must be enforced.

- Formal. This provides statements conforming to a more closely defined syntax with particular mathematical properties. For example:

{ X , Y, (customer X) (creditAccount Y) (holder X Y) } ==> (ge (age X) 18)

[4] proposes to start documentation of BR from selecting of the pattern from the list of the available one. These patterns are formed from the pattern elements presented in table 1.1.

Morgan proposes five patterns for the documentation of BR:

Basic constraint: This pattern, the most common business rule pattern, establishes a constraint on the subject of the rule.

List constraint: This pattern also constrains the subject, but the constraining characteristic(s) is (are) one or more items taken from a list.

Classification: This pattern establishes a definition for a term in the fact model.

Computation: This pattern establishes a relationship between terms in the fact model sufficient to allow the computation or the establishment of a value.

Enumeration: This pattern establishes the range of values that can legitimately be taken by a term in the fact model

These patterns can be modified by the user. For complex cases, Morgan recommends to group rules into rule sets, or one "master" rule with a number of subsidiary rules.

The rule statements refer to various terms – used as the definition of <subject> and <constraint> – that are visible in a supporting fact model. What ties a rule down to a particular situation is the explicit reference to something that's visible in the fact model. There is no strict restrictions put on what model use should be used the specification of facts. As a viable option Morgan propose to use UML or ER models.

Although providing structure for the BR documentation Morgan's patterns elements are not detail enough to be suitable for automatic processing. The notion of <fact>, <algorithm> and other pattern elements should be distilled, in order to be implemented in application. At the moment of writing thesis there was no known implementation of Morgan method.

He states that the ultimate goal is an owner having direct control over the rule definitions. The most importantly Morgan states the problem that is addressed in this thesis: because present tools are not sufficiently mature, however, this is not a practical option today, but it's definitely the direction we should be taking.

1.2.2. Kapočius et al. method

Authors of [48], [69], [44] discuss the merger of elements from two different requirements specification methods, thus trying to find a compromise between the needs of stakeholders, analysts and designers. The first one is the Output

Driven Requirements Specification Method (ODRES) [67], [68] and the second BR-based requirements specification method. BRS RuleSpeak-based [2] BR submodel of this method's requirements metamodel was applied for the extension of ODRES. BR templates are one of the techniques that are used in the method to simplify development of the information system. The authors of the method does not analyse automatic implementation/transformation issues of business rules.

From the BR templates perspectives restrictions of the proposed BR and their templates repository are not always easily explainable for to an average user. For example, the template for computation rules [2]:

<Subject> must/should [not] BE COMPUTED as <mathematical formula> [if/while <condition>]

consists of the following five elements [65]:

- subject (can be a computed term or data item),
- predefined text "must/should" with additional optional expression "not" (symbol "/" separates the available options),
- key-phrase "BE COMPUTED as",
- mathematical formula,
- condition with an additional mandatory expression "if/while", that must go before the condition (symbol "/" plays the same role as in case of predefined text).

From the BR repository description presented in [69] and examples it is unclear if the method supports specification of complex BR templates. Presented business rule template consists only from atomic rule parts. However practical applications we have experience with often require usage of composite rule parts (rule parts composed from another rule parts) as in the following example:

[Rule Part | [Composite Rule Part1 [Composite Rule part2 [One more rule part]]]]

From the description of the repository it is possible to make conclusion that condition (logical formulation, mathematical formula parts etc.) of the rules are stored in informal natural text format and could not be process automatically. It is claimed that such representation does not limit rule editor. However, from the transformational point of view such representation of condition makes very hard to execute transformation to the other models and implementation.

The method and corresponding repository were designed to support BRS RuleSpeak presented BR graphical modelling notation and BR templates. Additional BR templates can be introduced to reflect user needs. Usability of the BR application's templates interface was checked using only very simple rules. Presented experiment [69] concentrates on the graphical interface for the specification of BR and omits templates part. Therefore it is impossible to draw any conclusion about applicability of the templates presented in such way to any practical application domain.

One of the declared distinctive features of the method is an ability to reference almost any conceptual model structure (including BR itself) within the body of business rule. This increases the consistency of BR specification.

In order to define BR as it was discussed in the previous approach it is feasible to relate BR to some facts. The authors of the method propose to store fact model in the EER diagram [66]. Despite numerous EER benefits, these models do not have direct mappings to the natural language.

1.2.3. Business rules-driven object oriented design

The BROOD (Business Rules-driven Object Oriented Design) [70] approach addresses business modelling and the linking of business model components to software architecture components. By focusing on the conceptual level, BROOD attempts to externalising changes from software components. This user-oriented view enhances understandability and maintainability since it encourages the direct involvement of business stakeholders in the maintenance of their BR.

The initial concept of the metamodel was introduced in [71]. The metamodel is complemented by a language definition based on the context-free grammar EBNF. The language definition defines the allowable sentence patterns for business rule statements and describes the linking elements between BR and the related software design elements.

There are three BR types supported in BROOD, namely: Constraints, Derivation, and Action assertion. Constraint rules specify the static characteristics of business entities, their attributes, and their relationships. Action assertion specifies the action that should be activated on the occurrence of a certain event and possibly on the satisfaction of certain conditions. A derivation rule derives a new fact based on existing facts. For the described BR types the BROOD provides the number of patterns consisting of one or more well-defined rule phrases. The templates are not intended to be modifiable by the user and at the moment support only one language English.

In addition to the metamodel BROOD provides process of software development using BR. For our research the most interesting phases are design and evolution. During design phase according to BROOD BR phrases are linked to software design component. This activity is executed manually by software designer and insures rules traceability to software design components.

As far as evolution phase concerned ordinarily, simple BR changes could be performed by business users. The implementation of a complex business rule change requires more effort than that of simple change. It involves the introduction of new rule phrases or design elements, which is needed to be performed by an individual with the knowledge of software design.

1.2.4. Object role modelling

Object Role Modelling (ORM) [8] was originally intended for modelling and querying databases at a conceptual level where the data requirements of applications need to be represented in a readily understood manner, thus enabling non-IT professionals to assist the modelling, validating, and maintaining processes. ORM offers a number of possibilities for managers, analysts, or domain experts to be involved in the modelling of entity types, domain constraints and BR by using their own terminology. It is perhaps worthwhile to note that ORM derives from NIAM (Natural Language Information Analysis Method), which was explicitly designed to be a stepwise methodology arriving at "semantics" of a business application's data based on this kind of natural language communication.

ORM has an extensive and powerful graphical notation for representing a domain in a declarative manner as a network of elementary facts and their constraints. Elementary facts are represented in terms of object types that play roles. This graphical representation can be fairly easily re-verbalized into statements in pseudo natural language in a structured and fixed syntax. Therefore business rule modellers could represent a business policy either graphically or textually or both, which will in general improve, simplify, help to validate, and therefore speed up the modelling process.

Modelling BR requires an expressive modelling language in order to capture the business complexity. For this, ORM allows representing information structures in multiple ways as unary, binary, as well as n-ary facts. It has a sophisticated object type system that distinguishes between representations of lexical and non-lexical objects, and has strict “is a” relationships with "clean" multiple inheritance as in frame systems. ORM has an a priori given set of static and certain dynamic constraint types and derivation rules that turned out to be suitable and expressive enough to cover a significant part of the needs emerging from enterprise modelling. Such constraints and rules include classical ones such as uniqueness and mandatory roles, as well as less common ones such as subset, equality, ring, derivation, and/or stored rules, etc.

ORM has well-defined semantics, and the specified facts and constraints can easily be mapped into e.g. first order logic [75]. The finiteness and selection of the set of predetermined constraint types permitted the development of formal validation and consistency analysis tools that check the correctness and the consistency of specified BR [76].

The authors of [74] present a novel approach to support multilingual verbalization of logical theories, axiomatizations, and other specifications such as BR. This engineering solution is demonstrated with the Object Role Modelling language and the ontology engineering tool DogmaModeler, although its underlying principles can be reused with other conceptual models and formal

languages, such as Description Logics, to improve its understandability and usability by the domain expert. Lithuanian template for the verbalization of the ORM model is available in [73]

Structural business rule modelling issues are discussed in depth in a series of publications [77]-[91]. The authors of [72] propose an extension to the Object-Role Modelling approach to support formal declaration of dynamic rules. Dynamic rules differ from static rules by pertaining to properties of state transitions, rather than to the states themselves. In this [72], application of dynamic rules is restricted to so-called single-step transactions, with an old state (the input of the transaction) and a new state (the direct result of that transaction). Such restricted rules are easier to formulate (and enforce) than a constraint applying historically over all possible states. These dynamic rules are formulated in a syntax designed to be easily validated by non-technical domain experts.

Author of [26] analyzes UML data models from ORM perspective and identifies ORM constructs that can be transformed to UML. He also compares UML associations and related multiplicity constraints with ORM relationship types and related uniqueness, mandatory role and frequency constraints, discusses exclusion constraints, and summarizes how the two methods compare with respect to terms and notations for data structures and instances. Finally authors of [26], [27] draw to the conclusion that ORM set constraints are lost when transformed to UML. It is presented in the paper [27] how to compensate these defects by augmenting UML with concepts and techniques from the Object Role Modelling (ORM) approach. In general, set constraints in UML would normally be specified as textual constraints (in braced comments) or OCL should be used in more complicated cases.

The author of [29] provides a way to map ORM facts to UML constructs, leaving out the rest elements of the model. Although several papers [27], [28], [29] show how fragments of ORM model can be potentially encoded as fragments of UML models, a formal procedure for mapping onto logical schemas [28] that specifies how a target UML class diagram and OCL constraints can be created for any given ‘source’ ORM model is lacking. Both papers [29] and [26] propose to map ORM n-ary fact type to ternary associations in UML which is rarely supported in UML tools.

Despite of the fact that ORM has been used for three decades and now has industrial modelling tool support, it has no official, standard meta-model necessary for the MDA transformations. Authors of [93] discusses in their recent research to pave the way for a standard ORM metamodel. Our approach may be understood as one specific variant of metamodel proposed in [92]. The speciality of our presented approach is that differently from suggested in [92], where the ORM metamodel extends UML metamodel, we use independent ORM metamodel implemented in open source tool [9A].

1.3. Conclusions of the 1st chapter

The literature review clearly demonstrates that BR should be externalised and stored centrally in some particular repository. However, the approaches to enter BR to the repository differ significantly. Some authors propose to use formal languages to enter BR, but this limits the opportunity for business people to change them because the verbalization of formal languages is not mature enough.

Another option is to store BR in an informal way as a natural language statement or using very general templates. But in this case, BR cannot be automatically processed. This approach only ensures traceability of BR to implementation form. The existing natural language templates are not suitable for any real situation. In order to use them it is necessary to rephrase BR. There is no mechanism allowing the correct creation of automatically processable custom rephrased templates. Template application for practical domain has shown that standard templates should be rephrased in order to support the required functionality.

The analysis of the existing tools demonstrates that some of them have template specification possibilities. However, the specified templates are quite trivial and do not support the actual needs of business user. None of the analysed products provided functionality for the model-driven transformation of BR, as it is understood by OMG. In particular, it was not possible to control the transformation process, to access product metamodels and to specify transformation algorithm.

One more option to enter BR is a controlled natural language. Despite numerous works in this area, the usage of controlled natural languages is limited to the simple BR only. Besides, the majority of natural language processors recognize English as the main language for processing. As far as other languages are concerned, it is unlikely that there will be any significant progress in recognizing them in the nearest future. It particularly applies to the small nation languages, where the possible benefits and revenues are very low.

From the discussion presented above it is possible to draw a conclusion that user-defined templates for the BR authoring are a necessity. Furthermore, these BR templates should be clearly defined and automatically processed. The word ‘Processed’ in this thesis is perceived in a broad sense. The first and the obvious meaning is that BR, using custom natural language templates, could be entered to some particular repository. The second viable alternative is to use them for the transformation of BR to formal languages. One more alternative is to automatically propagate the BR, captured using custom templates, to the execution environment (e.g. executable code, security policies application engine, decision rules engine). It is worth mentioning that not only the new rules, but the changed rules should be processed as well.

2

Domain specific languages, transformations and tools

Tackling the problem of BR entering and transformation through templates, it is necessary to investigate the area of domain specific languages and transformations of these languages. This chapter presents the methods of specification and transformation of domain specific languages within the context of model-driven engineering.

2.1. Introduction to the 2nd chapter

Productivity gains brought by Domain Specific Languages (DSL) [136] have shown the importance of using appropriate modelling languages in the early phases of the software lifecycle. DSLs have triggered the new trend of language-centric methodologies [137], [138] and are based on the idea that the first step to efficiently treat a problem is to create or to customize a language that allows to describe the problem adequately. The precise definition of DSLs is in practice often a task for domain or methodology specialists who have only basic knowledge on language design.

Domain specific languages (DSL) have a crucial advantage over general purpose languages. They allow us to describe the elements, relations and constraints of a certain domain a lot more concise and compact. However, there

are many domains and technical spaces out there and we will see a lot of new DSLs coming up in the future, many of them to be used in software development projects. Especially these types of DSLs are predestined for a textual realization [131].

Papers which present graphical modelling tools usually do not compare their approaches to textual versions. They [implicitly assume that "graphical representations are better simply because they are graphical" which is questioned in [130]. Results of a case study in which concrete problems are modelled using textual and graphical notations are analysed. The authors argue that both text and graphics have their limitations and quality is not achieved automatically, although the authors reason that graphics have a higher potential of misleading the reader.

A modelling language (textual and visual) is usually defined in three major steps. The first one is to define concepts of the language, i.e. its vocabulary and taxonomy, as captured by its abstract syntax. Then, its semantics should be described in such a form that the concepts are clearly understood by the users of the language. Finally, it is necessary to precisely describe the notation, as captured by its concrete syntax.

The clear separation between abstract and concrete syntax is a technique to cope with the complexity of real-world language definitions since it allows to define the language concepts independently from their representation. For language designers, it is of primary importance to agree on language concepts and on the semantics of these concepts. The graphical representation of the concepts is often considered less important and is described in many language specifications only informally. However, an intuitive graphical representation is crucial for usability and indispensable for tool vendors who want to support a new modelling language with graphical editors, model animators, debuggers, etc. Sometimes, it is appropriate to have for one language more than one graphical representation, for instance when different stakeholders use the same language but need different views on the model. An example of such a language is ORM [8] that provides a graphical syntax intended for ontology engineers and a pseudo natural syntax intended for non-specialists.

Metamodelling is a widely used technique to capture the abstract syntax of a language. A well defined set of metamodelling constructs such as classes, associations, attributes, etc., complemented with a constraint language such as Object Constraint Language (OCL) allows one to define the concepts of the language and the relationships between them [139]. The abstract syntax is doubtlessly one of the most important parts of language definitions. Each sentence of the language can be represented without loss of semantic information as an instance of the metamodel. Such an instance can be represented in a standardized, textual format based on the general-purpose representation language XMI [140]. Model representations based on XMI are useful for

interchanging models between tools but humans need more comprehensible views on models.

Seeking to increase information systems design support level many researcher were seeking for the methods, which can enrich MDA methods. In Lithuania information systems development methods research is executed in Information systems department of Kaunas University of Technology, Vilnius Gediminas Technical University, in Mathematics and Informatics institute and other institutions. It is possible to find link of this thesis with componential development methods researched in Information systems department [46], integrated data and program component design [49], [50], [51], [52], [53], [54], transformation approaches [55], BR based methods [56], [57], [58], [59], [60], [61], [62]. Works of the institute of mathematics and informatics concentrates on the evaluation of formal methods [64]. The variety of the research in this area demonstrates that Lithuanian researches contribute to the information systems development methods. However no one of mentioned above authors concentrates on the problems analysed in this work.

2.2. Language design concepts

As it was mentioned before the main components of the language are its abstract syntax, concrete syntax and semantics. These concepts of language used in language centric methodologies although mostly overlap but in some sense differs from classical concepts of programming languages like notation, syntax, semantics and pragmatics [141].

The abstract syntax of a language describes the vocabulary of concepts provided by the language and how they may be combined to create models. It consists of a definition of the concepts, the relationships that exist between concepts and well-formedness rules that state how the concepts may be legally combined.

It is important to emphasise that a language's abstract syntax is independent of its concrete syntax and semantics. Abstract syntax deals solely with the form and structure of concepts in a language without any consideration given to their presentation or meaning.

All languages provide a notation that facilitates the presentation and construction of models or programs in the language. This notation is known as its concrete syntax. There are two main types of concrete syntax typically used by languages: textual syntax and visual syntax.

A textual syntax enables models or programs to be described in a structured textual form. A textual syntax can take many forms, but typically consists of a mixture of declarations, which declare specific objects and variables to be available, and expressions, which state properties relating to the declared objects

and variables.

An important advantage of textual syntaxes is their ability to capture complex expressions. However, beyond a certain number of lines, they become difficult to comprehend and manage.

A visual syntax presents a model or program in a diagrammatical form. A visual syntax consists of a number of graphical icons that represent views on an underlying model. A good example of a visual syntax is a class diagram, which provides graphical icons for class models. It is particularly good at presenting an overview of the relationships and concepts in a model.

The main benefit of a visual syntax is its ability to express large amounts of detail in an intuitive and understandable form. Its obvious weakness is that only certain levels of detail can be expressed beyond which it becomes overly complex and incomprehensible.

In practice, utilising a mixture of diagrammatical and textual syntaxes gains the benefits of both forms of representation. Thus, a language will often use visual notations to present a higher level view of the model, whilst textual syntax will be used to capture detailed properties [126].

An abstract syntax conveys little information about what the concepts in a language actually mean. Therefore, additional information is needed in order to capture the semantics of a language. Defining semantics for a language is important in order to be clear about what the language represents and means. Otherwise, assumptions may be made about the language that leads to its incorrect use. For instance, although we may have an intuitive understanding of what is meant by a state machine, it is likely that the detailed semantics of the language will be open to misinterpretation if they are not defined precisely. What exactly is a state? What does it mean for transition to occur? What happens if two transitions leave the same state? Which will be chosen? All these questions should be captured by the semantics of the language.

There are many different approaches to describing the semantics of languages in a metamodel. All the approaches are motivated by approaches to defining semantics that have widely been applied in programming language domains. The main difference is that metamodels are used to express the semantic definitions.

The approaches include:

- Translational semantics. Translating from concepts in one language into concepts in another language that have a precise semantics.
- Operational semantics. Modelling the operational behaviour of language concepts.
- Extensional semantics. Extending the semantics of existing language concepts.
- Denotational semantics. Modelling the mapping to semantic domain concepts.

2.3. Abstract syntax specification techniques

Abstract syntax of the language is specified using metamodel and metamodelling language. Metamodelling language places requirements on there being specific metamodelling architecture. This architecture provides a framework within which some key features of a metamodel can be realised. The traditional metamodel architecture, proposed by the original OMG MOF 1.X standards is based on 4 distinct meta-levels. These are as follows:

M0 contains the data of the application (for example, the instances populating an object-oriented system at run time, or rows in relational database tables).

M1 contains the application: the classes of an object-oriented system, or the table definitions of a relational database. This is the level at which application modelling takes place (the type or model level).

M2 contains the metamodel that captures the language: for example, UML elements such as Class, Attribute, and Operation. This is the level at which tools operate (the metamodel or architectural level).

M3 the meta-metamodel that describes the properties of all metamodels can exhibit. On this level only one meta-metamodelling language is defined (i.e. MOF).

This is an architecture that from the basis of MDA. Another possible architecture is Eclipse Modelling Framework (EMF), which is different from MDA just in using ECore on the M0 layer instead of MOF.

Although the 4-layer metamodel is widely cited, its use of numbering can be confusing. An alterative architecture is the golden braid architecture [142]. This architecture emphasises the fact that metamodels, models and instances are all relative concepts based on the fundamental property of instantiation.

The idea was first developed in LOOPS (the early Lisp Object Oriented Programming System, and then became a feature of both ObjVLisp [143] and also CLOS (the Common Lisp Object System).

Emfatic [145] Language for Eclipse Modelling Framework (EMF) Development is a language for representing EMF ECore models in textual form. The advantage of Emfatic is that it represents an entire ECore model in a single source file and it uses a Java-like syntax familiar to many programmers. One more textual language for defining metamodels is KM3 [145] (Kernel MetaMetaModell). As a metametamodell, KM3 is simpler than MOF 1.4, MOF 2.0 and ECore. It contains only 14 classes whereas, for instance, ECORE has 18 classes and MOF 1.4 has 28 classes. Only the core concepts of these other metametamodells are available in KM3.

[126] proposes the five levels of maturity of language metamodel:

Level 1 is the lowest level. A simple abstract syntax model must be defined, which has not been checked in a tool. The semantics of the language it defines

will be informal and incomplete and there will be few, if any, well-formed rules.

Level 2 at this level, the abstract syntax model will be relatively complete. A significant number of well-formedness rules will have been defined, and some or the entire model will have been checked in a tool. Snapshots of the abstract syntax model will have been constructed and used to validate its correctness. The semantics will still be informally defined. However, there may be more in the way of analysis of the language semantics.

Level 3 The abstract syntax model will be completely tried and tested. Concrete syntax will have been defined for the language, but will only have been partially formalised. Typically, the concrete syntax will be described in terms of informal examples of the concrete syntax, as opposed to a precise concrete syntax model. Some consideration will have been given to the extensibility of the language architecture, but it will not be formalised or tested.

Level 4 at level 4, the concrete syntax of the language will have been formalised and tested. Users will be able to create models either visually and textually and check that they result in a valid instance of the abstract syntax model. The language architecture will have been refactored to facilitate reuse and extensibility. Models of semantics will have begun to appear.

Level 5 is the topmost level. All aspects of the language will have been modelled, including its semantics. The semantic model will be executable, enabling users of the language to perform semantically rich operations on models written in the language, such as simulation, evaluation and execution. The language architecture will support good levels of reuse; it will have been proven to do so through real examples. Critically, the completed metamodel will not be reliant on any external technology – it will be a fully platform independent and self contained definition of the language that can be used ‘as is’ to generate or instantiate tools.

The authors of [126] notice, that most of the metamodels do not achieve a level greater than 2. Even international standards such as UML do not exceed level 3.

2.4. Concrete syntax specification techniques

Concrete syntaxes are traditionally expressed with rules, conforming to EBNF-like grammars, which can be processed by compiler compilers (e.g. ANTLR [158]) to generate parsers. Unfortunately, these generated parsers produce concrete syntax trees, leaving a gap with the abstract syntax defined by metamodels, and further ad-hoc hand-coding is required.

The Eclipse platform [132] is an ideal target for this kind of approach. First, the platform supports the user by a full-functional Java-IDE including among others an incremental compiler. Second, the user is supported by various build

and version management utilities which are essential for the client software development. Third, due to Eclipse's extensible nature, it allows integrating new plugins for DSLs. The downside of this approach is, that due to the extensibility and power of the Eclipse platform, the provided interfaces for DSL-tooling are rather complex and still lack stability. This means a lot of experience is needed to develop and maintain a sophisticated language support [133].

Being able to parse a text and transform it into a model, or being able to generate text from a model are concerns that are being paid more and more attention in industry. For instance Microsoft with the DSL Tools [125] or Xactium with XMF Mosaic [126] in the domain-specific language engineering community, are two industrial solutions for language engineering that involve specifications used for the generation of tools such as parsers and editors. A new OMG standard, MOF2Text [127], is also being developed regarding concrete-to-abstract mapping. Although this paper focuses on textual concrete syntaxes, it is worth noticing that there are also ongoing researches about modelling graphical concrete syntax [129], [128].

The most noticeable and well supported tool is openArchitectureWare's xText [5] framework that allows one to create a DSL infrastructure (including parser and Eclipse-based editor with syntax highlighting, code completion and error markers) by providing rather simple grammar-based definition.

The main idea is to create a grammar language that allows building not only the parser but also a text-to-AST transformation. The AST meta-model is described in EMF terms. Main principle is that a grammar rule having non-terminal X on the left side defines an AST class X. The right side of such a rule refers to other rules by assigning them to X's features. This implicitly defines features' types. Nothing but such a grammar has to be provided to define AST meta-model and text-to-AST transformation.

This tool is syntax-centric. Its main goal is to build parser that produces AST, not target model. So it does nothing about AST-to-target transformations and lookup (xText allows to perform some semantic analysis but only through constraints checking, it also has connection with xTend [152] transformation language, but all the transformations must be written manually) and the concrete syntax grammar is the main artefact they operate on.

XText enables text-to-model transformations and was also submitted as a proposal for TMF. In contrast to TCS the metamodel is derived from an XText grammar file. The grammar describes the syntax of the DSL and is then transformed to an ANTLR grammar and an ECore metamodel. The parser generated of the ANTLR grammar creates model elements conforming to the metamodel. The generated metamodel corresponds to an AST specification for the DSL. This approach suffers of the inability to create a custom metamodel. The authors propose to transform from the generated metamodel into a "real" metamodel. This helps separating concerns (parsing, linking) but additional effort

is needed compared to TCS. Furthermore the XText language itself is limited, so only simple DSLs are possible. The XText grammar which is similar to BNF consists of a set of rules. Each rule can be composed out of a combination of three token types (keyword, identifier or string) and references to other rules. Modifiers such as multiplicity, optionality or alternation are available. How a string token or a identifier token is build, is not customizable, nevertheless it is possible to define a simple custom token type for strings. XText generates an Eclipse based editor including an outline view, syntax highlighting and checking for a DSL.

The MontiCore framework [134] can be used for the agile development of textual languages, in particular domain-specific modelling languages (DSMLs). In order to reduce the abovementioned redundancy, one of the main design goals of the underlying grammar format was to provide a single language for specifying concrete as well as abstract syntax in a single definition. Associations, compositions, and inheritance as known from meta-modelling can directly be specified in this format. Such a language definition can be mapped to an object oriented programming language where the each production is mapped to a class with strongly typed attributes. A parser is generated to create instances of the abstract syntax from a textual representation.

The work [149] considers the concrete syntax facet of DSLs, when it is textual. The objective is to enable translation from text based DSL sentences to their equivalent model representation, and vice-versa. Such a feature is essential to the development of tools for text-based DSLs. Both model-to-text and text-to-model translations can be performed using a single specification. A grammar can thus be generated from both the metamodel and the TCS model to perform text-to-model translation. Grammar annotations that build the model while parsing can be automatically generated. Model-to-text translation can also be performed with the same information. To this end, a generic interpreter has been defined to traverse the model following the syntactical path specified in TCS. Keywords and symbols are written alongside model information.

The work presented in [150] proposes an approach for defining visual syntaxes for modelling languages. It is based on defining a set of mediator classes that relate language metamodel elements and the classes for visual elements (boxes, arrows, etc.).

