MODELLING OF THE BUSINESS RULES USING UML/OCL

Kęstutis Normantas*, Olegas Vasilecas

Information Systems Research Laboratory,
Vilnius Gediminas Technical University,
Sauletekio 11, LT20221, Vilnius, Lithuania

Received Februar 6, 2009 , accepted August 21, 2009

Abstract. Business rules are a crucial business category because they describe how enterprises are conducting business. Their value in developing software systems, which must be susceptible to fit rapidly changing business requirements, has made them attractive also within information system domain. As the formalization of business rules becomes a part of the commonly practiced systems analysis process, it is desirable for there to be a single, coherent representation for all kinds of business rules. The *Object Constraint Language* (OCL) as a part of the *Unified Modelling Language* (UML) provides the possibility to express business rules in formal and unambiguous manner. In this paper we investigate possibilities how to express different kinds of business rules with the UML/OCL, and discuss their advantages and disadvantages.

Keywords: Business rules, UML, OCL

Short title: Modelling business rules

** Corresponding author, email: kestutis.normantas@isl.vgtu.lt*

Introduction

Business success requires flexibility to fit rapidly changing business environment. Therefore, business supporting information systems (IS) should be able to adjust to those changes in time. Unfortunately, existing systems development approaches are not so flexible as business requires. It is difficult to readjust traditional business supporting IS to business changes because this process involves many troublesome tasks, e.g. revision of the system specification, redesigning, recoding etc. More often, in order to satisfy business needs in time, changes are made directly in code, without additional documentation. Later these changes grow over to a headache to those who are responsible for their management, and system specification looses its significant value. Moreover, Morgan in Ref. [1] notices that many research projects have shown that vast majority of software problems originates from specification error, not from the code as such. Therefore, demand on different approach to systems development arises.

According to the well-known Zachman Framework – see Ref. [2], in IS development process every system is represented by a single or several models depending on different aspects of modelling system and different points of view to the system. The business rules (BR) as their predecessors (business scope, motivation and strategy) and as their successors (business rules model and executable code) are concurrent with other aspects of enterprise system (data, functions, places, people and time). Therefore, it is very important that integration of BR model with other system's models as well as clear and unambiguous understating of BR and possibility to access and manage them should be guaranteed.

Regarding Barbara von Hale – see Ref. [3], an enterprise operates according to many different kinds of rules, such as legal mandates and rules it constructs for itself. The basic element of a BR is the language used to express it [4]. The most understandable form of BR is natural language, however, this form is ambiguous and informal to use BR in IS development process. As the formalization of business rules becomes a part of commonly practiced systems analysis process, it is desirable to be a single, coherent representation for all kinds of business rules.

The *Unified Modelling Language* (UML) – see Ref. [5] – has established itself as the leading object-oriented (OO) analysis and design methodology. UML is used for modelling systems within different abstraction levels [5]. The *Object Constraint Language* (OCL) has been developed as business modelling language within IBM Insurance division [6]. Recently, the second version of the OCL has been adopted as a part of the UML standard. The OCL supplements the UML methodology with possibility to specify system models in more detailed and unambiguous manner. According to Ref. [7], OCL is easy to use for an average business or system analyst, because its syntax is more relative to the natural language than traditional programming languages. Nonetheless, it is a formal language. Thus, the combination of the UML and the OCL is a formal way to express BR in IS development process.

Unfortunately, not much research is made on this topic. Erricson and Penker presented all possibilities to model a business with the UML – see Ref. [8] – and append a section with a description of expressing BR with the OCL. However, they discussed it considering few types of BR. Moreover, they did not educe advantages and disadvantages of the OCL as a language for expressing BR.

In Ref. [9], the main focus is set to the realization of BR of constraint type expressed with the OCL into database systems. However, the authors do not consider other types of BR though the OCL can be used as a query language as well as definition of derived values. More research is made in Ref. [10], where authors consider expressiveness of the OCL according to different types of BR. Apparently, the investigation has been made using the first version of the OCL. Therefore, many BR expressions with the OCL might be limited due to its first version of provided syntax.

The main aim of this research is to examine expressiveness power of the UML/OCL to model different types of BR. For the UML support possibility to represent systems in different abstraction levels, BR specified in the IS level model could be preserved until the implementation of specific level model. The existing tools provide opportunities for automated generation of those models. Therefore, BR specified in IS model, after elaboration and refinement, could be implemented in executive code.

For the examination of rules, widely adopted BR classification scheme, proposed by the GUIDE project – see Ref. [4], was selected. In this research we appeal to the extended list of action assertion BR types.

The paper is structured as follows. A brief overview of business rules is presented in Section 1. The objectives are discussed in Section 2. Section 3 deals with the tasks realization considering example business system is presented.

