COMPARATIVE ANALYSIS OF THE LANGUAGES FOR BUSINESS RULES SPECIFICATION

Krassimir Manev, Neli Maneva

Abstract*: One of the goals of authors of the Business Rules Approach, besides making the process of specification of software more precise and adequate, was to provide a rigorous basis for reverse engineering of business rules from existing software systems. By different reasons this goal was no more a subject of interest of the involved researchers. One of the general problems of using business rules (BRs) in the reverse engineering of software is the selection of language to present the rules, extracted from the source code. In principle, it is not impossible to use for this purpose the languages dedicated for the straightforward task – using BRs for specification of software systems. The current paper presents an overview of different such languages in order to decide whether they are appropriate for rules extraction and to outline those features which will be helpful in language comparison and selection of a specific language for BR.*

Keywords*: software development, software modernization, software reengineering, business rules, languages for specification of business rules, automated business rules extraction, comparative analysis.*

ACM Classification Keywords*: D.2.1 Requirements/Specifications, D.2.7 Distribution, Maintenance, and Enhancement.*

1. Introduction

In the fundamental report [Hay & Healy, 2000] the researchers from the Business Rules Group (formerly known as the GUIDE Business Rules Project) postulated, among the other goals of their project, the following:

- To provide a rigorous basis for reverse engineering business rules from existing software systems.

Then, involved too much in activities to achieve the main purpose of the project – to develop and implement the idea for specification of corporative software systems with *business rules* (BRs) – they did not implemented an approach for reverse engineering and extracting BR from existing systems.

Meanwhile, the necessity of modernization of huge amount of stable and helpful but created on old or/and outdated platforms, called *legacy systems*, became more and more important. One possibility for modernization of a legacy system is to *extract* (rather *automatically* than manually) the *business logic* of the system embedded in its program code (as well as in its database definitions and queries, if they exist) in form of BRs. This process has been called here *automatic BR extraction* (ABRE). After that the extracted rules could be (also automatically) "wearied" in the dialect of the specific domain and could be used by the client for reengineering of the legacy system or for a formal specification of a new one. For this purpose, the ABRE has to present the obtained BRs

with sentences of some *formal* (or at least *well structured*) *BRs extraction language* (BREL) in order to embed them in the ontology of the client's domain.

It seems quite natural to use as BREL some of the languages for the straightforward problem – specification of systems with business rules. Unfortunately, due to different reasons, the languages proposed for specification of systems with BRs are either subset of some general purpose specification languages or specialized, but becoming more and more complicated and inappropriate for ABRE from legacy code. Anyway, in our efforts to create good BREL, we have no other possibility than to use some features of the languages dedicated to specification of software systems with BRs.

In this paper we will consider some of these languages in order to outline their features, from the point of view of ABRE process that will be helpful in the process of creating a language for ABRE. Section 2 presents briefly some of the languages used for specification of software systems with BRs. Section 3 describes the basic principles of the comparative analysis (CA) method and its application for comparison of specification languages. The performed experiments and their results are presented. In Conclusion the pros and cons of the analysis are summarized and a few ideas for future research and development work are mentioned.

2. Languages for specification with BRs

The necessity of some language for writing BRs was stressed even in the mentioned above report of the GUIDE Business Rules Project [Hay & Healy, 2000]. There, on page 12, we read:

> "Note also that each BUSINESS RULE may be *expressed in* one or more FORMAL RULE STATEMENTS. A FORMAL RULE STATEMENT is an expression of a BUSINESS RULE in a specific formal grammar. A FORMAL RULE STATEMENT must be *in the convention of* a particular FORMAL EXPRESSION TYPE, which is to say one of the formal grammars for representing BUSINESS RULES. Examples of a FORMAL EXPRESSION TYPE are structured English, IDEF1X, Oracle's CASE*Method, Object Role Modeling, Ross's notation, and so forth."

So we will consider some examples of formal (or at least structured) languages which are candidates to be used in ABRE, including the pointed by the GUIDE Business Rules Project.

2.1. "Programming languages" - like

The most popular instrument from the group of languages that are similar to programming languages is the *structured English.* It is not a formal language. But something which is very similar and very helpful is popular in the domain of Computer science under the name *pseudo code,* especially for presenting of algorithms at some informal, high level. Really, each author of book on algorithms or professor teaching Algorithms is using her/his own pseudo code and structured English has to be the core of all such pseudo codes.