The Textual Concrete Syntax Specification Language (TCSSL) [73] is a metamodel-aware specification language for grammars. It promises a bidirectional mapping text-to-model mapping. A new grammar language consisting of three different types of rules is proposed:

- Simple rules are EBNF rules instantiating a model element.
- Seek rules are used to resolve references by looking for existing model elements matching a given condition.

- Singleton rules work like simple rules but do not instantiate a new model element if an instance already exists.

In addition several elements a rule can contain like alternatives, actions with guards, model expression (inside a language such as Kermeta or MTF), model queries (side effect free expressions), sub rule calls or multiplicities are defined.

Currently a number of different frameworks for model-driven DSL implementation are developed. While documentation for XText, TCS and TCSSL is available to some extend and this is not the case for some newer frameworks like IBM SAFARI [156] or TEF [157].

The initial presentation of SAFARI was at the EclipseCon 2006 [155] but there is no documentation and no public binary/source code available. SAFARI allows the generation of Eclipse based DSL environments offering a rich user experience such as syntax highlighting, source-text folding, hyperlink detection, content outlining, content assist, hover annotations, hover help, parsing, and project building. The DSL has to be specified using the LPG [154] parser generator.

The Textual Editing Framework (TEF) [157] is announced by the Humbolt-University of Berlin and offers similar feature as SAFARI. Yet only a website and the source code of an unusable alpha version is available at the moment. TEF is also based on Eclipse and allows the generation of convenient DSL editor.

The set of BR for one particular domain can be treated as an example of a domain specific language. However template specification and the language used for the template specification can be treated as metadomain-specific language. Existing frameworks for the development of concrete syntax address development of DSL not meta-DSLs and are not suitable for definition of template language. It is not feasible to use such framework for the specification of BR templates as they require comprehensive understanding of metamodelling and language development practices.

At the same time the DSL frameworks hardly could be applied for the specification of BR templates as they require comprehensive understanding of metamodelling and language development practices what is often a lack of a domain experts. For the specification of BR templates much less functionality is needed comparing to the development of DSL.

2.5. Transformation definition

Kleppe et al. [1] provide the following definition of model transformation. A transformation is the automatic generation of a target model from a source model, according to a transformation definition. A transformation definition is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language. A

transformation rule is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language. [147] suggest that this should be generalised, in that a model transformation should also be possible with multiple source models and/or multiple target models.

For instance, the QVT (Query\View\Transformation) [148] standard specifies a language in which one is able to express transformation definitions that consist of a number of mapping rules. The mapping rules may be combined by internal (or fine-grained) composition of transformations. Although existence of QVT standard the transformation languages that were used as a proposal for the standard continue to evolve almost independently. Detailed feature based survey of model transformation approaches is provided in [164]. We will give a brief overview of the languages that have affected our research.

Thales QVT proposal is a transformation language called TRL (Transformation Language) [161]. The language can be used for querying models as well as for transforming models. It reuses and extends the selection and filtering capabilities already available in OCL 2.0. The type of the data returned by a query may be a composite type (collection types, tuple types, dictionary types) or maybe provided by a metamodel (in which case the query is a special kind of transformation program).

The ATL [162] is a QVT-based transformation language, developed by the INRIA Atlas team. An implementation of ATL is currently available as open source under an Eclipse project called Generative Model Transformer (GMT) project. It is developed as a set of Eclipse plugins and works as a development IDE for transformations, with execution and debugging. Currently integrates with EMF and MDR.

Transformation programs written in ATL are inherently unidirectional. Source models, which are only navigable (e.g. read-only), and target models, which are not navigable (e.g. write-only), are clearly identified at development time. ATL offers two imperative constructs: called rule and action block. A called rule is explicitly called, like a procedure, but its body may be composed of a declarative target pattern. Matched rules and called rules may be used together in a single transformation program. Action blocks are sequences of imperative instructions that can be used in either matched or called rules. The recommended style is declarative (e.g. no called rules and no action blocks). Imperative style should only be used when no declarative language construct provides the capabilities required by a particular case.

MOdel transformation Language (MOLA) is combination of traditional structured programming in a graphical form with pattern-based rules. The loop concepts enable the iterative style for transformation definitions, while other languages rely on recursion [163].

OpenArchitectureWare (oAW) is a framework consisting of a set of modules

that assist model driven development. The manufacturers themselves describe it as a "tool for building MDSD/MDA tools" [165]. It is completely based on Eclipse and is part of the Eclipse Generating Modelling Technologies (GMT) project. Strong parts of oAW are on one hand the integrated workflow engine, which executes self-defined model transformation or code generation workflows, or processes one of the numbers of workflows already included. These contain various solutions for reading, instantiation and validation of models, model transformation or generation of code.

On the other hand, oAW currently supports probably the largest number of modelformats as input (namely EMF-models, UML models from Magic Draw, Poseidon, Rational Rose XDE and many, many others), which can even be expanded creating a compatible instantiator. The generated output in oAW is defined using the proprietary template language Xpand. Similarly, proper model-to-model transformations are achieved, by executing a user-defined workflow, written in the xTend language. It is closely related to Eclipse EMF, since they both share similar functionalities [165].

2.6. Conclusions of the 2nd chapter

The overview of the language design methods demonstrates that BR templates can be treated as DSL. Moreover, DSL development methods can be applied to the development of BR templates specification. However, the methods and frameworks designated to the DSL development are too complicated to be effectively used by domain experts responsible for the BR. They require enormous knowledge of language development concepts and techniques. Furthermore, the development of full scale DSL is not appropriate for specifying 2–20 BR templates.

Therefore, it is necessary to develop meta-DSL language for the specification of BR templates. This language can be treated as a framework for the development of DSL of one particular type – BR templates. Because of its narrow specialisation it is believed that this language will be easier to use than a general DSL development framework.

The first idea was to use DSL development framework for the development of BR templates specification language. Whereas further investigation of this idea and the existing frameworks and tools demonstrated that these frameworks, though intended for DSL development (as BR template specification language was considered to be) were not suitable for the development of DSL frameworks (as BR template specification language actually is). Therefore, the classical methods of concrete language syntax specification were used (ANTLR, in particular).

The first version of abstract syntax for BR template specification language

was developed using a repository of MOF and MDR models. However, in the course of research, it became obvious that MDR repository would not evolve into a stable platform. As a result, upon the availability of a more advanced model repository EMF, abstract syntax was transformed to the ECORE as a more stable one.

As for the majority of DSL, which are developed to be transformed to some general problem language, translational semantics is the most appropriate approach for the specification of the semantics of BR templates specification language. Translational semantics for the BR templates specification language could be expressed by defining the mapping to SBVR. The semantics of BR is intended to be specified using transformation specification language.

Over the period of more than six years of research into BR transformation, we have used 3 different transformation languages. This can be explained by the fact that these languages up to now are research languages. The tools supporting these languages are not user friendly and are unstable. TRL language implementation was the first tool we have success with. It was used for the transformation of OCL to SQL (2002–2004). ATL language and tool were used for defining the transformation of BRTL/ORM to UML/OCL (2004–2006). And, finally, upon the appearance of the implementation of transformation language xPand (part of openArchitectureWare), it was used for the experiment with the financial reporting domain that will be described below.

3

The business rules templates approach

In this chapter the BR templates specification approach and the methods to design BR templates are presented. The last section of the chapter describes the implementation of the BRTL approach in BRidgeIT tool.

3.1. Introduction to the 3rd chapter

The usual form for BRs to appear in the enterprise is to be buried in the numerous guidelines, policies and other documents. But BRs specified only in natural language are largely inaccessible to computer programs, decision making, quality assurance initiatives and management of the enterprise. In contrast, using BR templates (the kind of structured natural language with defined structure and empty slots to be filled in later) users feel comfortable as if they were working with natural language statements. At the same time BR templates are not limited by the shortcomings of the natural language. In particular it is very difficult to define BR templates acceptable for each enterprise. Especially it becomes crucial in the world wide context when adaptation to different cultures is needed. Therefore it is necessary to allow users to specify BR templates themselves.

BRs have to be in consistency with other enterprise models. Data and fact models are most often referred by BR. Differently from BR for data model it is

natural to have graphical notation. Opportunity to express fact model in natural language relating it with BR templates would be a beneficial advantage of the presented model. One of the fact based notations which is transformable to natural language is object role modelling ORM. A number of rules, in particular, schema rules, as they are defined in, can be expressed in ORM. BR templates can complement ORM with instance level rules. In this case both ORM and BR templates will benefit from formal integration.

There are several possible integration levels of ORM and BR templates [166]:

- technical integration of tools considering APIs and tools interfaces,
- conceptual integration of metamodels of description formalisms combined with hard and soft constraints,
- semantically integration of semantics of description techniques using a common semantic model,
- methodical integration by an embedding in the development process.

In this thesis conceptual integration of metamodels for ORM and BR templates was selected as the most appropriate.

The organization of BR template metamodell is presented in figure 3.1 Metamodel is constructed from three packages. The main aim of such organisation is to provide metamodel integrity with object role modelling. So BR resulting from such metamodel can seamlessly refer to corresponding ORM model. The package "Templates" contains metaclasses which enable creation of templates from the BR. UML 2.0 template creation principles are adopted for business rule. Lastly the most important package is "BRT template". It contains metaclasses which describe possible parts of the BR.

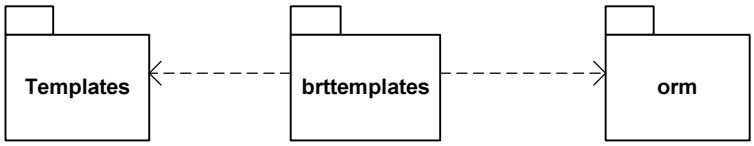

Figure 3.1. *Packages used by business rule template metamodel*

3.2. ORM elements

ORM is primarily a method for conceptual fact modelling. In Europe the method is often called NIAM (Natural Language Information Analysis Method). ORM is so called because it pictures the world in terms of objects (entities or values) that play roles (parts in relationships). In contrast to other modelling such as Entity-Relationship (ER) and Object-Oriented (OO) approaches, ORM makes no explicit use of attributes.

As it was mentioned before ORM metamodel is necessary in order to make ORM available for transformations. We start developing formal definition of ORM from defining ORM Model metaclass which aggregates all elements of ORM model. All basic model elements are inherited from abstract "ModelElement" metaclass. Figure 3.2 gives an overview of model elements available in ORM. After defining basic elements we proceed with relationships between elements (Figure 3.3). There are two types of constraints in ORM one for roles and another for the values of object types. Roles constraints have variety of specialised constraints (Figure 3.4). Relation of ORM elements to metaclasses is detailed in Table 3.1

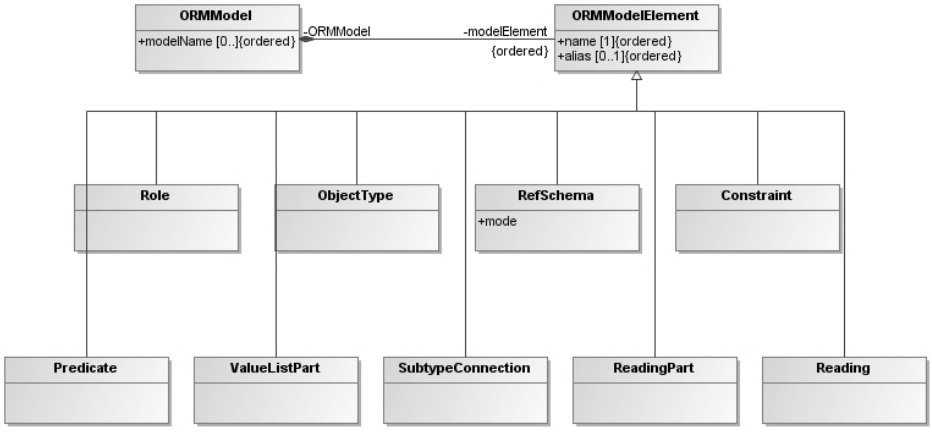

Figure 3.2. *Top elements of ORM metamodel*

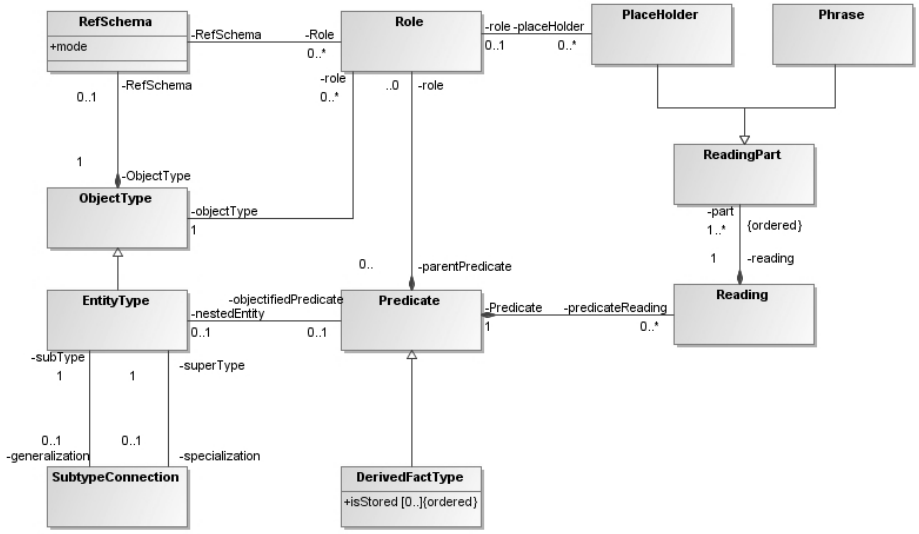

Figure 3.3. *ORM metamodel*

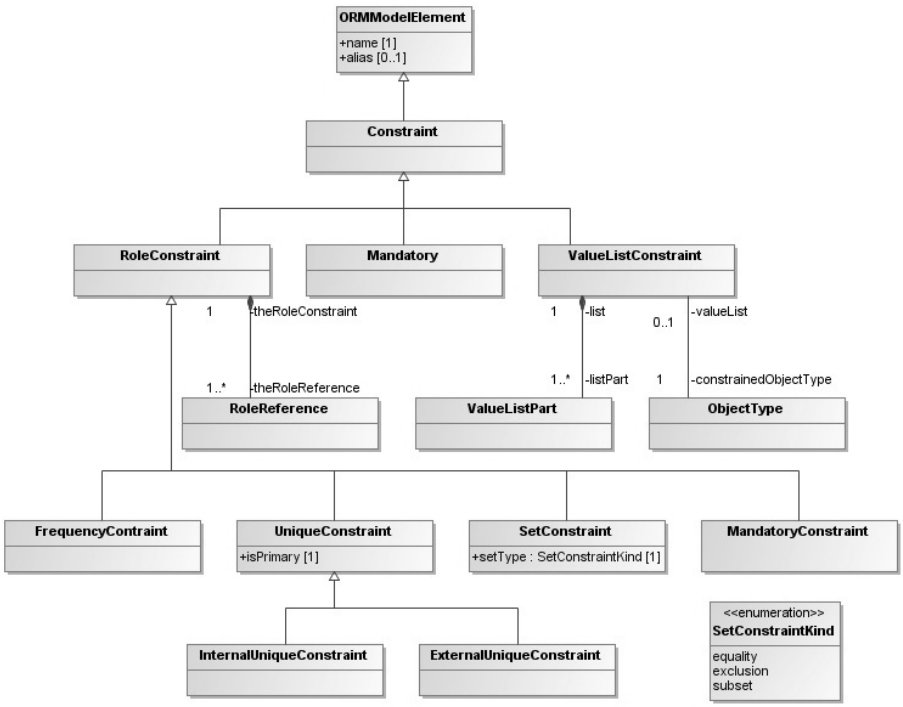

Figure 3.4. *Constraints of ORM metamodel*

Table 3.1 Mapping *of ORM elements to metaclasses*

ORM modelling element	**Metaclass**	**Description**
	ORMmodel	Represent ORM model
	ORMModelElement	Denotes general ORM model element. Abstract. Name attribute is inherited by each model element
Object type	ObjectType	Represents entity type
Value type	ValueType	Denotes a lexical object type
Reference scheme	RefSchema	Indicates how each instance of the entity type may be mapped via predicates to a combination of one or more values
Reference mode	RefSchema.mode	Indicates how values relate to the entities (e.g. plus sign “+”

ORM modelling element	Metaclass	Description
		may be added if the values are numeric)
Predicate	Predicate	Denotes n-ary predicate
Role	Role	ORM role, each role have player
Object holes	PlaceHolder	Denote object holes of ORM predicate
Internal uniqueness constraints	InternalUniqueness	Internal uniqueness constraints are placed over one or more roles in a predicate to declare that instances for that role (combination) in the relationship type population must be unique A predicate may have one or more uniqueness constraints, at most one of which may be declared primary by adding a "P"
External uniqueness constraint	ExternalUniqueness	External uniqueness constraint may be applied to two or more roles from different predicates by connecting to them with dotted lines
Objectified predicate	Association: Predicate-ObjectType	Object type made from relation type
Mandatory role constraint	Mandatory	Mandatory role constraint declares that every instance in the population of the role's object type must play that role
Disjunctive mandatory constraint	DisjunctiveMandatory	Disjunctive mandatory constraint applied to two or more roles to indicate that all instances of the object type population must play at least one of those roles.
Value constraints	ValueConstraint	To restrict an object type's population to a given list, the relevant values may be listed in braces

ORM modelling element	Metaclass	Description
Set comparison constraints	SetConstraint	Set comparison constraints may only be applied between compatible role sequences
Subset constraint	SetConstraintKind. Subset	Subset constraint restricting the population of the first sequence to be a subset of the second
Equality constraint	SetConstraintKind. Equality	Equality constraint indicate that the populations must be equal
Exclusion constraint	SetConstraintKind. Exclusion	Exclusion constraint indicate that the populations are mutually exclusive
Subtype	SubTypeConnection	Subtype indicates that the first object type is a (proper) subtype of the other
Frequency constraint	FrequencyConstraint	Frequency constraint applied to a sequence of one or more roles, these indicate that instances that play those roles must do so exactly n times, between n and m times, or at least n times
Ring constraint	RingConstraint	Ring constraint may be applied to a pair of roles played by the same host type. These indicate that the binary relation formed by the role population must be irreflexive (ir), intransitive (it), acyclic (ac), asymmetric (as), antisymmetric (ans) or symmetric (sym). Precise ring constraint type is saved in the name attribute of metaclass.

3.3. Elements of business rules templates language

The package “BRT template” contains “BRuleExp” metaclass which aggregates other elements of the BR (Figure 3.5.). Two metaclasses

RulePartExpComposite and RulePartExpAtomic are derived from RulePartExp. The former is necessary to enable internal structuring of BR and definition of complex rule template expressions. It plays a major role when it is necessary to present optional or repeating parts of the BR template. RulePartExpAtomic is an abstract metaclass from which others main part of BR are derived. These metaclasses are described more thoroughly:

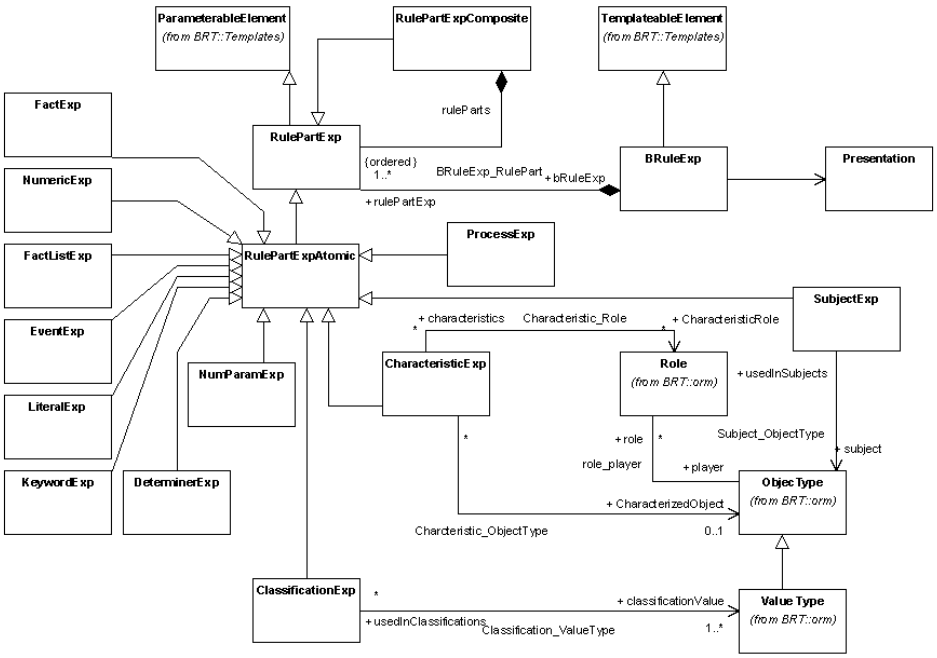

Figure 3.5. *Business rule template metamodel fragment*

RulePartExp – abstract Metaclass representing all possible template expressions

BRuleExp (BR) – BR expression, metaclass aggregating BR parts;

DeterminerExp (DE) – the determiner for the subject; from the following, the one that makes the best business sense in the statement. Possible values for the name attribute: each, any, etc.;

SubjectExp (SE) – a recognizable business entity.

SubjectExp metaclass contains reference to the ObjectType metaclass from ORM metamodel;

CharacteristicExp (CE) – the business behaviour that must take place or a

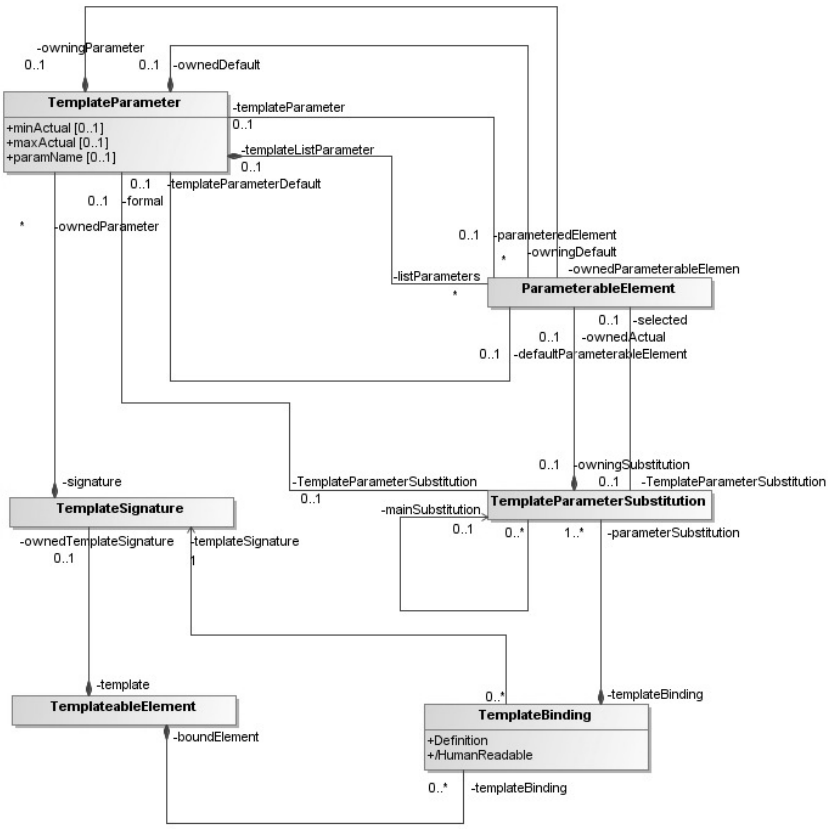

Figure 3.6. *Template metamodel fragment*

relationship that must be enforced. CharacteristicExp metaclass refers to the Role and ObjectType from the ORM metamodel;

FactExp (FE) – a relationship between terms identifiable in the fact model, together with defined constants. The relationship may be qualified by other descriptive elements in order to specify the applicability of the rule precisely.

LiteralExp (LE) – instance of metaclass LiteralExp contains character string in the name attribute;

NumericExp (NE) – instance of metaclass NumericExp contains numeric value in the name attribute;

NumParamExp (NPE) – numeric parameters;

ClassificationExp (ClE) – This typically defines either the value of an attribute, perhaps called "state" or something similar, or a subset of the objects in an existing class. May contain reference to the value entity from ORM model storing “state” value;

KeywordExp (KE) Represents keyword expression of the business rule

template;

Three metaclasses from ORM metamodel are referred in BR template metamodel. Subject expression refers to the ORM Object type. Usually subject expression is mandatory expression in BR templates. Classification expression refers to ORM Value type to denote where actually the classification value is stored, however this reference is optional and the value can be stored by the instance of classification expression. Characteristic expression refers to the Role metaclass from ORM metamodel.

The content of Templates package is presented in more details in figure 3.6. The Templates package specifies how BRuleExp can be parameterized with RulePartExp template parameters. The package introduces mechanisms for defining templates, template parameters and bound elements in general, and the specialization of these for BRuleExp and RulePartExp. The metaclasses of this package:

ParameterableElement – a parameterable element is an element that can be exposed as a formal template parameter for a template, or specified as an actual template in a binding of a template;

TemplateableElement – a templateable element is an element that can optionally be defined as a template and bound to other templates;

TemplateBinding – a template binding represents a relationship between a templateable element and a template. A template binding specifies the substitutions of actual parameters for the formal parameters of the template.

TemplateParameter – a template parameter exposes a parameterable element as a formal template parameter of a template;

TemplateParameterSubstitution – a template parameter substitution relates the actual parameter(s) with the formal template parameter within the context of a TemplateBinding;

TemplateSignature – a template signature bundles the set of formal template parameter for a templated element.

A command is the basic instruction that a script file contains. Some commands require parameters that further define what the command should do. An expression is a combination of operators and arguments that create a result. Expressions can be used as values in any command. Examples of expressions include arithmetic, relational comparisons, and string concatenations.

3.4. Precise notation of the business rules templates language

Precise notation of the BR template is used for the definition of abstract and domain-specific templates during the design of BR templates as it will be described in the following section. BR templates are not strictly bound to the one

particular notation. On the contrary like each model driven knowledge representation approach it can have several notations. In this paper we present one of the possible notations, which will be used for the examples in the following chapter.

Each business rule template is constructed from well defined parts called template expressions. Template expressions are separated from each other by template expression metaclass name or short notation keywords and a semicolon (e.g. LiteralExp: or LE:).

Template expression can have several representations even within the same language (e.g. the verb “to be” may have the presentation “is” or “was”). Template expression from its presentation is separated by the white space and each presentation is separated from each other by “|” (e.g. LiteralExp: be|is|was).

Three types of template expressions are distinguished: atomic, composite, reference expressions. Atomic template expressions do not contain other template expressions (e.g. LiteralExp:, KeywordExp:, NumericExp: etc.).

Reference template expressions contain references to the other models (e.g. BPMN, ORM). Referenced elements are written after the name of reference template expression (e.g. SubjectExp: ObjectType: Conference Paper).