1 Business rules: overview

The interest in business rules has been shown for several decades. Many definitions of business rules concept have been presented as well as techniques to discover and express the rules, and a lot of classification schemes for the categorization of them have been proposed. Unfortunately, there is no industry standard definition for the term *business rules*.

Regarding Morgan, see Ref. [1], a business rule is a compact statement about some aspect of a business: it can be expressed in terms that can be directly related to the business, using simple, unambiguous language that is accessible to all interested parts i.e. a business owner, a business analyst, a technical architect and so on. In general, business rules describe how a company conducts its business.

The Business Rules Group (BRG) – see Ref. [4] – defined business rule in both business perspective and information system perspectives: from the business perspective, a business rule is guidance that there is an obligation concerning conduct, action, practice or procedure within a particular activity or sphere, and from the information system perspective, a business rule is a statement that defines or constrains some aspect of the business. Business rules may be captured by business experts, business owners or end users for keeping business works. While IT professionals, who are also in charge of BR capture, aim at making their applications usable in reality.

The BRG formalized an approach for identifying and articulating the rules which define the structure and control the operation of an enterprise. They had presented business rules classification scheme in the GUIDE project report – see Ref. [4]. According to the GUIDE, a statement of a business rule falls into one of the four categories.

1. Definitions of business terms – terms in glossaries or

entities, objects, classes depending on specification language. Typical examples could be presented by words: "Customer", "Party", "Employee" etc.
2. Facts relating terms to each other – natural language sentences or relationships, attributes and generalization structures in a graphical representation of the model. Typical examples could be presented by short sentences: "Customer address", "Party identification number", "Employee is absent" etc.
3. Constraints (Structural/Action Assertions) – constraints the structure or the behaviour of enterprise. Full length sentences express behaviour of constrains: "Customer may be one of the following status: gold, silver or bronze" etc.
4. Derivations – definitions of how knowledge in one form may be transformed into the other knowledge. For instance, sentence containing *know-how* elements: "For each item price ratio is calculated comparing with the previous month".

Application of BR in IS development process differs into three domains: business system, information system, and software system (SS). The last one is closely related with implementation of BR in application and is not in the scope of this paper.

From the business system perspective, business rules should be described in a form relative to business people. The most understandable form is natural language. However, this form is ambiguous to be used for specification of rules from IS perspective. For this purpose, BRG has provided the specification of Semantics of Business Vocabulary and Business Rules (SVBR), see Ref. [11]. SVBR is oriented to business people and is designed to be used for business purposes independently of information systems designs.

From the IS perspective, BR must be defined in unambiguous, clear and precise manner to be implemented within SS later. For this purpose many different BR modelling methods and techniques are provided. The comparison of these techniques is presented in Ref. [12], [13]. Substantially, selection of one of these techniques depends on IS development approach, adopted by IS development team. Object-oriented approach is widely adopted and differs from others by its capability to capture the structure (data) and the behaviour (functions) of business system, while others are able to capture only one of them. Recently, the UML is the de-facto standard for object-oriented analysis and design. The OCL as adopted part of the UML, supplements this methodology with possibility to specify system models in more detailed and unambiguous way. Therefore, in the third section, investigation on modelling different types of BR with the UML/OCL will be provided.

2 Object Constraint Language: objectives

2.1 Main characteristics

Object Constraint Language (OCL) is a formal language allowing the specification of constraints in context of the UML model. The OCL has been developed as a business modelling language and has its roots in the Syntropy method, see Ref. [6]. It has been adjusted to the UML to compliment its modelling because it is not expressive enough to provide all the relevant details of a specification. However, the OCL is side-effects free language, therefore it cannot change anything in the model: the state of the system will never change because of the evaluation of an OCL expression, even though an OCL expression can be used to *specify* a state change (e.g., in a post-condition), see Ref. [7]. Furthermore, it is not possible to write program logic or flow control in the OCL as well as express implementation details.

The main characteristics of the OCL are presented as follows according to Ref. [7].
1. Both query and constraint language – it is possible to write a query expression of a body of an operation as well as define constraint to some attribute's value or existence of some object.
2. Mathematical foundation – it is based on mathematical set theory and predicate logic and it has a formal mathematical semantics.
3. Strongly typed language – model elements used in the OCL expressions must conform with types, therefore the OCL expressions can be checked during modelling.
4. Declarative language – modeller can make decisions at a high level of abstraction without going into details how something should be calculated.
5. Object-Oriented analysis and design method.

The main purpose of the OCL could be formulated as follows according to Ref. [6]: i) specify invariants on classes, types or stereotypes in class model; ii) specify pre and post conditions of operations; iii) describe guards on transitions in state diagrams; iv) specify target for messages and actions; v) specify derivation rules for any expression over a UML model. Thus, considering the OCL characteristics and using the OCL 2.0 notation [7],[16],[17] different types of business rules will be discussed in the next sections.

