Modeling Business Rules with UML State Diagrams

Konrad Kułakowski and **Grzegorz J. Nalepa** [1]

Abstract. Recently, in response to growing market demand, several different techniques of business rules representation have been created. Some of them try to present business rules in a visual manner. However, due to the complexity of the problem, the graphic representations that are proposed seem to be far from perfection. In this paper we would like to describe a visual rules modeling method based on UML classes and UML state diagrams. The strength of this approach relies on reusing classical notions provided by UML 2.0, e.g. an action, guard, etc., in a way which is close to theirs original meaning.

1 INTRODUCTION

Rules constitute a commonly recognized mechanism for representing knowledge about the world. In particular they are suitable for specifying the behavior and properties of different complex artifacts like information systems [14]. The rule-based approach is a foundation of various engineering and business systems. It is helpful for formulating business knowledge about the problem domain, defining the way in which systems interact with the changing environment and performing inference upon the knowledge. With time, rules applied to business problems have gained the name *business rules* and have become a separate notion. From the very beginning, business rules have aimed to be precise enough for professional software engineers and easy to use and to understand for all parties involved in the modeling of business domain concepts [6]. This is especially important since domain experts usually do not have mathematical knowledge indispensable for using formalisms like Prolog, Datalog or Process Algebras, for example.

The simplicity and expressiveness have also been very important for UML's authors. Since UML is perceived as a universal modeling language, in a natural way there is a tendency to use it for rule modeling [8, 7]. A very promising area for modeling rules in UML are business rules. UML, thanks to being popular with the software and business community, has an emerging opportunity to became a widely recognized language for business rules modeling.

The paper is organized as follows: In Sect. 2 related research in the area of business rules is discussed. Next, in Sect. 3, selected rule modeling aspects are summarized. Then, in Sect. 4 a new approach to rule representation with UML is proposed. Finally, in Sect. 5, concluding remarks, as well as directions for future work are contained.

2 RELATED WORKS

The first known usage of the term "Business Rules" comes from 1984 [1]. In fact, applying rules to business logic started in the late 1980s and the early 1990s[12, 4] and focused mainly on using business rules for data base modeling and programing. A serious attempt to make business rules better defined is "The Business Rules Book" written by Ronald G. Ross [13] and the report of the IBM GUIDE "Business Rules" Project [6]. In these works the authors define the scope of the problem domain, and identify core categories and patterns of business rules.

There is no one uniform business rule format [15, 2]; however, there are some standardization efforts in this area [3]. Also the idea of using UML together with business rules is not completely new. Usually, UML is treated as a language for expressing facts about terms in a model [5], whilst the rules themselves are not written in UML. In this context, the applying of UML/MOF to modeling rules, not only to the terms or facts, seems to be a very interesting perspective. There are several projects that try to propose UML/MOF representation for rules. One of them is *Production Rules Representation* (PRR) proposed by OMG [15]. PRR has been developed to address the need for a representation of production rules in UML models (business rules modeling as part of a modeling process). It proposes a meta-model for production rules and defines several notions like condition, action, binding and rulesets. The relationship between PRR and OMG model driven architecture is also discussed.

Another interesting initiative developed in order to exchange rules between communities is a general markup framework for integrity and derivation rules (R2ML) [16, 18]. The authors of R2ML define the rule concepts on the basis of RuleML [2] and Semantic Web Rule Language (SWRL) in terms of MOF/UML. On the top of the list of concepts provided by RuleML [17], a UML-Based Rule Modeling Language (URML) has been developed. It extends UML meta-model with the notion of a rule and defines new diagram elements supporting visual notations for rules [8, 19]. In this approach, modeling rules is done with the help of a class diagram enriched by one new diagram element called a *conclusion arrow*. A created model must conform to the *URML* meta-model, defining the semantics for all indispensable notions, i.e. rules, conditions, conclusions, etc.

Besides the indisputable benefits like providing visual rule notation in accordance with *UML/MOF*, a disadvantage of this approach is the relatively high number of classes required for defining a single rule. This complexity might be especially hard to accept for people who are familiar with simple if-then-else statements broadly used in several different tools such us *ILOG JRules* or *JBoss* rules. Also a new diagram elements such as rule's circle, may be a problem for people equipped with standard *UML* tools.

