UDC:007
Original scientific paper

FORMAL MODELLING OF BUSINESS RULES: WHAT KIND OF TOOL TO USE?

Sandra Lovrenčić, Kornelije Rabuzin, Ruben Picek
Faculty of Organization and Informatics, Varaždin, Croatia
{sandra.lovrencic, kornelije.rabuzin, ruben.picek }@foi.hr

Abstract: *Business rules are today essential parts of a business system model. But presently, there are still various approaches to, definitions and classifications of this concept. Similarly, there are also different approaches in business rules formalization and implementation. This paper investigates formalization using formal language in association with easy domain modelling. Two of the tools that enable such approach are described and compared according to several factors. They represent ontology modelling and UML, nowadays widely used standard for object-oriented modelling. A simple example is also presented.*

Keywords: *business rules formalization, ontology modelling, UML.*

1. INTRODUCTION

For leading a successful business it is critical to constantly increase a competitiveness of a company and all means that contribute to that goal will be used. Therefore, hardly any business system operates today without knowing its mission, goals and, of course, business rules. They are nowadays a very important component of a business system of any kind. Although a variety of methods to write down and formalize business rules exist, their definition is also very divers and still not completely agreed upon.

Formalization and implementation of business rules comprises several well accepted approaches or some special models developed from various book authors. Each of those approaches gives its perspective on business rules and supplements other parts of business system, for example, business processes. The authors wanted to use more formal business rules representations that allow usage of some formal language, but still have easy to understand domain model. According to that, this paper compares two such approaches: ontology modelling and UML, using tools that have possibility to express rules with formal logic, including a simple example in a representative tool of each approach.

2. BUSINESS RULES BASICS

Business rules influence considerably the way the business is managed. For that reason they are in the focus when developing formal model of a business system. But not only business system: rules are everywhere. If a person always wakes up at 7 o'clock, it is a rule; if a student must pass a written exam to approach an oral exam, that is also a rule. Rules can be found in many forms, from simple written sentences to integrated business support applications. And although they exist in all areas of our life, there is still no general agreement about their definition and ways of their classification. Rules range from simple to very complex and even fuzzy, that are based on fuzzy logic [8].

The Business Rules Group (BRG) in their final report about defining business rules [23] declares that "a business rule is a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behaviour of the business". Well known expert R.G. Ross says that it is "a directive intended to influence or guide business behaviour" [18] or "a constraint or a test exercised for the purpose of maintaining the integrity (i.e., correctness) of data" [19]. And for B. von Halle they are "the set of conditions that govern a business event so that it occurs in a way that is acceptable to the business" [22].

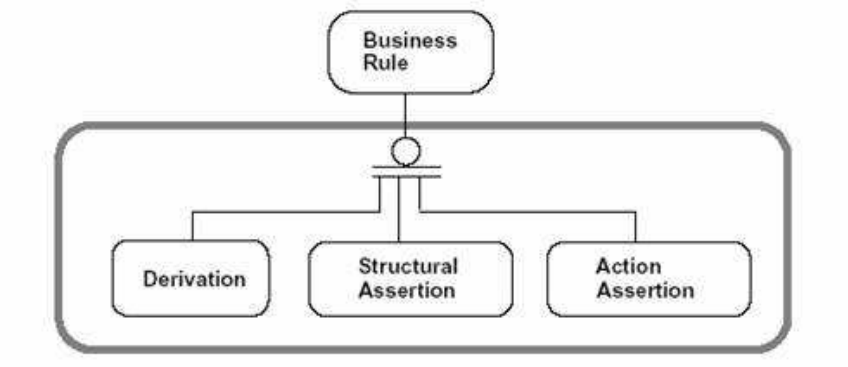
According to many different definitions of this term, there are also many different classifications available, as already mentioned. In [22] business rules are classified into following groups:

- Term – a phrase with agreed upon definition;
- Fact – a statement that connects terms;
- Mandatory constraint – an unconditional circumstance that must be true/false;
- Guideline – a warning about a circumstance that should be true/false;
- Action enabler – test conditions and initiates events, i.e., activities;
- Computation – provides an algorithm for computing a value of a term;
- Inference – test conditions and establishes the truth of a new fact.