2.2 Simplified document management system as an example.

Fig. 1 represents the class diagram of simplified document management system based on model requirements for electronic document and records management systems according to Ref. [14]. In presented system every document must have a type, a kind and some priority. Every document must be created by some author which may be a user or a group as well as some contact or a contact group. Document kind of receivable must be received by some address and kind of outgoing must be sending to some addressee. According to the processes with documents they may be in some state. Some documents may be involved in a file, which is a virtual catalogue for collecting documents. For system specification we used standard UML notation, which description and modelling guidelines could be found in Ref. [5].

The presented example is composed of different BR. For example document type may be one of the enumeration document types: a letter, an invoice, a memorandum, etc. To collect and check different BR classification scheme is required. To understand the nature of BR and the categories into which they fall classification scheme is presented in the GUIDE Business Rules Project report, see Ref. [4]. Though there are a lot of proposals how to describe and classify business rules in business rules research community, however GUIDE presents a formal approach for identifying and articulating the rules that define the structure and control the operation of an enterprise; moreover the classification proposed by this project involves others presented in Ref. [1],[3],[15]

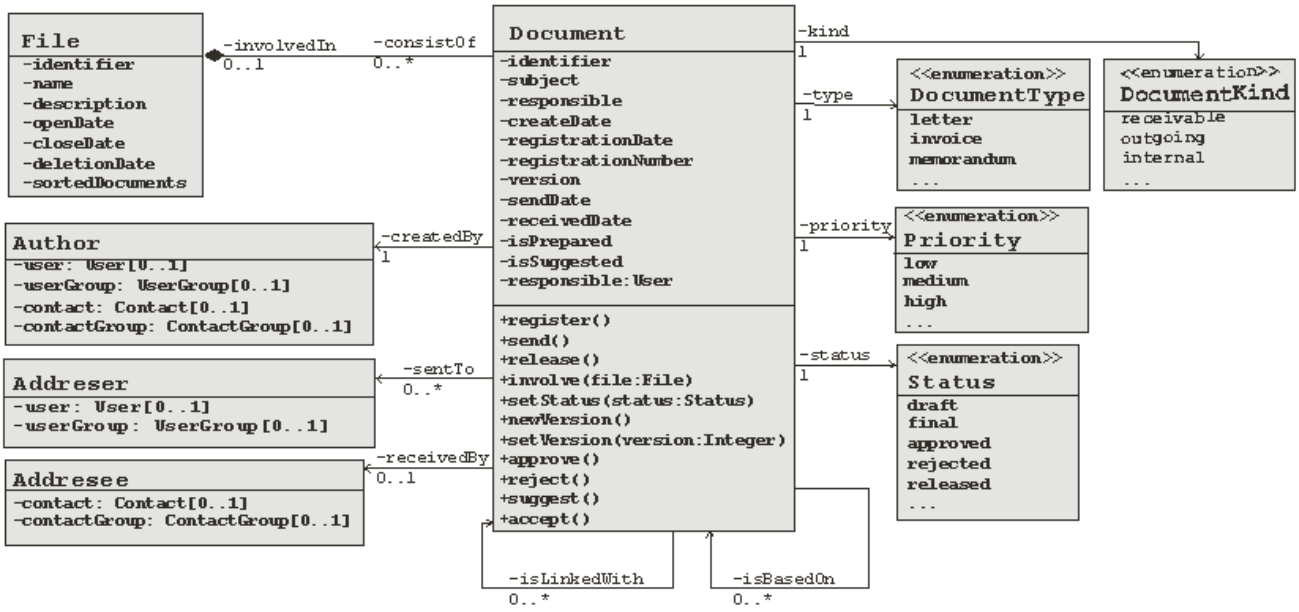

Fig. 1. Document management system: class diagram

3 Realisation of tasks using Object Constraint Language

In this section, considering the example discussed above, we will check extended list of action assertion type rules. This list involves the following types of BR: i) instance verifiers; ii) type verifiers; iii) sequence verifiers; iv) position verifiers; v) functional evaluators; vi) comparative evaluators; vii) calculators; viii) update controllers; and ix) timing controllers. Possibilities to model these types of rules using the UML/OCL will be discussed in the following subsections.

3.1 Instance verifiers

Each rule describes the effect of a correspondent (constraining object) upon an anchor (constrained object) [15]. Instance verifiers pertain to individual instances or occurrences of correspondent object classes [4]. Instance verifiers type includes the following subtypes: i) mandatory constraint (requiring occurrence of some object), ii) limited constraint (constraining number of population of object), iii) restricted (involving recursive structures), iv) pre-existing (requiring occurrence of some corresponded object class to exist before anchor object class and existence at the moment of rule check) and v) antecedent (requiring occurrence of correspondent object class to exist before anchor object class and it is not important or it still exists during check of rule).

