Issues and Challenges of Business Rules Modeling in Software Systems for Business Management

Anca ANDREESCU, Marinela MIRCEA
Bucharest University of Economic Studies, Romania
anca.andreescu@ie.ase.ro, mmircea@ase.ro

Software systems for business management appeared as a result of the growing need to ensure a consistent IT support for most of the business activities that organizations have to deal with. Moreover, organizations continue to struggle for obtaining competitive advantages on the business market and to lower the cost of developing and maintaining computer systems to support their operations. As business rules play an important role within any organization, they should be taken into consideration as distinct elements when developing a software system that will operate in a collaborative environment. The paper addresses the problem of business rules modeling, with special emphasis on incorporating business rules in Unified Modeling Language (UML) models.
Keywords: *Business Rules, Software Systems, Business Management, UML Models, OCL*

1 Introduction

While software development environments productivity is still growing strong, studies on the causes of software projects failure consistently places poor quality requirements on top of the hierarchy of these causes [1]. An explanation of this situation is that the development teams allocated too little time understanding the real business problems, the user needs or the nature of the underlying environment in which the system will run. Moreover, developers are trying to provide technical solutions as quickly, but based on insufficient understanding of the problem's requirements [2].

In most cases, difficulties in requirements modeling and analysis arise from insufficient understanding of the logic part of the application, known as business logic. Business logic is the defining element for a business being in the process of modeling and automation, and it includes both business rules (BR) and workflow (process), which describes the transfer of documents or data from one participant (person or software system) to another.

It is common knowledge that every organization operates according to a set o business rules. These may be external rules, coming from legal regulations that must be observed by all organizations acting in a certain field, or internal rules which define the organization's business politics and aim to ensure competitive advantages in the market. Starting from the previous observations, it is obvious the important role that business rules play within the development process of an organization's software system [3]. And this especially applies to Software Systems for Business Management (SSBM), as they are suitable to incorporate a large amount of business rules.

Doing business today is mainly about creating and maintaining strategies and connections. While strategies must comply with business rules, an organization is more likely to succeed in its business activities if it creates a support for strong collaborations between managers, employees, clients and any other stakeholders. Thus, using a collaborative SSBM is a must, not a need in order to operate in a collaborative environment.

An analysis of SSBM's components, in terms of their functionality and how they relate to business rules reveals the important role that the business rules play in the development process of these systems. The results of this analysis are summarized in Figure 1 and describe hereinafter.

Very often, software components are discussed in the context of component-based software systems. In this paper a software component will be perceived in a wider sense, as an element of the system which

DOI: 10.12948/issn14531305/18.1.2014.15

provides implementation for a service or predefined event, being encapsulated and able to communicate with other system components.