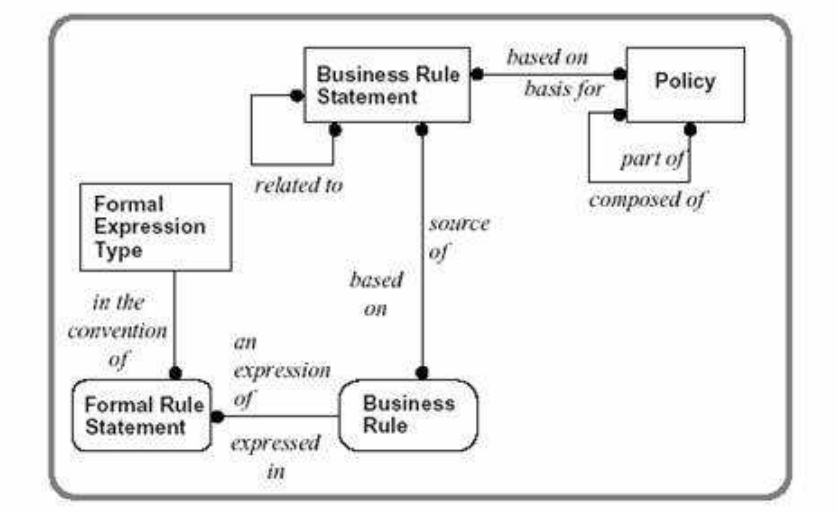
For the purpose of presenting variety of authors' views, here is also presented a general classification of business rules into three main groups that Ross gave in [18]:

- Rejectors – reject any event that would cause violation to occur;
- Producers – never reject update events, but accept them and derive something for the user (can be divided into Computation and Derivation rules);
- Projectors – accept update events and take other actions (produce some event; can be divided into Enabler, Copier or Executive rules).

Abd, as a third view, the BRG states that each business rules must be either (Picture 1)[23]:



Picture 1: Business rule types according to BRG



Picture 2: The origin of business rules

A structural assertion is a defined concept or a statement of a fact that expresses some aspect of the structure of a domain, an action assertion represents a constraint or condition that limits or controls various actions and a derivation is a statement of knowledge derived from other knowledge.
The origin of business rules is described by the BRG as a conceptual model, in the form of entity/relationship (E/R) diagram, presented in Picture 2 [23]. This model defines what business rules are and what kinds of them exist. More information about Picture 2 can be found in [23].

3. BUSINESS RULES FORMALIZATION

Formalization and implementation of business rules is today an object of many explorations. It is critical to constantly increase a competitiveness of a company and all means that contribute to that goal will be used. So, it is not only important to know business rules, but it is also important to structure and connect them well and maybe even make a business support system in which they are incorporated.
Business rules were at first and still are recorded in forms of simple sentences. Then, various models for their representation were developed. Now there are many tools that can be more or less used for business rules formal description and a number of integrated business support applications. Rules can be, for example, incorporated in ERA model [22], ontology [6], UML [1], active databases [15] or F-logic [6]. If we want to extend them to a business support application, we could use one of tools for Knowledge-Based System (KBS) development. But, no matter what method of formalization is chosen, rules should be written according to following principles [23]:

- Explicit expression – graphically or with a formal language (ERA model, F-logic);
- Coherent representation – unique formal representation of all kinds of rules in a company;
- Evolutionary extension – addition of constraints to existing representation, e.g., to diagrams;
- Declarative nature – by nature, rules are declarative, not procedural.

It can bee seen from above mentioned that there has to be some kind of formal language that enables proper formalization of business rules and that enables rule verifying. The Business Rules Manifesto [17] also states: "Formal logic, such as predicate logic, are fundamental to well-formed expression of rules in business terms, as well as to the technologies that implement business rules". Since most rules actually represent constraints on various business concepts and processes, a tool for defining rules has to enable some form of constraint making. Of course, if someone is developing a KBS, he would mostly use typical if-then expressions.

Although one can use only formal language for rules formalization, for example Flora-2 [6], authors wanted to explore approaches that enable easy domain modelling, which can be used as a starting point for different application systems development, but also enable usage of some kind of formal language for rules description, that is, constraint making. According to that goal, in this paper two of approaches are compared: ontology modelling and UML. For the realization of a simple example two representative tools were used and then compared: Protégé (http://protege.stanford.edu) and ArgoUML (http://argouml.tigris.org). They were chosen because they, as most important, enable constraint making and are similar in the way that both are free, Java based, open source tools.

3.1. ONTOLOGY MODELLING

Ontology is, as a much cited definition says [12], "a formal explicit specification of a shared conceptualisation for a domain of interest", meaning that it "has to be specified in a language that comes with formal semantics". Business domain ontology gives a good insight into all concepts of a specific business domain and its hierarchical structure. It shows connections among domain objects and can be one of starting points for defining business rules, along with business processes.

