ISSN 1392 – 124X INFORMATION TECHNOLOGY AND CONTROL, 2008, Vol. 37, No. 2

PRACTICAL APPLICATION OF BRTL APPROACH FOR FINANCIAL REPORTING DOMAIN[1]

Olegas Vasilecas, Sergejus Sosunovas

Vilnius Gediminas Technical University
Sauletekio al. 11, LT-10223 Vilnius, Lithuania

Abstract. Business rules are evidently important for organisations as they describe how they are doing business. Business rules templates are often proposed as a means of the specification of business rules. Business rules templates language (BRTL) is a language developed for the specification of business rules templates. This paper documents the findings of an experiment aimed at determining the extent to which business rules 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 the four historical projects of the same domain.

1. Introduction

1.1. Business Rules and Templates

Business rules are evidently important for organisations as they describe how they are doing business. Their value has also been recognised within the information system (IS) domain, mostly because of their ability to make applications flexible and amenable to change. [1].

Natural language is an initial requirement for the business rules representation language [2]. Templates are a popular way of knowledge representation. It already has showed its effectiveness in information extraction and ontology axioms specification. Ability to create templates can help enforce the consistent deployment of rules across different business scenarios, applications, projects and business units. Templates 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.

BRTL [3] is a language developed for the specification of business rules templates. It was prototyped in the tool BRidgeIT [4]. The language complies with the MDA [5] requirements and is fully transformable. It is intended that users of BRTL will develop a custom business rules template, integrated with ORM [6] model, specify business rules using this template and transform these business rules to PSM.

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 [2] 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.

1.2. 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 no technical user with the reporting information. Our main arm in this experiment is to trace the report algorithm specified using business rules 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.

[1] The work is supported by Lithuanian State Science and Studies Foundation according to High Technology Development Program Project "Business Rules Solutions for Information Systems Development (VeTIS)" Reg. No. B-07042

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 business rules 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. These templates are described within Section 2.

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 3.

Section 4 compares experiment results with historical data from the previous projects.

1.3. Goals

Figure 1 illustrates an overview of the experiment structure in which the top and bottom entities represent the PIM and PSM respectively.

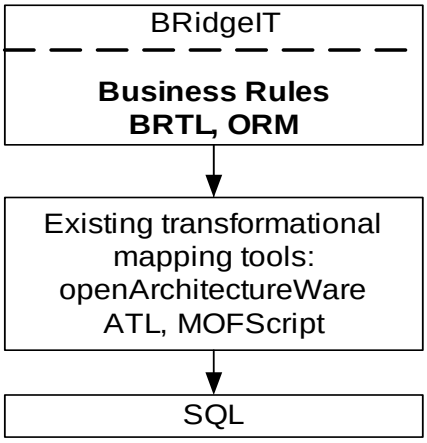

Figure 1. Domain structure

The BRTL supporting BRidgeIT tool and transformations appearing in the centre of the diagram represents the experiment objective. As well as creating workable business rules templates for the specification of business rules 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 business rules specification to facilitate their use.

Some existing research address the issues surrounding code generation from business rules specifications. Armonas and Nemuraite [8] present transformation principles for business rules between PIM and PSM levels. Vasilecas and Valatkaite [9] concentrates on relational database trigger generation from business rules presented as conceptual graphs. However, the approaches proposed in these papers currently work at a high level of abstraction. Within our paper, we aim to present a framework through which executable part of code can be created from a supporting PIM business rules specification using the MDA.

2. Platform Independent Model

The Platform Independent Model (PIM) has been developed using BRTL and ORM implementation in prototype tool BRidgeIT.

2.1. ORM model of the test application

ORM model in Figure 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 2. Each entity has a reference schema specified in the brackets that is used to identify instance of an entity.

Moving towards analyzing the model presented in Figure 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 business rules related with our test-application financial report. It is only the structure that will be used for the development of business rules template. It is obvious that in this form it is possible to 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.