As there is no formal specification for this instrument, for our discussions we will use the description, which could be considered as commonly accepted [Structuted, 2013]. By this description the structured English "*… aims at getting the benefits of both the programming logic and natural language. Program logic helps to attain precision*

while natural language helps in getting the convenience of spoken languages." Additionally it is mentioned that: "*Structured English or "pseudo code" consists of the following elements:*

- Operation statements written as English phrases executed from the top down;
- Conditional blocks indicated by keywords such as IF (N.B. the couple of keywords IF … ENDIF, indeed), THEN, and ELSE;
- Repetition blocks indicated by keywords such as DO, WHILE, and UNTIL.

and propose to …use the following guidelines when writing Structured English:

- Statements should be clear and unambiguous;
- Use one line per logical element;
- All logic should be expressed in operational, conditional, and repetition blocks;
- Logical blocks should be indented to show relationship;
- Keywords should be capitalized.

We will accept without serious objection only the first of the requirement, **extending it** with "*as much as possible"* because including of operational instructions in English, formulated by even most experienced human being, could not be without any ambiguity.

For the third requirement we have a serious objection. From [Hay & Healy, 2000] it is clear that "fathers" of BR-approach **did not even imagine a possibility to have BRs with cyclic structure**. It is not expected the person that analyses business processes of an enterprise and formulates the rules governing it, to know the conception of Iteration (as well as recursion). So, for the purposes of the BR-approach we probably have to drop the possibility to use the repetition constructors *DO, WHILE, and UNTIL.*

The other three requirements could be classified as optional. They are a matter of syntax and the same result could be obtained in another ways, too. About the last one, it is important the keywords *IF, ENDIF, THEN* and *ELSE* to be easy recognizable and not mixed with the same words, used in operational statement as regular English words. But this could be achieved by placing a special character in front of each keyword also. For example, underscore character – _if, _endif, _then and _else. The fourth requirement is pure "cosmetic" and is dedicated just to make the specification more readable. The second requirement also seems cosmetic but in fact it is semantically important. It is introduced to separate clearly different operational statement one from the other. Practically it could be reformulated in that sense, including the possibility to use, beside a new line character, other separators, too. For example, such separator can be the common accepted in programming languages semicolon sign (;).

Let us consider as an example the rule for an employer of the enterprise to obtain paid leave of absence: In order to obtain a paid leave of absence the employer has to receive the approval of her/his closest chief which is not in absence herself/himself and the number of remaining paid absence days of the employer to be at least as much as the asked days of absence. This rule expressed by structured English will look as follows:

IF *remaining days* are at least as much as *asked days* THEN

IF there is approval from closest chief not in absence THEN

Issue an order for leave of absence

ELSE

Refuse leave of absence

ENDIF

ELSE

Refuse leave of absence

ENDIF

Another very popular language which is “programming language” - like is the *language of flowcharts*. Essential for this language are two kinds of blocks:

- Rectangles with many inputs and one output represent operation statements, including a unique start point and one or more end points of the diagram. The corresponding sentence is written inside the rectangle;
- Rhombs with many inputs and two outputs labeled with **true** and **false**, respectively, representing conditions. The condition itself is written inside the rhomb.