For a formal development of specific ontology there are a number of languages and language definitions, for example: Description Logic (DL) [3], Frame logic (F-logic) [2], Resource Description Framework (RDF) and RDF Schema [13]. Also, there are various tools for building ontologies: OntoStudio (http://www.ontoprise.de/, formerly OntoEdit),), OilEd (http://oiled.man.ac.uk/), Protégé, Ontolingua (http://www.ksl.stanford.edu/software/ontolingua/), OWL - Web Ontology Language (http:// www.w3.org/2004/OWL/) and others. The RDF and OWL imply that ontologies are now domesticated on the Web also. Comparisons of a certain number of tools and languages can be found in [7] and [21] and in the reports from OntoWeb project [10] and its OntoWeb Portal (http://www.ontoweb.org/).

Ontology modelling in general, and in Protégé also, includes [14]:

- Determining the domain and the scope of the ontology;
- Considering a reuse of existing ontologies – with possible usage of knowledge-representation systems;
- Enumerating important terms in the ontology – depends on the first step;
- Defining classes in the ontology – describe concepts which are the focus of ontology;
- Arranging classes in a subclass-superclass hierarchy – dividing more specific concepts from general ones;
- Defining slots – properties of classes;
- Defining facets on slots – various features and attributes with allowed values;

- Filling in values for slots and instances – concrete examples of concepts.

Protégé, as an ontology development tool, exists already 18 years and today is widely used. It is made in Java, it's free and open source. It was originally developed to support knowledge acquisition for specialized medical expert systems, but today it has many plug-ins for various features, from making constraints on attribute values to exporting ontology in different formats (CLIPS, OWL, RDF, RDF Schema and HTML are delivered in standard distribution) and importing concepts from other tools into Protégé. It can be used as a basis for KBS development. Comparisons of Protégé and other ontology modelling tools can be found in [10] and [21]. It is already suggested that it can be used for business rules formalization [6].

For constraint development in Protégé is used PAL – Protégé Axiom Language [11,25], created as a plug-in that is delivered together with the tool. It enables writing of logical constraints and queries about frames of a knowledge base (classes, instances, slots and facets). PAL works with model-checking, making "closed-world" assumptions, so it can be used only for constraints on existing knowledge and not for asserting new one. It has constraint-checking engine that can be run against the knowledge base to find frames that violate constraints.

PAL plug-in provides [25]:

- A language to express logical constraints and queries about frames in a knowledge base;
- A set of special purpose frames to model constraints and queries as part of a knowledge base;
- A structured PAL expression editor that provides context-sensitive help to write PAL sentences;
- A constraint checking engine, which can be invoked with the Pal Constraints Tab or programmatically;
- A querying engine which can be invoked with the pal Queries Tab or programmatically.

A constraint in PAL includes a set of variable range definitions and a logical statement that must hold on those variables. The syntax is a variant of Knowledge Interchange Format (KIF), but, although it supports KIF connectives, it does not support all constants and predicates. A constraint statement is a sequence of sentences that use a number of predefined predicates and functions, ontology and other predicates linked with logical connectives (`<=>`, **`=>`**(if), `and`, `or`) in a prefix notation. `Not` is, of course, also used. Every variable in the constraint has to be quantified with universal (`forall`) or existential quantifier (`exists`). Examples of PAL constraints can be seen in example section.

PAL variable range definition must be in the form:
`(defrange var type [additional information])`, where is

- `defrange` – reserved word for denoting a variable range definition
- `var` – the name of the variable, begins with `?` for local and with `%` for global variables
- `type` – the type of the variable, can be `:SET`, `:FRAME`, `:SYMBOL`, `:STRING`, `:INTEGER`, `:NUMBER`

- `[additional information]` – depends on the type of the variable, for example `:FRAME [classname][slotname]`

A syntax of predicate and function assertions is the same:
`(pred-name/func-name symbol-1 symbol-2 symbol-3 ...)`, where is

- `pred-name/func-name` – the name of the predicate or function
- `symbol-n` – slot value, class or instance

Connectives and not are used as follows:
`(not assertion)`
`(connective-name assertion assertion)`, where is

- `assertion` – predicate assertion described above
- `connective-name - <=>, =>, and, or`

Quantifier syntax is in the form:
`(forall ?name full-assertion)`
`(exists ?name full-assertion)`, where is

- `?name` – the name of the variable being quantified
- `full-assertion` – assertion with connectives and not

3.2. UML

UML (Unified Modelling Language) is a standardised notation for object oriented system modelling and a basic platform for good development of business application systems. The definition of UML in the UML specification for version 1.5 [24] (UML 2.0 is still being upgraded by OMG and convenience separate documents are available on [26]) is: "UML is a graphical language for visualizing, specifying, constructing and documenting the artefacts of a software-intensive system", meaning:

- Visualizing – it is a graphical, visual language that enables building of a certain number of diagrams according to a specific notation - diagrams represent different views on the system and alleviate their comprehension during the model development;
- Specifying – enables forming of precise, unambiguous and complete models;
- Constructing – an application system that will be implemented is constructed;
- Documenting – user requests, architecture, the project, program code.