Instance verifiers require possibility to manipulate a population of corresponded object class. The manipulation of collections of objects is very common in object-oriented systems. The OCL support different operations for collections which may be adopted by expressing this type of BR. All operations for collections are denoted in the OCL expressions using an arrow; the operation following the arrow is applied to the collection before the arrow [6].

Though most rules of this type could be modelled using standard UML notation, e.g. mandatory or limiting constraint as cardinality of association between classes or cardinality of object attribute, there is possibility to express them using the OCL. For example, the rule stating that *„Every document must have a responsible user assigned*" can be expressed as cardinality of attribute (or association) „responsible" or following the OCL expression as presented in the example 1.

```
// example 1
context: Document
inv: self. responsible->not Empty()
```

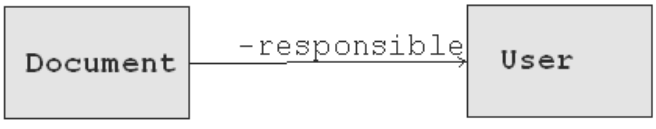

Fig. 2. Mandatory constraint rule: class diagram

The OCL expression is stated as invariant in the context of document. An invariant is a constraint that should be true for an object during its complete lifetime [7]. Thence this rule states while document exists it must have responsible user assigned to it. In the same manner it could be modelled limited constraint, e.g. *„A file must not have more that 1000 document involved in*" as presented in the example 2.

```
// example 2
context: File
inv:self.consists Of -> size()<1000
```

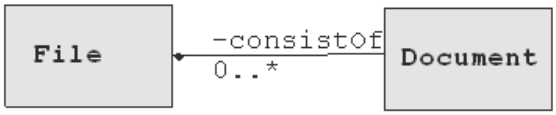

Fig. 3. Limited constraint rule: class diagram

The invariant above states that the size of the collection (number of all elements) of involved in file document objects must be less than or equal to 1000. Similar operation *count*() could be used to check the number of occurrences of some object in the related collection.

For restriction of recursive structures it should introduce recursive associations in class model. In the considering example recursive associations are modelled as document class attributes "isBasedOn" and "isLinkedWith". Suppose rule stating that "*A document cannot be based on itself*". In this case operation *reject*() could be involved as presented in the example 3.

// example 3
context:Document
inv:self.isBasedOn->reject(self)

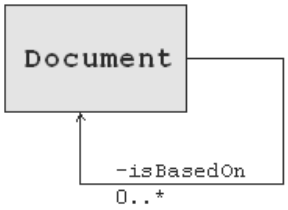

Fig. 4. Restricted constraint rule: class diagram

Operation *reject*() in this expression is used to state that in associated with document object collection „isBasedOn" cannot be document object itself. Similar operation *rejectAll*() could be used to except a collection of objects from related collection. On the contrary, operations *include*() and *includeAll*() could be used to preserve an object or object collection respectively to be in related collection.

Consider pre-existing constraint from involved example: „*Every outgoing document must be linked with some document*". The OCL expression for this example could be modelled in the following way as presented in the example 4.

// example 4
context Document
inv: self. kind=Document Kind::outgoing
implies self.isLinkedWith->notEmpty()

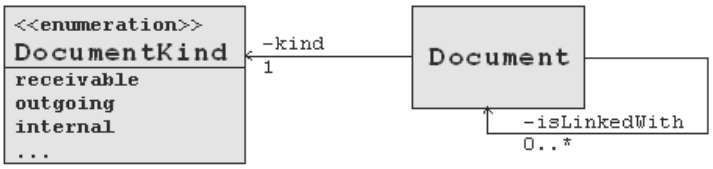

Fig. 5. Pre-existing constraint rule: class diagram

The expression above states that the fact that "*document is kind of outgoing*" implies the collection of objects "isLinkedWith" to be not empty. It follows thence that while the document is kind of "outgoing" it must be linked with the other document.

3.2 Type verifiers

Type verifiers control the creation of multiple instances in various object classes [4]. Type verifiers control occurrence of objects in object classes and may be one of four types: i) mutual (requiring that correspondent objects exist simultaneously), ii) mutually exclusive (requiring that no more than one correspondent object exist simultaneously), iii) mutually dependent (requiring that either one instance of every correspondent object class exists, or that no instances of any correspondent object class exist) and iv) mutually prohibited (requiring that at least one of the correspondent object classes has no instances).