3 MODELING BUSINESS RULES

Let us take a closer look at the modeling concept first. The situation is as follows: by having a natural language description of a certain problem area, we aim at providing a declarative rule-based description of this area. The rule-based description is somehow formalized (or at

[1] Institute of Automatics, AGH – University of Science and Technology, Al. Mickiewicza 30, 30-059 Kraków, Poland email: kkulak@agh.edu.pl gjn@agh.edu.pl

least disciplined) compared to the original one. Rules are a knowledge representation method that captures regularities, constraints and relations. While formalized, this description is a high-level one, close to the original natural language-based one. So the basic sense of rule modeling is to build a rule-based knowledge representation of the problem. It is a classic case of knowledge engineering, where a designer, knowledge engineer has to identify, extract, describe and represent knowledge possessed by domain experts, or possibly embedded in an information system, such as an enterprise.

The rule representation should meet certain requirements. It should:

- be easy to grasp by non-technical individuals,
- be possible to process automatically and to integrate with a certain runtime (rule engine),
- formalized to some degree,
- meet certain quality standards (e.g. completeness, lack of redundancy),
- be suitable for interchanging and integration with other systems,
- be manageable.

The emphasis on these aspects can differ, depending on the goal of providing the rule-based description. This could be describing system requirements, including constraints, or building a complete system from scratch. Rules can also be thought of as a certain means of formalized communication.

Rule modeling methods and approaches should be considered with respect to other modeling methods such as software engineering methods and methodologies (e.g. UML, MDA). Since rules are often an essential part of business systems, business process modeling methods, such as workflows or BPMN, have to be taken into consideration in the chapter.

When it comes to the modeling *process*, different aspects can be pointed out:

- identifying concepts and their semantics,
- determining high-level structure ruleflow, rulebase contexts,
- building rules capturing the knowledge,
- integrating the ruleset,
- analyzing the quality of the model.

A rule-based model expressed by means of a certain rule language.

While modeling rules, some other important factors have to be taken into consideration. These include:

- rule applications and types, e.g. constraint handling, facts, derivation, etc., and
- rule inference model, mainly the forward and backward chaining case.

These issues can have an important influence on the rule language.

3.1 MODELING LANGUAGES

Rule modeling is a classic problem in the field of AI (Artificial Intelligence). It is a question of knowledge engineering (KE) and building rule-based expert systems that have strong logical foundations. In this chapter, some fundamental logical rule formats are considered, based upon the propositional or predicate calculus. The formats are a basis for rule languages. Rules can be practically written and processed in the logic programming paradigm, e.g. in Prolog. Even though the language uses a subset of first order predicate logic (restricted to Horn clauses), it is easy to write meta-interpreters working with languages of another order.

Within the AI, a number of *visual* knowledge representation methods for rules have been considered. These methods include:

- decision tables, that help combining rules working in the same context,
- decision trees, that support visualization of the decision making process, and
- decision graphs and lists, a less common but powerful method of control specification.

Two important factors for using these methods are:

1. design support – all of these methods help the designer (knowledge engineer) develop the rule-based model in a more rapid, and scalable manner, and
2. logical equivalence – all of these formally correspond to rules on the logical level.

These methods are used to model rules in practical applications. They also influenced some classic software engineering languages, e.g. UML.

A common approach to model rule-based systems is to use UML, considered by some as a universal modeling language. UML offers a visual or semi-visual method for different aspects of information modeling. By using this, it is possible to model some specific rule types. However, when it comes to practical knowledge engineering, it has some major limitations due to the different semantics of rules and the object-oriented paradigm. In particular cases, some of these shortcomings can be overcome by the use of OCL, which allows for constraint specification for UML classes.

One area where UML or UML-related methods are more useful is the conceptual modeling, which supports practical rule authoring. UML class diagrams are suitable to capture relations between concepts present in rule vocabularies. A new specialized UML-based method for specifying these kinds of information is SBVR from OMG. It also provides methods that support transforming natural language rule specification to a formalized rule format, mainly the so-called “structured English” format.