Composite template expressions do not have special keyword. Different types of brackets are used instead. Parentheses “(” and “)” are used to enclose a group of mandatory template expressions. Square brackets “[” and “]” are used to enclose optional composite template expression. The number after the second angle bracket denotes the cardinality of the composite template. No number or the asterisk “*” sign denotes infinity. Vertical bars “|” are used to separate alternative template expressions of any type.

Each template expression can be parameterized. Composite template expressions are parameterized by the Template parameters are marked by question mark after the keyword of the template. It is possible to omit question mark in this case template expression without presentation will be treated as parameterized template expression. It is possible to add presentation after the parameterization question mark in order to provide parameter example or a default value.

Concrete syntax of the language:

```
bRuleExp   :(rulePartExp)*;

rulePartExp    : (rulePartExpComposite
  |rulePartExpAtomic
  )*;
rulePartExpComposite  :rulePartExpCompositeMandatory
  | rulePartExpCompositeOptional;

rulePartExpCompositeMandatory     :"("rulePartExp("|"rulePartExp)*")";
```

```
rulePartExpCompositeOptional  :"["rulePartExp("|"rulePartExp)*"]";

factExp :(rulePartExp)* ;
rulePartExpAtomic  :(numericExp
  | literalExp
  | eventExp
  | subjectExp
  | keywordExp
  | characteristicExp
  | classificationExp
  | processExp
| numParamExp
  | determinerExp) ;

subjectExp :("SubjectExp"|"SE")":"
  (IDENT|nameParamExp);

characteristicExp:("CharacteristicExp"|"CE")":"
  (IDENT|nameParamExp)?(".")?
  (IDENT|nameParamExp)?;

classificationExp :("ClassificationExp"|"ClE")":"
  (IDENT|nameParamExp);

eventExp  :("EventExp"|"EE")":"
  (STRING|paramExp);

numericExp  :("NumericExp"|"NE")":"
  (INT|paramExp)?;

literalExp  :("LiteralExp"|"LE")":"
  (STRING|paramExp);

keywordExp  :("KeywordExp"|"KE")":"
  (STRING|paramExp);

processExp:("ProcessExp"|"PE")":"
  (STRING|paramExp);

numParamExp:("NumParamExp"|"NPE")":"
  (STRING|paramExp);
```

```
determinerExp:("DeterminerExp"|"DE")":"
  (STRING|paramExp);

paramExp  :("?"("("defaultExp("|"defaultExp)*")")?)":" ;

nameParamExp: ("?"("("nameDefaultExp("|"nameDefaultExp)*")")?)":";

defaultExp :(STRING);

nameDefaultExp  :(IDENT);
```

3.5. Designing business rules templates

In this thesis we propose three level presentations of templates. The first level (abstract templates) is the most abstract level and depicts only the essential spirit of the template, excluding inessential application-domain-specific details. The second level (domain specific templates) contains references to the domain fact and process models, but it still contains empty slots. The final level templates are created when user provides necessary values to the empty slots hence forming resulting BR.

Our process for designing BR templates consists of the following steps:

1. The first one is when BR patterns that are mostly relative to the domain of interest are identified. There are many sources for the mining of such patterns. For example existing BR specified in informal manner in enterprise documents. Existing BR templates (e.g. [2], [4], [5]) and patterns can form the basis and provide inspiration for the creation of the patterns. Resulting set of BR patterns is directly influenced by the purpose of specifying BR. The result is the set of BR patterns in semiformal notation. For example Morgan BR basic constraint pattern:

<det> <subject> (must | should) [not] <characteristic> [(if | unless) <fact>].

2. Similar patterns are then grouped, integrated and specified in abstract templates. In this stage almost all parts of the template are parameterised. The template relation with the domain is very weak and it can be used without modification in the other domains. The result of this step is the set of abstract BR templates. For example Morgan business constraint pattern result in the following abstract template:

DeterminerExp: SubjectExp:
(KeywordExp: must | KeywordExp: should)
[KeywordExp: not]

CharacteristicExp:
[(KeywordExp: if | KeywordExp: unless)
FactExp:].

3. Then abstract templates are adapted for the particular domain of interest. The main task within this step is to determine which parts of the template are persistent and which are temporal with respect to the application of the BR resulting from the template. But what counts as permanent and what as short-lived is itself dependent on specification interests and purposes, both theoretical and practical. An analysis of the kind of changes that are of interest should determine, even if only roughly, a temporal interval or length of time as its focus or window. Template parts that are apt to change within that time interval are temporal. Those that are likely to hold through the designed interval are with respect to this task permanent. The values from the domain fact and process model are provided for the permanent parts. Default values and values to select from are defined for temporal template properties. After this step the template is usually bound to the domain models and can not be used separately. For example basic constraint domain-specific template from the conference organization domain is bound to ORM model presented in Fig. 3.7:

DE: Each SE: ObjectType: Conference Paper
KE: must CE: Role: be accepted|is accepted
KE: if CE: ObjectType: overall evaluation
KE: greater than NE:?
User view of the template:
Each Conference Paper must be accepted to conference proceedings if overall evaluation is greater than __

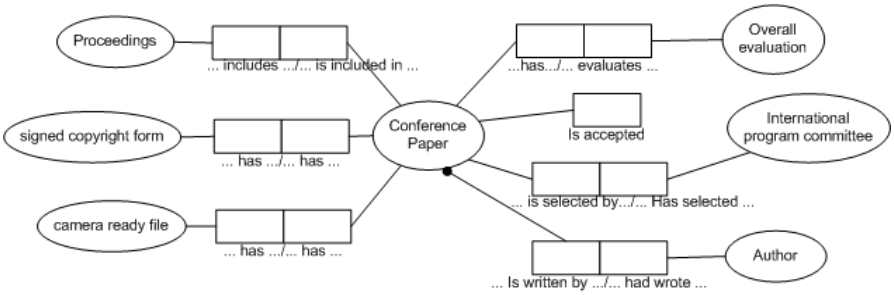

Figure 3.7. *ORM Conference model*

4. During the last step parameter values are provided for the template and business rule is created. It is still necessary to maintain BR relationship with the template for the purpose of the future change of BR parameters, effective search and presentation of the BR. Resulting BR:

Each Conference Paper must be accepted to conference proceedings if overall evaluation is greater than 5.

3.6. BRidgeIT tool

The proposed BR specification approach is implemented in the tool BRidgeIT. BRidgeIT is a (slightly permuted) acronym for BR bridge to IT through Templates. It is a system for specifying BR using templates and transforming them to other forms (e.g. OCL, SQL).

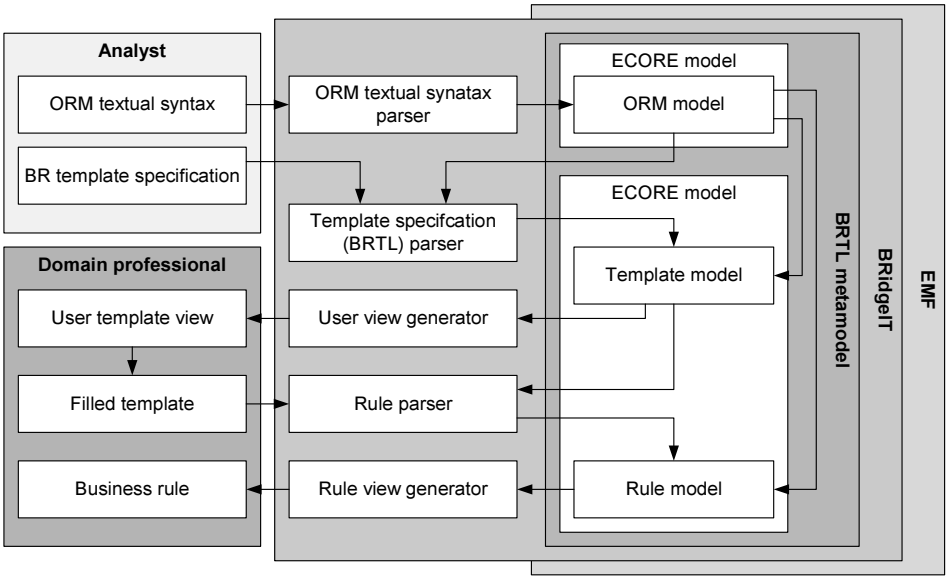

Figure 3.8. *BRidgeIT tool architecture*

The architecture of the BRidgeIT is presented in Figure 3.8. The tool is implemented as an Eclipse plug-in, however it can be used stand alone. It employs Eclipse EMF models repository for the storage ORM, BR templates specification and BR models and metamodels. Each model stored in the tool corresponds to BRTL approach metamodel.

In order to enter ORM into the BRidgeIT special textual notation is used (the syntax is described in the Appendix A). Graphical ORM user interface is planned for the development. Additionally BR templates can be used for the definition of ORM fact model.

After the model is defined the analyst defines BR templates using specification approach BRTL as it was described in the previous chapters. The system parses specification and provides user view of the template. The user fills in the gaps in the template and creates business rule.

3.7. Conclusions of the 3rd chapter

In this chapter, a metamodel for the Object Role Modelling (ORM) language and BR template are presented. The application of the metamodel is demonstrated on the concrete BR templates statements.

The metamodel delivers a more precise and detailed view of the ORM and BR templates. As a result of using only the well-known modelling concepts of UML which are compliant with Meta Object Facility (MOF), the metamodel can easily be read by everybody familiar with UML.

The provided integrated metamodel allows precise and consistent with data model definition of BR. It should be noted that, though the metamodel provides a precise description of the abstract syntax of ORM and BRT, it does not define any specific template. BR templates allow abstracting the complexity of the future realizations of the BR. In particular the template hides possible complex interface of the target system. The BR created according to this process can be further transformed to executable code and commands for the business rule engine or expert system. It is also feasible to transform BR to OCL in order to have a formal unambiguous presentation.

The preliminary version of the tool supporting specification of abstract templates, domain specific templates and export to XMI BRidgeIT is available at http://isl.vgtu.lt/BRidgeIT.

4

Business rules transformations

The analysis presented in the previous chapters emphasizes that BR should be transformed from the representation close to the natural language to the formal or semi-formal languages. This chapter demonstrates that the BR specified using BRTL approach can be transformed into the UML/OCL.

In the first part of the chapter, ORM models are transformed into UML models constrained by Object Constraint Language (OCL). The approach precisely describes the main features of the transformation. This opens the approach for seamless refining of the resulting models using UML tools and transformation to executable code.

Differently from ORM which has limited variation of constructs, transformation of BR templates (which can be defined by the user) is more complicated. The user can define actually the unlimited variety of templates. Therefore, the transformation of the BR specified through templates is demonstrated using several well known Morgan templates.

Both transformation specifications are validated for correctness using widely known UML/OCL tools.

4.1. ORM to UML/OCL transformation

4.1.1. Motivation of the ORM to UML/OCL transformation

Within the concept modelling community the object role modelling (ORM) [8] models have been studied and used for decades. These models are subject to introductory courses in database and software engineering education. A typical course will introduce the main concepts in an informal way, explain how to transform ORM schemas into Relational database schemas and will deepen the subject by practical exercises using a design tool and a database system. Conceptual modelling intends to support the quality checks needed before building physical systems by aiming at the representation of data at a high level of abstraction, and therefore acquire a high degree of, often implicit, semantics.

Within the software engineering community, Unified Modelling Language (UML) [168] has gained much attention, in particular in connection with the Model Driven Architecture (MDA) [169]. This paper proposes approach to transform ORM models to UML and OCL [22] using transformation languages and tools that satisfy MDA requirements. Making transformation specification design decisions we will use only such UML and OCL features that are implemented in the popular UML and OCL tools [170], [171], [172], [173], [174]. In contrast to known ORM – UML transformation approaches, this thesis however describes with its transformation specification not only the basic ORM concepts but also, an important ORM part, ORM constraints that vaguely can be presented in pure UML. The paper formally connects ORM constraints to OCL constraints. Furthermore, the transformation between models is also described in formal executable language ATL [162]. Resulted UML models and OCL constraints are validated by before mentioned tools. We are not aware of another approach handling these two classical models with respect to practical applicability and their transformation in a rigorous and uniform way. In particular, we are not aware of an approach being able to express the ultimate goal of the model transformation process, namely the equivalence between the constraints for the different models, in a formal and explicit way.

4.1.2. Object type and value type transformation rules

According to UML metamodel [168] each class should belong to the package and the package should be in the model namespace. Therefore the first rule in the transformation specification creates UML package and appropriate UML model. ORM model is composed from entity types and value types. These are the first ORM model elements that should be transformed. ORM entity types

are proposed to map to UML classes within the namespace of the created model. Reference schemas of the entity types are transformed to the attributes of the appropriate classes. Value types are transformed to UML attributes if they are connected to one fact type and to the classes otherwise. We argue that it is expedient to transform ORM value type to class (Fig. 4.1,b) in case of participation in several fact types (Fig. 4.1,a) than to attribute (Fig. 4.1,c) because of the existence of explicit associations between value type UML class and object type UML classes, besides connection names to both directions are preserved. The overview of transformation approach is presented in table 4.1.

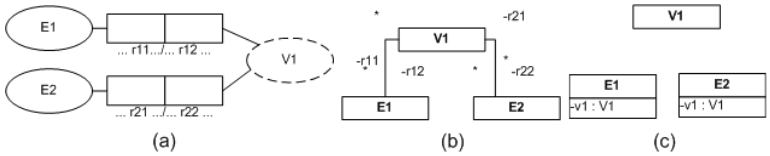

Figure 4.1. *(a) Value type of source ORM model can be transformed to several types of UML class diagrams (b, c)*

Table 4.1. *Overview of proposed approach for ORM transformation to UML/OCL*

ORM model elements	UML/OCL model elements
Entity Type	Class
Value Type	Class, Attribute
Fact Type	Class, Attribute, Association
Objectified Fact Type	Class
Subtype	Generalization
Mandatory constraint	Association end multiplicity range lower value
Uniqueness constraint	Association end multiplicity range upper value, OCL constraint
Frequency constraint	Association end multiplicity range lower value, Association end multiplicity range upper value
Set constraint	OCL constraint
Value constraint	OCL constraint
Ring constraints	OCL constraint (limited)

4.1.3. Fact type transformation rules

Transformation rules for fact types can be divided to three groups based on the cardinality of fact types: unary fact-types, binary-fact types and n-ary fact-types. Unary fact types attached to entity types are transformed to binary attributes. Unary fact types attached to value type that was not transformed to class results in an exception, it is treated as illogical model.

Binary fact types attached to the entity types results to binary association. Association end names are provided based on the first ORM phrase with the first appropriate role. Binary fact types with one value type, as it is stated earlier, are transformed to attribute or to the association if the value type is connected to several fact types. Binary fact types with two value types are transformed to the association or to the attribute based on the rules provided earlier.

N-ary fact types (Fig. 4.2, a) differently from the proposed in [26], [29] ternary association (Fig. 4.2, c) are transformed to UML class that have 1 multiplicity connections to participating entity types and value types (Fig. 4.2, b). The main reason for such transformation is that ternary associations are rarely supported by the UML tools. Objectified fact-types of any arity are transformed to UML classes as well.

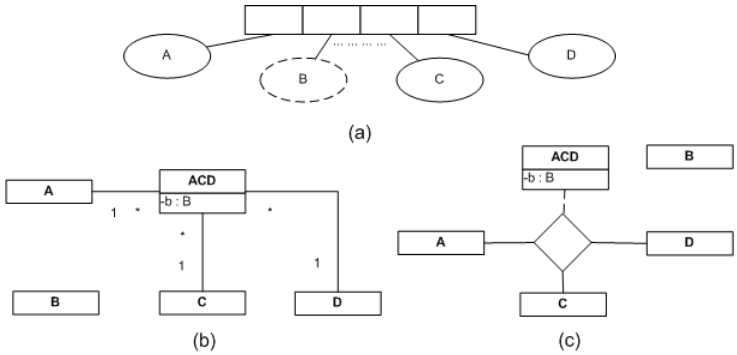

Figure 4.2. *Transformation of n-ary fact (a) type to combination of association and class (b) and to UML ternary association (c)*

4.1.4. Constraint transformations

Uniqueness, frequency and mandatory constraints

Internal uniqueness constraints are depicted as arrow tipped bars, and are placed over one or more roles in a fact type to declare that instances for that role (combination) in the relationship type population must be unique. For the transformation purposes we have identified three cases internal uniqueness constraints: one-role, two role on binary fact-type and n-ary role on n-ary fact-type.

One role internal uniqueness constraint is transformed to the multiplicity range upper value 1 of the appropriate association end for binary and n-ary fact types that was transformed to association. If it is applied on unary fact-type or on the fact type that was transformed to attribute then the multiplicity range upper value 1 is applied to attribute. If the constraint's binary or n-ary fact type was transformed to attribute and internal uniqueness constraint was applied to value type's role it constraints the following OCL constraint is generated for the UML model presented in Fig. 4.3,b:

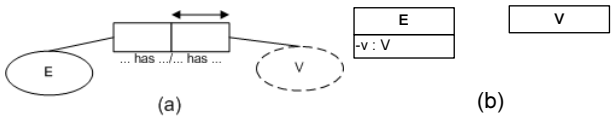

Figure 4.3. *(a) Internal uniqueness constraint on one role value type role of binary fact type, (b) resulted UML model*

Figure 4.4. *(a) two role internal uniqueness constraint on binary fact type, (b) resulting UML model*

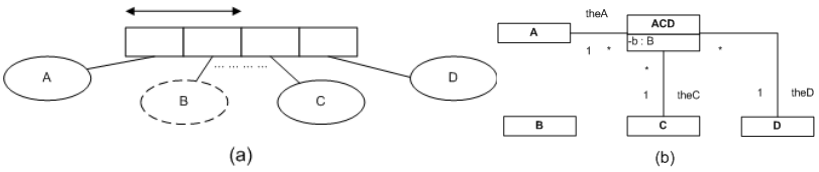

Figure 4.5. *(a) N-ary role internal uniqueness constraint, (b) resulting UML model*

Context E

inv: let a: Set(E) = E.allInstances in not a->exists(b|b.v=self.v)

Two role internal uniqueness constraints (Fig. 4.4.a) is transformed to following OCL statement for UML model in Fig. 4.4,b:

Context A

inv: not (self.r1->exists(b|b.r2->includes(self)))

N-ary role internal uniqueness constraints (Fig. 4.5.a) is transformed to following OCL statement for UML model in Fig. 4.5,b:

context ACD

inv: let a:Set(ACD)=ACD.allInstances in

not (a->exists(it|it.theA=self.theA and it.b=self.b))

An external uniqueness constraint (Fig. 4.6, a) shown as a circled "u" may be applied to two or more roles from different fact types by connecting to them with dotted lines. This indicates that instances of the combination of those roles in the join of those fact types are unique. In order to efficiently implement this constraint we have had to introduce ORM model wellformedness constraint on scope of the external uniqueness constraint. It constrains external uniqueness constrain to be put only on roles of the fact types connected to the same value or entity types. The necessity of introducing such wellformedness constraint arises because of inability of OCL to iterate through the model and find joins that external uniqueness constraint requires. The OCL constraint's context in this case

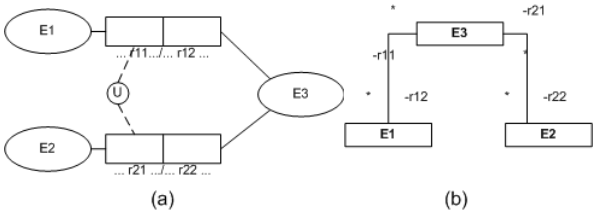

Figure 4.6. *(a) External uniqueness constraint on the binary fact type, (b) resulting UML model*

is any class that is attached to all fact types constrained by the ORM external uniqueness constraint. OCL constraint on UML model in Fig. 4.6 b following:

context E3
inv: let a: Set(E3)=E3.allInstances in
(not a->exists(b|b.r12=self.r12 and b.r22=self.r22))

A mandatory role constraint declares that every instance in the population of the role's object type must play that role. Mandatory constraint is transformed to association's other's end multiplicity range lower value. Default value is 0 if the role does not have mandatory constraint [26], [29].

Frequency constraint applied to a sequence of one or more roles, these indicate that instances that play those roles must do so exactly n times, between n and m times, or at least n times. This type of constraints is transformed to appropriate multiplicity range lower and upper value of the association end or attribute.

Set constraints

A dotted arrow (Fig. 4.7, a) from one role sequence to another is a subset constraint, restricting the population of the first sequence to be a subset of the second. Resulting OCL constraints for UML model in Fig. 4.7, d:

Context E2 inv: self.r11->includesAll(self.r21)
Context E1 inv: self.r12->includesAll(self.r22)

Equality constraint (A double-tipped arrow Fig. 4.7, b) indicate the populations must be equal. Resulting OCL constraints for UML model Fig. 4.7, d:

Context E2 inv: self.r11=self.r12
Context E1 inv: self.r12=self.r22

A circled "X" (Fig. 4.7, c) is an exclusion constraint, indicating the populations are mutually exclusive. Exclusion constraints may be applied between two or more sequences. Resulting OCL constraints for UML model in Fig. 4.7, d:

Context E2
inv: self.r11->isEmpty() or self.r12->isEmpty()
Context E1
inv: self.r12->isEmpty() or self.r22->isEmpty()

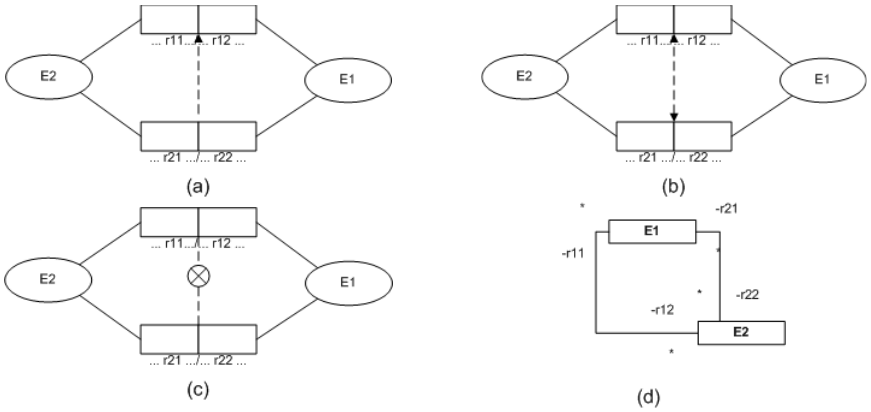

Figure 4.7. *(a) subset, (b) equality, (c) exclusion constraint on binary fact type, (d) resulting UML model*

Value constraint

To restrict an object type's population to a given list, the relevant values may be listed in braces (Fig. 4.8, a). If the values are ordered, a range may be declared separating the first and last values by "." (Fig. 4.8, b). OCL constraint for range value constraints for UML model in Fig. 4.8, c:

context A inv: self.code>=a1 and self.code<=a2

OCL constraint for list value constraint for UML model in Fig. 4.8, d):

context B

inv: self.code='b1' or self.code='b2' or self.code='b3'

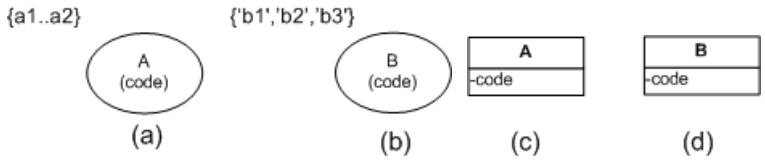

Figure 4.8. *Entity type with (a)value range constraint and (b) value list constraint, (c,d) resulting UML model*

Ring constraints

Ring constraint that may be applied to a pair of roles played by the same host type. These indicate that the binary relation formed by the role population must be irreflexive (ir), intransitive (it), acyclic (ac), asymmetric (as), antisymmetric (ans) or symmetric (sym). We will illustrate OCL constraints for the ORM ring constraints using UML model presented in Fig. 4.9, b. Ring constraints can be put on roles that can be transformed to association end of different multiplicity. Therefore we are presenting OCL constrains with navigation statements for one to many multiplicity case (r2 association end in

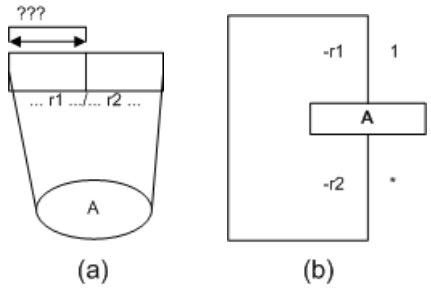

Figure 4.9. *(a) Role with undefined ring constraint, (b) resulting UML model*

Fig. 4.9, b) and constraint for single value for many to one case (r1 association end in Fig. 4.9, b).

Irreflexive means the object cannot bear the relationship to itself. OCL constraint for navigation to set:

Context A inv: self.r2->excludes(self)

OCL constraint for single value:

Context A inv: not (self.r1=self)

Intransitive means that if the first bears the relationship to the second, and the second to the third, then the first cannot bear the relationship to the third.

Intransitive OCL constraint for navigation to set:

context A

inv: self.r2->collect(b|b.r2)->excludesAll(self.r2)

Intransitive OCL constraint for single value:

context A inv: not(self.r1.r1=self.r1)

Asymmetric means that if the first bears the relationship to the second, then the second cannot bear that relationship to the first

Asymmetric OCL constraint for navigation to set:

context A inv: self.r2->collect(b|b.r2)->excludes(self)

Asymmetric OCL constraint for single value:

context A inv: not (self.r1.r1=self)

Anti-symmetric means that if the objects are different, then if the first bears the relationship to the second, then the second cannot bear that relationship to the first.

Anti-symmetric OCL constraint for navigation to set:

context A

inv: self.r2->select(a|not(a=self))->collect(a| a.r2)->excludes(self)

Anti-symmetric OCL constraint for single value for:

context A

inv: not (self.r1=self) implies (self.r1=self.r1)

Symmetric means that if the first bears the relationship to the second, then the second bears that relationship to the first.

Symmetric OCL constraint for navigation to set:
context A inv: self.r2->collect(a|a.r2)->includes(self)
Symmetric OCL constraint for single value:
context A inv: self.r1.r1=self

Acyclic means that a chain of one or more instances of that relationship cannot form a cycle (loop). It is the only type of ORM constraint that cannot be fully implemented in OCL. This constraint requires recursive OCL statement; however recursion is still unsolved issue of OCL [175]. But it is possible to generate through transformation specification OCL constraint of practically unlimited depth. We have shown in bold repeatable part of OCL constraints.

