UML Representation Proposal for XTT Rule Design Method*

Grzegorz J. Nalepa[1] and Krzysztof Kluza[1]

Institute of Automatics,
AGH University of Science and Technology,
Al. Mickiewicza 30, 30-059 Kraków, Poland
gjn@agh.edu.pl, kluza.krzysztof@gmail.com

Abstract In the paper practical issues concerning the use of UML as a knowledge representation method for rules are discussed. A proposal of an UML-based representation for the XTT structured representation for rules is presented. Since some deep semantical differences between UML and rule-based representation exist, several possible UML representations are evaluated. A practical algorithm for building an UML representation using activity diagrams from XTT tables is proposed.

1 Introduction

Using Knowledge Engineering (KE) methods in practical Software Engineering [1] (SE) has gained some momentum in recent years. One of the best examples is the business rules approach [2]. The fact is that KE has been developed in parallel with SE, and both approaches use different methods and tools to actually model and build systems. Important semantical differences between these two make the use of the KE methods in SE non-trivial, whereas using SE methods with KE problems is often of limited use.

The paper focuses on analyzing possibilities of the *practical* use of the UML language to model XTT rule-based systems. XTT [3] is a structured knowledge representation for rules, based on some classic KE notions of decision tables and decision trees. Representing XTT with UML artifacts encounters number of issues addressed in the paper. A new algorithm for encoding an XTT diagram using UML is introduced.

The paper is organized as follows. In the Sect. 2 possibilities of using UML as a knowledge representation language are discussed. Then, in Sect. 3 knowledge representation issues in the HeKatE project are presented. The paper presents an UML-based representation for the XTT rule representation method. The application of this method is discussed using the example thermostat control system presented in Sect. 4, whereas the method itself is introduced in Sect. 6 and 5. The UML representation of the thermostat is then discussed in Sect. 7. Directions for future work are presented in the final section.

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

2 UML as Knowledge Representation

UML as a design language identifies two distant domains of Software Engineering, that is software structure modelling, and behavior modelling. There are two main diagram classes. Structure Diagrams and Behavior Diagrams containing different types of diagrams.

Apparently that Structure Diagrams are often considered the basis of UML. They are fairly complete and allow for expressing software components and denoting relationship among them easily (i.e.: Class Diagram, Component Diagram etc.). Behavior diagrams model software logic at different abstraction levels. First of all there is a big picture perspective: modelling what particular software should do, from the user point of view (i.e.: Use Case Diagram). There is also a detailed perspective: what particular software components defined by the Structure Diagrams should do (i.e.: State Machine Diagram, Interaction Diagram etc.). The problem is that these two perspectives do not fit together very well with the Structure Diagram. While the detailed perspective corresponds to classes, the big picture one serves more as a guideline, than a real modelling tool. So it seems that there is hardly any real relationship between modelling software behavior and its structure.

Another issue regards the *semantic gap* between the design and the implementation [4]. Even if discussed diagrams support the implementation process by describing software in a comprehensive way, it is impossible to verify in a reasonable time if the implementation matches the design. There is also another gap in the specification-design-implementation process called *analysis specification gap* [5]. It regards a difficulty with the transition from a specification to the design. Formulating a specification which is clear, concise, complete and amenable to analysis turns out to be a very complex task, even in small projects.

Applying UML as a Knowledge Engineering method is not straight forward [6]. Existing diagrams are not suitable for rule modeling or expressing knowledge in general. Using an UML profile, which is a redefinition of the semantics of certain diagrams, does not help much, and in some cases might complicate the design. It forces the use of existing diagrams for purposes they were not designed for i.e. representing rules is tricky and inefficient.

There are several possible approaches when it comes to practical UML application for knowledge engineering. The first solution is the "classic" and definitely the easiest one. It consists in modelling the system with a knowledge-based approach, that uses some classic knowledge representation method, such as decision trees, then design the software implementation using UML, and generate an object-oriented (OO) code. In this case, KE methods are used in the "design" stage, while SE methods provide "implementation" means. The problem is, that there is a fundamental difference in the semantics of the KE methods, such as decision rules, and UML.

In the second approach the rule-based knowledge is modelled with UML diagrams, and then the corresponding OO code is generated. This approach relies on either extending, or redefining the original semantics of UML. Some early beginning can be observed in *OMG Production Rule Representation* [7], where

some ideas of extending existing semantics of UML were contained. However, a complete example of this approach may be found in the *Unified Rule Modelling Language* (URML), (see [8]). In this case, existing UML diagrams are used to model different type of rules.

The last one is possibly the most complicated approach. It relies at incorporating a complete rule-based logic core into an OO application. It aims at minimizing the semantic gap between SE and KE. Such a solution is being developed in the *HeKatE* project, where a declarative, rule-based core is integrated into an OO application as a logical model (as in the Model-View-Controller design pattern [9]). In this paper it is proposed to find a UML representation corresponding, to XTT –a custom rule representation formalism. Such a representation could then be used to present a rule-base application model designed with XTT in UML.