Since UML is a de facto standard information modeling method in software engineering approaches and tools, it can be treated as a low-level language on top of which a richer semantics is provided. This is possible for the standardized MOF and UML profiles formats. By building upon these, a dedicated rule modeling language can be built, e.g. URML or PRR.

An important community is built around the W3C and the so-called *Semantic Web Initiative*. The methods built on top of XML, RDF and OWL allow also for rule modeling for both web and general purposes. Rule interchange is possible using the XML-based RIF format.

The rule-based model can be used as a stand-alone logical core of a business application. However, in practice, this model should be somehow integrated with other models, and components of a heterogeneous application. Examples of integration discussed in this chapter include integration with business processes described with BPMN, as well as interfaces on the Java platforms. A number of approaches to the integration can be enumerated, with Model-View-Controller being a prime example.

Finally, the multilayer aspect of the rule language should be considered. A useful and expressive rule language should provide:

- rich, but well-defined semantics,
- formally defined syntax with clear logical interpretation,
- scalable visual representation, which allows for the visualization of many rules,

- machine readable encoding for model interchange and integration.

Using this criteria, it is easier to analyze selected languages.

3.2 HEKATE APPROACH

Some of the main concepts behind the HEKATE project [9], which aims at providing a complete rule modeling and implementation solution, are:

- providing an integrated design and implementation process, thus
- closing the semantic gap, and
- automating the implementation, providing
- an executable solution, which includes
- an on-line formal analysis of the design, during the design.

HeKatE uses methods and tools in the areas of:

- knowledge representation, for visual design,
- knowledge translation, for automated implementation,
- knowledge analysis, for formal verification.

Currently, development within the project is focused on the:

- conceptual design method, codename ARD+ [11], which allows for attribute (vocabulary) specification,
- logical design, codename XTT+ [10], for rule design using a hybrid decision tables and tree based method.

For project progress see `hekate.ia.agh.edu.pl`

A principal idea in this approach is to model, represent, and store the logic behind the software (sometimes referred to as business logic) using advanced knowledge representation methods taken from KE. The logic is then encoded with the use of a declarative representation. The logic core would be then embedded into a business application or embedded control system. The remaining parts of the business or control applications, such as interfaces or presentation aspects, would be developed with classic object-oriented or procedural programming languages such as Java or C.

4 REPRESENTING PRODUCTION BUSINESS RULES IN UML

Rules are widely recognized as a critical technology for building various types of knowledge-based applications. Rules are also important in information systems engineering, where they constitute a natural way of expressing business application logic. The classical form of a rule is a plain, textual if-then-else statement defining a rule's condition and rule's conclusions. On the other hand, there are a few propositions of visual rules modeling, e.g. URML [8].

In response to the market gap for visual rules modeling and taking into account the great popularity of UML as a general purpose modeling language, we would like to propose another UML-based approach to visual rules modeling. In this approach a rule is expressed as a class with a stereotype *rule*. Such a class has its own *state diagram*, which is used for expressing the rule's condition, conclusion and action.

4.1 INTERPRETABLE LANGUAGE

In the presented approach, UML is used for modeling a system and creating schemes of the rules. Other important parts of the model, such as events, guards conditions, actions and conclusions, are written in an *interpretable language*. In the context of UML the natural choice of language for expressing e.g. guard conditions is OCL. It has appropriate expression power and proven syntax constructions suitable for expressing statements upon the UML abstracts. Since actions are also important parts of some kinds of rules, there is still a need for another language for actions modeling. OCL as a constraint language does not seem to be the optimal choice for this purpose. On the other hand, a situation in which there are several different languages for modeling several different aspects of the systems is not convenient. The optimal solution should consists of UML and one interpretable language having mechanisms allowing for the expression of all non-UML terms like constraints and actions. Such a language should be easy to use by rule architects, and it also needs to be understood by the rule interpreter. Because people for whom the idea of using UML and state diagrams appears attractive should not be forced to use a certain implementation of a rule engine, the question of an interpretable language to expressing non-UML model elements remains open. Thus, however, the OCL language seems to be useful in the context of specifying logical statements upon the UML terms, in this article, it is treated rather like one possible and well documented proposition, not like a part of the final method's specification.