For this type of rule comparison of collections should be involved. Combination of operations for collection with logical operator (OCL support *and*, *or*, *xor* operators) could be used to express most of this type of a rule. Consider

following the rule "*Document can be created only by one of following: a user, a users group, a contact or a contacts group*". This rule could be referred to mutually exclusive type of type verifier. Depending on example model, the document is created by some author which can be a user, a user group, a contact or a contact group. The following OCL expression mutually excludes candidate subjects to author as presented in the example 5.

The invariant above states that one of the collections must not be empty while others must. Other types of type verifiers could be expressed in the same manner.. Consider mutually prohibiting rule: "Internal document cannot be received by or send to any subject(s)". Corresponding the OCL expression could be modelled as presented in the example 6.

```
// example 5
context Document
inv:
 self.createdBy.user->notEmpty() and
(self.createdBy.userGroup -> isEmpty() and
 self.createdBy.contact -> isEmpty() and
 self.createdBy.contactGroup -> isEmpty() )
 or
 self.createdBy.userGroup -> notEmpty() and
(self.createdBy.user -> isEmpty() and
 self.createdBy.contact -> isEmpty() and
 self.createdBy.contactGroup -> isEmpty())
 or
 self.createdBy.contact -> notEmpty() and
(self.createdBy.user -> isEmpty() and
 self.createdBy.userGroup -> isEmpty() and
 self.createdBy.contactGroup -> isEmpty())
 or
 self.createdBy.contactGroup -> notEmpty() and
(self.createdBy.user -> isEmpty() and
 self.createdBy.userGroup -> isEmpty() and
 self.createdBy.contact -> isEmpty())
```

```
// example 6
context Document
inv: self.kind = DocumentKind::internal implies
self.sentTo -> isEmpty() and self.receivedBy -> isEmpty()
```

The invariant above involves implication operator and comparison of the collections. It states that document of kind "internal" implies associated collections of objects *sentTo* and *receivedBy* to be empty. In the same manner it could be possible to express more of this type BR.

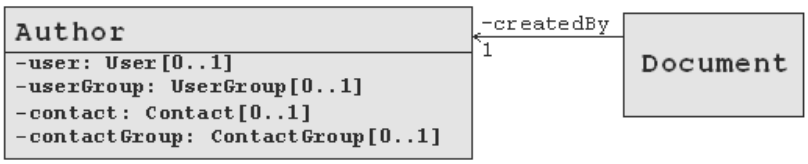

Fig. 6. Mutually exclusive constraint rule: class diagram

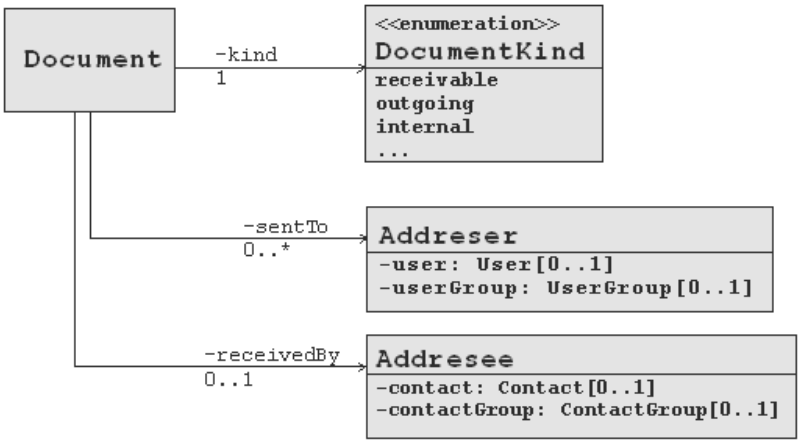

Fig. 7. Mutually prohibited constraint rule: class diagram

3.3 Sequence verifiers

According to Ref. [4], sequence verifiers control changes in object state. If object may be in multiple states then these rules determine the sequence in which instances of that object class may assume those states. Sequence verifier may be of one of the following types: i) initialling (requiring some state on object initialization); ii) forward (requiring transition to a higher state of object); iii) progressive and retrogressive (requiring transition to a next higher or lower state of object accordingly); iv) re-initializing (defining that when object moves to a lower state it should be moved to initial state first) and v) cyclical (defining that object can be moved to a lower state before it moves to highest state and vice versa).

This type of rules can not be fully defined in the OCL because the OCL is a declarative language, thus can not be used to define actions. The best way to express changes of object states in the UML is state machine diagrams [8]. The UML state machine diagrams describe the behaviour of a class over time of the states and transitions of a single object progressing through its lifetime. In the UML state machine diagrams the OCL expressions may be used in a number of ways – see Ref. [7], but commonly used are guards on states transitions and restrictions on states. The guard is condition on transition in a state machine diagram that must be met to change a state of object. Usually, restrictions on states are restrictions on values of links and attributes when an object is in a certain state. Using the UML state machines to define a sequence of object states and the OCL to express restrictions on model elements is possible to express most of the rules of the sequence verifiers type.