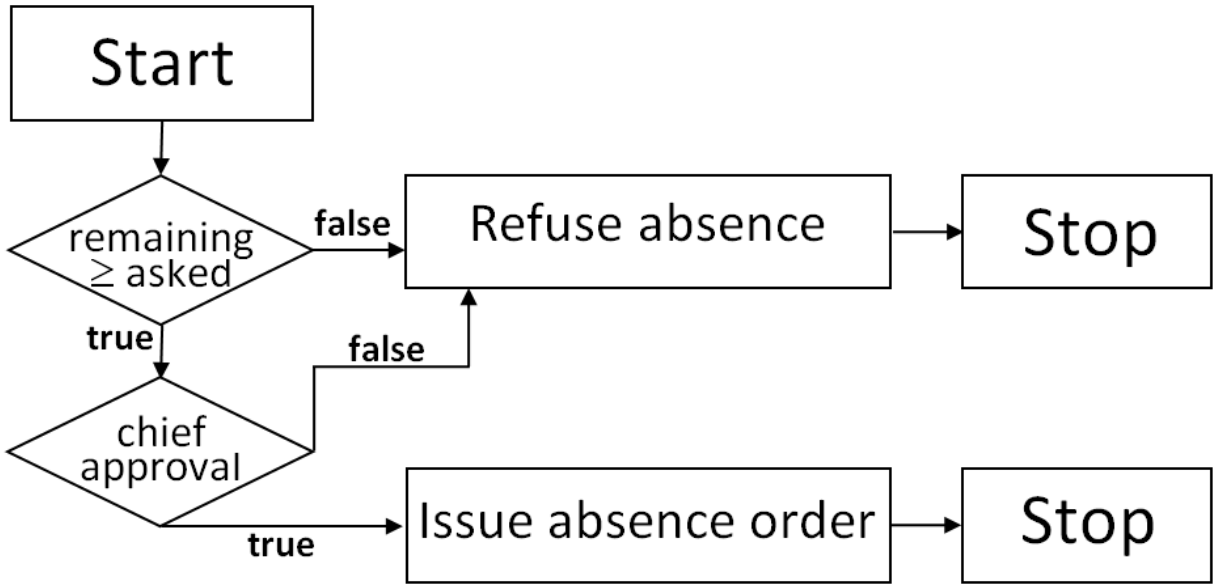

Figure 1. Presenting the business rule by a block-diagram

Rectangles and rhombs are linked with arrows that control the “flow” through the chart. The flowchart in Figure 1 represents the same rule, specified above with structured English.

There are no principle differences in expressive power of the structured English and the flowcharts language. So the structured language is more appropriate for ABRE process and flowcharts are more appropriate for visualization of the extracted rules.

2.2. “Entity relationship” - like

The languages IDEF1X [IDEF1X, 1993], Case*Method (of Oracle) [Barker, 1990] and Object-Role Modeling language, mentioned in [Hay & Healy, 2000] are representatives of a class of languages, the main purpose of which is to specify the data model (or data scheme) of an information system. To the same class could be added also the Bachman notation, Barker's Notation, EXPRESS, Martin notation, Z-notation (of Jean-Raymond Abrial), UML, Merise, etc. These languages are very popular under the general name *Entity-relationship* languages. The descriptions created with these languages are called *Entity relationship diagrams* (ERD) [Chen, 1976]. That is why we will consider very briefly the possibility to extract BRs in form of ERD.

Basically, the entity relationship languages are graphical. ERD is a graph with labels on the vertices and edges. Vertices represent entities of the modeled domain and the edge that links two vertices represents the existing between them relationship. The label of a vertex usually includes the name of the entity and its structure when the entity is a complex object. The label of an edge contains the name of the presented relationship and its type (one-to-one, one-to-many, and so on). The part of the data model of a corporative system concerned with “leave of absence asking” example, presented with ERD, is shown in the Figure 2.

It is obvious that graphical languages are very helpful for visualization of the data model of an information system and for manual processing but could not be used as input or output of some automated procedures. The relation between ERDs and the specifications which are more appropriate for automated processing has been studied in [Chen, 1997]. There the author made very interesting parallel of “ER vs. structured English” relationship and the relationship between Chinese characters (hieroglyphs) and European alphabets (which are to very high degree phonetic). In the same paper the author proposed a simple procedure for translation of an ERD to specification, written in structured English.

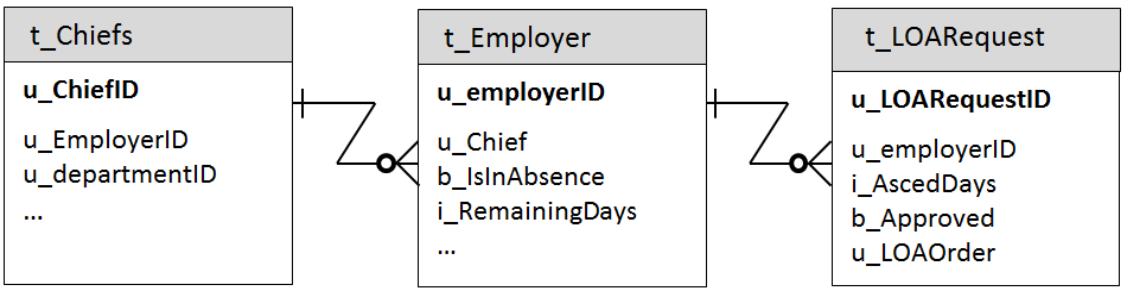

Figure 2. ERD presentation of a business rule

A similar approach could be found in the document, describing the IDEF1X language [IDEF1X, 1993]. In the Annex B of this document, called Formalization (page 130), for each IDEF1X is proposed to map it “to an equivalent set of sentences in a formal, first order language. “The author statement is that in such a way ERD could be “considered a practical, concise way to express the equivalent formal sentences”.