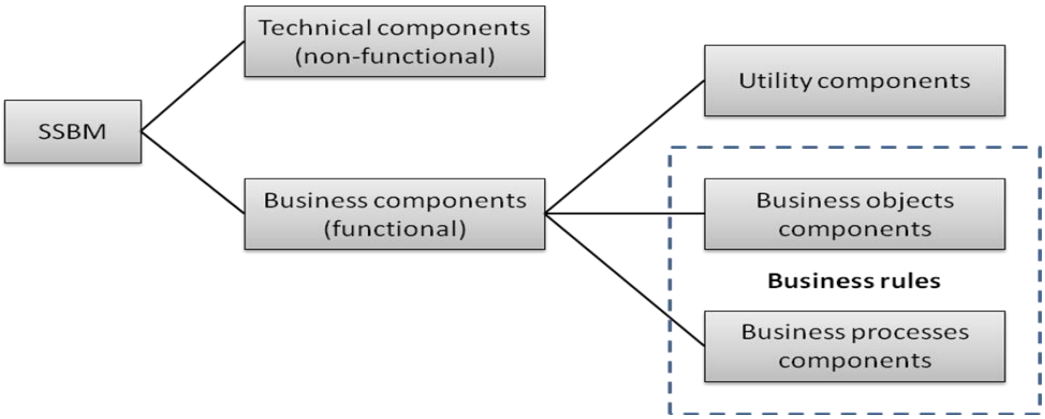

Fig. 1. Components of a software system for business management

According to [4], a software application may contain two types of components: Technical components and Business components. Technical components are non-functional components used to build the technical architecture by providing reliable and reusable solutions that have a recurring problem (ensuring, for example, networks communication or persistence). On the other hand, a Business component is a representation of the nature and behavior of real world entities, as they can be found in an organization's vocabulary (customer, account etc.).

In terms of independence from specific organization's requirements, at the lowest level of a SSBM we can identify components that supply business utility services [5]. For example, an address book, a catalogue or a component that deals with interest rates. This type of component encapsulates little or no business rules and can be regarded as a utility function. A "function " component can act as a lookup table, indexed by a key, a mathematical function or a combination of input parameters that will provide (almost) always the same result. These components are, by their nature, very stable and could be reused within a particular business area.

At the next level there are components that encapsulate business objects which could manage, for example, customers, a bank account or book copies. These components are also relatively stable in the sense that once developed, they will subsequently undergo minor changes, mostly due to the need to add information or additional roles. Many business rules can be found in these components, so their reusability is likely to be restricted to a particular business area.

On the highest level there are the components that manage business processes. They contain objects that store events such as borrowing a book or ordering a product. Business objects will play a role in the related process event and, therefore, will be recorded in that event. A process component is less reusable than the other two types and includes business rules governing the process. These components are less stable and they tend to change frequently as the organization seeks better ways to conduct the business. Business rules can be stored in the process related component or can be encapsulated in a separate component that will act as a plug-in. By using the latter approach, a process will become more general because we introduce certain flexibility in the execution order of activities and impose business rules to restrict this order.

2 Explicit Manipulation of Business Rules

Businesses are controlled by rules that regulate how the business operates and is structured. Often they are not even considered rules but are referred to as "facts" of the business. Rules ensure that the business is run according to predefined external laws or regulations or internal restrictions or goals

DOI: 10.12948/issn14531305/18.1.2014.15

[6]. Enforcing business rules will make the business to function as efficiently and profitably as possible, while fulfilling its goals.

Given these facts, it is obvious the important role that business rules play within the development process of SSBM. From the business rules manipulation perspective, the support offered by the traditional software development methodologies (both structured and object oriented) is very limited, because they: a) do not define a structured process for business rules identification, specification, analysis and implementation; b) allow business rules to be scattered in different parts of the system, which cumbers the possibilities to track and change rules or to ensure their uniqueness.

These are motivations that have entailed business rules to be treated as distinctive elements of the software development process, movement that had started almost twenty years ago, as conceptual studies, and has been consolidated during the last ten years, through the so-called *business rules approaches* [7]. Such an approach formalizes business rules that are critical within an organization and specify them in a language that can be easily understand by all the stakeholders.

It is obvious that, for software technologies, the adoption of a business rules approach is a natural step forward in increasing productivity. Promoters of the business rules approaches, such as Barbara Von Halle [8] or Ronald G. Ross [9] advocate that many problems related to frequently requirements changes could be solved using such an approach and some of them assert the advantages of using commercial rule based software products, also known as Business Rules Management Systems (BRMS). Essentially, this kind of systems externalizes business rules and provides facilities for a centralized business rules management.

Though it is an important software development strategy in the context of the new challenges brought in by the on-going extension of electronic businesses, a business rules approach can significantly increase the development efforts, because, at methodological level, it extends the software development cycle, and, at the technological level, commercial BRMS (such as IBM ILOG Business Rule Management Systems or Visual Rules) imply major costs. These represent restrictions that have limited the use of business rules approaches mainly to systems that are specific to large organizations or belong to a very specialized business domain, such as insurance or telecommunications.