Referring to considering example, document lifecycle could be explained as state machine diagram as presented in Fig. 8.

The process could be described as follows. A new document saved in system is being prepared for release. After acceptance it may be suggested for release. After suggestion for release document may be approved or rejected. If a document is approved then it could be released, otherwise if a document is rejected the new version of document must be created referring to the cause of rejection. As sequence verifiers require, the sequence of object states is defined (by state machine diagram) in a model. The mentioned-above guard for transition is described as transition from the state *draft* to the state *final* which is an entry point to release process by the UML notation meaning. The restriction is enclosed in brackets and denotes that transition is possible only if document is prepared (by checking Boolean type attribute *isPrepared* value). Restriction on a state is described as *releaseCandidate* state denoting that document can be in this state only if it is suggested (by checking Boolean type attribute *isSuggested* value).

Considering the sequence verifier subtypes, the control of object states could be preserved by operations on any the OCL instance *oclInState*() or *oclInState* (str:StateName). Supposing the following rule: "*Every newly created internal document is draft until it is accepted that document is prepared for release*". Firstly, the rule denotes that initial state where document must be in is a draft. Corresponding the OCL expression is presented in the example 7.

```
// example 7
context Document::accept()
pre: self.oclInState(draft)
post: self.isPrepared=true and self.oclInState(final)
```

The OCL expression above restricts an operation *accept*() denoting that before execution of the operation document must be in a draft state and then it must be prepared in a final state. Thus control of changes of object states using the OCL could be applied to the other sequence verifiers.

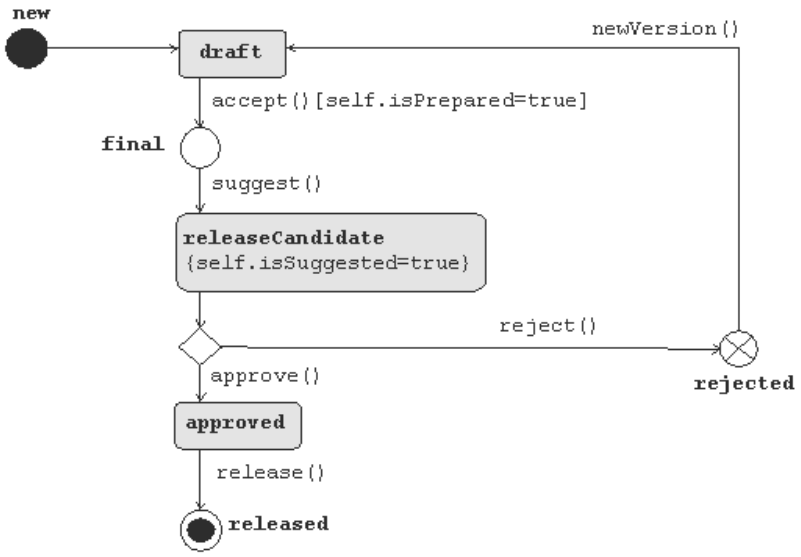

Fig. 8. States of document: state machines diagram

3.4 Position selectors

Position selectors pertain to a value, either in a value sequence or in a chronological sequence [4]. This type of rules is of two types: i) positioned – the lowest and the highest, and ii) chronological – the oldest and the newest. Supposing the rule: "*Document version must increase by one on set of version*". Respective OCL expression could be modelled as presented in the example 8.

```
// example 8
context Document::setVersion(version: Integer)
pre: --none
post: version=@pre + 1
```

The expression above denotes that after execution of operation *setVersion*() document attribute version must increase by one. It is expressed using keyword @pre which refers to an attribute value at the start of operation.

Another way of expressing position selectors could be usage of the OCL loop operations. Different iterator (loop operation) types may be chosen depending on business rule nature. For example, the rule mentioned above could be expressed using iterator *forAll*() as presented in the example 9.

```
// example 9
context Document
inv: self.allInstances->forAll(doc1,doc2 | doc1 <> doc2
implies doc1.version = doc2.version + 1)
```

Iterator *forAll()* returns true if all elements of collection used expression are true. In this case the operator *forAll()* is used to denote, that every different element of collection attribute's version value should be greater by one. Thus expression of positioned type of position selectors could be applied to the other business rules. However, the chronological sequence rules require additional constructs of model because the OCL does not support date or time data types. Therefore there should be involved some utility elements to support chronological sequence of elements of the collection.