The focus of UML is a standard modelling language and not a standard development process. It is independent of the development process [9], because different domains and organizations need different development processes. It can be expanded without redefining the UML core. The important characteristic of UML is that it is designed as a general-purpose language and represents a set of best engineering practices. There are many tools that enable UML modelling, importing to UML or importing from it. Along with ArgoUML, there is its commercial version Poseidon for UML (http://www.gentleware.com/index.php), that has more possibilities. UML CASE tools are also RationalRose and Rational XDE of IBM (http://www-306.ibm.com/software/rational/),

Borland Together (http://www.borland.com), Visual UML (http://www.visualuml.com/) and others. The usage of UML for defining business rules is explored in [1] and [6].

UML notation consists of robust concepts that are used through all development phases of an application system. The parts of the notation are:

- Building blocks – model elements, relations and diagrams;
- Well-formedness rules – for blocks;
- General mechanisms – specification of ornaments and expansion mechanisms.

As already mentioned, different views on the system enable a number of diagrams (thirteen in UML 2.0). They are classified into three groups [1]:

- Behaviour diagrams – show behavioural features of a system or business process: activity, state machine, use case and all interaction diagrams;
- Interaction diagrams – a subset of first group that is concentrated at object interactions: communication, interaction overview, sequence and timing diagrams;
- Structure diagrams – show static elements that are irrespective of time: class, composite structure, component, deployment, object and package diagrams.

Basic object oriented concepts used in UML for static view (class diagram, which authors used) are, according to [1]:

- Class – abstraction of an object;
- Object – a construct that mirrors a concept in the real world;
- Attribute – property of an object;
- Method (and/or Operation) – a function or a procedure that access and modify attributes of an object or do other needed actions.

Objects in the diagram are connected with various kinds of associations that are explained in [1].

As Protégé, ArgoUML is also made in Java, it's free and open source and enables exporting to XMI. Current version of ArgoUML doesn't support all diagrams listed before. Since authors wanted to show formal means to implement business rules solely, without other parts of a business domain, only class diagram is used. UML's class diagram can be easily compared with hierarchical ontology made in Protégé.

In the same way Protégé uses PAL for expressing constraints, UML tools use OCL – Object Constraint Language, developed as a business modelling language within IBM Insurance division as a language for business modelling within IBM. It is not a stand-alone language, but an integral part of the UML and an OCL expression needs to be placed within the context of a UML model. Description of the OCL given below is based on OCL [24] specification and OCL2 specification convenience document [26].

OCL is a formal language to express side-effect-free constraints and has following characteristic:

- Pure specification language – without side effect – only returns value and cannot change anything in the system model;
- Modelling language – programs cannot be written in it;
- Typed language – each expression has type that cannot be compared with other type;
- Specification language – implementation issues cannot be expressed.

It is stated that it can be used:

- As a query language;
- To specify invariants on classes and types in the class model;
- To specify type invariant for Stereotypes;
- To describe pre- and post conditions on Operations and Methods;
- To describe Guards;
- To specify target (sets) for messages and actions;
- To specify constraints on operations;
- To specify derivation rules for attributes for any expression over a UML model.

The OCL expression is written in the context of an instance of a certain type, where reserved word `self` refers to that instance, and context declaration begins with a `context` keyword. However, if constraint is properly shown in the diagram, explicit context declaration in the constraint text is not needed. There are a number of predefined OCL types. After context, other stereotypes are used depending on the purpose, but constraints are mainly expressed as:

- Invariant (`inv`) - for invariants on classes and types in the class model, where all instances of that type must be true at any time;
- Precondition (`pre`) and postcondition (`post`) for pre- and post conditions on operations and methods.

Expressions usually apply to properties, which can be attributes, association ends, operations or methods and are written after a dot. The operators (`*`,`-`,…) are used in infix notation. OCL constraint examples are in the example section.