However, there is the possibility to apply the underlying principles of business rules approaches, at some extent, also for the SSBM that do not fit the categories mentioned above. In this context, we use the concept of *explicit manipulation of business rules* in a software system, designating any attempt to treat business rules as independent assets in any stage of the development cycle. This means that the developers might use the advantages of the underlying principles of business rules approaches without being compelled to use a business rules engine. The design of a general development process (based on the Unified Software Development Process [10]) that is capable to integrate a set of necessary activities for the explicit manipulation of business rules was presented in [3]. Figure 2 depicts the implications of using business rules within the SSBM development process, starting from the four main necessary activities related to business rules: identification, specification implementation and management.

The above mentioned activities are all equally important in successfully managing business rules. Business rules identification may be the hardest part, because, depending on the enclosed information, business rules could be based on explicit or tacit (implicit) knowledge [11]. Once identified, business rules have to be specified in an appropriate manner in order to be understood by all persons involved in the development process. While business people are not so often familiar with specification languages that require a higher level of formalization, developers require that business rules statements to be unambiguous in order to allow an easy transition towards source code. This challenge can

DOI: 10.12948/issn14531305/18.1.2014.15

lead to the following conclusion: business rules must be specified at different levels of formalization, starting from natural language and ending to formal descriptions.

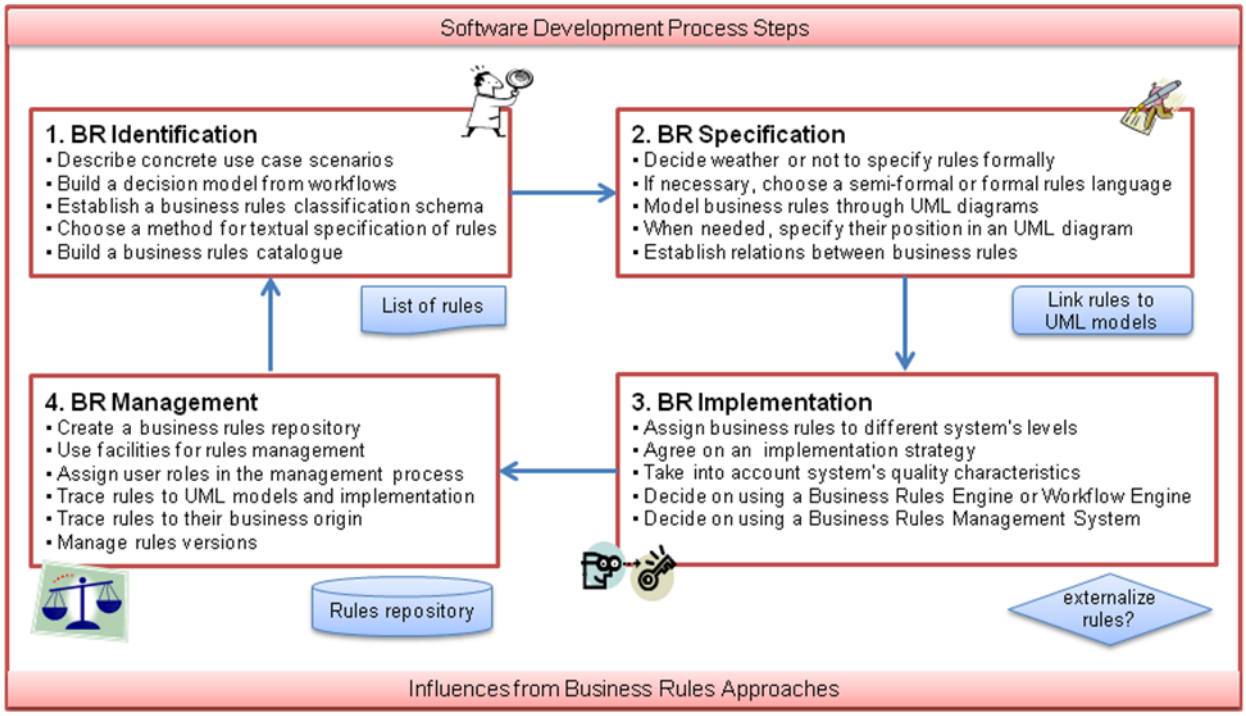

Fig. 2. Implications of business rules explicit manipulation

The Unified Modeling Language (UML) is a standard semi-formal specification language and is able to describe, through its models, many aspect of a software system. Furthermore, several types of constraint can be applied to the models' elements in order to add supplementary information. In many cases, this information actually represents business rules. Hereinafter the article focuses on visual and formal representation of business rules within UML models.

3 Business Rules Modeling in UML

Business rules modeling aims at representing business rules in various models in order to be more easily understood by developers. This largely depends on the system's type, the development methodology, the type of rule and so on, and can vary from simple graphics to complex representations such as decision tables, decision trees or activity diagrams.