3.5 Functional evaluators

Functional evaluators take care of sequence in which instances of object class are defined [4]. There are the following types of functional evaluators: i) unique (requiring unique sequence of values), ii) ascending and descending (requiring sorting of values in sequence), iii) non-renewable (requiring that any given value of the correspondent object, if used more than once, may be used only in strictly successive instances of the anchor object), iv) patterned (requiring that successive instances of the anchor object class be assigned in a specified sequence, or tests for that condition).

As it was discussed in the previous section control or modification on sequence of elements could be preserved using iterators. For example, unique values evaluator could be the following rule: "*Document registration number must be unique*". Respective OCL expressions are presented in example 10.

```
// example 10
context Document
inv: self.allInstances -> isUnique
(doc       | doc.registrationNumber )
context Document
inv: self.allInstances -> forAll
(doc1, doc2 | doc1 <> doc2 implies
doc1.registrationNumber <> doc1.registrationNumber)
```

The expressions above are identical because the first one uses operator *isUnique*() which is true if every document has a unique registration number and the second one is true if every pair of different document objects, registration numbers is not the same.

Sorting out the collection elements could be modelled using *sortedBy*() which simply sorts out collection's element in ascending mode. For example, the rule *„Document registration number must be in ascending order within a file*" could be expressed by defining attribute sorted Documents of collection type in context of file (example 11).

```
// example 11
context File
def: sortedDocuments: Sequence(Document) = self.
consistOf->sortedBy( registrationNumber )
```

The expression above collects sorted elements of documents collection into sequence. The sequence as well as other type of the OCL collections defines a sequence and ordering the elements of collection. Therefore, if it is required that collection elements are served out in some pattern operations *asSequence*(), *asBag*(), *asSet*(), *asOrderedSet*() could be applied, as it is described in Ref. [7]. Combining mentioned operators with iterators for collections could produce sequences of elements arranged in predefined way, thus, providing a possibility to express complex rules.

3.6 Comparative evaluators and calculators

Comparative evaluators describe comparisons between pairs of instances of object classes. The comparisons may be 'equal to', 'not-equal-to', 'greater-than', 'greater-than-or-equal-to', 'less-than', or 'less-than-or-equal-to', and so forth [4]. The OCL supports different logical operators for this purpose and some of them were discussed above. It should be noticed that the OCL is strongly typed language, therefore compared objects or values should be carefully chosen.

Calculators involve any standard computation, e.g. sum, subtract, max, min, med, etc. Calculations in the OCL are possible between two data types: integer and real. Calculators may be expressed using standard OCL calculating operations (summary, subtraction, division, modulus, etc.) but as well as in the case of comparative evaluators conformance of types is required.

3.7 Update controllers

Update controllers prescribe whether updates to a database may occur and may be of the following types: i) frozen (requiring existence of anchor object to make some operation to correspondent object); ii) frozen to users (the same as above, but restricted to specific list of users); iii) enabled (existence of every instance of anchor object enables operations on correspondent object), and iv) enabled with reversal (when anchor object is deleted the state of correspondent object is reversed).

The OCL is the constraint language therefore every expression in some way may confine an operation to a database. In object-oriented systems object attributes values are set or got by some defined operation. Therefore, if it is required to restrict some changes of value then pre-and-post conditions for operations could by applied. For example, restriction on change of value may be used for post condition for an operation denoting that value must be the same as at start of operation by introducing *@pre* keyword as it was shown in the previous examples. Certainly, if it is required to relate constraints on operations with some users, e.g. restrict execution of operation or change of value to specific users, additional constraint should be introduced into the model to check whether operation is executed by the related user or not.

Consider the following rule "*Only assistant can register internal documents*" requiring an assistant role to execute the operation *register*(). One of possible expressions of this rule may be introduction of a new operation into document class. The operation, e.g. *checkRole*(event: String, role: Role) could be of Boolean return type checking whether some event is made up by some role. Additionally, it should be mentioned that in Ref. [18] much attention is made on modelling access control with the UML/OCL, where suggestions on modelling such type of rules are proposed.

3.8 Timing controllers

Timing controllers prescribe tests for the length of time that instances of correspondent objects have (or have not) existed [4]. The OCL does not support the possibility to express time-based constraints which could confine objects states according to time. Therefore, additional elements (such as time utility) could be introduced to model or extension to the OCL could be provided as suggested in Ref. [19],[20].

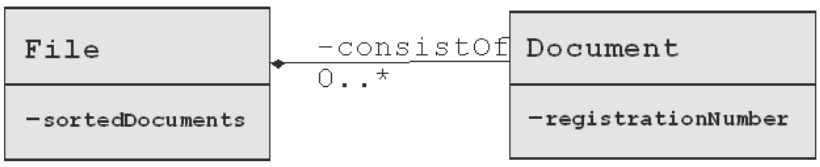

Fig. 9. Functional evaluator rule: class diagram

Conclusions

In this paper, research on possibilities to express different types of action assertion business rules with the UML/OCL was made. It was established that the combination of the UML and the OCL is expressible enough to capture most of action assertion type rules, both structural and behavioural.