Acyclic OCL constraint for navigation to set:
context A
inv: inv: (self.r2->collect(a|a.r2)->excludes(self))
Acyclic OCL constraint for single value:
context A
inv: not (self.r1.r1=self)
Acyclic deeper OCL constraint for navigation to set:
context A
inv: (self.r2->collect(a|a.r2)->
collect(a|a.r2)->excludes(self))
Acyclic deeper OCL constraint for single value:
context A inv: not (self.r1.r1.r1=self)
Acyclic even deeper OCL constraint for navigation to set:
context A inv: (self.r2->collect(a|a.r2)->
collect(a|a.r2)-> collect(a|a.r2)->excludes(self))
Acyclic even deeper OCL constraint for single value:
context A inv: not (self.r1.r1.r1.r1=self)

4.1.5. An example of ORM-UML/OCL transformation

We present a case study of the use of the transformation specification in ATL to create UML model constrained by OCL statements from ORM model.

For our case study, we consider a fragment of scientific conference management domain ORM model (Fig. 4.10). It is information system used by a conference programme committee chair to maintain details about submitted papers, reviewers and assigned reviews.

The source ORM model was encoded to XMI format according to ORM metamodel and transformed to UML model in appropriate XMI format using ATL language execution environment. OCL statements constraining resulted UML model were generated as textual strings.

We have mapped example ORM model constraints to the OCL statements. In the following part of the chapter we will provide ORM constraint textual

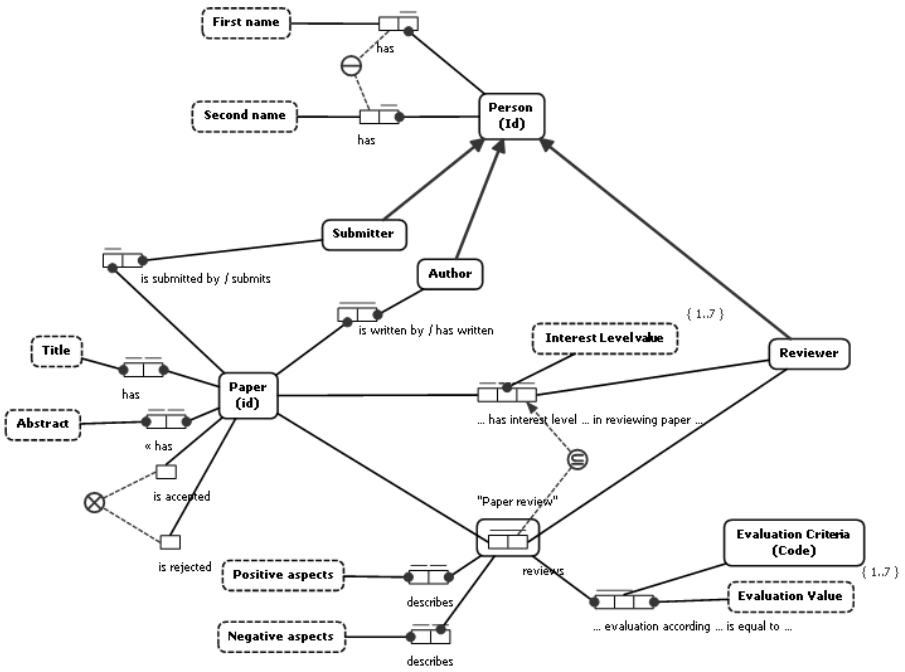

Figure 4.10. *Source ORM model for the transformation example*

description and appropriate OCL statement resulted from the transformation.

Uniqueness role constraint on "Author has written Paper":

context Paper inv:not self.iswrittenby ->

exists (a|a.haswritten->includes(self))

Uniqueness role constraint on n-ary fact type "Paper review evaluation according Evaluation Criteria is equal to Evaluation Value" is transformed to:

Context PaperreviewevaluationaccordingEvaluationCriteriaisequaltoEvaluationValue

inv:let a:

Set(PaperreviewevaluationaccordingEvaluationCriteriaisequaltoEvaluationValue)=PaperreviewevaluationaccordingEvaluationCriteriaisequaltoEvaluationValue.allInstances in not a->exists(a| a.thePaperreview = self.thePaperreview and a.theEvaluationCriteria = self.theEvaluationCriteria)

Uniqueness role constraint on objectified binary fact type "Reviewer reviews Paper" is transformed to:

context Paperreview

inv:let a: Set(Paperreview) =Paperreview.allInstances

in not a->exists(a| a.theReviewer = self.theReviewer and a.thePaper = self.thePaper)

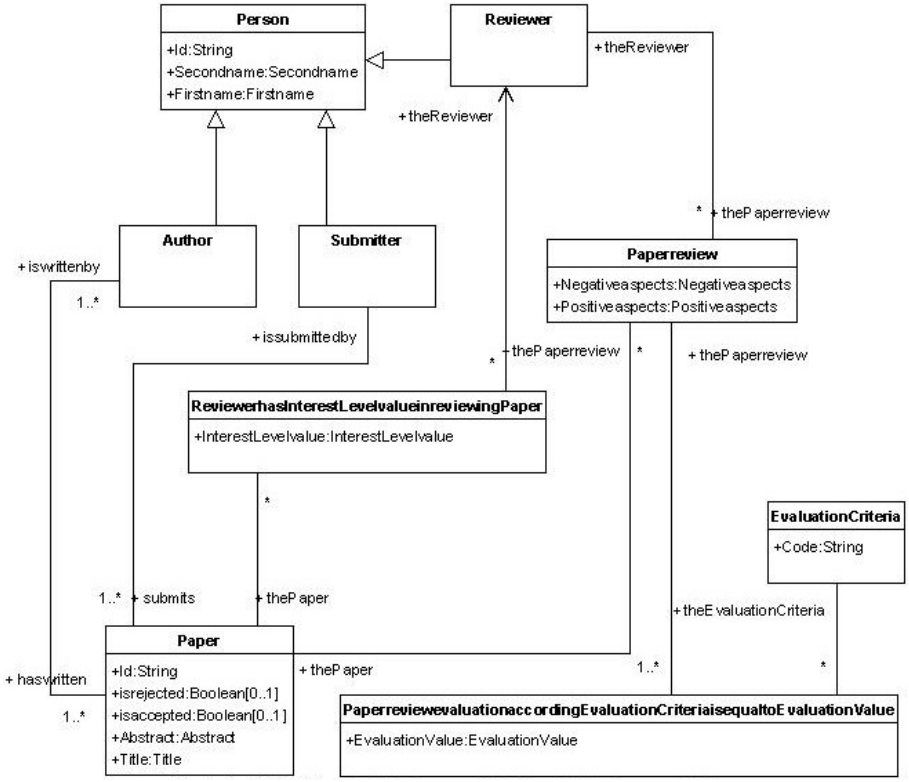

Figure 4.11. *Example of resulting UML model*

Uniqueness role constraint on n-ary fact type "Reviewer has interest level Interest Level value in reviewing Paper" is transformed to:

context ReviewerhasInterestLevelvalueinreviewingPaper inv: let a: Set(ReviewerhasInterestLevelvalueinreviewingPaper) = ReviewerhasInterestLevelvalueinreviewingPaper.

allInstances in not a->exists(a| a.theReviewer = self.theReviewer and a.thePaper = self.thePaper)

External uniqueness role constraint on fact types "Person has First name" and "Person has Second name" is transformed to:

context Person inv:

let a: Set(Person) =Person.allInstances in

not a->exists(a| a.Firstname = self.Firstname and a.Secondname = self.Secondname)

Subset constraint on fact types "Reviewer has interest level Interest Level value in reviewing paper Paper" and "Reviewer reviews Paper" is transformed to:

context Paper inv:

self.theReviewerhasInterestLevelvalueinreviewingPaper->
collect (a|a.theReviewer)->
includesAll(self.thePaperreview->
collect (b|b.theReviewer))

Exclusion constraint on fact types "Paper is accepted" and "Paper is rejected" is transformed to:

context Paper
inv: self.isaccepted or self.isrejected

We have checked all presented constrains for the syntactic and semantic correctness using OCL tool OCLE and Dresden OCL toolkit. Additionally in order to verify that the OCL constraint semantics fully represent ORM constraint semantics we used approach described in [176] and implemented in USE tool. The principle for the approach is to define properties that should be verified on the model. Then the USE tool checks whether it is possible to generate snapshots from the model that verify the property. Appropriate UML model snapshots were generated for the each OCL constraint.

4.2. Business rule templates to UML/OCL transformation

The main purpose of this chapter is to demonstrate the possible transformation of BR templates and resulting rules to the OCL statements. As it was mentioned before BR template metamodel do not provide any specific templates, therefore before processing with specification of BR it is necessary to select or construct new BR templates. Consequently one more purpose of this chapter is to present how the BR templates can be formally defined within the boundaries of provided metamodel.

Table 4.2. *ORM fact types of conference organization domain*

Fact number	Fact
F1	Conference paper is included in proceedings
F2	Conference paper has overall evaluation
F3	Conference paper is accepted
F4	Conference paper is selected by international program committee
F5	Conference paper has signed copyright form
F6	Conference paper has camera ready file
F7	Conference paper is written by author from country

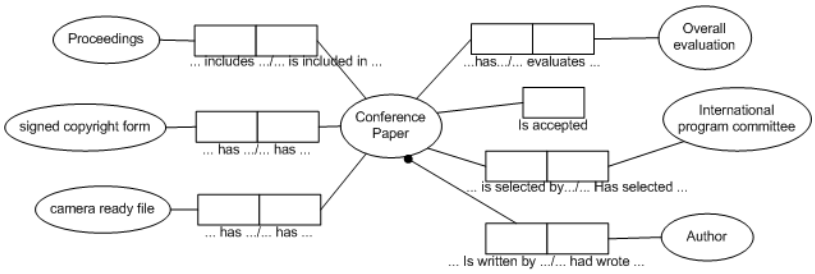

Figure 4.12. *Source ORM Conference model*

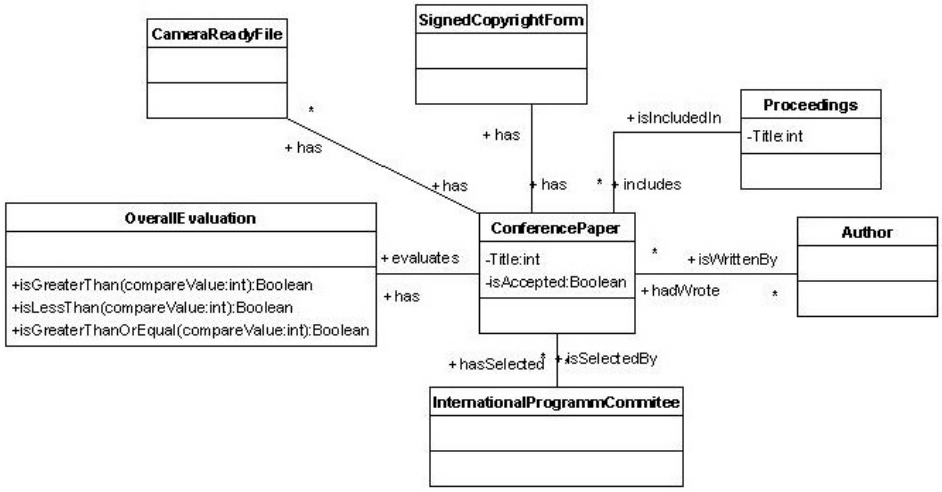

Figure 4.13. *Resulting UML model of Conference domain*

Conference organisation domain which is common for the majority of readers is selected for the examples. Although it is not pure enterprise business domain transformation principles presented in this chapter remains valid and in other domains. Table 4.2 presents the minimal number of ORM facts types from this domain and resulting ORM model is presented in Figure 4.12. ORM model transformed to UML model is presented in Figure 4.13.

4.2.1. Basic constraint template

This template, the most common business rule template, establishes a constraint on the subject of the rule. Two equally valid variants are provided. The optional word "should" in this template makes an easier-sounding expression in some circumstances. It does not make the rule optional in any way. Example of business rule based on basic constraint template is presented in table 4.3.

Table 4.3. *Basic constraint template transformation example*

Basic constraint template
Morgan notation: <det> <subject> (must | should) [not] <characteristic> [(if | unless) <fact>].
Precise notation of abstract template: <det> <subject> (<keyword: must> | <keyword: should>) [not] <characteristic> [(<keyword: if> | <keyword: unless>) <fact>].
Precise notation of domain template: <det: Each> <subject: Conference Paper> <keyword: must> <characteristic: be accepted> <keyword: if> <characteristic: overall evaluation> <keyword: greater than > <numeric:?>
User view of the template: Each Conference Paper must be accepted to conference proceedings if overall evaluation is greater than <?>
Resulting business rule: Each Conference Paper must be accepted to conference proceedings if overall evaluation is greater than 5
Transformation result to OCL *Context: ConferencePaper* *Inv r1: if self.OverallEvaluation.isGreaterThan(5) then* *isAccepted=true* *EndIf*

4.2.2. List constraint template

This template also constrains the subject, but the constraining characteristic(s) is (are) one or more items taken from a list. Again, two variants are provided, so you can choose the one that's the best fit to the particular situation. Example of business rule based on list constraint template is presented in table 4.4.

Table 4.4. *List constraint template transformation example*

List constraint template
Morgan notation: <det> <subject> (must | should) [not] <characteristic> (if | unless) at least <m> [and not more than <n>] of the following is true: <fact-list>.
Precise notation of abstract template: <det><subject><keyword: must ><characteristic><keyword: only if>

<keyword: at least><numeric: m><keyword: of the following> <keyword: is true>:<fact-list>
Precise notation of domain template: <det: Each><subject: conference paper><keyword: must ><characteristic: <role: be included in><objectType: the proceedings> ><keyword: only if> <keyword: at least><numeric: ?><keyword: of the following> <keyword: is true>:[<fact-list: <keyword: it><characteristic:?>>]*
User view of the template: Each conference paper must be included in: the proceedings only if at least <?> of the following is true:[it <?>]*
Resulting business rule: Each conference paper must be included in the proceedings only if at least 3 of the following is true: it is selected by the international program committee; it has the camera ready file; it has signed copyright form.

Transformation result to OCL
`Context: ConferencePaper`
`Inv r2:`
`self.InternationalProgramCommittee -> notEmpty()`
`and self.CameraReadyFile-> notEmpty()`
`and self.SignedCopyrightForm`

4.3. Discussion

In this section we want to debate typical questions that may show-up during discussions about the subject of this chapter.

What are the business cases of the approach? A 'business case' for our approach could be tuning of the general database model, developed by ORM, and application, developed using UML to handle that database. Constraints provided in ORM should be preserved in both of them.

What role plays tool support in the approach? Transformation rules and resulting UML and the OCL constraints are quite complex. Our experience shows that this complexity requires tool support in order to understand the consequences of design decisions, for example, the consequences of a particular constraint. We use OCLE and Dresden OCL for constraint validation and Poseidon for target UML model validation.

Is transformation extensible? Transformation specification is provided as a fully executable ATL file containing transformation rules. One can change the transformation specification and adopt it for its own needs.

Is transformation fully reversible? At the moment transformation is not fully

reversible. In case of reverse transformation of UML model to ORM objectified and n-ary fact types would be not recreated. Transformation of OCL constraint to ORM constraints is hardly possible at the moment. The alternative is to transform OCL to ConQuer language proposed in [177]. During reverse transformation only the basic phrases and sentences will be recreated.

4.4. Conclusions of the 4th chapter

In this chapter, the use of MDA as a framework for the transformation of ORM models to UML class models with the constraints represented in OCL is described. We have proposed and formally specified the transformation rules and transformation decisions for the resulting model to be accessible to the widely used UML tools (e.g. Poseidon for UML, Rational Rose, and Eclipse UML). Differently from the existing approaches, the presented transformation covers ORM constraints in addition to transformation of structural ORM elements. However, due to the limitations of OCL, there still exists unresolved unlimited iteration problem. Therefore, we had to limit transformation specification to the predefined iteration depth of resulting the OCL constraints in the case of transformation ORM set and acyclic ring constraints.

Transformation of BRs, specified by BRTL into UML/OCL, can improve the quality of BR design and facilitate the development of applications using BRs authored by the domain experts. The use of the integrated ORM/BRTL metamodel suggested in the thesis for the transformation of business rules enables us to use standard model-driven tools. Since only well-known modelling concepts of UML, which are compliant with Meta Object Facility (MOF), have been used, the BRTL metamodel suggested in the thesis can be understood by everybody familiar with UML.

The proposed approach has proven to be very effective for generating UML and OCL constraints from ORM by presentation of providing transformation examples. Each presented OCL statement was validated to be correct syntactically and semantically by using the OCLE tool. In order to prove the transformation of the semantics of ORM constraints, the snapshots of the resulting UML model generated using USE tool for each OCL constraint were used.

The suggested approach enables software system engineers to focus on the application domain and architectural design decisions without being limited by the tools used, because MDA ensures exchangeability of models. It is especially important if conceptual models were developed by separate teams and brought together for the creation of enterprise wide system.

5

Evaluation of the approach

This chapter documents the findings of the experiment aimed at determining the extent to which BR specified using BRTL can be used within the model-driven development of the financial reporting systems. The results of the experiment are compared with the data available from four historical projects of the same domain.

5.1. Experiment Overview

The experiment is concerned with the specification and implementation of a fragment of fully executable test code. The application chosen for development was a set of financial reports providing non technical user with the reporting information. Our main aims in this experiment are to trace the report algorithm specified using BR in the language acceptable for user to the executable SQL statement and evaluate results.

This type of application was chosen because of its wide distribution, reporting functionality is an eternal part of many enterprise systems. At the same time algorithms of these reports have to be constantly reviewed in order to insure confidence in reporting data. Changes to these algorithms happen on the regular basis.

The main tenet of MDA is to abstract away from particular implementation technologies (platforms) by modelling systems in a platform independent way and automating the process of developing implementations on particular platforms from those models. It is intended that a Platform Independent Model (PIM) is realized through the use of a modelling language such as UML and exists to document a technology independent architecture for a specific computing process at a high level of abstraction. Since the PIM is platform independent no specific implementation technology is specified. Mappings from these PIMs to Platform Specific Models (PSMs) are documented where a specific PSM models the architecture required for software deployment within a specific implementation technology.

To comply with MDA information systems development requirements, the experiment was initiated through the development of a test system PIM. It is important to note that while BR templates are platform independent in the respect that no implementation technology constraints are specified within the templates structure, they are domain specific because of the references to the domain model specified in ORM and elements of the domain language common to the user.

A PSM consisting of the architecture required for the implementing of the test system using a specific set of technologies was created in parallel to the PIM. By implementing the two models concurrently, the PIM architecture could be used within the relation of the PSM to create two complementing models with inherent similarities. These similarities could be exploited to facilitate the extraction of PIM to PSM mappings. The PSM is described within section 5.5.

5.2. Goals

Figure 5.1 illustrates an overview of the experiment structure in which the top and bottom entities represent the PIM and PSM respectively.

The BRTL supporting BRidgeIT tool and transformations appearing in the centre of the diagram represents the experiment objective. As well as creating workable BR templates for the specification of BR on platform independent level the experiment is aimed at an investigation into the extent to which transformational support for these templates can be realized thought the utilization of element held within BRTL and ORM. Therefore the experiment result will consist of a documented set of PIM to PSM transformations with indications to where extra information is required to be presented within transformable BR specification to facilitate their use.

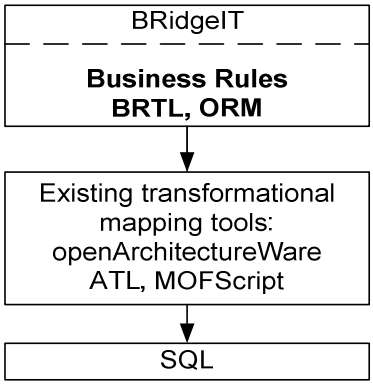

Figure 5.1. *Domain structure*

5.3. ORM model of the test application

ORM model in Figure 5.2 is used to present the main terms and their relations from the domain of interest. It is clearly seen that presented ORM model can be rewritten in natural language. Its development actually starts from the sentences that are used by the domain experts. At the same time ORM model does not seem close to any database model or any other formal model, it is just a graphical representation of every day phrases used by the domain profession and this, as consequence, minimizes any negative reaction of domain professionals.

The application domain model consists of the entities all together describing the reporting domain. *Report* is a report term that has relations with entities *Column* and *Row* as it is presented in Figure 5.2. Each entity has a reference schema specified in the brackets that is used to identify instance of an entity.

Moving towards analysing the model presented in Figure 5.2. it is possible to see that *Row* is related to three other entities *GL*, *ARP* and *CGR*. These entities are native for the domain of interest and are the acronyms of terms used in the ten years old legacy system. To be specific, *GL* is an acronym of "General Ledger". According to the same logic, CGR corresponds to "Customer GRoup". Unfortunately, we did not break the ARP code; however the meaning of these three letters is a more detailed grouping of GL records.

These entities represent terms used to describe the algorithm of mapping rows in the data source to rows in the report applying some aggregation operation. For example predicate "positive balance in" prescribes to include only positive balance of some particular GL to the corresponding row in the report. However this model is not enough to specify all BR related with our test-application financial report. It is only the structure that will be used for the development of BR template. It is obvious that in this form it is possible to

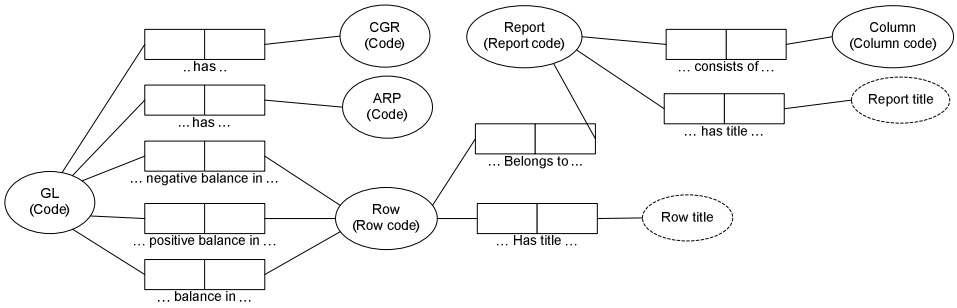

Figure 5.2. *Domain ORM model*

present only most simple rules, whereas complex rules requiring order of terms, optional and mandatory elements cannot be presented using this model.

BRidgeIT currently does not support graphical notation of the ORM model. We have used textual notation instead.

5.4. Specification of the test application

The next phase of development PIM is creation of BR template and specification of BR according to this template. Developed templates will have reference to the ORM domain model presented in the previous section.

The usual development of the template starts from the identification of the patters in the requirements. In our case we have used old user requirements describing report algorithm in order to develop templates. This approach insures that domain professionals will work with BR statements that are close to their everyday phrases. As a result of this activity, two templates were created.

The first one *Row name* is used to relate row code and row name. It is specified using BRTL:

SE "Row" LE ? CE "has title" LE ?.

Subject expression (BRTL keyword: SE) is used to refer to entity *Row* from the ORM model. Keyword characteristic expression (BRTL keyword: CE) is used to denote “has title” relation between entities *Row* and *Row title*. This template has two parameters of literal type (BRTL keyword: LE) expressed by two question marks. It is intended that such kind of templates would be developed by IT professionals. Domain professionals will work with user friendly presentation of the template:

Row {?} has title {?}

After the domain professionals have provided all necessary parameters there were developed more than 50 rules of such kind:

Row {1.} has title { Cash and Balances with Central Banks }

Row {2.} has title { Financial Assets Held For Trading Total }

Row {2.1.} has title {Financial Assets Held For Trading Derivatives }
Row {2.2.} has title {Financial Assets Held For Trading Equity Instruments}
Row {2.3.} has title {Financial Assets Held For Trading Other Debt Instruments}

This rule seems relatively simple and naturally can be implemented in one table of relational database. However in relational database case we would have rule interpretation difficulties by domain professionals. The support process of BR implemented as tables and corresponding forms is more resource intensive than in template case. This argumentation seems even more assured in more complicated template case (e.g. Report algorithm).

As it was mentioned before, for the experiment we have developed two BR templates. The second one is called "Report algorithm". This template is used to describe the most important part of the system under consideration. It is an algorithm intended to map records in the data sources to the rows in the report. The rules described using this template represent mapping criteria, which could be presented as logical statements. However, domain professionals prefer to work with natural language statements instead of the set of logical operators (e.g. "AND" and "OR"). Report algorithm template specification in BRTL is presented in the next paragraph:

```
[KE "Negative"]{paramMinus}
SE "GL" ( NE ? | NE ? CE "ARP" NE ? )      {paramGLARP}
[
   KE "All" CE "CGR" |
   [KE "except"]{parIskirCGR} CE "CGR" NE ?
]{parCGR}
(
   CE "positive balance in" LE ?|
   CE "negative balance in" LE ?|
   CE "balance in" LE ?
){parLikuciai}
[
   [KE "all these GL"| KE "GL" LE ?]
   KE "credit (negative) balance does not decrease them but is shown in row"
LE ?
   [
      KE "except account" NE ?
       KE "which negative balance is showed in " LE ?]
]
[KE "except GL" NE ? KE "for which the result is shown"]
[ KE "additionally" NE ? KE "negative balance with opposite sign"]
```

This template differently from the previous one has optional (BRTL keyword "[" and "]") and mandatory (BRTL keyword "(" and ")") elements. The

notation is very close to the regular expression notation. However differently from regular expressions BR specified using this template are stored in the ECORE model format and are acceptable for MDA transformations. Additionally, in order to simplify specification of transformation it is possible to define names of the composite rule parts within the template definition (BRTL keyword "{" and "}"). For example, elements paramGLARP and paramMinus allow direct reference to the rule parts which simplifies specification of transformation.

The template report algorithm allows specifying over 500 different variations of BR. We are presenting only the most typical variations of BR defining report algorithm as it is specified by the user:

GL {1111} ARP {3333} All CGR balance in {1.}

GL {4568} ARP {4789} balance in {1.} credit (negative) balance does not decrease them but is shown in row {24.}

GL {15987} ARP {4567} CGR {245} balance in {1.} credit (negative) balance does not decrease them but is shown in row {24.}

The first example rule says: GL {1111} ARP {3333} all CGR positive balances are presented in report row {1.}. It means that the generated code must select only positive records from the data source that have GL account number 1111 ARP number 3333 and any client group.

It should be noted that in our case one rule is not enough to provide algorithm for all rows in the report. Even more, BR corresponding to one template are not enough to generate even the simplest report, it is necessary to use a set of BR that correspond to different templates. ORM in this case serves as a structure that allows connection of BR specified using two different templates, however satisfying one common functional purpose.

5.5. Platform specific model and transformations

Existing data warehouse can be used in order to provide data source for test system report. According to MDA, code generation should be executed in two steps. During the first step BR are transformed to the SQL select statement ECORE model. The second step is when generation of code from SQL ECORE model is executed.