4.2 RULE DIAGRAMS

The rule diagram (Figure 1) is a UML class diagram containing the classes representing rules and some business terms that are directly used by the rules.

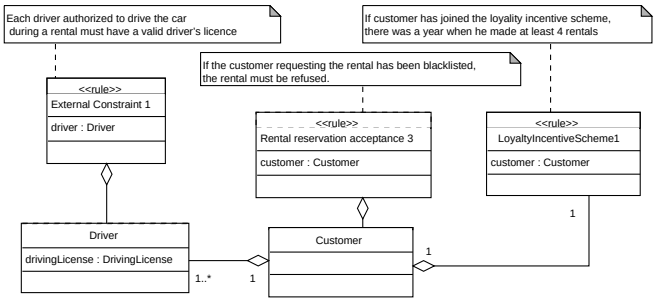

Figure 1. Rules as classes

The main role of the rule diagram is to show relationships between rules and business terms. Attributes of the class representing a rule should cover all of the business artifacts indispensable for a condition evaluation, a conclusion drawing or an action being performed. Every rule should have a textual comment informing about its business source.

4.3 BUSINESS VOCABULARY DIAGRAMS

Rules express some logical statements about terms and facts [6] that comes from the UML model. The set of all terms and facts will be called the *business vocabulary*. Thus, every UML diagram containing elements of *business vocabulary* which are neither *rule diagrams* nor *rule definition diagrams* associated with a rule will be called a *business vocabulary diagram*. The most popular kind of *business vocabulary diagram* is a class diagram (Figure 2).

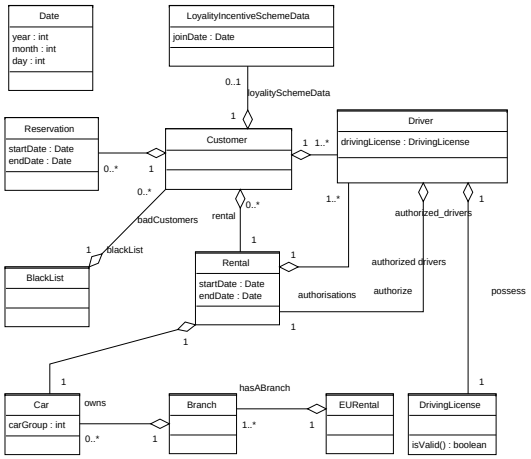

Figure 2. Business vocabulary diagram

4.4 RULE DEFINITION DIAGRAM

The definition of the rule has a form of an state diagram associated with the class representing the rule. This diagram will be called the *rule definition diagram*. The rule can be fired (can be applied) if, according to its *rule definition diagram*, it is able to change the state from a start state to a stop state. If there are some actions defined between a start state and a stop state, all of them have to be executed when the rule is triggered. If it is not possible for the rule to leave a start state, it means that the given rule is not active and cannot be executed at the moment. Following the conceptual rule classification [18, 8] we would like to define three types of business rules: Integrity Business Rules, Derivation Business Rules and Reaction Business Rules.

The integrity rule, also known as an integrity constraint, consists of a predicate function given producing a boolean value on the output. A very popular language for formulating constraints and expressing business knowledge in UML models is OCL (Object Constraint Language). It allows for precise formulation properties of relations between classes and objects. In fact, a guard condition may be formulated in any language understood by a rule interpreter; however, for the sake of examples' clarity, the OCL language is preferred. In the proposed approach the semantics of an integrity rule is given by a simple start-stop diagram containing one guard condition (Figure 3). Obviously, a condition of the integrity rule is met if the rule object changes its state to stop.

Figure 3. Integrity business rule

Derivation business rules have conditions and conclusions. Depending on the positive evaluation of the condition, a conclusion is drawn. In our approach, a condition is represented by a guard expression, whilst the conclusion is the action performed in the action state followed by a guard expression. The action state should have a stereotype *conclusion* (Figure 4). The action should have the form of a logical expression in a language understood by a rules interpreter. The action's logical expression represents a new knowledge derived from the existing facts (subjects of conditions) in the system.