Any software system can be represented by one or more models that correspond to different aspects of the system. Since most SSBM are complex, a complete and detailed model of such a system will include several models that can be handled separately. Object-oriented development methodologies recommend that a system should be built from different perspectives. Thus, Object Modeling Technique (OMT) proposes three models for three different purposes, namely models that can describe either objects or interactions or transformations: object model, dynamic model and functional model. On the other hand, the Unified Software Development Process recommends modeling a system around three visions: use case view, logical view and components view.

However, a complete model of a SSBM, must also address the problem of business rules modeling. UML does not have special notations for the visual description of business rules, but rules that are represented in or by different UML diagrams can stand for a business rules model. Problems arise when a complete business rules model should be represented and analyzed as a whole, because business rules are scattered in various system's models [12]. A viable solution is to place business rules in a repository, from

DOI: 10.12948/issn14531305/18.1.2014.15

where they can be analyzed in a systematic manner. Also, by establishing a relationship between different models and the rules in repository, rules' traceability can be achieved.

The following paragraphs contain an analysis of how business rules are included in or affect different types of UML diagrams. Synthesis of this approach is outlined in Figure 3.

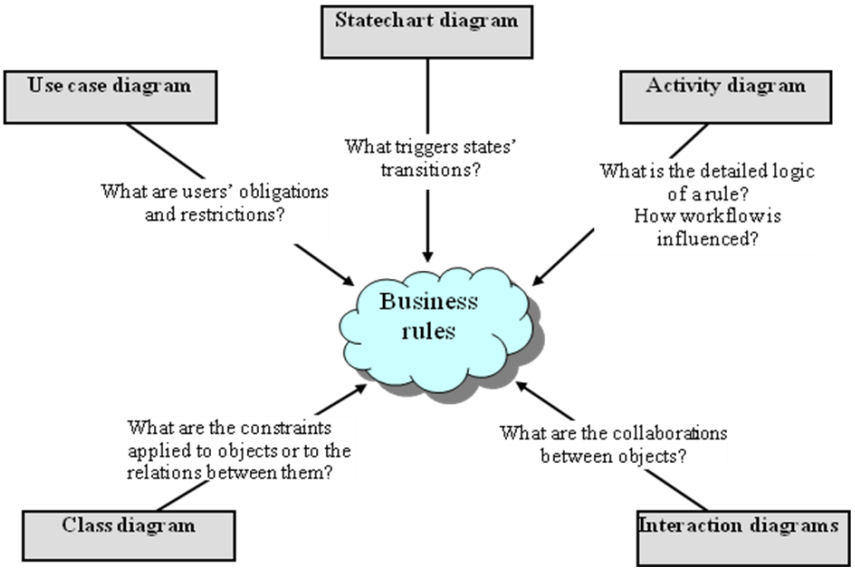

Fig. 3. Placing business rules in the context of UML diagrams

Activity diagrams can be used to model the logic of the operations described in one or more UML use cases. They are similar to technological process schema and data flow diagrams encountered in structured development methods. In fact, activity diagrams are created as a finite set of serial actions or a combination of serial and parallel actions [13]. In an activity diagram, a business rule can be associated to a ramification within the business process. For example, in Figure 4 there are two business rules that influence the workflow, corresponding to each decision point (represented as a diamond in the diagram). The first rule determines the Standard customer to pay the order before issuing the delivery, while the Premium customers can pay after the order was sent for delivery. The second rule is a condition that specifies that if an order is urgent, than it must be delivered within 12 hours, otherwise it will be delivered as usual.