In order to execute the first MDA transformation step two components are needed. The first one is SQL select statement metamodel, which will be used for the experiment, and the second one is model to model transformation tool [162]. At the moment of experiment there was no known mature enough SQL select statement metamodel available. Therefore the new one very simplified metamodel presented in Figure 5.3 was developed.

Our developed simplified SQL metamodel is very close by its nature to the

UML and OCL metamodels. The main element of the metamodel is select expression (metaclass SelectExp) which is contained within SQLModel metaclass. Select expression in our metamodel has only basic elements select list items, basic SQL formulas (metaclass OperationCallExp) and references to the database structures metaclasses ColumnCallExp and TableCallExp. Naturally we need to develop very basic metamodel of data base elements, they are represented by metaclasses Table and Column. Despite its simplicity this SQL metamodel is enough to experiment with code generation from BR specified in templates for the test application.

Despite of the fact that actual SQL code is generated only on the second transformation step, the main decisions regarding test system implementing code are made during the first step when model to model transformation is specified. Therefore it is feasible to discuss the code resulting from the BR transformation.

First of all, ORM model will be transformed to the SQL model. Mapping ORM model in transformation rules is necessary in order to provide rules with information about relying database structure, in particular tables and column names.

As it was mentioned before, test-system report will be using existing data warehouse structures, therefore the only thing that should result from transformations is correct select statement. The main intention of this statement is to map existing records to report rows according to BR. Resulting SQL statement is trivial by its nature; however because of the big number of rules (more than 500) its support is rather complicated.

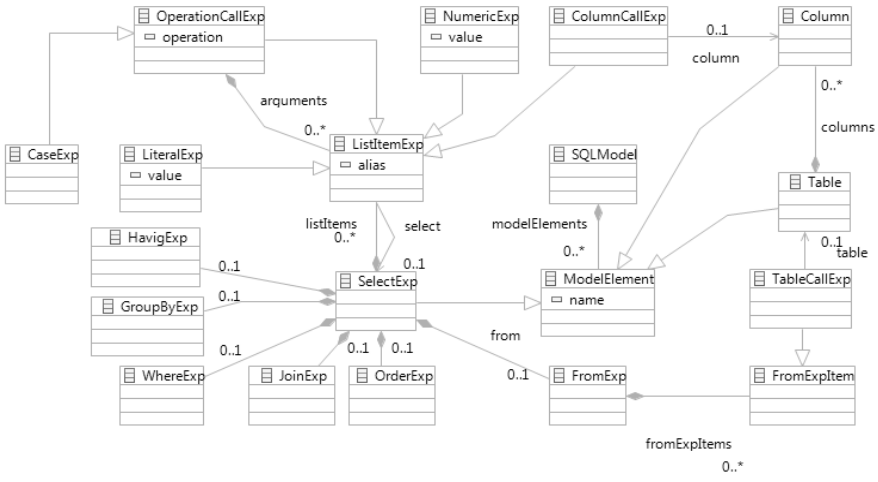

Figure 5.3. *SQL select statements simplified metamodel*

5.6. Evaluation of the results

In the previous sections we have described our experiment environment and technical implementation results. As it was mentioned earlier, one of the purposes of the experiment was to evaluate BRTL based MDA transformational approach comparing it to the alternative ones. For this purpose we have selected experiment domain that satisfies three requirements:

- Not difficult to implement.
- Many BR > 500.
- Availability of historical data from the previous implementation projects.

After the execution of the experiment we have recorded the time spend for the development of different test—system artefacts. It was compared to the historical data collected in one of the Lithuanian enterprises and presented in Table 5.1. In this section we will briefly describe historical scenarios, provide comments on the activities and time necessary to implement them.

The figures presented here should be understood as a relative measures and they might change from project to project and are highly depending on the qualification of the IT and domain professionals. The results might be different applying different software development t process methodologies. However, we

Table 5.1. *Comparison of the results in one enterprise case*

Scenarios / Activities	No code gen.	Custom repository with code generation			BRTL
		No inter.	Forms	Univ.	
Tool development	0 h.	160 h.	320 h.	600 h.	3200 h.
Tool customisation	0 h.	0 h.	0 h.	50 h.	20 h.
Specification of algorithm					
- Domain professional	80 h.	80 h.	80 h.	80 h.	100 h.
- IT professional	50 h.	50 h.	50 h.	50 h.	20 h.
Coding of algorithm	160 h.	120 h.	120 h.	700 h.	60 h.
- Lines of code to load repository	4000	3000	3000	5500	0
- Lines of code to generate code	0	3000	3000	6000	1200
Algorithm change (typical one change)					
- Domain professional	0,5 h.				
- IT professional	1 h.	2 h.	0 h.	1 h.	0 h.
Change delivery to the production environment	40 h.	40 h.	0 h.	40 h.	0 h.
Algorithm change (not typical)	20 h.	40 h.	40 h.	80 h.	15 h.

still believe that presented results are relevant because of the implementation of the scenarios in the same organization over the 3 years and without any explicit activity towards improving software development process. It is possible to state that these figures are accurate and are affected only by the technology being used.

The scenarios presented in Table 5.1 are the natural evolution towards increasing the effectiveness of IT professional's work and development of the tool that simplifies the life of the IT professionals. We do not distinguish separate group of graphical reporting solutions here because at the moment of experiment none of the major business intelligence consultants provided us with any solution that contradicts or affect our presented list. Even more, it is possible to make an assumption based on our experience with several Lithuanian enterprises that our presented list is a typical list of the most often implemented scenarios.

No code generation scenario is a straightforward approach to the problem. First of all, domain professionals specify in natural language algorithm for the report. Then IT professionals implement this algorithm in some programming language. After some testing phase the solution is presented for domain professionals. The change to the report requires repeating of all before mentioned steps.

Custom repository scenario includes development of data base based solution for the storage of report algorithm. This repository structure is suitable for the storage of only one type of algorithm that is described in the natural language. This scenario includes three possible options available in our analysed enterprise: No interface, Forms, Universal. Consequently, this scenario includes development of the software component implementing code generation from the repository.

No interface scenario omits the development of the interface available for the user. Database table storing an algorithm are edited by the IT professionals or advanced domain professionals.

Forms scenario involves development of the user interface in order the domain professionals would be able to enter and modify the algorithm.

Universal scenario differently from the previous two includes development of the universal repository. The developed repository was the most complex one comparing with No interface and Forms scenarios. The designed repository was intended to store any possible algorithm that could be specified within one SQL statement. Actually, this universal repository structure reminds simplified abstract syntax of SQL language with financial reporting domain specific additions. In order the user could use the user interface of Universal scenario he should have the basic understanding of SQL syntax and the principles the code was generated from repository. These requirements for the user qualification were too high and as consequence user interface was never used by the domain professionals. After unsuccessful implementation of user interface non MDA

domain specific language (DSL) was developed. This DSL was used to load algorithm to the repository. The main challenge with DSL is to develop a language that is common to the domain professionals and is not too technical. In our analysed enterprise, developed DSL was not accepted by the user, and as a result it was used solely by IT professionals.

BRTL scenario includes development of the BR templates, specification of the BR and MDA based model-to-model and model-to-code transformation as it was described in the previous sections.

The development time of all scenarios is separated to the following activities:

Tool development activity includes development of the algorithm storage tool. In no code generation scenario no tool was developed. In repository scenario this activity includes development of the repository database. In BRTL scenario it includes development of BRidgeIT. It is important to note that BRidgeIT differently from homemade repositories can be used to describe different types of templates from different domains.

Tool customisation activity is not applicable in No code generation and Repository scenario, because repository is created already customized for the particular algorithm. In BRTL case this includes development of templates.

Specification of algorithm activity is applicable for all scenarios. The time necessary to execute this activity is distributed between Domain professionals and IT professionals. This activity includes specification of algorithm by domain professionals and its understanding by IT professional. In BRTL scenario only domain professional is responsible for the specification of algorithm using predefined templates.

Domain professional is understood as a person familiar with domain application, however without programming background. This means that he has no experience of algorithms specification using programming language as well as using any formal language. Usually they are persons with understanding of trivial logical operations such as “AND” and “OR” but having difficulties with formulation of complex logical statements consisting of more than 3 such logical operations in the expressions with brackets. They also have no experience identifying logical contradictions within such statements.

IT professional is understood as a person with programming experience, with no or very little understanding of the domain logic and how it should be implemented in the information system. We do not distinguish systems analysts responsible for the requirement specification as it is intended that IT professionals have some basic background of requirements analysis.

Coding of algorithm is actual implementation of algorithm in programming language. In no code generation scenario this activity represents the classical coding of algorithm using some programming language. In custom repository scenario this activity includes development of code generation software

component and loading the repository with first version of the algorithm. Because of the usage of standard code generation facility in BRTL scenario specification, model-to-model and model-to-code transformations take less time. The usage of well formed templates provides IT professionals with already "filed repository".

Algorithm change activity represents a typical change of the algorithm. In our analysed algorithm it was addition/removal of one account to the row in the report. This requires relatively many effort of domain professional in Forms scenario. This is because of the necessity to browse over the number of complicated forms in order to make corrections. In BRTL scenario this activity requires to edit one particular business rule. However it takes a significant amount of time of IT professional in No code generation scenario. In repository scenario the time is used to fill in the repository, in no interface scenario to change repository manually, in Universal to edit DSL specification and update repository.

Change delivery to the production environment is a typical activity in the enterprises having several environments (e.g. development, testing and production) and implementing changes on the regular basis during service windows. In our analysed enterprise the changes were applied to the production environment once in two weeks. Therefore in some scenarios when the code migration to the production was necessary there is a time lag of 40 working hours.

Tool change activity is necessary to introduce changes that were not foreseen at the tool development time. In no code generation scenario it took 20 hours to change implementing code. In Repository scenario it was necessary to change repository structure and, as a consequence, edit code generation software component. In BRTL case modification of template and transformation specifications was necessary.

5.7. Conclusions of the 5th chapter

The results of the present experiment demonstrate the viability of the solutions based on the BR templates, BRTL and MDA transformations. A comparison of the experiment results with historical records shows that BRTL solution can be used in the constantly changing environment. Only in this case, a relatively high cost of developing the technology can be compensated by the time saved. BRTL technology allows reallocating of BR alteration costs from IT professionals to domain professionals.

The comparison of the experiment results with the historical data of an actual project clearly demonstrates that MDA-based solutions are economically not feasible in rarely changing environment and when cheap development

resources are available. Code generation from the repository scenario is feasible when the changes are typical and code generation from repository is not too complex. However, this scenario is not flexible enough to support any algorithm change. Even the addition of one column to the condition is a time-consuming task. Making these repositories more flexible and universal results in increased development time and makes the code generation a very complicated task. In this case, MDA-based tools allow us to reduce the development time significantly.

However, the wide use of transformations, as recognized by the previous researchers [178], [179], is limited by the lack of metamodels for the majority of programming languages. Anyone, planning to implement a transformation solution based on the language which is not very popular, is required to develop his/her own metamodel. Another less flexible option is to execute direct transformation of BR in templates to code, omitting model-to-model transformation.

The preliminary version of the tool supporting specification of abstract templates, domain specific templates and export to XMI BRidgeIT is available at http://isl.vtu.lt/BRidgeIT.

General conclusions

The development of the BRTL approach, BR specification based on user-defined templates, as well as transformation of BR to semi-formal language, conduction of an experiment and its verification, checking in practice and a comparative analysis of the historical data, allowed the author to draw the following conclusions having scientific and practical value:

1. The performed analysis of recent investigations aimed at capturing BRs reveals that the existing natural language templates are not suitable for the real cases of BR specification. In order to use the existing BR templates, it is necessary to rephrase the BR under consideration, and, then, the meaning of the BR vanishes for the BR owners. The suggested solution to the problem is to specify user-defined BR templates for each particular case of BR capturing.
2. The analysis of the existing BR specification approaches implemented in the tools shows that some of them have particular facilities to specify custom BR templates. However, the BR templates specified by these tools are quite trivial. Furthermore, the considered tools do not provide functionality for the model-driven transformation of BRs specified by means of suggested user-defined BR templates, as it is understood by OMG. In particular, it is not possible to manage the transformation process, to access metamodels used by tools and to

specify particular transformation rules.

3. Based on the research performed, we have concluded that the problem of employing user-defined templates for BR specification and further transformation of these BR can be solved by the BRTL approach suggested in the thesis.
4. Transformation of BRs, specified by BRTL into UML/OCL, can improve the quality of BR design and facilitate the development of applications using BRs authored by the domain experts. The use of the integrated ORM/BRTL metamodel suggested in the thesis for the transformation of business rules enables us to use standard model-driven tools. Since only well-known modelling concepts of UML, which are compliant with Meta Object Facility (MOF), have been used, the BRTL metamodel suggested in the thesis can be understood by everybody familiar with UML.
5. The experiment conducted and described in the thesis showed the advantages of the proposed approach in comparison with other commonly used approaches. Specification of BRs using templates and their further transformation to the executable code decrease the time of BR development up to 30 %, allowing us to reallocate the time from IT professionals to domain professionals.
6. The proposed approach eliminates the participation of IT professionals in the propagation of BR changes to the implementation platform if these changes are anticipated in the template. In the case, when the template is not designed for these changes, it saves up to 25 % of IT development time. Once implemented, the changes which were not designed become typical, with all beneficial outcomes.

References

[1] Editors of BRCommunity.com, "A Brief History of the Business Rule Approach", *Business Rules Journal*, Vol. 6, No. 1, 2005, [Online]. Available: http://www.BRCommunity.com/a2005/b216.html. [Accessed Jan. 15, 2008].

[2] Ross, R. G. *Principles of the Business Rule Approach.* Addison Wesley, 2003.

[3] The Business Rules Team (BRT). Adaptive et all. *Semantics of Business Vocabulary and Business Rules (SBVR) Revised Submission to BEI RFP br/2003-06-03*, OMG, 2005. [Online]. Available: http://www.omg.org/cgi-bin/doc?bei/05-01-01. [Accessed Jan. 20, 2007].

[4] Morgan, T. *Business Rules and Information Systems: Aligning IT with Business Goals*. Addison Wesley, 2002.

[5] Von Halle B. *Business Rules Applied: Building Better Systems Using the Business Rules Approach*. John Wiley & Sons, New York, 2002.

[6] Hay, D., Healy, K. A. *GUIDE Business Rules Project, Final Report,* GUIDE, October 1997.

[7] Gottesdiener, E. Eliciting Business Rules in Workshops (part 2), *Business Rules Journal*, 2003 January, Vol. 4, No. 1. [Online]. Available: http://www.brcommunity.com/p-b121a.php. [Accessed Nov. 12, 2007].

[8] Halpin, T.A. Object-role modeling (ORM/NIAM). *In: Bernus, P., Mertins, K., and G. Schmidt (Eds.): Handbook on Architectures of Information Systems*. Springer-Verlag, Berlin, (1998), pp. 81–101. [Online]. Available: http://www.orm.net/pdf/springer.pdf. [Accessed Jun. 23, 2007].

[9] Corticon Technolgies. [Online]. Available: http://www.corticon.com/. [Accessed Sep. 6, 2007].

[10] Fair Isaac, Blaze Advisor product. [Online]. Available: http://www.fairisaac.com/fic/en/product-service/product-index/blaze-advisor/. [Accessed Sep. 8, 2007].

[11] ILOG: Business Rules management systems. [Online]. Available: http://www.ilog.com/. [Accessed Sep. 16, 2007].

[12] Resolution iR. [Online]. Available: http://www.resolutionebs.com. [Accessed Jan. 23, 2007].

[13] Visual Rules BRMS. [Online]. Available: http://www.visual-rules.com. [Accessed Feb. 18, 2007].

[14] Protégé tool page. [Online]. Available: http://protege.stanford.edu/. [Accessed Feb. 15, 2008].

[15] Giurca, A., Lukichev, S. and Wagner, G. Modeling Web Services with URML. *In proceedings of Semantics for Business Process Management Workshop*, Budva, Montenegro, 2006. [Online]. Available: http://km.aifb.uni-karlsruhe.de/ws/sbpm2006/papers/sbpm06_Giurca.pdf. [Accessed Feb. 16, 2008].

[16] Nicolae, O., Diaconescu, I., Giurca, A., Wagner, G Towards a financial service Rule-Based implementation using Jena and Jboss. In N Tandareanu and I. Iancu (Eds.) *Proc. of 7th International Conference on Artificial Intelligence and Digital Communication,* AIDC'2007, September 15-16, 2007, Craiova, Romania, pp. 59–69. [Online]. Available: http://inf.ucv.ro/~aidc/proceedings/2007/AIDC07_07.pdf. [Accessed Jan. 7, 2008].

[17] Strelka. Strelka - The UML-based Visual Rule Modeling Tool: Intro, Download and Examples. REWERSE Working Group I1 Home Page. [Online]. Available: http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=Strelka. [Accessed May. 11, 2008].

[18] Milanović, M., Gašević D., Giurca, A., Wagner, G., and Devedžić, V. Model Transformations to Share Rules between SWRL and R2ML. *Proceedings of 3rd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2007),* Innsbruck, Austria, 2007. (May,2008) [Online]. Available: http://swese2007.fzi.de/papers/02.Model_Transformations.pdf. [Accessed May. 22, 2008].

[19] Milanović, M., Gašević, D., Giurca, A., Wagner, G., Lukichev, S., and Devedžić, V. Bridging Concrete and Abstract Syntax of Web Rule Languages. *Web Reasoning and Rule Systems*, LNCS: 4524/2007, 2007, pp. 309–318.

[20] Kilow, H. *Business models a guide for business and IT*. Prentice Hall, 2002.

[21] Wagner, G., Tabet, S., Boley, H. MOF-RuleML: The Abstract Syntax of RuleML as a MOF Model. *Integrate 2003 proceeding*. [Online]. Available: http://www.omg.org/docs/br/03-09-01.doc. [Accessed Aug. 14, 2006].

[22] Warmer, J., Kleppe, A. *The object constraint language. Precise modeling with UML.* Addison Wesley Longman, 1999.

[23] Hobbs, J., Israel, D., Principles of template design. *Proceedings of the ARPA Workshop on Human Language Technology.* M. Kaufmann, 1994, pp. 172–176.

[24] Sowa J., F, Relating Templates to Language and Logic. *Information Extraction: Towards Scalable, Adaptable Systems, edited by M. T. Pazienza*, LNAI 1714, Springer-Verlag, Berlin, 1999, pp. 76–94

[25] Hou, C.-S., Noy, M., Musen., A. A template based approach towards acquisition of logical sentences. *Proceedings of the IFIP 17th World Computer Congress*, Kluwer, B.V., 2002, pp. 77–89.

[26] Halpin, T.: UML data models from an ORM perspective: Parts 1–10, In: *Journal of Conceptual Modeling*, Inconcept, (1998-2001) (July, 2006) [Online]. Available: http://www.orm.net/. [Accessed Nov. 20, 2007].

[27] Halpin, T., Augmenting UML with Fact-orientation, *In: workshop proceedings: UML: a critical evaluation and suggested future, HICCS-34 conference,* 2001. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=00926348. [Accessed May. 14, 2006].

[28] Halpin, T., *Information Modeling and Relational Databases 3rd edn.,* Morgan Kaufmann Publishers, 2001.

[29] Bollen, P., A Formal ORM-to -UML Mapping Algorithm. [Online]. Available: http://arno.unimaas.nl/show.cgi?fid=465. [Accessed Jun. 7, 2006].

[30] OMG. Business Semantics of Business Rules RFP. OMG document: br/03-06-03, 2003. [Online]. Available: http://www.omg.org/cgi-bin/apps/
do_doc?br/03-06-03.pdf. [Accessed Aug. 12, 2006].

[31] Boley, H. The Rule Markup Language: RDF-XML Data Model, XML Schema Hierarchy, and XSL Transformations. *Proceedings of INAP2001* LNCS 2543. Invited Talk, Tokyo, Springer-Verlag, 2003, pp. 5–22.

[32] Demuth, B., Hussmann, H., Loecher., S. OCL as a specification language for business rules in database applications. In: Martin Gogolla, Cris Kobryn (Eds.), *<<UML>>2001 – The Unified Modeling Language. 4th International Conference,* Toronto, Canada, LNCS 2185, Berlin Heidelberg: Springer-Verlag, 2001, pp. 104–117.

[33] Alagar, V.,S., Periyasamy, K., P. BTOZ: A formal specification language for formalizing business transactions. *Proceedings of the 39th Int'l conf. and exhibition of technology of object-oriented languages and systems (TOOLS'01)*, 2001, pp. 240–252.

[34] Braun, P., Lötzbeyer, H., Schätz, B., Slotosch, O. Consistent Integration of Formal Methods. In S. Graf, M. Schwartzbach (Eds.): *Tools and Algorithms for the Construction and Analysis of Systems: 6th International Conference, TACAS 2000*, LNCS 1785, Springer-Verlag Berlin, Germany, March/April, 2000, pp. 48–62.

[35] Chisholm, M. *How to Build The Business Rules Engine*. San Francisco, Morgan Kaufmann Publishers, 2004.

[36] Gudas, S., Skersys, T. The Enhancement of Class Model Development Using Business Rules. *Advances in Informatics, 10th Panhellenic*

Conference on Informatics PCI 2005, Volas, Greece, November 11-13, 2005, Panayiotis B., Elias N. H. (Eds.), Lecture Notes in Computer Science, Vol. 3746. Springer-Verlag, Berlin, pp. 480–490.

[37] Wangler, B., Wohed, R., Ohlund, S. E. Business Modelling and Rule Capture in a CASE Environment. 4th European workshop on the Next Generation of CASE-Tools, Paris, 1993.

[38] VeTIS 1 project report. Vol 1, 2007.

[39] Leap SE project Home Page. Leap Systems. 2007. [Online]. Available: http://www.leapse.com. [Accessed Oct. 8, 2007].

[40] Key. KeY Project Home Page. 2007. [Online]. Available: www.key-project.org. [Accessed Oct. 8, 2007].

[41] Valatkaite, I., Vasilecas, O. A Conceptual graphs approach to business rules modelling. In: Kalinichenko, L. et al. (eds.): *Proc. Of Seventh East-European Conference on Advance in Databases and Information Systems (ADBIS'2003)*, Springer-Verlag. LNCS 2798, pp. 178–189.

[42] Bevington. D. ASL – A Formal Language For Specifying A Complete Logical System Model (Zachman Row 3) Including Business Rules. *Business Rules Journal*, Vol 5, No. 1, 2004. [Online]. Available: http://www.BRCommunity.com/a2004/b167.htm.
[Accessed Apr. 11, 2007].

[43] Ambler. S. W. Business Rule Overview. [Online]. Available: http://www.agilemodeling.com/artifacts/businessRule.htm.
[Accessed Jan. 16, 2007].

[44] Butleris, R., Kapocius, K. The Business Rules Repository for Information Systems Design. *6th East-European Conference ADBIS'2002*. Research Communications. Vol.2, Bratislava: STU, pp. 64–77.

[45] Gudas, S., Skersys, T., Lopata, A. Framework for Knowledge-based IS Engineering. *In: Advances in Information Systems ADVIS'2004.* LNCS Vol. 3261. Springer-Verlag, Berlin, pp. 512–522.

[46] Stulpinas, P., Stulpinas, R., Nemuraitė, L. Adaptyviojo tiekimo grandinių tinklo modelis ir jį realizuojanti elektroninių paslaugų sistema:

Informacinės technologijos verslui – 2005: tarptautinės konferencijos pranešimų medžiaga. ISBN 9955-09-871-6. - Kaunas, 2005, pp. 245–250

[47] Butleris, R., Kapočius, K. Business Rules Approach in Information Systems Development. *Proceedings of International Conference "Business Operation and Its Legal Environment: Processes, Tendencies and Results"*, Riga: Turiba, 12 April, 2002, pp. 121–128.

[48] Butleris, R., Danikauskas, T., Kapočius, K. Enrichment of Functional Requirements Specification Method with Business Rules. *Proceedings of the Thirteenth International Conference on Information Systems Development.* Vilnius, Technika, 2004, pp. 194–205.

[49] Butleris, R., Danikauskas, T. Reikalavimų informacijos sistemai specifikavimo Oracle CASE terpėje plėtra. *Informacijos mokslai*, Vilnius:VU, 19, 2001, pp. 49–60.

[50] Aleksandravičienė, A., Butleris R., Danikauskas, T., Šidlauskas, K. Duomenų modelio automatizuoto sudarymo prototipo funkcionalumo tyrimas. *Informacijos mokslai*, Vilnius:VU, 32, 2005, pp. 118–127.

[51] Armonas, A., Nemuraitė, L. Pattern Based Generation of Full-Fledged Relational Schemas from UML/OCL Models: *Information Technology And Control*, Kaunas, Technologija, 2006, Vol. 35, No. 1, pp. 27–33.

[52] Butkienė, R., Butleris, R. Extending Functionality of CASE Tools to Support Requirements Engineering. *Proceedings of the 1st Workshop on Emerging Database Research in Eastern Europe*, co-located with VLDB 2003. Humboldt-Universität Berlin, Germany, September 8 2003, pp. 12–16.

[53] Miliauskaite, E., Nemuraite, L. Taxonomy of integrity constraints in conceptual models. In: P.Isaias et all. (Eds.): *Proceedings of the IADIS Virtual Multi Conference On Computer Science and Information Systems* 2005, April 11–29, IADIS Press, pp. 247–254.

[54] Nemuraitė, L., Paradauskas, B., Salelionis, L. Extended Communicative Action Loop for Integration of New Functional Requirements. *Information technology and control*. Kaunas: Technologija, 2002, No. 2(23), pp. 18–26.

[55] Nemuraitė L., Čeponienė L. Reikalavimų transformavimas į projektą kuriant paslaugų informacines sistemas, *Informacinės technologijos*

2005: konferencijos pranešimų medžiaga. Kaunas, Technologija, 2005, pp. 601–609

[56] Butleris, R., Danikauskas, T., Kapočius, K. Enrichment of Functional Requirements Specification Method with Business Rules. *Proceedings of the Thirteenth International Conference on Information Systems Development.* Vilnius, Technika, 2004, pp. 194–205.

[57] Butleris, R., Motiejunas, L. Invoking business rules from database triggers. *Proceedings of 9th World Multi-Conference on Systemics, Cybernetics and Informatics*, July 10–13, 2005, Orlando, USA, 2005, Vol. IV, pp. 271–275.

[58] Butleris, R., Motiejūnas, L. Metadata for business rules integration with database schema. *Proceedings of the IADIS virtual multiconference on computer science and information systems 2005*. IADIS press, 2005, pp. 263–270.

[59] Kapočius, K., Butleris, R. Structuring of Business Rules During the Information System Development. Proceedings of the 1st Workshop on Emerging Database Research in Eastern Europe, co-located with VLDB 2003. Humboldt-Universität Berlin, Germany, 2003, pp. 17–21

[60] Motiejūnas L., Butleris R. The Requirements of Business Rules Managing. *Information Technology and Control*. Kaunas, 2004, No. 1(30), pp. 56–63.