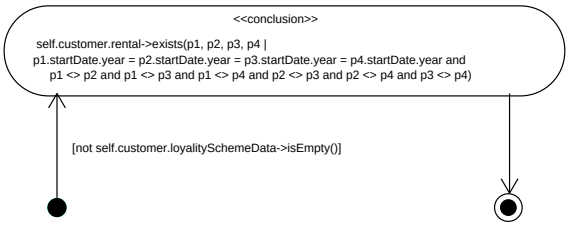

Figure 4. Derivation business rule

Reaction business rules (Figure 5) may have conditions, triggering events and actions. For the given rule, one condition or one triggering event (at least one is obligatory) and one action should be defined. In general, such a kind of rule allows for the modeling of an event-condition-action behavioral pattern, in which execution of the action is preceded by event triggering and guard condition evaluation. The absence of a condition is allowed only if a triggering event is defined, and inversely, a triggering event is not required if only a condition is defined. Thus, there are three possible subtypes of the reaction business rules:

- reaction business rule with a non-empty event and a non-empty condition
- reaction business rule with a non-empty event and an empty condition
- reaction business rule with an empty event and a non-empty condition

The first kind of rule is triggered by the event only if a guard condition is true. The second one models the executing action in response to the raised event, whilst the third kind of rule models the action execution as a result of changes in the system that make the guard condition true. The condition is modeled as a guard expression; thus, it may have any boolean form suitable for a rule interpreter (OCL is the preferred language). The action state in the *rule definition diagram* has a stereotype *action*.

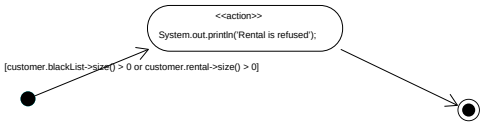

Figure 5. Reaction business rule

5 CONCLUSIONS

In this paper, the main assumptions of a new approach to representing business rules in UML has been presented. This approach allows for modeling business rules as UML state diagrams. It makes modeling rules similar to modeling system behavior, which may shorten the time required for modeling the system. A rule is represented by a well-known concept of a stereotyped class; thus, there is no need to

define any new UML artifacts except for stereotypes. Consequently, almost every UML 2.0 compatible modeler might be used for rule modeling. It is easy to find the business vocabulary since it is explicitly shown in UML diagrams. With the help of stereotyped rules, the well known statechart concepts, such as action, guard and event, retain as much as possible from their original meaning. E.g. since applying the rule is represented by following the transitions of an state diagram, a guard concept remains a kind of expression deciding whether we may apply the rule, i.e. whether we may follow the transition. Moreover, a rule may formulate statements upon the all the classes that existed in a given model. Since every rule is a class in a model, it is possible to create rules that apply to other rules.

Since some of a rule's components are written in OCL or other interpretable languages, rule modeling may seem to be a little bit harder than using a graphical notation. On the other hand, such languages are usually quite simple; e.g. in OCL some more sophisticated constructions like nested collections has been abandoned [20]. Thus, after getting a bit of practice in the chosen language, working with rules written in UML and state diagrams should not be a problem.

Regardless of the lack of a strictly defined language for actions and guards expressions, some experiments in these areas are being conducted. The aim of the authors is to build a complete solution containing a full-featured example that might be analyzed with the help of the selected rule framework. To achieve this, an appropriate transformation from a UML model to a terms and facts model has to be defined. Term and fact representation, as well as the choice of the interpretable language, has to depend on the selected rule processing environment. One of the considered platforms for rule processing is the Prolog language.

The work presented here will be integrated within the *HeKatE* approach briefly discussed in Sect. 3.2. The basic idea is to model a rule-based logical application core with the visual representation presented here. The OCL expression can be replaced by Prolog-based rules, since Prolog is the language of choice for the HeKatE prototype implementation. Since HeKatE aims at designing applications using the Model-View-Controller pattern, using an UML-based representation greatly improves the possibility of integration with the UML-based view design.

ACKNOWLEDGEMENTS

The paper is supported by the HeKatE Project funded from 2007–2009 resources for science as a research project.