Use case diagrams model the behavior of a system by linking system's functions with its actors. Business rules within use case diagrams are mainly statements that describe the duties and powers of actors in the system. It is important to describe how tasks are assigned to actors, by including some degree of obligation. It influences the way business rules are formulated. Mandatory requirements are designated by use of the expression "must do", while non-binding requirements are designated by using "can / could do" expressions.

Other types of rules that naturally belong in a use case are those that describe the conditions representing exceptions to the baseline scenario. For example, in the above activity diagram, a Premium customer is allowed to pay an order after or while it was delivered. Suppose we introduce a supplementary business rules according to which, for orders that have the value greater than 500 Euro, Premium customers must pay a deposit representing 20% of the order value. This rule will be an exception (and will generate an alternate flow) to the basic flow of events in the use case "Premium customer pays order".

Sequence and collaboration diagrams are used to describe how users accomplish their tasks. These include business rules that determine the exact order of actions to be performed by user and system in order to carry

DOI: 10.12948/issn14531305/18.1.2014.15

out a particular task.

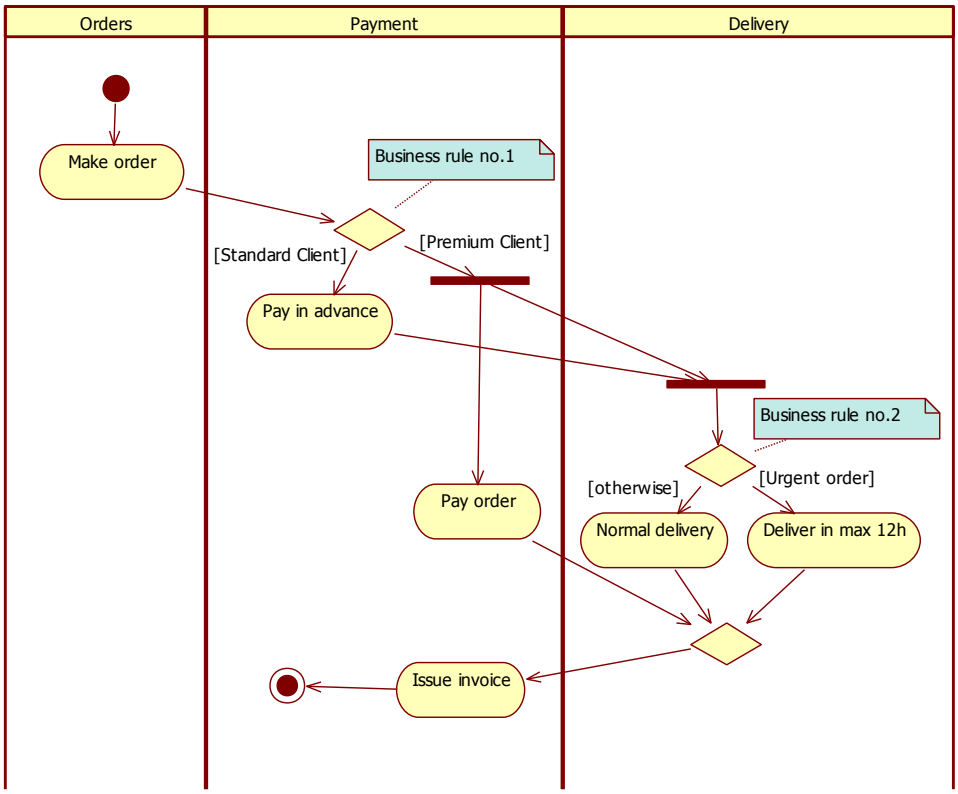

Fig. 4. UML activity diagram for orders processing

Statechart diagrams are useful for modeling the life cycle of an object by specifying the sequence of object's states in response to the occurrence of events and under certain conditions. In most cases, these are Event - Condition - Action (ECA) business rules. Frequently, ECA rules are represented by statechart or activity diagrams.