[61] Skersys T., Gudas S. The enhancement of class model development using business rules. *Lecture Notes in Computer Science: Advances in Informatics: 10th Panhellenic Conference on Informatics, PCI 2005, Volos, Greece, November 11–13, 2005: proceedings*. Berlin. 2005, Vol.3746, pp. 480–490.

[62] Vedrickas G., Nemuraitė L. Multipartite structure model of business rules: *Informacinės technologijos verslui 2005: tarptautinės konferencijos pranešimų medžiaga.* Kaunas, 2005, pp. 271–276.

[63] Čeponienė L., Nemuraitė L., Ambrazevičius E. Using state coordinator pattern for transition from design independent to platform independent model *Informacinės technologijos ir valdymas*. Kaunas 2005, T. 34, nr. 3, pp. 263–268.

[64] Čaplinskas Albertas, Gasperovič Jelena. An approach to evaluate quality in use of IS specification language. *Frontiers in artificial intelligence and applications*, Vol. 118 (2005), pp. 152–166

[65] Kapočius K., Butleris R. Repository for Business Rules Based IS Requirements. *INFORMATICA*, 2006, Vol. 17, No. 4, pp. 503–518, 2006.

[66] Emasri, R., Navathe, S.B. *Fundamentals of database systems*, Third Edition. Addison-Wesley, 2002.

[67] Butkienė, R, Butleris, R., Danikauskas, T. The approach to consistency checking of functional requirements specification. *The 6th World Multiconference on Systematics, Cybernetics and informatics. Proceedings of International Conference,* Vol.18, Orlando, USA, 2002, pp. 67–72.

[68] Butkienė, R., Butleris., R. The Approach for the User Requirements Specification. *5th East-European conference ADBIS'2001, Research Communications*, Ed. by A. Čaplinskas, J. Eder, Vilnius, 2001, pp. 225–240.

[69] Kapočius. K. *Business rules structurization models and their application developing information systems*. Doctoral dissertation. Kaunas University of technology, 2006.

[70] Loucopoulos, P., Wan Kadir M., N. BROOD: Business Rules-driven Object Oriented Design. Journal of Database Management, Volume 19, Issue 1 edited by Keng Siau 2008, IGI Global [Online]. Available: http://www-staff.lboro.ac.uk/%7Ebspl/Journals/JDM%202008.pdf. [Accessed Aug. 8, 2008].

[71] Loucopoulos, P., Wan Kadir M., N. Relating evolving business rules to software design. *Journal of Systems Architecture*, 50(7), pp. 367–382.

[72] Balsters, H., Carver, A., Halpin, T., Morgan, T. Modeling dynamic rules in ORM.2nd International Workshop on Object-Role Modelling (ORM 2006). LNCS 4278.Berlin: Springer-Verlag, 2006, pp. 1201–1210.

[73] Jarrar, M., Keet, M., Gordevicius, J. A Lithuanian Verbalization Template for ORM conceptual models and rules. [Online]. Available: http://www.starlab.vub.ac.be/staff/mustafa/orm/verbalization/Lithuanian/

verbalization_LithuanianLithuanian_Ver02.pdf. [Accessed Nov. 19, 2007].

[74] Jarrar, M., Keet, M., and Dongilli, P.: Multilingual verbalization of ORM conceptual models and axiomatized ontologies. *Technical report. STARLab,* Vrije Universiteit Brussel. [Online]. Available: http://www.starlab.vub.ac.be/staff/mustafa/publications/[JKD06a].pdf. [Accessed Feb. 11, 2006].

[75] Troyer, O., Meersman, R.: A Logic Framework for a Semantics of Object-Oriented Data Modeling. OOER 1995, pp. 238–249.

[76] De Troyer, O., Meersman, R., A., Verlinden, P. RIDL* on the CRIS Case: a Workbench for NIAM. In: Computerized Assistance during the Information Systems Life Cyle, eds. Olle T.W., Verrijn-Stuart A.A., Bhabuta L., Elsevier Science Publishers B.V. (North-Holland), 1988, pp. 375–459.

[77] Halpin, T. 2003, 'Verbalizing Business Rules: Part 1', *Business Rules Journal*, Vol. 4, No. 4 [Online]. Available: http://www.BRCommunity.com/a2003/b138.html. [Accessed April. 8, 2004].

[78] Halpin, T. 2003, 'Verbalizing Business Rules: Part 2', *Business Rules Journal*, Vol. 4, No. 6. [Online]. Available: http://www.BRCommunity.com/a2003/b152.html. [Accessed Jun. 8, 2004].

[79] Halpin, T. 2003, 'Verbalizing Business Rules: Part 3', *Business Rules Journal*, Vol. 4, No. 8 [Online]. Available: http://www.BRCommunity.com/a2003/b163.html.

[80] Halpin, T. 2003, 'Verbalizing Business Rules: Part 4', *Business Rules Journal*, Vol. 4, No. 10. [Online]. Available: http://www.BRCommunity.com/a2003/b172.html. [Accessed Jun. 8, 2004].

[81] Halpin, T. 2004, 'Verbalizing Business Rules: Part 5', *Business Rules Journal*, Vol. 5, No. 2. [Online]. Available: http://www.BRCommunity.com/a2004/b179.html. [Accessed Jun. 8, 2004].

[82] Halpin, T. 2004, 'Verbalizing Business Rules: Part 6', *Business Rules Journal*, Vol. 5, No. 4. [Online]. Available: http://www.BRCommunity.com/a2004/b183.html. [Accessed Jun. 8, 2004].

[83] Halpin, T. 2004, 'Verbalizing Business Rules: Part 7', *Business Rules Journal*, Vol. 5, No. 7. [Online]. Available: http://www.BRCommunity.com/a2004/b198.html. [Accessed Jan. 15, 2005].

[84] Halpin, T. 2004, 'Verbalizing Business Rules: Part 8', *Business Rules Journal*, Vol. 5, No. 9. [Online]. Available: http://www.BRCommunity.com/a2004/b205.html. [Accessed Jan. 15, 2005].

[85] Halpin, T. 2004, 'Verbalizing Business Rules: Part 9', *Business Rules Journal*, Vol. 5, No. 12. [Online]. Available: http://www.BRCommunity.com/a2004/b215.html. [Accessed Jan. 15, 2005].

[86] Halpin, T. 2005, 'Verbalizing Business Rules: Part 10', *Business Rules Journal*, Vol. 6, No. 4. [Online]. Available: http://www.BRCommunity.com/a2005/b229.html. [Accessed Apr. 6, 2005].

[87] Halpin, T. 2005, 'Verbalizing Business Rules: Part 11', *Business Rules Journal*, Vol. 6, No. 6. [Online]. Available: http://www.BRCommunity.com/a2005/b238.html. [Accessed Nov. 24, 2005].

[88] Halpin, T. 2005, 'Verbalizing Business Rules: Part 12', *Business Rules Journal*, Vol. 6, No. 10. [Online]. Available: http://www.BRCommunity.com/a2005/b252.html. [Accessed Nov. 24, 2005].

[89] Halpin, T. 2005, 'Verbalizing Business Rules: Part 13', *Business Rules Journal*, Vol. 6, No. 12. [Online]. Available: http://www.BRCommunity.com/a2005/b261.html. [Accessed Nov. 24, 2005].

[90] Halpin, T. 2006, 'Verbalizing Business Rules: Part 14', *Business Rules Journal*, Vol. 7, No. 4. [Online]. Available: http://www.BRCommunity.com/a2006/b283.html. [Accessed May. 26, 2006].

[91] Halpin, T. 2006, 'Verbalizing Business Rules: Part 15', *Business Rules Journal*, Business Rules Journal, Vol. 7, No. 6. [Online]. Available: http://www.BRCommunity.com/a2006/b294.html. [Accessed May. 26, 2006].

[92] Krogstie, J., Halpin, T., Siau, K.: Two Meta-Models for Object-Role Modeling. *In: Krogstie, J., Halpin, T., Siau, K. (Eds.): Information Modeling Methods and Methodologies*, Idea Group Publishing, 2005, pp. 17–42

[93] Cuyler, D., Halpin, T., Metamodels for Object-Role Modeling, 2003). [Online]. Available: http://www.emmsad.org/2003/Final%20Copy/26.pdf. [Accessed May. 26, 2006].

[94] Herbst, H., "Business Rules in Systems Analysis: A Meta-model and Repository System," *Information Systems*, vol. 21, no. 2, April 1996, 1996, pp. 147–66.

[95] Herbst, H. *Business Rule-Oriented Conceptual Modeling*. Germany: Physica-Verlag, 1997.

[96] Loucopoulos, P., & Layzell, P. J. Rubric: A rule based approach for the development of information systems. *Paper presented at the 1st European workshop on fault diagnosis, reliability and related knowledge based approaches*, Rhodes, 1986.

[97] van Assche, F., Layzell, P. J., Loucopoulos, P., & Speltinex, G. Rubric: A rule-based representation of information system constructs. *Paper presented at the ESPRIT Conference*, Brussels, Belgium. 1998.

[98] Martin, J. *Information engineering.* Prentice- Hall, 1998.

[99] Rosca, D., Greenspan, S., Feblowitz, M., & Wild, C. A decision support methodology in support of the business rules lifecycle. *Paper presented at the International Symposium on Requirements Engineering (ISRE'97)*, Annapolis, MD., 1997.

[100] Rosca, D., Greenspan, S., & Wild, C. Enterprise modeling and decision-support for automating the business rules lifecycle. *Automated Software Engineering*, 9(4), 2002, pp. 361–404.

[101] Rosca, D., Greenspan, S., Wild, C., Reubenstein, H., Maly, K., & Feblowitz, M. Application of a decision support mechanism to the business rules lifecycle. *Paper presented at the 10th Knowledge-Based Software Engineering Conference (KBSE95)*, Boston, MA., 1995.

[102] Hay, D., & Healy, K. A. Business rules: What are they really? GUIDE (The IBM User Group), 1998, [Online]. Available: http://www.BusinessRulesGroup.org/. [Accessed May. 28, 2006].

[103] Hay, D., & Healy, K. A. Defining business rules ~ what are they really? (No. Rev 1.3): the Business Rules Group. 2000.

[104] Bajec, M. and Krisper, M., "A methodology and tool support for managing business rules in organisations," *Information Systems*, vol. 30, no. 6, 2005, pp. 423–443.

[105] Damianou, N., Dulay, N., Lupu, E., Sloman, M. The Ponder Policy Specification Language. *In proceedings of Workshop on Policies for Distributed Systems and Networks (POLICY 2001)*. Springer-Verlag, LNCS 1995, Bristol, UK, 2001, pp. 18–39.

[106] Bradshaw, J. M., Beautement, P., Bunch, L., Drakunov S. V., et al: Making agents Acceptable to People. *In N. Zhong, J. Liu (eds): Handbook of Intelligent Information Technology*. IOS Press, Amsterdam, the Netherlands, 2003.

[107] Wright, S., Chadha, R., Lapiotis G. (eds.): *Special Issue on Policy Based Networking. IEEE Network*, Vol. 16, No. 2, March, 2002, pp. 8–56

[108] Brickley, D., and Guha, R. V. *RDF vocabulary description language 1.0: RDF schema. Technical report*, W3C Working Draft, 2004.

[109] The Rule Markup Initiative. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*, version 0.6., 2004.

[110] Kagal, L., Finin, T., and Johshi, A. A Policy Language for Pervasive Computing Environment. *Policy 2003: Workshop on Policies for Distributed Systems and Networks.* Springer-Verlag, 2003.

[111] Uszok, A., Bradshaw, J., Jeffers, R., Suri, N., et al. 2003. KAoS Policy and Domain Services: Toward a Description – Logic Approach to Policy Representation, Deconfliction, and Enforcement. *Policy 2003: Workshop on Policies for Distributed Systems and Networks*. Springer-Verlag.

[112] Hirtle, D. TRANSLATOR: A TRANSlator from LAnguage TO Rules. *Canadian Symposium on Text Analysis (CaSTA)*, Fredericton, Canada, 2006.

[113] Marchiori, M. Towards a People's Web: Metalog. *In IEEE/WIC/ACM International Conference on Web Intelligence*, 2004, pp. 320–326.

[114] Business Rules Team. Semantics of Business Vocabulary & Business Rules (SBVR). *W3CWorkshop on Rule Languages for Interoperability Position Paper*. [Online]. Available: http://www.w3.org/2004/12/rules-ws/paper/85. [Accessed Oct. 8, 2006].

[115] Sowa, J. F., Common Logic Controlled English, 2004. [Online]. Available: http://www.jfsowa.com/clce/specs.htm. [Accessed Nov. 8, 2006].

[116] Pease, A., and Murray, W. An English to Logic Translator for Ontologybased Knowledge Representation Languages. *In Proceedings of the 2003 IEEE International Conference on Natural Language Processing and Knowledge Engineering*, 2003, pp. 777–783.

[117] Schwitter, R. and Tilbrook, M. Controlled Natural Language meets the Semantic Web. *In Proceedings of the Australasian Language Technology Workshop*, 2004, pp. 55–62.

[118] Schwitter, R., Ljungberg, A., and Hood, D. ECOLE – A Look-ahead Editor for a Controlled Language. *In Controlled Translation, Proceedings of EAMT-CLAW03*, 2003, pp. 141–150.

[119] Krammer, M. I. Business rules: Automating business policies and practicies. *Distributed Computing Monitor*, 1997.

[120] OMG, SBVR – Semantics of Business Vocabulary and Business Rules", Adopted Specification, 2006, [Online]. Available: http://www.omg.org/ docs/dtc/06-03-02.pdf. [Accessed May. 8, 2007].

[121] Ahrendt, W., et al. *The KeY Tool. Software and Systems Modeling*, Springer, 2005.

[122] Beckert, B., U. Keller, P. H. Schmitt. Translating the Object Constraint Language into first-order predicate logic. *Proceedings, VERIFY, Workshop at Federated Logic Conferences (FLoC)*, Copenhagen, Denmark, 2002.

[123] Hobbs, J., Israel, D., Principles of template design. *in Proceedings of the ARPA Workshop on Human Language Technology*, M. Kaufmann, 1994, pp. 172–176

[124] Filatova, E., Hatzivassiloglouy, V., McKeown, K. Automatic Creation of Domain Templates. Proceedings of the COLING/ACL, 2006, pp. 207–204.

[125] Greenfield, J., Short K., Cook S. and Kent S., *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*, Wiley, 2004.

[126] Clark, T., Evans A., Sammut P. and Willans J., *Applied metamodelling: A foundation for language-driven development*, 2005. [Online]. Available: http://albini.xactium.com. [Accessed Oct. 8, 2006].

[127] OMG, *MOF Model to Text Transformation Language* (Request For Proposal), OMG Document ad/2004-04-07, OMG, 2004.

[128] de Lara, J. and Vangheluwe H., Using AToM3 as a meta-case tool. *Proceedings of the 4th International Conference on Enterprise Information Systems (ICEIS)*, 2002, pp. 642–649.

[129] Fondement, F. and Baar T., Making Metamodels Aware of Concrete Syntax. *European Conference on Model Driven Architecture (ECMDA)*, LNCS 3748, 2005, pp. 190–204.

[130] Petre, M. Why looking isn't always seeing: readership skills and graphical programming. *Commun. ACM,* 38(6):33–44, 1995.

[131] Groonniger, H., Krahn, H., Rumpe, B., Schindler, M., and Voolkel, S. Text-based Modeling. 4th International Workshop on Software Language Engineering, 2007. [Online]. Available: http://www.sse-tubs.de/ publications/Groenniger_et_al_ATEM_07.pdf. [Accessed Sept. 8, 2007].

[132] Eclipse Website. [Online]. Available: http://www.eclipse.org. [Accessed Dec. 11, 2007].

[133] Krahn, H., Rumpe, B., and Voolkel, S. Eficient Editor Generation for Compositional DSLs in Eclipse. The 7th OOPSLA Workshop on Domain-Specific Modeling, 2004. [Online]. Available: http://www.sse-tubs.de/publications/KRV_Editor.pdf. [Accessed Sept. 24, 2007].

[134] Krahn, H., Rumpe, B. and Volkel, S. Integrated Definition of Abstract and Concrete Syntax for Textual Languages. *In Proceedings of Models 2007*, 2007.

[135] Matula, M. NetBeans Metadata Repository. [Online]. Available: http://mdr.netbeans.org/MDR-whitepaper.pdf [Accessed Sept. 24, 2007].

[136] Kieburtz, R.,B., McKinney, L., Bell, J., M., Hook, J., Kotov, A., Lewis, J., Oliva, D., Sheard, T., Smith, I., and Walton, L. A software engineering experiment in software component generation. *In*

Proceedings of the 18th International Conference on Software Engineering (ICSE), 1996, pp. 542–552.

[137] Kent, S. Model driven engineering. *In Michael J. Butler, Luigia Petre, and Kaisa Sere, editors, Proceedings of Third International Conference on Integrated Formal Methods (IFM 2002)*, volume 2335 of LNCS, Springer, 2002, pp. 286–298.

[138] Mellor, S.,J., Clark, A.,N., and Futagami, T. Guest editors' introduction: Model-driven development. *IEEE Software*, 20(5), 2003, pp. 14–18.

[139] OMG. *Meta Object Facility (MOF) Core, v2.0, OMG Document formal/06-01-01*, 2005, [Online]. Available: http://www.omg.org/cgi-bin/doc?formal/2006-01-01. [Accessed Jan. 24, 2007].

[140] OMG. *XML Metadata Interchange (XMI) 2.0.* OMG Document formal/03-05-02, May 2003.

[141] Kurtz, B., L. *Formal Syntax and Semantics of Programming Languages, A Laboratory Based Approach*, Addison-Wesley Publishing Company, 1995.

[142] Hofstadter, D. R., et. al. *An eternal golden braid*. Vintage Books, New York, 1979.

[143] Cointe, P. Metaclasses are first class: the ObjVlisp model. *In Norman Meyrowitz, editor, Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications*, ACM, October 1987, pp 156–165.

[144] IBM, Emfatic, [Online]. Available: http://www.alphaworks.ibm.com/tech/emfatic. [Accessed Sept. 24, 2007].

[145] Jouault F., Bézivin J., KM3: A DSL for Metamodel Specification, *FMOODS 2006*, pp. 171–185.

[146] Laukaitis, A., Vasilecas, O. Natural Language as Programming Paradigm in Data Exploration Domain. *Information Technology And Control, Kaunas, Technologija,* 2007, Vol. 36, No. 1, pp. 30–36. [Online]. Available: http://itc.ktu.lt/itc361/Laukait361.pdf. [Accessed Mar. 24, 2008].

[147] Mens T. and Van Gorp P., A Taxonomy of Model Transformation and Its Application to Graph Transformation, *Proceedings of the International Workshop on Graph and Model Transformation*, Tallinn, Estonia, 2005, pp. 7–23.

[148] OMG. *MOF QVT Final Adopted Specification, OMG Document ptc/2005-11-01*, 2005. [Online]. Available: http://www.omg.org/docs/ptc/05-11-01.pdf. [Accessed Sept. 24, 2007].

[149] Jouault, F., Bezivin, J., Kurtev, I. TCS: a DSL for the Specification of Textual Concrete Syntaxes in Model Engineering *GPCE '06,* October 22–26, 2006, Portland, Oregon, ACM.

[150] Fondement, F., Baar, T.: Making Metamodels Aware of Concrete Syntax. *In Hartman, A., Kreische, D., eds.: Model Driven Architecture - Foundations and Applications, First European Conference, ECMDA-FA 2005,* Nuremberg, Germany, November 7-10, 2005, Proceedings. Volume 3748 of Lecture Notes in Computer Science, Springer, 2005, pp. 190–204.

[151] Xtext Reference Documentation, [Online]. Available: www.eclipse.org/gmt/oaw/doc/4.1/r80_xtextReference.pdf. [Accessed Jan. 28, 2008].

[152] Efftinge, S., Extends language reference. [Online]. Available: http://www.eclipse.org/gmt/oaw/doc/4.1/r25_extendReference.pdf. [Accessed Jan. 29, 2008].

[153] Fondement, F., Schnekenburger, R., Gérard, S., and Muller, P.-A. *Metamodel-Aware Textual Concrete Syntax Specification.* Technical report, 2006. [Online]. Available: http://fondement.free.fr/lgl/docs/papers/tcss.pdf. [Accessed Apr. 24, 2008].

[154] LPG parser generator. [Online]. Available: http://sourceforge.net/projects/lpg. [Accessed Apr. 24, 2008].

[155] EclipseCON. [Online]. Available: http://www.eclipsecon.org/2006/Home.do. [Accessed Apr. 24, 2008].

[156] Charles, P., Dolby, J., Fuhrer, R., M., Sutton Jr., S., M., and Vaziri, M. SAFARI: a meta-tooling framework for generating language-specific IDE's. *In OOPSLA Companion*, 2006, pp. 722–723.

[157] Textual editing framework tool project homepage. [Online]. Available: http://www2.informatik.hu-berlin.de/sam/meta-tools/tef/tool.html. [Accessed Apr. 24, 2008].

[158] Parr, T., and Quong, R. ANTLR: A Predicated-LL(k) parser generator. *Journal of Software Practice and Experience*, 25(7), 1995, pp. 789–810.

[159] Dirckze, R. (spec. leader), "Java Metadata Interface (JMI) Specification Version 1.0". [Online]. Available: http://jcp.org/aboutJava/communityprocess/final/jsr040/index.html. [Accessed Apr. 24, 2008].

[160] ModFact Project Home Page. [Online]. Available: http://modfact.lip6.fr. [Accessed Apr. 24, 2008].

[161] Alcatel, Softeam, Thales, TNI-Valiosys, Codagen Corporation, et al. *MOF Query/Views/Transformations, Revised Submission*. OMG Document: ad/03-08-05., 2005.

[162] Jouault, F., Kurtev, I.: Transforming models with ATL. *In: Satellite Events at the MoDELS 2005 Conference*. Volume 3844 of Lecture Notes in Computer Science., Springer-Verlag, 2006, pp. 128–138.

[163] Kalnins, A., Celms, E., Sostaks, A. Tool support for MOLA. *Fourth International Conference on Generative Programming and Component Engineering (GPCE'05), Workshop on Graph and Model Transformation (GraMoT)*, Tallinn, Estonia, September 2005. [Online]. Available: http://melnais.mii.lu.lv/audris/MOLAtoolGRAMOTFin.pdf. [Accessed Apr. 24, 2008].

[164] Czarnecki, K., and Helsen, S. Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3), 2006. pp. 621–645.

[165] Völter, M.: *openArchitectureWare 4 – Fact Sheet*, 2007.

[166] Braun,P., Lötzbeyer, H., Schätz, B., Slotosch, O., Consistent Integration of Formal Methods *S. Graf, M. Schwartzbach (Eds.): Tools and Algorithms for the Construction and Analysis of Systems: 6th International Conference, TACAS 2000*, LNCS 1785, Springer-Verlag Berlin, Germany, March/April, 2000, pp. 48–62.

[167] Hendryx, S., A User's Perspective of the OMG Business Rules Proposal. BRCommunity portal. [Online]. Available: http://www.brcommunity.com/b228.php. [Accessed Apr. 24, 2008].

[168] OMG, *UML 2.0 Superstructure Specification, OMG document ptc/03-08-02*, September 2, 2003.

[169] OMG, *MDA Guide Version 1.0.1, in Object Management Group, J. Miller, et al., Editors*. 2003.

[170] IBM: Rational Rose tool, 2006. [Online]. Available: http://www-306.ibm.com/software/rational/ [Accessed Apr. 24, 2008].

[171] Eclipse Project: Eclipse UML2. [Online]. Available: http://www.eclipse.org/uml2. [Accessed Apr. 24, 2008].

[172] LCI team: Object constraint language environment. Computer Science Re-search Laboratory,"BABES–BOLYAI" University, Romania, (2005) [Online]. Available: http://lci.cs.ubbcluj.ro/ocle/. [Accessed Apr. 24, 2008].

[173] Gentleware: Poseidon UML tool. [Online]. Available: http://www.gentleware.com/. [Accessed Apr. 25, 2008].

[174] Demuth B.: The Dresden OCL Toolkit and its Role in Information Systems Development. *In O. Vasilecas at al (Eds). Proc. of Thirteenth International Conference on Information Systems Development. Advances in Theory, Practice and Education*. Vilnius, Technika, 2004. [Online]. Available: http://dresden-ocl.sourceforge.net/. [Accessed Apr. 24, 2008].

[175] Brucker A.D., Doser J., Wolff B.: Semantic Issues of OCL: Past, Present, and Future. *In: OCL for (Meta-)Models in MultipleApplication Domains, Available as Technical Report*, University Dresden, number TUD-FI06-04-Sept, 2006, pp. 213–228. http://www.brucker.ch/bibliography/download/2006/brucker.ea-semantic-2006-b.pdf. [Accessed May. 10, 2008].

[176] Gogolla M., Bohling, J., Richters, M.: Validation of UML and OCL Models by Automatic Snapshot Generation. *In: Booch, G., Stevens, P., Whittle, J., (Eds.): Proc. 6th Int. Conf. Unified Modeling Language (UML'2003)*. Springer, Berlin, LNCS 2863, 2003, pp. 265–279.

[177] Bloesch, A., Halpin, T.: ConQuer: a conceptual query language. *In: Thalheim B. (Eds.):Proc. 15th International Conference on Conceptual Modeling ER'96* (Cottbus, Germany), LNCS 1157, Springer-Verlag, 1996, pp. 121–133

[178] Bézivin, J., Hammoudi, S., Lopes, D., Jouault, F. *An Experiment in Mapping Web Services to Implementation Platforms. Technical report: 04.01*, LINA, University of Nantes. [Online]. Available: http://lina.atlanstic.
net/documents/RR_pdfs/RR-LINA-0401.pdf. [Accessed Apr. 10, 2008].

[179] Staron, M., Kuzniarz, L., Wallin. L. A Case Study on a Transformation Focused Industrial MDA Realization. 3rd Workshop in Software Model Engineering, 2004. [Online]. Available: http://www.metamodel.com/ wisme-2004/present/7.pdf. [Accessed Apr. 11, 2008].