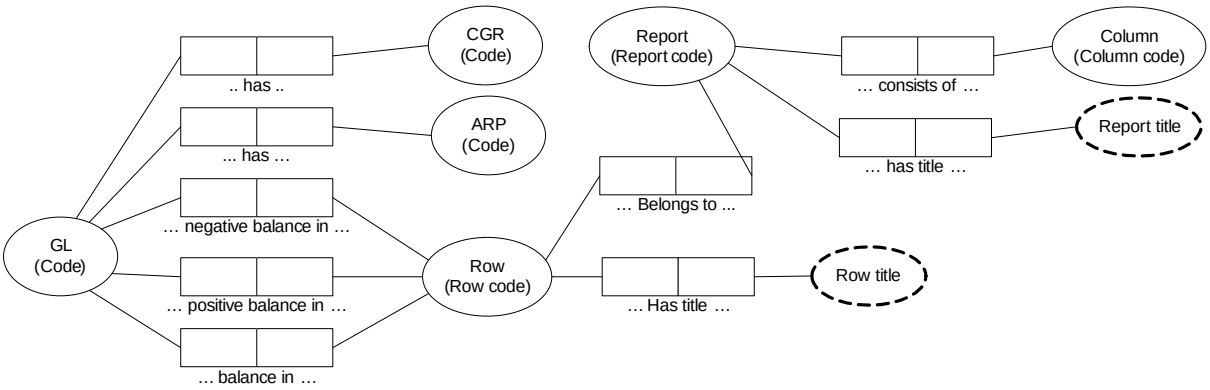

Figure 2. Domain ORM model

2.2. BRTL specification of the test application

The next phase of development PIM is creation of business rules template and specification of business rules 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 describeing report algorithm in order to develop templates. This approach insures that domain professionals will work with business rules 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 professionnals 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 professionnals. The support process of business rules 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 business rules 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 business rules 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 business rules. We are presenting only the most typical variations of business rules defining report algorithm as it is specified by the user:

GL {1111} ARP {3333} All CGR balace 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, business rules corresponding to one template are not enough to generate even the simplest report, it is necessary to use a set of business rules that correspond to different templates. ORM in this case serves as a structure that allows connection of business rules specified using two different templates, however satisfying one common functional purpose.

3. 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 business rules 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 [10]. 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 3 was developed.

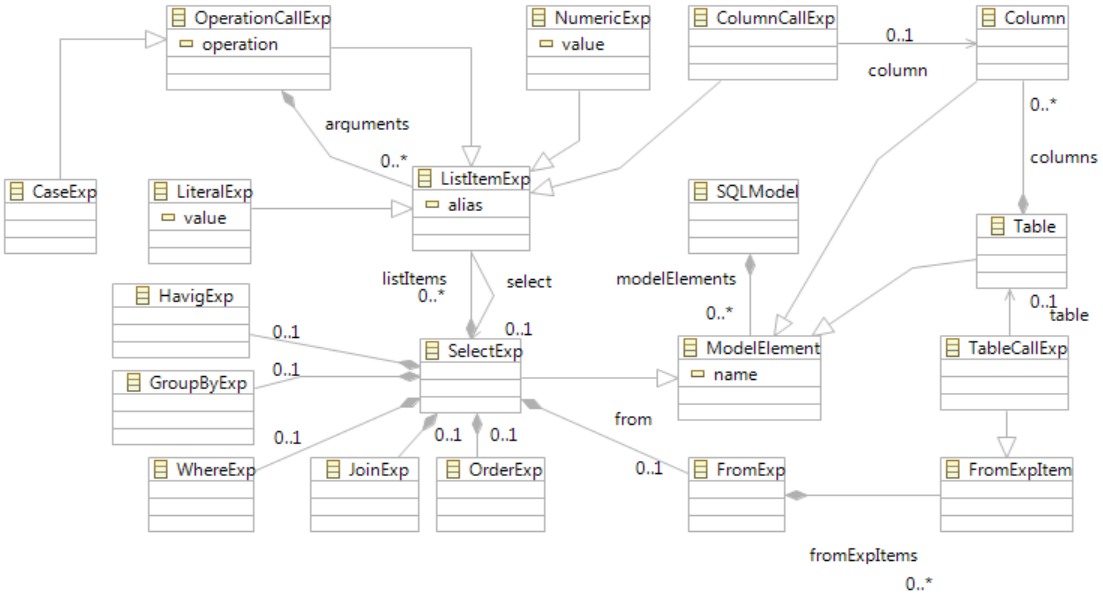

Figure 3. SQL select statements simplified metamodel

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 business rules 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 business rules 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 business rules. Resulting SQL statement is trivial by its nature; however because of the big number of rules (more than 500) its support is rather complicated.

For the purpose of the experiment MOFScript [11] model to code generation tool was used. The MOFScript language has been submitted as a proposal for a model to text transformation language to the OMG. MOFScript is based on the QVT-Merge [12] specification in terms of metamodel extensions and lexical syntax. A MOFScript rule is a specialisation of QVT-Merge operational mappings, and MOFScript constructions are specialisations of QVT-Merge constructions. The main goals with the language are to provide ease-of-use, minimize additions to QVT, as well as providing flexible mechanisms for generating text output.

4. 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 business rules > 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 artifacts. It was compared to the historical data collected in one of the Lithuanian enterprises and presented in Table 1. In this section we will briefly describe historical scenarios, provide comments on the activities and time necessary to implement them.

Table 1. Comparison of the results in one enterprise case

Scenario / Activities	No code generation	Custom repository with code generation			BRTL (experiment)
		No interface	Forms	Universal	
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.