An OCL expression is written as follows:

```
context Type stereotype [constraintName]
self.property = expression
```

, where is

- `context` – reserved word, as explained above
- `Type` – context type
- `stereotype` – for example: `inv`, `pre`, `post`
- `[constraintName]` - optional, name of the constraint
- `self` – reserved word, as explained above
- `property` – instance of the `Type` type
- `expression` – the constraint itself

4. EXAMPLE

As an example for business rules formalization, here is used a simple set of eleven rules that are part of the case study in [22]. Case study is called Virtual Child International and represents a virtual play park for children at home. Selected rules are referring to main business objects in park and they, along with objects, can be found on page 226 in [22]. The whole example is presented at Web site [27].

Main objects in the example are:

- Sponsor (Guardian) – usually parent of a member, has attributes: Sponsor known, Credit, Special Deal and Credit Rating + inherited from superclasses;
- Member – child that uses the park, has attribute Age Appropriate + inherited;
- Park Keeper (Park Ranger) – a person that takes care of the park, has no attributes except inherited.

In our tools those object will represent classes. Several other classes are created, according to rules:

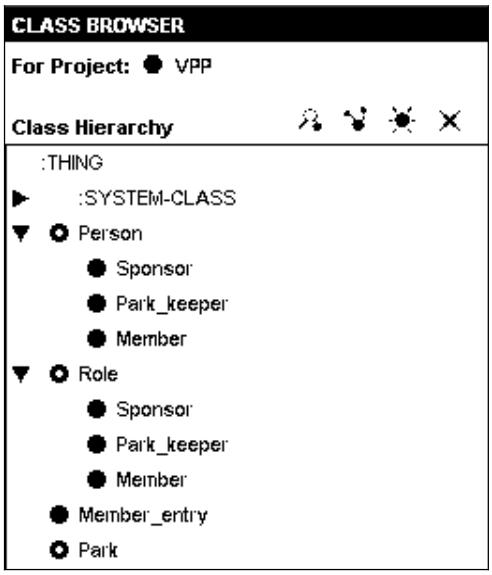
- Person – a superclass of three main classes, has attributes: Person Key, Surname, Name, BirthDate, Age;
- Role – a superclass of three main classes, has attributes: Role Key, Role Name;
- Member Entry – for information about enrolled members, has attributes: Start Date, End Date, Membership Status, Maximal Time, Person Key, Park Key, Role Key;
- Park – information about park, has attributes Park Key, Park Name.

It must be noted that this is only a small part of the model and that not all attributes are created, but only those needed to create used eleven rules. As an example of PAL and OCL rule expressions here will be shown their equivalent for those two rules from the set:

1. If member age < 16 and member age > 6 then member.age is appropriate.
2. If a guardian's credit rating code is not = "A" and the guardian's special deal flag = "yes" then the guardian has good credit.

4.1. PROTEGE

Since Protégé is ontology development tool, all classes and their attributes in the example are created as a small ontology. All inherited attributes are in Protégé automatically written within subclass slot window. In Picture 3 can be seen hierarchical structure of example classes.

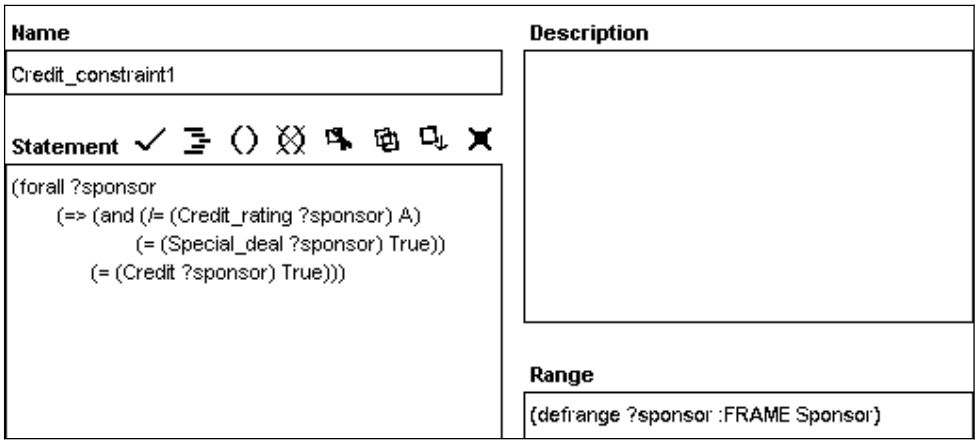


Picture 3: Class hierarchy in Protégé

Since Protégé has possibility to create restrictions on attributes in forms of allowed values or minimum and maximum values, first rule can be defined just by setting minimum to 6 and maximum to 16. When PAL is used, rule would be expressed, using attribute Age_appropriate, as:

```
(defrange ?Member :FRAME Member)
(forall ?Member
        (=> (or (<(Age ?Member) 6)
                (>(Age ?Member) 16))
            (=(Age_appropriate ?Member) True)))
```