The OCL supports operations on collections of the UML model elements, therefore restrictions on population of object instances could be expressed as it is required by instance verifiers.

The OCL provides calculative and logical operators, which may be used to express functional evaluators. The combination of operators and operations for collections may be used to express type verifiers.

The OCL has pre and post conditions on operations of classes, therefore restrictions on data manipulation could be applied in many different ways.

The OCL provides possibility to query the UML model; therefore, the OCL may be used to express derived values. Supported collection data types enable to predefine required structure of the collection; in addition, iterators enable to make a more complex structure of the collection.

The OCL supports only limited set of data types, therefore there is no possibility to express BR requiring comparison of variables with constants (e.g. it is not possible to compare dates).

The OCL does not provide a possibility to express time-based constraints which could constraint objects states according to time, therefore suspending model with additional constructs is required, or extension for the OCL should be provided.

Acknowledgements

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.

References

[1] Morgan T. Business Rules and Information Systems: Aligning IT with Business Goals. Addison Wesley. 2002, pp. 384

[2] Hay D.C., Requirements Analysis: From Business Views to Architecture. Prentice Hall, 2002

[3] Von Halle B. Business Rules Applied: Building Better Systems Using the Business Rules Approach. John Wiley and Sons. 2002, pp. 495

[4] Hay D.C. et al. Defining Business Rules - What Are They Really?//GUIDE project. 2001, available from: [http://www.businessrulesgroup.org/first_paper/br01c0.html], retrieved 2009 01 10

[5] Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language User Guide, Second Edition. Addison Wesley. 2005.

[6] OMG. OCL 2.0 specification. 2005. Available from: [http://www.omg.org/docs/ptc/05-06-06.pdf], retrieved 2009 01 15

[7] Warmer J., Kleppe A. Object Constraint Language, The: Getting Your Models Ready for MDA, Second Edition. Addison Wesley. 2003, pp. 347

[8] Eriksson H.E., Penker M., Business Modelling with UML: Business Patents at Work. John Wiley and Sons. 2000.

[9] Zimbrao G. et al. Enforcement of Business Rules in Relational Databases Using Constraints//UMFG Database Group. 2007, available from: [http://www.lbd.dcc.ufmg.br:8080/sbbd2003/artigos/paper010.pdf], retrieved 2008 12 20

[10] Sosunovas S., Vasilecas O. Verslo taisyklių modeliavimas OCL kalba (in Lithuanian) // *Lietuvos matematikos rinkinys.* Vilnius. 2004, pp. 369-376

[11] OMG. Semantics of Business Vocabulary and Business Rules, v 1.0. 2008, available from: [http://www.omg.org/docs/formal/08-01-02.pdf], retrieved 2008 10 15

[12] Lebedys E., Vasilecas O. Verslo taisyklių modeliavimo kalbų analizė (in Lithuanian) // "Informacinės Technologijos'2004". Technologija. 2004, pp. 487-494

[13] Herbst H. et al. The Specification of Business Rules: A Comparison of Selected Methodologies. Methods and Associated Tools for the Information System Life Cycle, Amsterdam et al.: Elsevier 1994, pp. 29-46.

[14] MoReq2, Model Requirements Specification for the Management of Electronic Records, available from: [http://www.moreq2.eu/], retrieved 2008 11 21

[15] Ross G.R., Principles of the Business Rule Approach. Addison Wesley. 2003, pp. 352

[16] Warmer J., Kleppe, A., Cook, S., Informal Formality? The Object Constraint Language and its application in the metamodel. in Proc. International Conference on the Unified Modelling Language (UML), Springer-Verlag. 1999.

[17] Warmer J., Kleppe, A., OCL : The Constraint Language of the UML. Journal of Object-Oriented Programming, 12(1):10–13. 1999.

[18] Ahn G.J., Shin M.E., Role-Based authorization constraints specification using Object Constraint Language// Enabling Technologies: Infrastructure for Collaborative Enterprises, Proceedings tenth IEEE international Workshops. 2001, pp. 157-162

[19] Flake S. Temporal OCL extensions for specification of real-time constraints. In Workshop Specification and Validation of (SVERTS'03). 2003.

[20] Hamie A., Mitchel R., Howse J. Time-Based Constraints in the Object Constraint Language. School of Computing and Mathematical Sciences, University of Brighton. 2006, available from : [http://www.biro.brighton.ac.uk/biro], retrieved 2008 12 14