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 process methodologies. However, we 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 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 professsionals 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. This 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 analyzed 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 business rules templates, specification of the business rules 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 differrently 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 analyzed 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 analyzed 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. Conclusions

The results of the present experiment demonstrate the viability of the solutions based on the business rules templates, BRTL and MDA transformations. Comparison of the experiment results with historical records demonstrates that BRTL solution is feasible to use in a very often changing environments. Only in this case relatively high technology development price can be compensated by the saved time. BRTL technology allows reallocating change prices from IT professional to domain professionals.

The comparison of experiment results with real project historical data clearly demonstrate that MDA based solutions is economically not feasible in rarely changing environments and in case of availability of cheap development resources. 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 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 development of code generation very complex task. In this case MDA based tools allow reduce development time significantly.

However the wide usage of the transformations as it was recognised by the previous researches [13, 14] is limited by the lack of metamodels for the majority of programming languages. Anyone who is planning to implement transformation solution based on the language not belonging to the most popular one is required to develop it own metamodel. The other less flexible opportunity is to execute direct transformation of business rules in templates to code omitting model-to-model transformation.

References

[1] **M. Bajec, M. Krisper.** A methodology and tool support for managing business rules in organizations. *Information Systems, Vol.*30 , *Issue* 6, 2005, 423- 443. *URL: http://infolab.fri.uni-lj.si/marko/downloads/Bajec%20&%20Krisper%202004,%20A%20methodology%20and%20tool%20support%20for%20managing%20BR%20in%20organisations.pdf, retrieved on* 2007.12.10.

[2] **T. Morgan.** Business Rules and Information Systems. *Addison Wesley*, 2002.

[3] **S. Sosunovas, O. Vasilecas.** Precise notation for business rules templates databases and information systems. *Proceedings of the* 7*th International Baltic Conference on Databases and Information Systems, Technika*, 2006, 55-60. *http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1678474, retrieved on* 2007.12.16.

[4] **S. Sosunovas.** Open source tool Bridge IT. *http://isl.vtu.lt/BRidgeIT/, retrieved on* 2007.12.08.

[5] **J. Miller, J. Mukerji.** Model Driven Architecture (MDA). *Technical Report, ormsc*/2001-07-01.

[6] **T. Halpin.** Object-Role Modeling (ORM/NIAM). *P. Bernus, K. Mertins and G. Schmidt (Eds.): Handbook on Architectures of Information Systems, Springer-Verlag,* 1998, *http://www.orm.net/pdf/springer.pdf, retrieved on* 2007.12.09

[7] OMG. *UML* 2.0 *Superstructure Final Adopted specification. OMG document: ptc*/03-08-02,*OMG*, 2003, *URL: http://www.omg.org/cgi-bin/doc?ptc/03-08-02.pdf, retrieved on* 2007.12.15.

[8] **A. Armonas, L. Nemuraite.** Traceability of Business Rules in Model Driven Development. *Proceedings of the 6th International Conference on Perspectives in Business Information Research – BIR*'2007, *Jyrki Nummenmaa and Eva Söderström (eds.)*, 2007, 22-36. *http://www.cs.uta.fi/reports/dsarja/D-2007-13.pdf, retrieved on* 2007.12.21.

[9] **I. Valatkaite, O. Vasilecas.** A Conceptual Graphs Approach for Business Rules Modeling. *Advances in Databases and Information Systems*, *LNCS* 2796, 2003, 178-189.

[10] **F. Jouault, I. Kurtev.** Transforming Models with ATL. *Proceedings of the Model Transformations in Practice Workshop at MoDELS* 2005, *Montego Bay, Jamaica*, (*January*, 2008). *http://www.sciences.univ-nantes.fr/lina/atl/bibliography/MTIP05, retrieved on* 2007.12.24.

[11] Softeam. *MOFScript Revised Submission to the MOF Model to Text Transformation RFP*. *OMG document ad*/05-05-04, *http://www.omg.org/cgi-bin/apps/doc?ad/05-05-04.pdf, retrieved on* 2007.12.21.

[12] QVT-Merge Group. *Revised submission for MOF* 2.0 *Query/Views/Transformations RFP version* 2.0. *OMG document id ad*/2005-03-02, *http://www.omg.org/cgibin/apps/doc?ad/05-03-02.pdf, retrieved on* 2007.12.15.

[13] **J. Bézivin, S. Hammoudi , D. Lopes, F. Jouault.** An Experiment in Mapping Web Services to Implementation Platforms. *Technical report*: 04.01, *LINA, University of Nantes. http://lina.atlanstic.net/documents/RR_pdfs/RR-LINA-0401.pdf, retrieved on* 2007.12.15.

[14] **M. Staron, L. Kuzniarz, L. Wallin.** A Case Study on a Transformation Focused Industrial MDA Realization. 3*rd Workshop in Software Model Engineering, http://www.metamodel.com/wisme-2004/present/7.pdf, retrieved on* 2007.12.15.

Received February 2008.

DOI: 10.5755/j01.itc.37.2.11940