Class diagrams are used to depict classes of business objects and the relationships between them. These always include specific business rules that express the constraints applied to objects or the properties that govern the relationships between objects. In other words, we can say that the UML diagrams have built-in visual syntax support for defining certain types of business rules. A class diagram has structural constraints in its relationships to depict the multiplicity of an association. UML requires that the multiplicity between classes be defined when defining an association between two classes. The multiplicity is actually a rule that defines how many objects of one class can or must be associated with an object of the other class. For example, we will consider the following two facts which can be found in a banking system: a) **Person** applies for **Credit** and b) **Consumer** and **Mortgage** are types of **Credit**. The first fact was represented in Figure 5 as an association relationship between class **Person** and class **Credit,** while the second fact was described as a generalization relationship between super-class **Credit** and subclasses **Consumer** and **Mortgage**. More details on how to textually specify business rules using a pattern language can be found in [14].

The previous examples described business rules that are included by default in a class model. In addition, a class can add supplementary documentation in the form of notes or constraints. Constraints are generic UML elements for defining formal rules. They are expressed in UML within curly braces close to the model element that it affects and can

DOI: 10.12948/issn14531305/18.1.2014.15

be specified either with a formal language or more informally through natural language. The advantage of specifying via a formal language is that it is easier to ensure unambiguous specifications. The note attached to the **Account** class in Figure 5 shows that for this class the minimum amount deposited in an account must be 100 Euro.