Obviously, the graphical form of specifications with the numerous ER languages is not intrinsic negative. The serious negative feature of these languages is that they are rather “static”, dedicated to specify the structure of

the data of the system, not the operations with data. Of course, looking at the ERD of Figure 2 an experienced system designer could imagine what the rule for approving an employer's leave of absence request is. But in principle it will be better for the adequateness of the design not to rely on the designer's imagination.

2.3. Ross's notation

Ronald Ross is considered as a "father" of the Business Rules approach. His first works on the concept appeared in the middle of 90's of the past century, before joining to the GUIDE Business Rules Project. In parallel with developing of the concept for specification of systems with BRS, he was working on creating of a language, especially dedicated for writing BRs. As a result he has created a description tool called in that time *Ross' notation* or *Ross' method*. With the developing and ameliorating of the approach Ross developed and ameliorated his own description tool. Nowadays the notation created by Ross is popular under the name RuleSpeack (a trademark of the created and ruled by Ross company *Business Rules Solutions, LLC*).

The idea of RuleSpeak is, besides putting a particular structure on the English sentences, to introduce some limitations on the used word in order to make the structured English sentence clearer and less ambiguous. For example, it is strongly recommended to use only the word *must* instead of sophisticated phrases of obligation as *is strictly required,* as well as only *must not* instead of sophisticated phrases of defending as *is not allowed.* The notation RuleSpeak, also, strongly recommends avoiding the word *can* that gives the impression for existence of some degree of freedom. Instead it is recommended to replace the word *can* with *… may …. only if …* to eliminate any freedom.

Another principle of RuleSpeak is to keep the sentences as short as possible. So, in our example, the rule under consideration will be expressed in RuleSpeak as follows:

*The employer **may** take leave of absence **only if** received the chief's approval.*

and

*Asked days leave of absence **must be** no more than remaining days.*

2.4. OMG Semantic and Vocabulary of Business Rules

Soon after publishing the report of the GUIDE Business Rules Project it attracted the attention of the specialists from the powerful OMG which decided to support the project and to make Business Rules approach a standard of OMG for specification of systems knows as Semantic and Vocabulary of Business Rules (SVBR) [SVBR, 2008]. This standard incorporates three different specification tools – structured English, RuleSpeac and Object-Role Modeling language. With this act OMG really confirmed the expectations that a simple language concept will not be expressive enough for writing BRs.

2.5. Other BR-related languages and their comparison

A study of some BR modeling languages is presented in [Rima, 2013]. A number of BR specification languages (Simple Rule Markup Language, Semantics of Business Vocabulary and Rules, Production Rule Representation, Semantic Web Rule Language, Object Constraint Language) and Business Process specification languages

(Unified Modeling Language, Data Flow Diagram, Colored Petri Nets, Event-driven Process Chain, IDEF3, Business Process Modeling Notation) have been briefly described. The main purpose of the performed comparative analysis has been to show the representational capabilities of the selected BR specification languages and to describe which modeling aspects have been covered by them within the three layer framework, proposed for BR-based software modeling.

We think that a more flexible approach to comparison of BR languages should be developed. Next follows a brief description of our ideas.

3. An approach to comparison of languages for specification with BRs

The results of the performed study can be summarized as follows:

- There is a variety of BR specification languages, used to achieve specific goals at different stages of the software systems life cycle;
- Most of BR specification languages are especially designed and used for the purposes of the straightforward task – when a new software system should be developed from scratch and the process of requirements elicitation is just started.
- None of the existing BR specification languages has been recognized as a universal and accepted as a standard till now.

Taking into account these conclusions, we try to select a procedure for comparison of BR languages so as to meet the following requirements:

- Comparison should be flexible, easily adjusted to the context - particular circumstances, in which it is accomplished;
- The analyzed and evaluated characteristics (i.e. quality content of the BR languages) and the set of compared languages should be defined in accordance with the currently defined context;
- Comparison should be supported by a formal method and a systematic procedure for its application.

Next follows a brief description of our approach, based on the Comparative analysis method and an example, illustrating its feasibility.

3.1. The essence of the Comparative Analysis method

The method of Comparative Analysis (CA) has been introduced in [Maneva, 2007]. It shares the main objectives and methods of the Multiple Criteria Decision Making theory, trying to specify and apply them systematically.

Generally speaking, the ***Comparative Analysis*** method is a study of the quality content of a set of homogeneous objects and their mutual comparison so as to select the best, to rank them or to classify each object to one of the predefined quality categories.