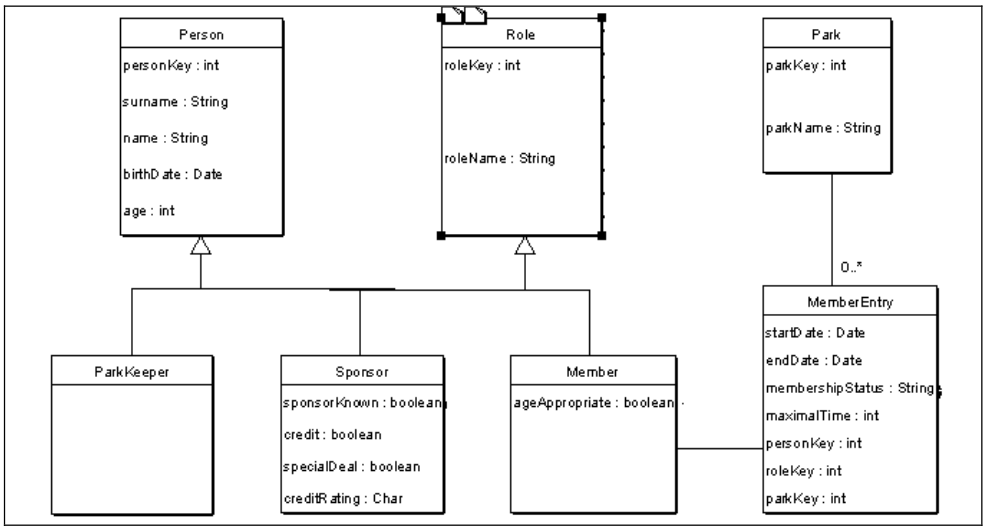
The second rule can be seen in Picture 4.



Picture 4: Business rule expressed in PAL

4.2 ARGOUML

Classes in ArgoUML are represented graphically, along with hierarchical connections and associations between them. Each object that represents a class normally consists of three parts: class name, class attributes and class operations. The last part is omitted and hidden from all classes, because authors restricted themselves strictly on business rules and also operations were unnecessary for this small example (although they can be used in OCL). Attributes that are inherited aren't automatically denoted in a subclass. An ArgoUML class diagram can be seen in Picture 5.

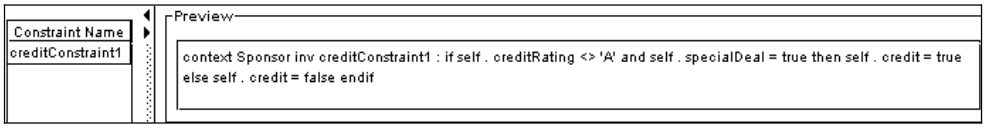


Picture 5: Class diagram in ArgoUML

In a first rule, created in OCL, operation `oclAsType()` is used for accessing (inheriting) the property `age` from the superclass `Person` to subclass `Member`:

```
context Member inv ageConstraint :
if self.oclAsType (Person).age > 6 and self.oclAsType
(Person).age < 16 then
   self.ageAppropriate = true
else
   self.ageAppropriate = false
endif
```

The second rule can be seen in Picture 6.



Picture 6: Business rule expressed in OCL

5. COMPARISON

Searching the literature authors couldn't find an explicit comparison between usage of UML and ontology modelling for business rules formalization. The comparisons of such

tools are based on their syntax differences and on means of representation of certain concepts [5] (In presented example, slots in Protégé are attributes in ArgoUML). Therefore, the authors selected ten factors that, according to business rule literature, primarily according to Business Rules Manifesto[17], can have impact on selection of means of their modelling. Comparison is made between two specific tools, not between methods, but can be taken as a general recommendation. Factors are graded by authors using 1 to 5 Likert type scale, based on satisfaction (1 – tool doesn't satisfy the factor, 5 – tool completely satisfies the factor). Main guidance in grading was authors' experience and statements regarding individual factors from literature (Table 1).