References

[1] D. S. Appleton, 'Business rules: the missing link', *Datamation, 15*, **30**(16), (October 1984).
[2] H. Boley, 'The ruleML family of web rule languages', in *PPSWR*, eds., José Júlio Alferes, James Bailey, Wolfgang May, and Uta Schwertel, volume 4187 of *Lecture Notes in Computer Science*, pp. 1–17. Springer, (2006).
[3] H. Boley and M. Kifer. RIF basic logic dialect. World Wide Web Consortium, Working Draft WD-rif-bld-20071030, October 2007.
[4] C. C. Fleming and B. von Halle, *Handbook of Relational Database Design*, Addison-Wesley Professional, Reading, 1989.
[5] T. Halpin. Verbalizing business rules: Part 1, April 03 2004.
[6] D. Hay and K. A. Healy, 'Defining business rules what are they really?', Technical report, the Business Rules Group, (2000).
[7] S. Lukichev and G. Wagner. UML-Based Rule Modeling with Fujaba, 2006.
[8] S. Lukichev and G. Wagner, 'Visual rules modeling', in *Ershov Memorial Conference*, eds., Irina Virbitskaite and Andrei Voronkov, volume 4378 of *Lecture Notes in Computer Science*, pp. 467–473. Springer, (2006).
[9] G. J. Nalepa and I. Wojnicki, 'A proposal of hybrid knowledge engineering and refinement approach', in *FLAIRS-20 : Proceedings of the 20th International Florida Artificial Intelligence Research Society Conference : Key West, Florida, May 7-9, 2007*, eds., David C. Wilson, Geoffrey C. J. Sutcliffe, and FLAIRS, pp. 542–547, Menlo Park, California, (may 2007). Florida Artificial Intelligence Research Society, AAAI Press.
[10] G. J. Nalepa and I. Wojnicki, 'Proposal of visual generalized rule programming model for Prolog', in *17th International conference on Applications of declarative programming and knowledge management (INAP 2007) and 21st Workshop on (Constraint) Logic Programming (WLP 2007) : Wurzburg, Germany, October 4–6, 2007 : proceedings : Technical Report 434*, eds., Dietmar Seipel and et al., pp. 195–204, Wurzburg : Bayerische Julius-Maximilians-Universitat. Institut fi£¡r Informatik, (september 2007). Bayerische Julius-Maximilians-Universitat Wurzburg. Institut fi£¡r Informatik.
[11] G. J. Nalepa and I. Wojnicki, 'Towards formalization of ARD+ conceptual design and refinement method', in *FLAIRS2008*, (2008). submitted.
[12] R. Ross, 'Entity modelling: Techniques and application', Technical report, Database Research Group, Boston, MA, (1987).
[13] R. G. Ross, *The Business Rule Book*, Business Rule Solutions, 1994.
[14] S. Russell and Norvig P., *Artificial Intelligence: A Modern Approach*, Prentice-Hall, Englewood Cliffs, NJ, 1995.
[15] S. Tabet, G. Wagner, S. Spreeuwenberg, P. D. Vincent, J. Gonzaques, M. C. de Sainte, J. Pellant, J. Frank, and J. Durand, 'OMG production rule representation - context and current status', in *Rule Languages for Interoperability*. W3C, (2005).
[16] G. Wagner. How to design a general rule markup language, June 2002. Invited talk at the Workshop XML Technologien für das Semantic Web (XSW 2002), Berlin.
[17] G. Wagner, G. Antoniou, S. Tabet, and H. Boley, 'The abstract syntax of ruleML - towards a general web rule language framework', in *Web Intelligence*, pp. 628–631. IEEE Computer Society, (2004).
[18] G. Wagner, A. Giurca, and S. Lukichev, 'A general markup framework for integrity and derivation rules', in *Principles and Practices of Semantic Web Reasoning*, eds., François Bry, François Fages, Massimo Marchiori, and Hans-Jürgen Ohlbach, number 05371 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, (2006). <http://drops.dagstuhl.de/opus/volltexte/2006/479> [date of citation: 2006-01-01].
[19] G. Wagner, A. Giurca, and S. Lukichev, 'Modeling Web Services with URML', in *Proceedings of Workshop Semantics for Business Process Management 2006, Budva, Montenegro (11th June 2006)*, (2006).
[20] J. Warmer and A. Kleppe, *The Object Constraint Language: Precise Modelling with UML*, Object Technology Series, Addison-Wesley, Reading/MA, 1999.