List of author's papers on the topic of dissertation

In the reviewed scientific periodical publications

[1A] S. Sosunovas, O. Vasilecas. Practical application of BRTL approach for financial reporting domain. *Information Technologies and Control*, Kaunas, Technologija, ISSN 1392-124X, 2008, Vol. 37, No. 2, pp. 106–113. (Thomson ISI Web of Science)

[2A] S. Sosunovas, O. Vasilecas. Transformation of business rules models in information systems development process. *Scientific Papers University of Latvia, Database and Information Systems*. Latvias Universitate. ISSN 1407-2157. 2004, Vol. 672, pp. 373–384. (Thomson ISI Master Journal)

[3A] S. Sosunovas, O. Vasilecas. Tool-supported method for the extraction of OCL from ORM models. W. Abramowicz (Eds.): *Proc. 10th International Conference on Business Information Systems BIS 2007*, Poznan, Poland. LNCS 4439. Springer-Verlag. ISSN 0302-9743. 2007, pp. 449–463. (Thomson ISI Proceedings)

[4A] S. Sosunovas, O. Vasilecas. Verslo taisyklių modeliavimas OCL kalba. *Lietuvos matematikos rinkinys*, Vilnius, MII. TEV, ISSN 0132-2818. Vol. 44, 2004, pp. 369–376.

[5A] S. Sosunovas, O. Vasilecas. A model driven approach to business rules model transformations. *Frontiers in Artificial Intelligence and Applications*. IOS Press. Vol. 118, ISSN 0922-6389, 2005, pp. 198–208. (Thomson ISI Proceedings)

[6A] S. Sosunovas, O. Vasilecas. Principles of business rules transformation in BRMD approach. *Scientific Proceedings of Riga Technical University, 5th Series, Computer Science, Applied Computer Systems*, Riga. RTU Publishing, vol. 22, ISSN 1407-7493, 2005, pp. 49–56

In the other editions

[7A] D. Bugaite, O. Vasilecas, S. Sosunovas. Template-based approach for rules transformation into SQL triggers. In: *Proc. of 8th International IEEE Baltic Conference on Databases and Information Systems*, Tallinn, Estonia. Tallinn University of Technology Press, 2008, ISBN 978-9985-59-789-7, pp. 253–263. (Thomson ISI Proceedings)

[8A] S. Sosunovas, O. Vasilecas. An experiment in model driven specification and transformation of business rules. In: *Proceedings of 14th International Conference on Information and Software Technologies: Information Technologies 2008 (IT 2008)*, Kaunas, Lithuania, Technologija, ISSN 2029-0020, 2008. pp. 343–352. (Thomson ISI Proceedings)

[9A] S. Sosunovas, O. Vasilecas. Precise notation for business rules templates. In: O. Vasilecas et al. (eds.), *Proc. of 7th International IEEE Baltic Conference on Databases and Information Systems*, Vilnius, Lithuania. Technika. ISBN: 1-4244-0345-6. 2006, pp. 55–60. (Thomson ISI Proceedings)

[10A] O. Vasilecas, S. Sosunovas. Preparing business rules templates and abject role modelling for transformations. *In the proceedings of CompSysTech'05 International Conference. 16–17 June 2005, Varna, Bulgaria.* ISBN 954-9641-42-2. 2005, pp. II.2-1–II.2-6.

[11A] O. Vasilecas., S. Sosunovas. Metamodel based integration of ORM and business rules templates. In: *R. Simutis (Eds) Proceedings of the International Conference Information Technologies for Business 2005*. Vilnius University, Kaunas Faculty of Humanities. ISBN 9955-09-871-6. 2005, pp. 77–83.

[12A] S. Sosunovas, O. Vasilecas. Business rules based model driven approach in information systems development. In: *R. Romansky (Eds) Proceedings of the 18th International Conference on Systems for Automation of Engineering and Research.* September 24–26, 2004, Varna, Bulgaria. ISBN 954-438-428-6. 2004, pp. 35–40.

[13A] S. Sosunovas, O.Vasilecas. On formalizing the ORM and business rules templates. In: *Proc. of the Fourteenth International Conference on Information Systems Development 2005 (ISD'2005)*, Karlstad, Sweden. 2005, pp. 171–182.

[14A] S. Sosunovas, O. Vasilecas. On transformation of business rules templates to the OCL, In: *Proceedings of "Informacinės technologijos'2006"*. Kaunas, Technologija. 2006, pp. 632–638.

[15A] S. Sosunovas, O. Vasilecas. Verslo taisyklių modelių transformacija meta-modelių pagrindu. *Lietuvos mokslas ir pramonė. Konferencijos "Informacinės technologijos'2004" pranešimų medžiaga.* Kaunas, Technologija. ISBN 9955-09-588-1. 2004, pp. 585–592.

APPENDIXES

Appendix A. ORM textual syntax

```
ormStatementSet
    :   (   objectTypeExp
        |   predicateExp
        |   eUniquenessConstraintPExp
        |   eUniquenessConstraintExp
        |   subsetExp
        |   equalityExp
        |   exclusionExp
        )*
    ;
objectTypeExp
    :   (   entityExp
        |   valueExp
        )
        SEMI
    ;
predicateExp
    :   "Predicate"
        (   IDENT
            (   EXCLAMATION
            |
            )
        |
```

```
        )
        ( LPAREN predicateRoleExp ( COMMA predicateRoleExp )* RPAREN )
        (   aliasExp
        |
        )
        LCURLY ( sentenceExp )* ( internalUniqConstraintsExp )* RCURLY
    ;
eUniquenessConstraintPExp
    :   "EUniquenessConstraintP" contraintBodyExp SEMI
    ;
eUniquenessConstraintExp
    :   "EUniquenessConstraint" contraintBodyExp SEMI
    ;
subsetExp
    :   "Subset" contraintBodyExp SEMI
    ;
equalityExp
    :   "Equality" contraintBodyExp SEMI
    ;
exclusionExp
    :   "Exclusion" contraintBodyExp SEMI
    ;
entityExp
    :   ( "Entity" identExp ( COMMA identExp )* )
    ;
valueExp
    :   ( "Value" identExp ( COMMA identExp )* )
    ;
identExp
    :   IDENT
        (   LPAREN IDENT RPAREN
            (   PLUS
            |
            )
        |
        )
        (   COLON
            (   ( IDENT )
```

```
            |   ( LPAREN IDENT ( COMMA IDENT )* RPAREN )
            )
        |
        )
        (   LCURLY valueListExp RCURLY
        |
        )
    ;
valueListExp
    :   ( STRING ( COMMA STRING )* )
    |   ( INTLIT DOTDOT INTLIT )
    ;
predicateRoleExp
    :   ( IDENT )
        (   DOT
        |
        )
        (   aliasExp
        |
        )
    ;
aliasExp
    :   "as" IDENT
    ;
sentenceExp
    :
        (   "Reading"
        |   "Sentence"
        |
        )
        ( ( IDENT
            |   INTLIT
            |   DOTDOTDOT
            )* ) SEMI
    ;
internalUniqConstraintsExp
    :   LT ( ( IDENT
            |   INTLIT
            )
```

```
            (   COMMA IDENT
            |   INTLIT
            )* ) GT
    ;
contraintBodyExp
    :   ( LPAREN ( predicateRoleRefExp ( COMMA predicateRoleRefExp )* )
RPAREN )
    ;
predicateRoleRefExp
    :   IDENT DOT IDENT
    ;
```

Appendix B. Experiment models and rules

B 1. Experiment ORM model (textual notation)

```
Entity GL(kodas), CGR(kodas), ARP(kodas), ACC(kodas);
Entity Ataskaita([Ataskaitos kodas]), Eilute([Eilutes kodas]);
Value [Ataskaitos pavadinimas], [Eilutes pavadinimas];
Entity Stulpelis([Eilutes kodas]);


Predicate (Ataskaita, Eilute)
{
    1 sudaryta is eiluciu 2;
}
Predicate (Ataskaita, Stulpelis)
{
    1 sudaryta is stulpeliu 2;
}
Predicate (Ataskaita, [Ataskaitos pavadinimas])
{
```

```
    1 turi pavadinima 2;
}
Predicate (Eilute, [Eilutes pavadinimas])
{
    1 turi pavadinima 2;
}
Predicate (GL, CGR)
{
    1 turi CGR;
}
Predicate (GL, ARP)
{
    1 turi 2;
}
Predicate (GL, ACC)
{
    1 turi 2;
}
Predicate (GL, Eilute)
{
    1 teigiami likuciai rodomi eiluteje 2;
}
Predicate (GL, Eilute)
{
    1 neigiami likuciai rodomi eiluteje 2;
}
Predicate (GL, Eilute)
{
    1 likuciai rodomi eiluteje 2;
}
```

B 2. Experiment rules

B.2.1. "Row name" rules

Eilute {1.} turi pavadinima {Grynieji pinigai ir lėšos centriniuose bankuose}

Eilute {2.} turi pavadinima {Prekybinis finansinis turtas}

Eilute {2.1.} turi pavadinima {Išvestinės finansinės priemonės}

Eilute {2.2.} turi pavadinima {Nuosavybės vertybiniai popieriai}

Eilute {2.3.} turi pavadinima {Skolos vertybiniai popieriai}

Eilute {2.4.} turi pavadinima {Paskolos ir kiti išankstiniai mokėjimai}

Eilute {3.} turi pavadinima {Tikrąja verte vertinamas finansinis turtas}

Eilute {3.1.} turi pavadinima {Nuosavybės vertybiniai popieriai}

Eilute {3.2.} turi pavadinima {Skolos vertybiniai popieriai}

Eilute {3.3.} turi pavadinima {Paskolos ir kiti išankstiniai mokėjimai}

Eilute {4.} turi pavadinima {Parduoti turimas finansinis turtas}

Eilute {4.1.} turi pavadinima {Nuosavybės vertybiniai popieriai}

Eilute {4.2.} turi pavadinima {Skolos vertybiniai popieriai}

Eilute {4.3.} turi pavadinima {Paskolos ir kiti išankstiniai mokėjimai}

Eilute {5.} turi pavadinima {Paskolos ir gautinos sumos įskaitant išperkamąją nuomą}

Eilute {5.1.} turi pavadinima {Skolos vertybiniai popieriai}

Eilute {5.2.} turi pavadinima {Paskolos ir kiti išankstiniai mokėjimai}

Eilute {6.} turi pavadinima {Investicijos laikomos iki termino pabaigos}

Eilute {6.1.} turi pavadinima {Skolos vertybiniai popieriai}

Eilute {6.2.} turi pavadinima {Paskolos ir kiti išankstiniai mokėjimai}

Eilute {7.} turi pavadinima {Išvestinės finansinės priemonės apsidraudimo sandoriai}

Eilute {7.1.} turi pavadinima {Tikrosios vertės apdraudimas}

Eilute {7.2.} turi pavadinima {Pinigų srautų apdraudimas}

Eilute {7.3.} turi pavadinima {Grynosios investicijos į užsienio įmonę apdraudimas}

Eilute {7.4.} turi pavadinima {Tikrosios vertės apdraudimas nuo palūkanų normos rizikos}

Eilute {7.5.} turi pavadinima {Pinigų srautų apdraudimas nuo palūkanų normos rizikos}

Eilute {8.} turi pavadinima {Apdraustųjų straipsnių tikrosios vertės pokyčiai sudarant portfelio apdraudimo nuo palūkanų normos rizikos sandorius}

Eilute {9.} turi pavadinima {Materialusis turtas}

Eilute {9.1.} turi pavadinima {Nekilnojamasis turtas įranga ir įrenginiai}

Eilute {9.2.} turi pavadinima {Investicinis turtas}

Eilute {10.} turi pavadinima {Nematerialusis turtas}

Eilute {10.1.} turi pavadinima {Prestižas}

Eilute {10.2.} turi pavadinima {Kitas}

Eilute {11.} turi pavadinima {Investicijos į dukterines asocijuotąsias ir bendrąsias įmones įskaitant prestižą kai apskaitai taikomas nuosavybės metodas}

Eilute {12.} turi pavadinima {Mokestinis turtas}

Eilute {12.1.} turi pavadinima {Einamojo laikotarpio mokesčių}

Eilute {12.2.} turi pavadinima {Atidėtųjų mokesčių}

Eilute {13.} turi pavadinima {Kitas turtas}

Eilute {14.} turi pavadinima {Parduoti laikomas ilgalaikis turtas ir perleidžiamos turto grupės}

Eilute {15.} turi pavadinima {Centrinių bankų indėliai}

Eilute {16.} turi pavadinima {Prekybiniai finansiniai įsipareigojimai}

Eilute {16.1.} turi pavadinima {Išvestinės finansinės priemonės}

Eilute {16.2.} turi pavadinima {Įsipareigojimai parduoti pasiskolintą finansinį turtą kuris nėra nuosavybė}

Eilute {16.3.} turi pavadinima {Kredito įstaigų indėliai}

Eilute {16.4.} turi pavadinima {Indėliai išskyrus kredito įstaigų indėlius}

Eilute {16.5.} turi pavadinima {Skolų įsipareigojimai įskaitant obligacijas kurias ketinama atpirkti artimiausiu metu}

Eilute {16.6.} turi pavadinima {Kiti įsipareigojimai}

Eilute {17.} turi pavadinima {Tikrąja verte vertinami finansiniai įsipareigojimai}

Eilute {17.1.} turi pavadinima {Kredito įstaigų indėliai}

Eilute {17.2.} turi pavadinima {Indėliai išskyrus kredito įstaigų indėlius}

Eilute {17.3.} turi pavadinima {Skolų įsipareigojimai įskaitant obligacijas}

Eilute {17.4.} turi pavadinima {Subordinuotosios paskolos}

Eilute {17.5.} turi pavadinima {Kiti įsipareigojimai}

Eilute {18.} turi pavadinima {Amortizuota savikaina vertinami finansiniai įsipareigojimai}

Eilute {18.1.} turi pavadinima {Kredito įstaigų indėliai}

Eilute {18.2.} turi pavadinima {Indėliai išskyrus kredito įstaigų indėlius}

Eilute {18.3.} turi pavadinima {Skolų įsipareigojimai įskaitant obligacijas}

Eilute {18.4.} turi pavadinima {Subordinuotosios paskolos}

Eilute {18.5.} turi pavadinima {Kiti įsipareigojimai}

Eilute {19.} turi pavadinima {Finansiniai įsipareigojimai susiję su perleidžiamu finansiniu turtu}

Eilute {20.} turi pavadinima {Išvestinės finansinės priemonės apsidraudimo sandoriai}

Eilute {20.1.} turi pavadinima {Tikrosios vertės apdraudimas}

Eilute {20.2.} turi pavadinima {Pinigų srautų apdraudimas}

Eilute {20.3.} turi pavadinima {Grynosios investicijos į užsienio įmonę apdraudimas}

Eilute {20.4.} turi pavadinima {Tikrosios vertės apdraudimas nuo palūkanų normos rizikos}

Eilute {20.5.} turi pavadinima {Pinigų srautų apdraudimas nuo palūkanų normos rizikos}

Eilute {21.} turi pavadinima {Apdraustųjų straipsnių tikrosios vertės pokyčiai sudarant portfelio apdraudimo nuo palūkanų normos rizikos sandorius}

Eilute {22.} turi pavadinima {Atidėjiniai}

Eilute {22.1.} turi pavadinima {Restruktūrizavimui}

Eilute {22.2.} turi pavadinima {Nebaigtoms teisinėms byloms ir mokestiniams ginčams}

Eilute {22.3.} turi pavadinima {Pensijoms ir kitoms išmokoms darbuotojams}

Eilute {22.4.} turi pavadinima {Kreditavimo įsipareigojimams ir garantijoms}

Eilute {22.5.} turi pavadinima {Įsipareigojimams pagal sutartis}

Eilute {22.6.} turi pavadinima {Kiti atidėjiniai}

Eilute {23.} turi pavadinima {Mokestiniai įsipareigojimai}

Eilute {23.1.} turi pavadinima {Einamojo laikotarpio mokesčių}

Eilute {23.2.} turi pavadinima {Atidėtųjų mokesčių}

Eilute {24.} turi pavadinima {Kiti įsipareigojimai}

Eilute {25.} turi pavadinima {Akcinis kapitalas apmokamas pareikalavus}

Eilute {26.} turi pavadinima {Įsipareigojimai susiję su parduoti laikomomis perleidžiamomis turto grupėmis}

Eilute {27.} turi pavadinima {Kapitalas}

Eilute {27.1.} turi pavadinima {Apmokėtasis kapitalas}

Eilute {27.2.} turi pavadinima {Neapmokėtasis kapitalas}

Eilute {28.} turi pavadinima {Emisinis skirtumas}

Eilute {29.} turi pavadinima {Kita nuosavybė}

Eilute {29.1.} turi pavadinima {Su sudėtinėmis finansinėmis priemonėmis susijusi nuosavybė}

Eilute {29.2.} turi pavadinima {Kita}

Eilute {30.} turi pavadinima {Perkainojimo rezervai kiti vertės koregavimai}

Eilute {30.1.} turi pavadinima {Materialiojo turto}

Eilute {30.2.} turi pavadinima {Nematerialiojo turto}

Eilute {30.3.} turi pavadinima {Grynosios investicijos į užsienio įmonę apdraudimas}

Eilute {30.4.} turi pavadinima {Valiutos keitimo skirtumų}

Eilute {30.5.} turi pavadinima {Pinigų srautų apdraudimas}

Eilute {30.6.} turi pavadinima {Parduoti turimo turto}

Eilute {30.7.} turi pavadinima {Parduoti laikomo ilgalaikio turto ar perleidžiamų turto grupių}

Eilute {30.8.} turi pavadinima {Kiti}

Eilute {31.} turi pavadinima {Rezervai}

Eilute {32.} turi pavadinima {Supirktos nuosavos akcijos}

Eilute {33.} turi pavadinima {Einamųjų metų pelnas}

Eilute {34.} turi pavadinima {Išankstiniai dividendai}

Eilute {35.} turi pavadinima {Mažumos nuosavybė}

Eilute {35.1.} turi pavadinima {Perkainojimo rezervai}

Eilute {35.2.} turi pavadinima {Kita}

B.2.2. "Report algorithm" rules

GL {3201} ARP {1003} VISI CGR likuciai rodomi eiluteje {1.}

GL {3301} ARP {1004} likuciai rodomi eiluteje {1.} siu saskaitu kreditiniai (neigiami) likuciai ju ne mazina, o rodomi eil. {24.}

GL {73103} ARP {1010} CGR {28} likuciai rodomi eiluteje {1.} siu saskaitu kreditiniai (neigiami) likuciai ju ne mazina, o rodomi eil. {24.}

GL {3001|3101|30101|6001|6101|6102|6103|6104|6105|30102|30103|30104|30105|30106|300108|300109|300138|300210|300227|300113|300117|300118|300219|300223|300240|300244|300033|300056|300063|300102|300133|300126|300131|300156|300163|300186|300196|300197|300146|300178|300182|300253|300230|300171|300173|300247|300250|300071|300203|300473|300095|300127|300152|300361|300437|300474|300123|300149|300154|300168|300169|300185|300236|300362|300501|300503|300504|300505|300506|300509|300510|300511|300512|300513|300514|300519|300530|5001|16101|81301|16102|6106|300094|300096|300532|300533|300540} likuciai rodomi eiluteje {1.} siu saskaitu kreditiniai (neigiami) likuciai ju ne mazina, o rodomi eil. {24.}

GL {90441|90442|90444|90445|90446|90447|90448|90451|90453|90454|90455|90456|90457|90458|90459|90460|90461|90462|90602} likuciai rodomi eiluteje {2.1.} GL {90455|90456} kreditiniai (neigiami) likuciai ju ne mazina, o rodomi eil. {16.1.} papildomai {90493|90494|90495|90496} neigiami likuciai su priesingu zenklu

GL {19101} ARP {1074|1144} likuciai rodomi eiluteje {2.2.}

GL {19101} ARP {1071|1072|1073|1141|1142|1143} likuciai rodomi eiluteje {2.3.}

GL {19101} ARP {1071|1072|1073|1141|1142|1143} likuciai rodomi eiluteje {2.3.}

GL {19103} ARP {1079|1149} likuciai rodomi eiluteje {4.1.}

GL {7201|7205|7206|47702|47703|47704|47705|47711|47712|47716|47717|47720|90423|90429} likuciai rodomi eiluteje {5.2.} siu saskaitu kreditiniai (neigiami)

likuciai ju ne mazina, o rodomi eil. {24.} isskyrus sask. {7201} kurios neigiamas likutis rodomas eil. {18.}

GL {73103} ARP {1011|1013} likuciai rodomi eiluteje {5.2.} siu saskaitu kreditiniai (neigiami) likuciai ju ne mazina, o rodomi eil. {24.}

GL {73102} isskyrus CGR {28} likuciai rodomi eiluteje {5.2.} siu saskaitu kreditiniai (neigiami) likuciai ju ne mazina, o rodomi eil. {24.}

GL {7507|46701|71101|71102|71106|71109|71113} likuciai rodomi eiluteje {5.2.}

GL {12001|13001|14101|14201|46703|67301|69101} neigiami likuciai rodomi eiluteje {5.2.}

Minus GL {2002|2003|2006} likuciai rodomi eiluteje {5.2.}

Minus GL {2004} ARP {2164|2165} likuciai rodomi eiluteje {5.2.}

GL {19102} ARP {1075|1076|1077|1145|1146|1147} likuciai rodomi eiluteje {6.1.}

GL {73106|90411} likuciai rodomi eiluteje {6.1.}

GL {90425} ARP {1151} likuciai rodomi eiluteje {6.1.}

GL {90510} likuciai rodomi eiluteje {6.1.} siu saskaitu kreditiniai (neigiami) likuciai ju ne mazina, o rodomi eil. {20.4.}

Minus GL {92002} ARP {1502} likuciai rodomi eiluteje {9.1.}

Minus GL {92006} ARP {1507} likuciai rodomi eiluteje {9.2.}

GL {92004} ARP {1105|1325} likuciai rodomi eiluteje {10.1.}

Minus GL {1504} ARP {2083} likuciai rodomi eiluteje {10.1.}

GL {92004} ARP {1100|1101|1106|1320|1321} likuciai rodomi eiluteje {10.2.}

Minus GL {1504} ARP {2080|2081} likuciai rodomi eiluteje {10.2.}

GL {19505} ARP {1083|1193|1082|1192|1193} likuciai rodomi eiluteje {11.}

B 3. Experiment transformation definition (fragment of xText)

```
«IMPORT brtl::brtemplates»
«IMPORT brtl::templates»


«DEFINE root FOR brtemplates::RulePackage»
```

```
  «FILE 'selectStatement.sql'»
    «EXPAND selectStatement
      FOREACH bRules.select(e|e.name=='Ataskaitos
algoritmas')»
  «ENDFILE»
«ENDDEFINE»

«DEFINE selectStatement FOR brtemplates::BRuleExp»
  SELECT
  CASE
  «EXPAND caseStatementEilNr FOREACH templateBinding»
  ELSE '0' END as EILNR,
  TAL_NAME1, TAL_NAME5, TYPE,
  TC_PL_GROUP, BRA_NAME, CRR_NAME,
  RTYPE, DB_AMOUNT_LT, DB_ACC_NO, TC_CIF_NO, TC_SHORT_NAME

  from FINREP_BALANS_20071231

«ENDDEFINE»

«DEFINE caseStatementEilNr FOR templates::TemplateBinding»
  WHEN
    «EXPAND whenThenStatement
      FOREACH
parameterSubstitution.select(e|e.formal.paramName=='paramGL
ARP')»
    «EXPAND whenThenStatement
      FOREACH
parameterSubstitution.select(e|e.formal.paramName=='parIski
rCGR')»
    «EXPAND whenThenStatement
      FOREACH
parameterSubstitution.select(e|e.formal.paramName=='parCGR'
)»
  THEN
    «EXPAND thenStatement
      FOREACH
parameterSubstitution.select(e|e.formal.paramName=='parLiku
ciai')»
«ENDDEFINE»



«DEFINE thenStatement FOR
templates::TemplateParameterSubstitution»
    «IF this.formal.paramName=='parLikuciai'»
```

```
       «EXPAND inListExpString FOREACH
((brtemplates::RulePartExpComposite)selected)
         .ruleParts.last()
         .templateParameter.TemplateParameterSubstitution

  .select(e|e.templateBinding==this.templateBinding)»
    «ENDIF»
«ENDDEFINE»

«DEFINE whenThenStatement FOR
templates::TemplateParameterSubstitution»
  «IF this.formal.paramName=='paramGLARP' &&
    this.selected==this.formal.listParameters.first()»
    TAL_NAME5 IN (
      «EXPAND inListExpTalName FOREACH
((brtemplates::RulePartExpComposite)selected)
       .ruleParts.last()
       .templateParameter.TemplateParameterSubstitution

  .select(e|e.templateBinding==this.templateBinding)»
        )
   «ELSEIF this.formal.paramName=='paramGLARP' &&
    this.selected==this.formal.listParameters.last()»
    TAL_NAME5 IN (
      «EXPAND inListExpTalName FOREACH
((brtemplates::RulePartExpComposite)selected)
       .ruleParts.first()
       .templateParameter.TemplateParameterSubstitution

  .select(e|e.templateBinding==this.templateBinding)»
     ) AND
    TYPE2 IN(
      «EXPAND inListExpType FOREACH
((brtemplates::RulePartExpComposite)selected)
         .ruleParts.last()
         .templateParameter.TemplateParameterSubstitution

  .select(e|e.templateBinding==this.templateBinding)»
           )
   «ELSEIF this.formal.paramName=='parCGR'»
    CGR IN (
      «FOREACH
((brtemplates::RulePartExpComposite)selected)
         .ruleParts.last()
         .templateParameter.TemplateParameterSubstitution

  .select(e|e.templateBinding==this.templateBinding) AS e»
```

```
            «EXPAND inListExpString FOR e»,
        «ENDFOREACH»
        )
    «ENDIF»
«ENDDEFINE»


«DEFINE inListExpTalName
FOR templates::TemplateParameterSubstitution»
   «((brtemplates::RulePartExp)this.ownedActual).name.trim()
»,
«ENDDEFINE»

«DEFINE inListExpType
FOR templates::TemplateParameterSubstitution»
   '«((brtemplates::RulePartExp)this.ownedActual).name.trim(
)»',
«ENDDEFINE»

«DEFINE inListExpString
FOR templates::TemplateParameterSubstitution»
   '«((brtemplates::RulePartExp)this.ownedActual).name.trim(
)»'
«ENDDEFINE»
```