3 Knowledge in the HeKatE Design Process

The HeKatE project aims at applying selected AI tools into SE. The approach in based on incorporating an extended rule-based model as a logical application core. The model uses the *XTT* rule representation and design method, with the support of the ARD rule prototyping method. The project provides *hierarchical* design *process*, which should ultimately be represented by both custom XTT/ARD methods, as well as an UML-bases representation.

The main difference between the HeKatE knowledge representations and UML diagram is, that UML, after all, does not provide *a design process*. Whereas, HeKatE *is* about the integrated design process. So the methods on which HeKatE is based, have been invented with the design *process* in mind. Where UML provides means to model the system from different perspectives, or aspects; HeKatE focuses on the subsequent phases of the design process of the very same system. So different levels of abstraction (e.g. subsequent ARD levels, or XTT design iterations) describe *the very same holistic model* all the time (see also MDA [4]).

The *XTT* (*EXtended Tabular Trees*) knowledge representation [3], has been proposed in order to solve some common design, analysis and implementation problems present in RBS. In this method three important representation levels has been addressed: *visual* – the model is represented by a hierarchical structure of linked extended decision tables, *logical* – tables correspond to sequences of extended decision rules, and *implementation* – rules are processed using a Prolog representation. On the visual level the model is composed of extended decision tables, see Fig. 1. The table represents a set of rules, having the same attributes. On the logical level, a table corresponds to a number of rules, processed in a sequence. If a rule is fired and it has a link, the inference engine processes the rule in another table.

In addition to XTT which represents rules, there is an entire design process involved. XTT diagrams are at the very end of this process. It is an *Attribute Relationship Diagram* (ARD) based approach. It offers a process of identifying attributes and relationships among them. The key underlying assumption in the

ARD design with knowledge specification in attributive logics is that, similarly as in the case of Relational Databases [10], the attributes are *functionally dependent*. An ARD *diagram* is a conceptual system model at a certain abstract level. Attributes are subsequently identified at more and more detailed levels. The process includes all levels. At the most detailed level, XTT diagrams are added to define dependencies among attributes and to describe how to calculate attribute values. The ARD process is similar, in terms of its goals, to Structure Diagrams. However, while the Structure Diagrams tend to describe what elements the software consists of, ARD describes what is *known* about it.

4 Thermostat Case Study

The analysis of the UML representation is conducted using a classic rule-based control system example, a *Thermostat case*, found in [11]. The main problem consists in creating a temperature control system for an office. The system needs to take into account current date, including the day of the week, as well time of the day. The original design has 18 rules, and has been studied in detail in the HeKatE project. Here only the complete XTT design is presented in Fig. 1.

In the subsequent sections several approaches to the XTT representation in UML are presented, with the optimal one used to represent the whole XTT Thermostat design.

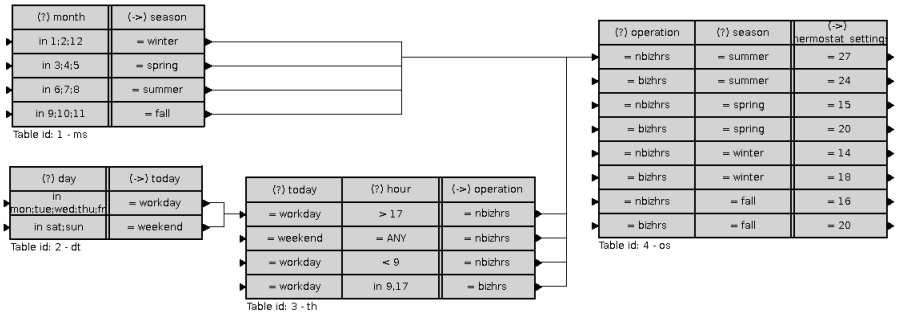


Figure 1. XTT Thermostat Design

5 Modeling XTT with UML

In this section the evolution of the UML representation for XTT is discussed. Several attempts to provide such a representation are presented, in order to expose the semantic and conceptual differences between UML and knowledge-based systems. Finally, an optimal solution is proposed.

Decision tables in XTT represent rules that have the same attributes. The rules in a single table are processed sequentially. So at this point a reasonable idea is to try to use diagrams that show not so much the structure of the system, but its behaviour (dynamics). It could be diagrams such as use case diagrams, activity diagrams or state diagrams, as well as diagrams of interaction (sequence or collaboration diagrams).

According to [6] State Machine Diagrams and Activity Diagrams seem to be the best UML candidates for rule modeling, but not good enough to serve the purpose of rule modeling with similar expressiveness as XTT. It is possible to use them to express rules in case of smaller systems. However, in case of larger systems, where the number of states grows fast their use poses some practical problems. So the first attempt to use UML for XTT is to investigate activity and state diagrams.

It is important to understand the differences and similarities between state machine diagrams and activity diagrams. Activity and state diagrams are related. A state diagram shows the possible states of the object and the transitions that cause a change in state. It focuses on an object undergoing a process (or on a process as an object). However, an activity diagram focuses on the flow of activities involved in a single process and shows how they depend on one another [12].

The first attempts were carried out using state diagram for XTT modelling. State diagrams capture the behavior of a software system and state machine uses graph notation to represent the behavior of a piece of a system [13]. The Table 1 lists the types of pseudostates used in state diagrams.