Table 1: Factors for business rules tool selection

Factor	Grade	
	Protégé	**ArgoUML**
Declarative knowledge is divided from procedural	5	3
Processes can be included	2	5
Supports concepts, attributes and constraints	5	5
Rules are explicit	3	3
Rules can be expressed by formal language	5	5
Formal language is easy to learn and use	4	2
Rule consistency can be checked	5	4
Queries are enabled	5	5
Can accept new rules without major changes	4	3
Support various saving formats	5	2

Declarative knowledge is divided from procedural – Protégé, as ontology tool, whose purpose is to model structure and constraints of a domain, represents mainly declarative knowledge by its definition. ArgoUML aims, as proper UML representative, at whole business model, and procedural knowledge is its natural part, in greatly interweaved with declarative knowledge.

Processes can be included – Processes can be subsumed using PAL, but using Java would be better option, although this isn't the purpose of Protégé. ArgoUML, as stated above, naturally includes processes.

Supports concepts, attributes and constraints – Both tools support them, using different names. They are important for proper expression of a business rule.

Rules are explicit – Rules can be explicitly stated in both tools, using formal language, but if it is possible, other simpler options are used (attribute value restrictions, for example, as stated earlier).

Rules can be expressed by formal language – In both tools, as represented in this paper.

Formal language is easy to learn and use – In authors' experience, PAL is relatively easy to learn and use, but that is not the case with OCL. Various authors also underline that fact [1,16,20], while others vote for opposite [4]. Of course, it depends on prior knowledge, but average users of UML modelling are usually not comfortable with formal language.

Rule consistency can be checked – PAL has its own constraint checking engine. Constraints incorporated in other ways don't allow instances that violate them. OCL is stated to be consistent too [4,20].

Queries are enabled – Both formal languages enable them, as can be seen from their documentation [11,24,25,26].

Can accept new rules without major changes – Both tools support that factor if it is a new constraint in PAL/OCL attached to existing class or attribute. According to authors' experience, new concepts cause fewer changes in Protégé.

Support various saving formats – Included in Protégé are (besides PPRJ) CLIPS, OWL, RDF and RDF Schema. Through plug-ins it supports also UML, XML, XML Schema, RuleML, XMI, html, pdf and simple text. ArgoUML supports (besides UML) only XMI. Since sharing and reuse of knowledge is nowadays more and more emphasized, support to more formats is very desirable option.

This comparison gives a general overview of characteristics important for business rules formalization. Average grade of 4,3 for Protégé and 3,7 for ArgoUML gives advantage to the first tool. But, if, for example, incorporation of processes is of utmost importance, it would be wrong. Therefore, each factor should be weighted, depending on its importance for a specific domain modelling. In that case, more accurate results would be obtained.

6. CONCLUSION

As definition of business rules become obligatory factor in successful organization management, more and more methodologies and tools for their formalization emerge. It is not only important to define the rules, but to formalize them well and incorporate them in various business support systems. It is also important to be able to transform them into various forms, according to current needs.

A representation of rules in a formal language with the possibility of a constraint checking ensures a good rule base. On the other hand, most business people aren't familiar with formal logic. They need visual understandable representations that are easy to follow. Therefore, to satisfy both sides, many tools combine those approaches.

Authors compared two such approaches: ontology modelling and UML, with the concrete example in Protégé and ArgoUML. From their description and above example, it can be seen that they have some joint points, although they start from different views at an organization. Along with their main goal, both have class (if ArgoUML's class diagram is used) as their starting entity, which is a good basis for rule formalization. Slots and attributes can be compared, but Protégé additionally supports concrete instances, which actually isn't necessary for rule development. On the other hand, ArgoUML has methods and operations that can be used in constraint definitions. Both tools have formal languages for rule definition with a substantial number of built-in predicates and operations to enable complex business rules implementation.

It is hard to determine which approach is better. The choice depends on various factors represented in Table 1. It also has to be known how deep formalization should be. Factors represented can be a general guidance to selection of appropriate method and tool to formally model business rules. Extended research could attach weight to each factor, since not all factors could be of same importance for the selection – generally and individually.

More formalism, of course, gives us a better control over the rules and enables constant verifying of constraints and inconsistencies. The further research in this direction could include other similar approaches in a more extent comparison with formal parameters. More connection among various tools through transformation into different formats could be made. Analysis of applicability of a certain approach on different domains could also be a subject of an exploration. With so many directions and new formalization approaches, this area still expands and gives many possibilities for research and solutions that would enhance business rules representation.

REFERENCES

[1] Ambler, S. W. (2004): *The Object Primer*, Cambridge University Press, New York.

[2] Angele, J.; Lausen G. (2004): *Ontologies in F-logic*, Handbook on ontologies, Springer-Verlag, Berlin.