B 4. Experiment transformation result (fragment of SQL code)

```
SELECT
  CASE
     WHEN TAL_NAME5 IN ( 3201 )
      AND TYPE2 IN( '1003' )
     THEN '1.'
     WHEN TAL_NAME5 IN ( 3301 )
      AND TYPE2 IN( '1004' )
     THEN '1.'
     WHEN TAL_NAME5 IN ( 73103 )
      AND TYPE2 IN( '1010' ) CGR IN ( '28' )
     THEN '1.'
     WHEN TAL_NAME5 IN ( 3001, 3101, 30101, 6001, 6101, 6102, 6103,
6104, 6105, 30102, 30103, 30104, 30105, 30106, 300108, 300109, 300138,
300210, 300227, 300113, 300117, 300118, 300219, 300223, 300240, 300244,
300033, 300056, 300063, 300102, 300133, 300126, 300131, 300156, 300163,
300186, 300196, 300197, 300146, 300178, 300182, 300253, 300230, 300171,
```

```
300173, 300247, 300250, 300071, 300203, 300473, 300095, 300127, 300152,
300361, 300437, 300474, 300123, 300149, 300154, 300168, 300169, 300185,
300236, 300362, 300501, 300503, 300504, 300505, 300506, 300509, 300510,
300511, 300512, 300513, 300514, 300519, 300530, 5001, 16101, 81301, 16102,
6106, 300094, 300096, 300532, 300533, 300540 )
      THEN '1.'
      WHEN TAL_NAME5 IN ( 90441, 90442, 90444, 90445, 90446, 90447,
90448, 90451, 90453, 90454, 90455, 90456, 90457, 90458, 90459, 90460,
90461, 90462, 90602 )
      THEN '2.1.'
      WHEN TAL_NAME5 IN ( 19101 )
        AND TYPE2 IN( '1074', '1144' )
      THEN '2.2.'
      WHEN TAL_NAME5 IN ( 19101 )
        AND TYPE2 IN( '1071', '1072', '1073', '1141', '1142', '1143' )
      THEN '2.3.'
      WHEN TAL_NAME5 IN ( 19101 )
        AND TYPE2 IN( '1071', '1072', '1073', '1141', '1142', '1143' )
      THEN '2.3.'
      WHEN TAL_NAME5 IN ( 19103 )
        AND TYPE2 IN( '1079', '1149' )
      THEN '4.1.'
      WHEN TAL_NAME5 IN ( 7201, 7205, 7206, 47702, 47703, 47704, 47705,
47711, 47712, 47716, 47717, 47720, 90423, 90429 )
      THEN '5.2.'
      WHEN TAL_NAME5 IN ( 73103 )
        AND TYPE2 IN( '1011', '1013' )
      THEN '5.2.'
      WHEN TAL_NAME5 IN ( 73102 ) CGR IN ( '28')
      THEN '5.2.'
      WHEN TAL_NAME5 IN ( 7507, 46701, 71101, 71102, 71106, 71109,
71113 )
      THEN '5.2.'
      WHEN TAL_NAME5 IN ( 12001, 13001, 14101, 14201, 46703, 67301,
69101 )
      THEN '5.2.'
      WHEN TAL_NAME5 IN ( 2002, 2003, 2006 )
      THEN '5.2.'
      WHEN TAL_NAME5 IN ( 2004 )
        AND TYPE2 IN( '2164', '2165' )
      THEN '5.2.'
      WHEN TAL_NAME5 IN ( 19102 )
```

```
      AND TYPE2 IN( '1075', '1076', '1077', '1145', '1146', '1147' )
    THEN '6.1.'
    WHEN TAL_NAME5 IN ( 73106, 90411 )
    THEN '6.1.'
    WHEN TAL_NAME5 IN ( 90425 )
     AND TYPE2 IN( '1151' )
    THEN '6.1.'
    WHEN TAL_NAME5 IN ( 90510 )
    THEN '6.1.'
    WHEN TAL_NAME5 IN ( 92002 )
     AND TYPE2 IN( '1502' )
    THEN '9.1.'
    WHEN TAL_NAME5 IN ( 92006 )
     AND TYPE2 IN( '1507' )
    THEN '9.2.'
    WHEN TAL_NAME5 IN ( 92004 )
     AND TYPE2 IN( '1105', '1325' )
    THEN '10.1.'
    WHEN TAL_NAME5 IN ( 1504 )
     AND TYPE2 IN( '2083' )
    THEN '10.1.'
    WHEN TAL_NAME5 IN ( 92004 )
     AND TYPE2 IN( '1100', '1101', '1106', '1320', '1321' )
    THEN '10.2.'
    WHEN TAL_NAME5 IN ( 1504 )
     AND TYPE2 IN( '2080', '2081' )
    THEN '10.2.'
    WHEN TAL_NAME5 IN ( 19505 )
     AND TYPE2 IN( '1083', '1193', '1082', '1192', '1193' )
    THEN '11.'
    ELSE '0'
END AS EILNR,
TAL_NAME1  ,
TAL_NAME5  ,
TYPE    ,
TC_PL_GROUP ,
BRA_NAME  ,
CRR_NAME  ,
RTYPE    ,
DB_AMOUNT_LT,
DB_ACC_NO  ,
TC_CIF_NO  ,
```

```
   TC_SHORT_NAME
FROM  FINREP_BALANS_20071231
```

Appendix C. Formal ORM to UML/OCL transformation specification in ATL

```
module orm2uml; -- Module Template
create OUT : UML from IN : ORM;


helper context ORM!Role def: getFirstRolePhrase() : String =
        if
        --if exists reading starting from role
            self.parentPredicate.predicateReading->select(reading|
                self.placeHolder->includes(
                    reading.part->select(part|part.oclIsKindOf(ORM!PlaceHolder))->last()
                )
            )->notEmpty()
        then
            self.parentPredicate.predicateReading->select(reading|
                self.placeHolder->includes(
                    reading.part->select(part|part.oclIsKindOf(ORM!PlaceHolder))->last()
                )
            )->first().part->select(
                    part|part.oclIsKindOf(ORM!Phrase)
                )->first().name
        else 'the'+self.name
        endif;


helper context ORM!Reading def: getReadingAsText() : String =
```

```
    let v : String = '' in self.part->collect(a|
        if a.oclIsKindOf(ORM!Phrase) then
            a.name
        else
            a.role.name
        endif
        )
    .toString()
    .regexReplaceAll('Sequence ','')
    .regexReplaceAll('[^a-zA-Z]','')
;
helper context ORM!Role def: lowerMultiplicity() : Integer =
        if self.roleReferences->select(roleRef|
            roleRef.theRoleConstraint.oclIsKindOf(ORM!MandatoryConstraint))->notEmpty()
        then
            1
        else
            0
        endif
;

helper context ORM!Role def: upperMultiplicityOnOneRole() : Integer =
        if self.roleReferences->select(roleRef|
            roleRef.theRoleConstraint.oclIsKindOf(ORM!InternalUniqueConstraint)
            and roleRef.theRoleConstraint.theRoleReference->size()=1)->notEmpty()
        then
            1
        else
            0-1
        endif
;
helper context String def: delSpaces() : String =
        self.regexReplaceAll(' ','')
;
helper context String def: roleName() : String =
        'the'+self.delSpaces()
    ;

helper context ORM!Predicate def:isObjectified() : Boolean =
        not self.nestedEntity.oclIsUndefined()
    ;

helper context ORM!Role def:isParentObjectified() : Boolean=
    self.parentPredicate.isObjectified()
    ;

helper context ORM!Predicate def:isBinary(): Boolean =
    self.role->size()=2
    ;

helper context ORM!Role def:isParentBinary(): Boolean =
    self.parentPredicate.role->size()=2
    ;

helper context ORM!Role def:isParentNary(): Boolean =
```

```
    self.parentPredicate.role->size()>2
    ;

--
helper context ORM!RoleConstraint def:isForOnePredicate(): Boolean =
    self.theRoleReference->forAll(ref1|
                    ref1.theRoleConstraint.theRoleReference->forAll(ref2|
                        ref1.role.parentPredicate=  ref2.role.parentPredicate
                        )
                    )
    ;
helper context ORM!RoleConstraint def:isForBinaryPredicate(): Boolean =
    self.theRoleReference->forAll(rRef|rRef.role.isParentBinary())
    ;
helper context ORM!RoleConstraint def:isForObjectifiedPredicate(): Boolean =
    self.theRoleReference->exists(rRef|rRef.role.isParentObjectified())
    ;

helper context ORM!RoleConstraint def:isForEntityRoles(): Boolean =
    self.theRoleReference->forAll(rRef|
        rRef.role.objectType.oclIsKindOf(ORM!EntityType)
        )
    ;

helper context ORM!Role def:isObjectTypeEntity(): Boolean =
        self.objectType.isEntity()
    ;

helper context ORM!ObjectType def:isEntity(): Boolean =
    self.oclIsKindOf(ORM!EntityType)
    ;
helper context ORM!ObjectType def:isInvolvedInOnePredicate(): Boolean =
    self.role->size()<2
    ;

helper context ORM!RoleConstraint def:isBinaryRoleConstraint(): Boolean =
    self.theRoleReference->size()=2
    ;

helper context ORM!RoleConstraint def:hasOneConnectionPoint(): Boolean =
    let pCol:Sequence(ORM!Predicate) = self.theRoleReference->
        collect(rRef|rRef.role.parentPredicate) in

    pCol->iterate(p;
        oCol : Sequence(ORM!ObjectType)=pCol->first().getObjectTypes()|
        p.getObjectTypes()->asSet()->intersection(oCol)
        )->size()=1
;

helper context ORM!RoleConstraint def:getOneConnectionPoint(): ORM!ObjectType =
    if self.hasOneConnectionPoint() then
    let pCol:Sequence(ORM!Predicate) = self.theRoleReference->
        collect(rRef|rRef.role.parentPredicate) in

    pCol->iterate(p;
```

```
        oCol : Sequence(ORM!ObjectType)=pCol->first().getObjectTypes()|
        p.getObjectTypes()->asSet()->intersection(oCol)
        )->asSequence()->first()
    else ''
    endif
;

helper context ORM!Predicate def: getObjectTypes():Sequence(ORM!ObjectType)=
    self.role->collect(r|r.objectType)
    ;



helper context ORM!RoleConstraint def:isNAryRoleConstraint(): Boolean =
    self.theRoleReference->size()>1
    ;

helper context ORM!Role def:isParentToClass(): Boolean=
    if self.isParentObjectified()
        or self.isParentNary()   then
        true
    else
        false
    endif
;

helper context ORM!ObjectType def:isObjectTypeToClass(): Boolean=
    if self.isEntity() or not self.isInvolvedInOnePredicate()
    then
        true
    else
        false
    endif
;

helper context ORM!Role def:resolveRoleObjectTypeClassName():String=
    if self.objectType.isObjectTypeToClass() then
                thisModule.resolveTemp(self.objectType, 'cl').name
    else ''
    endif
    ;

helper context ORM!Role def:resolveRolePredicateClassName():String=
    if self.isParentToClass() then
        self.parentPredicate.resolvePredicateClassName()
    else
            if self.isObjectTypeToClass() then
                self.resolveRoleObjectTypeClassName()
            else ''
            endif
    endif;

helper context ORM!Predicate def:resolvePredicateClassName():String=
    if self.isObjectified() then
        thisModule.resolveTemp(self.nestedEntity, 'cl').name
```

```
    else
        thisModule.resolveTemp(self, 'cl').name
    endif
    ;

helper context ORM!Role def:resolveRoleNavigationName():String=
    if self.objectType.isObjectTypeToClass() then
        if self.isParentObjectified() then
            self.objectType.name.delSpaces()
        else
            self.getFirstRolePhrase().delSpaces()
        endif
    else
        self.objectType.setAttributeName()
    endif


    ;
--How to access role end from objectType
helper context ORM!Role def:oclresolveRoleNavigationName(o:ORM!ObjectType):String=
    if self.isParentToClass() then
    --TODO
            ''
    else
            self.resolveRoleNavigationName()
    endif

    ;


helper context ORM!ObjectType def:setAttributeName():String=
            self.name.delSpaces()
    ;

helper context ORM!ObjectType def:setAttributeTypeName():String=
            self.name.delSpaces()
    ;

helper context ORM!ObjectType def:setClassName():String=
            self.name.delSpaces()
    ;

helper context ORM!ObjectType def:resolveObjectTypeName():String=
    if self.isObjectTypeToClass()then
            self.setClassName()
    else
            self.setAttributeName()
    endif
    ;



rule Model2Model{
    from a: ORM!ORMModel
    to
```

```
        z:UML!Package(
        name<- 'UMLPackage', --a.modelName,
      isSpecification <- false,
        isRoot <-false,
        isLeaf <- false,
        isAbstract <- false,
        namespace<-b
        ),

    b: UML!Model (
        name<- 'UMLModel', --a.modelName,
      isSpecification <- false,
        isRoot <-false,
        isLeaf <- false,
        isAbstract <- false
    )


}

rule EntityType2Class {
  from a:ORM!EntityType
    to cl:UML!Class(
        name<-a.name.delSpaces(),
        visibility <- #vk_public,
        isSpecification <- false,
        isRoot <- false,
        isLeaf <- false,
        isAbstract <- false,
        namespace <- a.ORMModel
        --isActive <- false
    )
}



rule RefSchema2Attribute{
from a:ORM!RefSchema
to at : UML!Attribute(
        name<-a.name.delSpaces(),
        visibility <- #vk_public,
        owner<-a.ObjectType,
        changeability <- #ck_changeable,
        ordering <- #ok_unordered,
        targetScope <- #sk_instance,
        -- End of bindings inherited from StructuralFeature
        initialValue <- OclUndefined,
        type<-t
    ),
    t:UML!Class(
        name<-if a.mode='Numeric'then 'Integer' else'String'endif,
        namespace <- a.ORMModel
    )

}
```

```
-------------------------------------------------------------------------------------------
--Not objectified predicate --
-------------------------------------------------------------------------------------------
rule UnaryPredicate2Attribute {
    from a:ORM!Predicate(
        --Only predicates attached to ValueTipes
        a.role->select(  pred | pred.objectType.oclIsKindOf(ORM!ValueType))->isEmpty()
        and
        a.role->select(  pred | pred.objectType.oclIsKindOf(ORM!EntityType))->size()=1
        and (a.nestedEntity.oclIsUndefined())
        )
        to ua : UML!Attribute (
            -- Begin bindings inherited from ModelElement
            name <- a.predicateReading->
            first().part->select(a|a.oclIsKindOf(ORM!Phrase))->first().name.delSpaces(),

            visibility <- #vk_public,
            owner <- a.role->first().objectType,
            changeability <- #ck_changeable,
            multiplicity <- um,
            ordering <- #ok_unordered,
            type <- boo,
            targetScope <- #sk_instance,

            initialValue <- OclUndefined
        ),

        um : UML!Multiplicity (
            range <- Set{ur}
        ),

        ur : UML!MultiplicityRange (
            lower <- a.role->first().lowerMultiplicity(),
            upper <- 1, --TODO
            multiplicity <- um
        ),
        boo :UML!Class(
            name<-'Boolean',
            isSpecification<-false,
            isRoot<-false,
            isLeaf<-false,
            isAbstract<-false,
            isActive<-false,
            namespace <- a.ORMModel)
}

rule BinaryPredicateEntytiType2Association {
    from a:ORM!Predicate(
        --Only entity types
        a.role->select(  role | role.objectType.oclIsKindOf(ORM!ValueType))->isEmpty()
        and
        a.role->select(  role | role.objectType.oclIsKindOf(ORM!EntityType))->size()=2
        --binary predicate
        and
        a.role->size()=2
```

```
        --not objectified
        and (a.nestedEntity.oclIsUndefined())
        )
    to b:UML!Association (
        connection <- Set{end1,end2},
        namespace <- a.ORMModel
    ),
     end1:UML!AssociationEnd(
          isSpecification <- false,
            visibility <- #vk_public,
            association <- b,
            participant <- a.role->first().objectType,
            name<- a.role->first().getFirstRolePhrase().delSpaces(),
            --predicate reading where the role is first
            isNavigable <- true,
            ordering <- #ok_unordered,
            aggregation <- #ak_none,
            targetScope <- #sk_instance,
            changeability <- #ck_changeable
        ),
        end2:UML!AssociationEnd(
          isSpecification <- false,
            visibility <- #vk_public,
            association <- b,
            participant <- a.role.last().objectType,
            name<- a.role->last().getFirstRolePhrase().delSpaces(),
            isNavigable <- true,
            ordering <- #ok_unordered,
            aggregation <- #ak_none,
            targetScope <- #sk_instance,
            changeability <- #ck_changeable
        )
    do{
            thisModule.roleMultiplicity(a.role->first(),end2);
            thisModule.roleMultiplicity(a.role->last(),end1);

    }
}

rule BinaryPredicateValueType2Attribute {
    from a:ORM!Predicate(
        --mixed predicate, at least one entity type
        a.role->select(  role | role.objectType.oclIsKindOf(ORM!EntityType))->size()=1
        and
        --binary predicate
        a.role->size()=2
        and
--      --valueType participate only in only predicate
        a.role->select(  role2 |
            role2.objectType.oclIsKindOf(ORM!ValueType)
            and role2.objectType.role->size()=1)->notEmpty()
        --not objectified
        and (a.nestedEntity.oclIsUndefined())
        )
    to ua : UML!Attribute(
```

```
        name<-a.role->select(role|
                role.objectType.oclIsKindOf(ORM!ValueType)
                )
            ->first().objectType.setAttributeName(),
        visibility <- #vk_public,
        owner<-a.role->select(role|role.objectType.oclIsKindOf(ORM!EntityType))-
>first().objectType,
        changeability <- #ck_changeable,
        ordering <- #ok_unordered,
        targetScope <- #sk_instance,
        -- End of bindings inherited from StructuralFeature
        initialValue <- OclUndefined,
        multiplicity <- um,
        type<-t
        ),
        t:UML!Class(
            name<-a.role->select(role|
                role.objectType.oclIsKindOf(ORM!ValueType)
                )
            ->first().objectType.setAttributeTypeName(),
            namespace <- a.ORMModel
        ),
        um : UML!Multiplicity (
            range <- Set{ur}
        ),
        ur : UML!MultiplicityRange (
            lower <- a.role->select(role|
                role.objectType.oclIsKindOf(ORM!EntityType))->first().lowerMultiplicity(),
            upper <- a.role->select(role|
                role.objectType.oclIsKindOf(ORM!EntityType))->first().upperMultiplicityOnOneRole(),
            multiplicity <- um
        )

}


rule NaryPredicate2Class {
    from a:ORM!Predicate(
        --n-ary predicate
        a.role->size()>2
        --not objectified
        and (a.nestedEntity.oclIsUndefined())
        )
    to cl : UML!Class(
        name<-a.predicateReading->first().getReadingAsText().delSpaces(),
        visibility <- #vk_public,
        isSpecification <- false,
        isRoot <- false,
        isLeaf <- false,
        isAbstract <- false,
        namespace <- a.ORMModel
     )
}

rule NaryPredicateRole2Association {
```

```
    from a:ORM!Role(
        --n-ary predicate
        a.parentPredicate.role->size()>2
        and
        a.objectType.oclIsKindOf(ORM!EntityType)
        --not objectified
        and (a.parentPredicate.nestedEntity.oclIsUndefined())
        )
    to b:UML!Association (
        connection <- Set{end1,end2},
        namespace <- a.ORMModel
    ),
     end1:UML!AssociationEnd(
          isSpecification <- false,
            visibility <- #vk_public,
            association <- b,
            participant <- a.objectType,
            name<- 'the'+a.objectType.name.delSpaces(),
            --predicate reading where the role is first
            isNavigable <- true,
            ordering <- #ok_unordered,
            aggregation <- #ak_none,
            targetScope <- #sk_instance,
            changeability <- #ck_changeable,
            multiplicity <- um
        ),
        end2:UML!AssociationEnd(
          isSpecification <- false,
            visibility <- #vk_public,
            association <- b,
            participant <- a.parentPredicate,
            name<- a.parentPredicate.predicateReading->first().getReadingAsText().roleName(),
            isNavigable <- true,
            ordering <- #ok_unordered,
            aggregation <- #ak_none,
            targetScope <- #sk_instance,
            changeability <- #ck_changeable

        ),
        um : UML!Multiplicity (
            range <- Set{ur}
        ),
        ur : UML!MultiplicityRange (
            lower <- 1,
            upper <- 1
        )
        do{
            thisModule.roleMultiplicity(a,end2);

        }
}
```

```
--Objectified predicate --
-----------------------------------------------------------------------------------------------

rule ObjectifiedNaryPredicateRole2Association {
    from a:ORM!Role(
        a.objectType.oclIsKindOf(ORM!EntityType)
        --objectified
        and (not a.parentPredicate.nestedEntity.oclIsUndefined())
        )
    to b:UML!Association (
        connection <- Set{end1,end2},
        namespace <- a.ORMModel
    ),
     end1:UML!AssociationEnd(
          isSpecification <- false,
            visibility <- #vk_public,
            association <- b,
            participant <- a.objectType,
            name<- a.objectType.name.delSpaces(),
            --predicate reading where the role is first
            isNavigable <- true,
            ordering <- #ok_unordered,
            aggregation <- #ak_none,
            targetScope <- #sk_instance,
            changeability <- #ck_changeable,
            multiplicity <- um
        ),
        end2:UML!AssociationEnd(
          isSpecification <- false,
            visibility <- #vk_public,
            association <- b,
            participant <- a.parentPredicate.nestedEntity,
            name<- a.parentPredicate.nestedEntity.name.roleName(),
            isNavigable <- true,
            ordering <- #ok_unordered,
            aggregation <- #ak_none,
            targetScope <- #sk_instance,
            changeability <- #ck_changeable
        ),
        um : UML!Multiplicity (
            range <- Set{ur}
        ),
        ur : UML!MultiplicityRange (
            lower <- 1,
            upper <- 1
        )
        do{
            thisModule.roleMultiplicity(a,end2);
        }
}

--Subtype rule

rule SubtypeConnection2Generalization {
    from a:ORM!SubtypeConnection
```

```
    to b:UML!Generalization (
        parent<-a.superType,
        child<-a.subType
        )
}
--Multiplicity
rule roleMultiplicity(role : ORM!Role, end: UML!AssociationEnd){
    to  um : UML!Multiplicity (
        ),
        ur : UML!MultiplicityRange (
            multiplicity <- um
        )
    do{
        end.multiplicity<-um;
        ur.upper <- role.upperMultiplicityOnOneRole();
        ur.lower <- role.lowerMultiplicity();
    }
}

--Uniquiness constraint on binary predicate
rule BinaryRoleInternalUniqueness2OCL{
    from a:ORM!InternalUniqueConstraint
    (
        a.isBinaryRoleConstraint()
        and
        a.isForOnePredicate()
        --and both roles refer to the entitytype
        and a.isForEntityRoles()
        --and refered predicates are binary
        and a.isForBinaryPredicate()
        --and not objectified
        and not a.isForObjectifiedPredicate()
        )
    to constr:UML!Constraint(
        constrainedElement<-a.theRoleReference->first().role.objectType,
        body<-exp
        ),
        exp:UML!BooleanExpression
        (
                language<-'OCL',

                body <-'context '+  a.theRoleReference-
>first().role.resolveRoleObjectTypeClassName()
                 +
                ' inv:'+'not self.'+a.theRoleReference->last().role.getFirstRolePhrase().delSpaces()
                +' -> exists (a|a.'+
                    a.theRoleReference->first().role.getFirstRolePhrase().delSpaces()
                +'->includes(self))'
            )
    do{
        exp.body.println();
    }
}

--Uniquiness constraint on nary predicate
```

```
rule NAryRoleInternalUniqueness2OCL{
    from a:ORM!InternalUniqueConstraint
    (
        a.isNAryRoleConstraint()
        --and those roles from the same predicate
        and
        a.isForOnePredicate()
        --and refered predicates are binary
        and not (a.isForBinaryPredicate()
        and a.isBinaryRoleConstraint()
        and not a.isForObjectifiedPredicate())
        )
    to constr:UML!Constraint(
        constrainedElement<-a.theRoleReference->first().role.objectType,
        body<-exp
        ),
        exp:UML!BooleanExpression
        (
                language<-'OCL',
                body <-'context '+  a
                    .theRoleReference->first()
                    .role.resolveRolePredicateClassName()
                 +
                ' inv:'+
                'let a: Set('+
                a.theRoleReference->first()
                    .role.resolveRolePredicateClassName() +
                ') ='+
                a.theRoleReference->first()
                    .role.resolveRolePredicateClassName() +
                '.allInstances in'+
                ' not a->exists(a|'+
                    a.theRoleReference->iterate(rRef; rStr : String = ' '|
                        if not( rRef=a.theRoleReference.last()) then
                            rStr+' a.'+
                            rRef.role.resolveRoleNavigationName()+
                            ' = '+
                            ' self.'+
                            rRef.role.resolveRoleNavigationName()+
                            ' and'
                        else
                            rStr+' a.'+
                            rRef.role.resolveRoleNavigationName()+
                            ' = '+
                            ' self.'+
                            rRef.role.resolveRoleNavigationName()

                        endif
                        )
                    +
                    ')'
            )
    do{
        exp.body.println();
    }
```

```
}

rule NAryRoleExternalUniqueness2OCL{
    from a:ORM!ExternalUniqueConstraint
    (
        a.hasOneConnectionPoint()
        )
    to constr:UML!Constraint(
        constrainedElement<-a.getOneConnectionPoint(),
        body<-exp
        ),
        exp:UML!BooleanExpression
        (
                language<-'OCL'
                ,

                body <-'context '
                +
                a.getOneConnectionPoint().resolveObjectTypeName()+
                ' inv:'+
                'let a: Set('+
                a.getOneConnectionPoint().resolveObjectTypeName()+
                ') ='+
                a.getOneConnectionPoint().resolveObjectTypeName()+
                '.allInstances in'+
                ' not a->exists(a|'+
                    a.theRoleReference->iterate(rRef; rStr : String = ' '|
                        if not( rRef=a.theRoleReference.last()) then
                            rStr+' a.'+
                            rRef.role.oclresolveRoleNavigationName(a.getOneConnectionPoint())+
                            ' = '+
                            ' self.'+
                            rRef.role.oclresolveRoleNavigationName(a.getOneConnectionPoint())+
                            ' and'
                        else
                            rStr+' a.'+
                            rRef.role.oclresolveRoleNavigationName(a.getOneConnectionPoint())+
                            ' = '+
                            ' self.'+
                            rRef.role.oclresolveRoleNavigationName(a.getOneConnectionPoint())

                        endif
                        )
                    +
                    ')'
            )
    do{
        exp.body.println();
    }
}
```

Sergejus Sosunovas

USER DEFINED TEMPLATES FOR THE SPECIFICATION AND TRANSFORMATION OF BUSINESS RULES
Doctoral Dissertation
Technological Sciences, Informatics Engineering (07T)

VARTOTOJŲ SUDAROMI ŠABLONAI VERSLO TAISYKLĖMS SPECIFIKUOTI IR TRANSFORMUOTI
Daktaro disertacijos
Technologijos mokslai, informatikos inžinerija (07T)

2008 10 28. 14,25 sp. l. Tiražas 20 egz.
Vilniaus Gedimino technikos universiteto leidykla „Technika“, Saulėtekio al. 11, LT-10223 Vilnius, *http://leidykla.vgtu.l*t
Spausdino UAB „Baltijos kopija“, Kareivių g. 13B, 09109 Vilnius *http://www.kopija.lt*