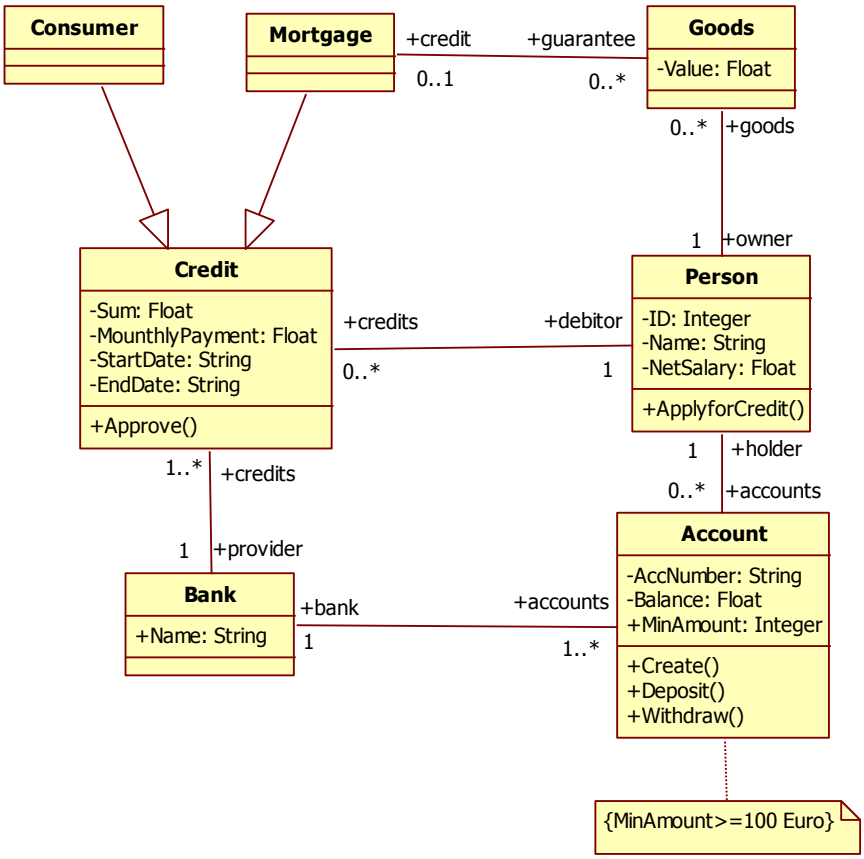

Fig. 5. UML class diagram for a banking system

Usually, constraints are specified in natural language and practice has shown that this method wills always generate ambiguities. In order to describe explicit and unambiguous constraints, the so-called formal languages have been created. Though, the main disadvantage of traditional formal languages is that they are difficult to use by people that do not have a solid mathematical background. Object Constraint Language (OCL) aims to fill this gap and it is a formal language, but at the same time, easy to read and write. OCL [15] was defined as a part of UML specification and became a standard for specifying rigorous expressions that can add essential information to object-oriented models and other object modeling artifacts.

OCL is not an implementation language and cannot be used to specify actions, such as what the result is of violating a specific rule or what is performed when a rule evaluates to a specific value. These actions are best depicted in a UML activity diagram or through writing pseudocode. OCL is a *typed language*, in which all the operators are of a specific type and each operator can only be applied on specific operand types [6]. This is why, for any OCL expression, a context must be provided.

There are many uses for OCL in order to augment UML models. Some of the most common are described in Table 1, together with examples of business rules specified in natural language and also in OCL. These business rules are applied to the UML banking model in Figure 5.

DOI: 10.12948/issn14531305/18.1.2014.15

Table 1. Uses of OCL expressions

Element	Description	Informal business rules	OCL expression
Invariant	Condition that must be true at all the times for all objects of a class.	The minimum amount deposited in an account must be 100 Euro.	**context** Bank **inv**: self.Accounts.MinAmount>=100
Pre-condition	Specifies what must be true before an operation on a class is performed.	Before creating an account, a person must deposit a minimum amount of money.	**context** Account :: Create (sum: Integer) **pre**: sum > MinAmount
Post-condition	Specifies what will be true after the operation has been performed.	When deposit money in an account, its balance will increase just with the added value amount of money.	**context** Account :: Deposit (s: Integer) **post**: Balance = Balance @pre + s
Derivation rule	Specifies how a specific value is calculated from other values.	Calculate the number of accounts a person owns.	**context** Person ::numberAccounts: Integer **derive**: self.accounts →size()

Guard	Specifies whether to perform a specific activity or, when several alternatives exist, an alternative.	In order to be approved a credit must be verified and preapproved.	context Credit::Approve( ) : void
			pre: state = #verified
			post: if
			(oclIsInState(#preapproved))
			state = #approved
			else
			state = #denied
			endif

However, we must also consider the limitations of this language. Even if by automatic code generation more complete code sequences are produced, not all OCL expressions may be directly executable. This may be a major drawback in an industry that requires shorter delivery times. According to [13], OCL must be used in situations where the ability to generate code may be a requirement, but seems harder to create UML models that generate granular code than it is to write the code itself.

4 Conclusions

Software Systems for Business Management must offer support for at least three key elements: automate organization's underlying activities and decisions, enforce internal and external business rules and operate in a collaborative environment through the communication with other systems. All the above elements are equally important for a business's success, but the first two may be considered critical. A good understanding of the system's business requirements and how business rules are included in and influence these requirements represent the very first step in developing a SSBM. However, an important distinction must be made between business rules and system rules because sometimes a system imposes rules that do not support any business rule [16]. In [9] Ross presented an extensive analysis of this distinction between rules. Since UML is a standard for software modeling, this paper presented various ways on how to include business rules into UML models, by using the language elements or additional constraints. Because business rules that are described more formally than in natural language are more suitable to be unambiguous and well understood by the development team, the use of OCL expressions for business rules specification was also addressed. Though, practice has shown reluctance in using OCL during the requirements analysis stage, when developers work with business people, considering that it is more appropriate for the design phase. Future research will focus on defining UML stereotypes for business rules and including these in Computer Aided Software Engineering (CASE) tools.