[3] Baader, F.; Horrocks, I.; Sattler, U. (2004): *Description Logics*, Handbook on ontologies, Springer-Verlag, Berlin.

[4] Chiorean, J.Č Paşca, M.; Cârcu, A.; Botiza, C.; Moldovan, S. (2003): *Ensuring UML models consistency using the OCL Environment*, Sixth International Conference on the Unified Modelling Language – the Language and its applications, San Francisco.

[5] Cranefield, S.; Haustein, S.; Purvis, M. (2001): *UML_Based Ontology Modelling for Software Agents*, Proceedings of Ontologies in Agent Systems Workshop, Agents 2001, Montreal.

[6] Čubrilo, M.; Maleković, M.(2004): *Business Rules Modelling by Means of F-logic, UML and Ontologies (problems and possible solutions)*, Intelligent Systems at the Service of Mankind, Volume I, Ubooks, Neusäß.

[7] Denny, M. (2002): *Ontology Building: A Survey of Editing Tools*, http://www.xml.com /pub/a/2002/11/06/ontologies.html, 23.09.2004.

[8] Eriksson, E. H.; Penker, M. (2000): *Business modelling with UML: business patterns at work*, John Wiley & Sons, Indianapolis

[9] Eriksson E. H., Penker M., Lyons,B., Fado, D. (2004): *UML 2 Toolkit*, John Wiley & Sons, Indianapolis.

[10] Gómez Pérez A. et al. (2002): *OntoWeb Deliverable 1.3: A survey on ontology tools*, http://ontoweb.org/About/Deliverables/D13_v1-0.zip/, 17.05.2004.

[11] Grosso, W.: *The Protégé Axiom Language: Overall Design Considerations*, http://protege.stanford.edu/plug-ins/paltabs/OverallDesignConsiderations.zip, 11.10.2004.

[12] Gruber, T. (1993): A *Translation Approach to Portable Ontology Specifications, Knowledge Acquisition*, Vol.5, in Staab, S.; Studer, R., eds. (2004): *Handbook on Ontologies*, Springer-Verlag, Berlin.

[13] McBride B. (2004): *The Resource Description Framework (RDF) and its Vocabulary Description Language RDFS*, Handbook on ontologies, Springer-Verlag, Berlin.

[14] Noy, N. F.; McGuinness, D. L. (2000): *Ontology Development 101: A Guide to Creating Your First Ontology*, http://smi-web.stanford.edu/pubs/SMI_Abstracts /SMI-2001-0880.html, 23.09.2004.

[15] Rabuzin, K.; Maleković M. (2004): *Implementing Business rules in active databases*, Proceedings of 15th International Conference on Information and Intelligent Systems, Varaždin.

[16] Ray, I.; Li, N.; France, R.; Kim, D.-K. (2004): *Using UML To Visualize Role-Based Access Control Constraints*, SACMAT'04, Yorktown Heights, New York.

[17] Ross, R. G., ed. (2003): *Business Rules Manifesto*, Business Rules Group, http:// www.businessrulesgroup.org/, 14.05.2004.

[18] Ross, R. G. (2003): *Principles of the Business Rule Approach*, Addison Wesley, Boston.

[19] Ross, R.G. (1997): *The Business Rule Book*, Database research group, Inc., Boston.

[20] Sourrouille, J.L.; Caplat, G. (2002): *Constraint Checking in UML Modeling*, Proceedings of the 14th international conference on Software engineering and knowledge engineering, Ischia.

[21] Staab, S.; Studer, R., eds. (2004): *Handbook on Ontologies*, Springer-Verlag, Berlin.

[22] von Halle, B. (2002): *Business Rules Applied*, John Wiley & Sons, Inc., New York.

[23] *** (2000): *Defining Business Rules ~ What Are They Really?*, http://www.businessrulesgroup.org /first_paper/br01c0.htm, 14.05.2004.

[24] *** (2003): *OMG Unified Modeling Language Specification, v. 1.5*, http://www.omg.org /cgi-bin/docs?formal/03-03-01.pdf, 12.11.2004.

[25] ***: *PAL Documentation*, http://protege.stanford.edu/plug-ins/paltabs/pal-documentation/ index.html, 11.10.2004.

[26] *** (2005): Unified Modelling Language (UML) Version 2.0, http://www.omg.org/ technology/documents/formal/uml.htm, 10.01.2006.

[27] *** (2001): *VCI Case Study Description*, http://www.kpiusa.com/brbook/VCIcasestudy.htm, 17.02.2005.