For CA use in practice, we distinguish two main roles: the ***Analyst***, responsible for all aspects of CA implementation, and a ***CA customer*** - a single person or a group of individuals, tasked with making a decision in a given situation. Depending on the identified problem to be solved by a customer at a given moment, a ***case***

should be opened to determine the context of the desired comparative analysis. Each case is specified by the following 6 elements:

case = { View, Goal, Object, Competitors, Task, Level}

The ***View*** describes the CA customer's role and the perspective from which the CA will be performed. Taking into account the responsibilities and some typical tasks of the main participants in the BR extraction, the following Customer's roles have been identified [Maneva & Manev, 2011]: ***Business Analyst, Policy Maker, Software Architect,*** and ***Software Developer.*** Thus a lot of cases for comparison of BR specification languages can be further described, reflecting the specific participant's point of view to the analyzed case.

The **Goal** expresses the main customer's intentions in CA accomplishment and can be to describe, analyze, estimate, improve, or any other, formulated by the Customer, defining the case. All of these goals can be stated for the CA of languages for specification with BRs.

The **Object** represents the item under consideration. For each object for CA application a *quality model* should be created – a set of characteristics, selected to represent the quality content in this context, and the relationships among them. For the problem under consideration, the investigated Object is a BR specification language.

According to the stated goal, the set C of **Competitors, C = {C_1, C_2, ..., C_n}** – the instances of the objects to be compared – should be chosen. When the Goal is to perform the CA so as to obtain the ranking of a number of objects, the set **C** comprises those competitive objects.

The element **Task** of a case can be *Selection* (finding the best), *Ranking* (producing an ordered list), *Classification* (splitting the competitors to a few preliminary defined quality groups) or any combination of them. There are no special considerations, when we define the element ***Task*** of a case for BR language comparison.

The Depth **Level** defines the overall complexity (simple, medium or high) of the CA and depends on the importance of the problem under consideration and on the resources needed for CA implementation.

3.2. Comparison of languages for specification with BRs – a quality model

It is obvious, that the above described approach meets all stated requirements, especially for flexibility, because by definition of a case we can describe completely the current situation, specifying who, why and what exactly should be analyzed in order to make a reasonable choice to support the corresponding decision.

One of the most difficult steps in the CA use is the creation of a model, adequate to the quality content of the studied object. As we have already stated, the model comprises different quality characteristics, identified as significant for the performed comparative analysis. Usually the Analyst is responsible for object modeling made with help and guidance of the Customer, ordered the CA. For the purposes of language comparison, we have to start with constructing a quality model for the object "BR specification language". From practitioner's point of view it is important to mention, that we will stick to the incremental object modeling, described in [Maneva, 2007]. When the object "BR specification language" appears in a case for the first time, a quality model for it is created and saved as a basic (generic) one in a repository. When the same object reappears in another case, its generic quality model is invoked and modified according to the context, defined by this new case. The modification can be to add some new quality characteristics at any level of the hierarchy or to delete some characteristics. The

changed version of the model is saved as a derivative model, together with the generic model, which can be enriched with some additional quality characteristics. In this way the generic model is always the complete hierarchical structure, comprising all quality characteristics ever considered.

In Table 1 is shown an initial quality model for the object “BR specification language”. It comprises two groups of quality characteristics - external and internal. Of course, for complete description of the quality model we have to provide precise definitions of all characteristics and to describe the metrics for leaves of the constructed hierarchical structure, allowing quantitative evaluations, necessary for CA Task accomplishment. So far this is beyond the scope of this paper.

Table 1.

External	stability
	fidelity
	usability
	availability of automating tools
Internal	expressiveness
	understandability
	validity
	cognitive complexity
	transformability
	degree of automation

3.3. Comparison of languages for specification with BRs – an example

The CA use can be illustrated by the next real-life ***example:*** The team, involved in a real-life modernization project, has to choose one specification language for Business rules, extracted from the source code of a legacy system.

With the help of the Analyst, a case has been defined:

case = { View, Goal, Object, Competitors, Task, Level},

where the elements, determining the context of the desired Comparative analysis, are:

- ***Goal:*** To compare some BR specification languages, recognized as appropriate.
- ***View:*** The identified point of view for this situation belongs to those participants in the modernization project, who are supposed to use a BR specification language, namely the Business Analyst and the Policy Manager.
- ***Object:*** A BR specification language.