References

[1] B. McGraw, "Why Software Projects Fail", May 2009, available at: http://fearnoproject.com/2009/05/01/why-software-projects-fail/

DOI: 10.12948/issn14531305/18.1.2014.15

[2] D. Leffingwell, D. Widrig, "Managing Software Requirements: A Use Case Approach, Second Edition", *Addison Wesley*, 2003.

[3] A. Andreescu, "A General Software Development Process Suitable for Explicit Manipulation of Business Rules," *The proceedings of the "CompSysTech '09" International Conference*, Ruse, Bulgaria, pp. IIIA.9-1, June 2009.

[4] J. Grodziski, „Méthodes de développement logiciel : Intégration de l'approche évènementielle à l'approche par composant", *Mémoire de DESS systèmes d'information et de communication, University of Paris*, 2000.

[5] I.Graham, „Business Rules Management and Service Oriented Architecture: A Pattern Language", *Wiley*, 2007.

[6] H.E. Eriksson and M.Penker, "Modeling with UML: Business Patterns at Work", *John Wiley & Sons*, 2000.

[7] Business Rules Group, "BRG's Business Rules Manifesto", 2003, available at: http://www.businessrulesgroup.org

[8] B. Von Halle, L. Goldberg and J. Zachman, "Business Rule Revolution: Running Business the Right Way", *Happy About*, Cupertino, 2006.

[9] R.G. Ross, „Business Rule Concepts: Getting to the Point of Knowledge, 3rd Edition", Business *Rule Solutions*, 2009.

[10] G. Booch, J. Raumbagh and I. Jacobson, „The Unified Software Development Process", *Addison-Wesley*, 1999.

[11] I. Nonaka and H. Takeuchi, „The Knowledge-Creating Company", *Oxford University Press*, Oxford. 1995.

[12] O.Vasilecas and E. Lebedys, „Moving business rules from system models to business rules repository", *INFOCOMP*, Vol.5, Nr.2, June 2006.

[13] P. Kimmel, "UML Demistified", *McGraw-Hill/Osborne*, 2006.

[14] M. Mircea, A. Andreescu, "Using Business Rules in Business Intelligence", *Journal of Applied Quantitative Methods*, Volume 4, Issue 3 - September 2009.

[15]- J. Warmer and A. Kleppe, „Object Constraint Language: Getting Your Models Ready for MDA, Second edition"**,** *Addison Wesley*, 2003.

[16]- G. Witt, "Writing Effective Business Rules", *Morgan Kaufmann Elsevier*, 2012.

Anca Ioana ANDREESCU, PhD is an associate professor at the Bucharest University of Economic Studies, Faculty of Cybernetics, Statistics and Informatics, Department of Economic Informatics and Cybernetics. She published over 20 articles in journals and magazines in computer science, informatics and business management fields, over 30 papers presented at national and international conferences, symposiums and workshops. In January 2009 she finished the doctoral stage, the title of her PhD thesis being: *The Development of Software Systems for Business Management*. She is the author of one book and she is coauthor of five books. Her interest domains related to computer science are: requirements engineering, business analytics, modeling languages, business rules approaches and software development methodologies.

Marinela MIRCEA, associate professor, PhD, currently working with Bucharest University of Economic Studies, Faculty of Cybernetics, Statistics and Informatics, Department of Economic Informatics and Cybernetics. Competence areas: information system, Business Intelligence. Research in the fields of in-formation system, Business Intelligence, classification techniques. Author of 6 books and more than 50 papers published in national and international journals.

DOI: 10.12948/issn14531305/18.1.2014.15