For this case a simple linear quality model has been constructed. It comprises only three quality characteristics, considered as equivalent (i.e. having equal weights – coefficients of importance):

- *Expressiveness* – the capability of the BR language to represent concepts and communicate ideas about business organization and management;
- *Usability* – the capability of the BR language to be understood, learned and used;
- *Efficiency* - the capability of the BR language to provide appropriate results relative to the amount of the resources used.

❖ ***Competitors***: The set C of Competitors comprises the languages, identified as appropriate, e.g. mentioned above in Section 2.

❖ ***Task***: Selection – finding the most appropriate among the compared languages.

❖ ***Level***: Simple.

Following the described in [Maneva, 2007] procedure for CA use, we can find the most appropriate (in accordance with the selected quality characteristics) specification language.

Conclusion

The paper explains the need of some languages for specification with BRs for the purposes of reverse engineering of legacy system. An overview of such languages has been made. In order to facilitate the selection of a BR specification language, which is the most appropriate in a given situation, some requirements to the method of comparison have been defined. The proposed method of CA is presented and illustrated by an example.

A few possible directions of further research are:

- To study each of the identified as appropriate languages for specification with BR so as to select only a few of them to be further used (separately or in a combination);
- To continue the quality content modeling for other objects related with the CA use in the process of the BR extraction: ***products*** as additional sources of BR information (documentation, developer's and user's stories, test data and scenarios), ***processes*** like BR tracking, change management in reverse engineering, etc.

Acknowledgments

This work is supported by the National Scientific Research Fund of Bulgaria under the Contract ДТК 02-69/2009.

Bibliography

[Barker, 1990] R. Barker, Case*Method: Entity Relationship Modelling, Addison-Wesley, 1990.

[Chen, 1976] P. Chen. The Entity-Relationship Model: Toward a Unified View of Data. In: ACM Transactions on Database Systems, v. 1, 1976, pp. 9-36.

[Chen, 1997] P. Chen. English, Chinese, and ER Diagrams. In: Data & Knowledge Engineering, v. 27, 1997, pp. 5-16

[Hay&Healy, 2000] D. Hay and K. A. Healy (eds.). Defining Business Rules ~ What Are They Really? GUIDE Business Rules Project Final Report, rev. 1.3., July, 2000.

[IDEF1X, 1993] INTEGRATION DEFINITION FOR INFORMATION MODELING (IDEF1X), Federal Information Processing Standards, Publication 184, December 1993. http://www.itl.nist.gov/fipspubs/idef1x.doc, last visited on 13.12.2013.

[Maneva & Manev, 2011] N. Maneva, Kr. Manev. Extracting Business Rules – Hype or Hope for Software Modernization. Int. Journal "Information Theories & Applications, vol. 18-4/2011, pp. 390-397.

[Maneva, 2007] N. Maneva. Comparative Analysis: A Feasible Software Engineering Method, Serdica J. of Computing, 1(1), pp. 1-12.

[Rima, 2013] A. Rima, O. Vasilecas, A. Smaizys. Comparative Analysis of Business Rules and Business process Modeling Languages. Computational Science and Techniques online J. Volume 1, Number 1, 2013, 52-60.

[Ross, 1994] R. G. Ross, The Business Rule Book: Classifying, Defining and Modeling Rules, Boston, Massachusetts: Database Research Group, Inc., 1994.

[Stineman, 2009] B. Stineman, IBM WebSphere ILOG Business Rule Management Systems: The Case for Architects and Developers. IBM Software Group, November 2009. http://www-01.ibm.com/software/websphere/products/business-rule-management/, last visited on 15.02.2012.

[Structured, 2013] Structured English, Wikipedia. The Free Encyclopedia, last visited on 13.12.2013. http://en.wikipedia.org/wiki/Structured_English

[SVBR, 2008] Semantics of Business Vocabulary and Business Rules (SBVR), v1.0 OMG Available Specification, 2008.

Authors' Information

Krassimir Manev *– Assoc. Professor, Ph.D., Department of Informatics, New Bulgarian University, 21 Montevideo str., Sofia 1164, Bulgaria; e-mail: kmanev@nbu.bg*

Major Fields of Scientific Research: Discrete mathematics and Algorithms, Formal Methods in Software Engineering, Source Code Analysis, Competitive Programming and Education in Informatics.

Neli Maneva *– Professor, Ph.D., Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, Acad. G. Bonchev str., bl. 8, Sofia 1113, Bulgaria; e-mail: neman@math.bas.bg*

Major Fields of Scientific Research: Software Engineering, Software Quality Assurance, Model-driven software development.