Table 1. Types of used pseudostates

	Initial pseudostate – the starting point of a state machine.
	Choice – allows the execution of a state machine to choose between several different states based on guard conditions on the transitions.
	Fork and join – represents a split in the execution of the state machine into orthogonal regions. The join reunites the regions into a single transition. The state machine won't transition from the join until all regions have transitioned to the join pseudostate.

The attempts to find a UML representation are presented on a single XTT table from the Thermostat systems, the *TH* table.

The first abandoned attempt The Fig. 2 shows the corresponding UML state diagram for the sample XTT table. However, in this case, where XTT is transformed into activity diagrams it is not clear which attributes should be transformed to the states and which to guard conditions.

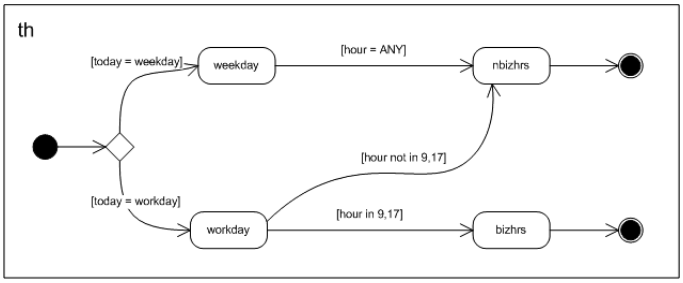


Figure 2. First attempt: State diagram corresponding to XTT TH diagram

The second abandoned attempt To avoid the problem from the first attempt (how to transform XTT attributes), let:

- values of the XTT output attributes (H in XTT) become states,
- individual rows from XTT (conjunction of values in their cells) become guard conditions.

The Fig. 3 shows the modified UML state diagram corresponding to the sample XTT table. Unfortunately, with when the number of rules in XTT table grows, the diagram becomes poorly readable.

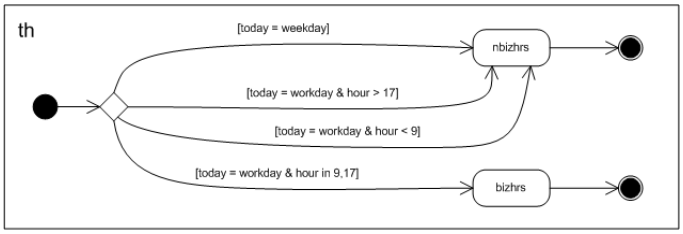


Figure 3. First attempt: State diagram corresponding to XTT TH diagram

The third abandoned attempt The diagram could be more readable if we use a fork pseudostate instead of a choice pseudostate. The Fig. 4 shows the corresponding UML state diagram for the sample XTT table with the fork pseudostate. If there is an deficiency in an XTT table (e.g. as a result of a mistake) and different rows will not exclude each other, than the fork pseudostate duplicates the input value and may transfer the control to more than one edge of the subsequent states.

The limitation of all of these three approaches is the lack of the output attribute naming, and geting the names of input attributes it is needed to search for them in the guard conditions.

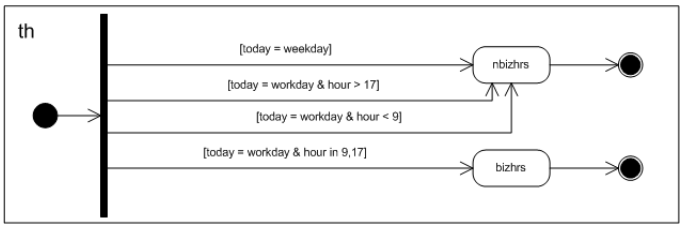


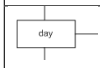
Figure 4. Third attempt: State diagram corresponding to XTT TH diagram

6 UML Model for XTT

Considering the previously analyzed approaches, in this section a more optimal translation is proposed.

In general, activity diagrams are related to flow diagrams and can illustrate the activities taking place in the system. The Table 2 lists the types of nodes used in activity diagrams.

Table 2. Types of used pseudostates

	Action
	Decision node
	Merge node
	Fork node
	Join node
	Partitions (swimlanes)
	Decision node
	Parameter of activity

Finally, an algorithm to transform XTT diagrams to UML activity diagrams has been formulated. The proposed transition algorithm from XTT table to UML activity diagram is as follows:

1. All input attributes become input parameters and output attribute becomes output parameter of an activity (for the demarcation of the diagram can be divided into the partitions with a swimlane).

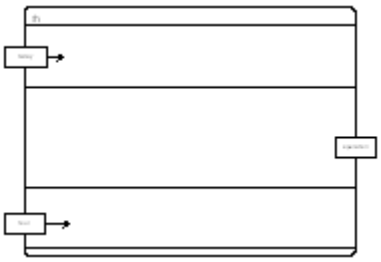


Figure 5. Example of applying of the first step of algorithm

2. For each attribute (activity parameter), if there is more than one unique value in the XTT, a decision node and for every unique value of attribute needs to be added:
   (a) the control flow with guard condition is introduced (with that unique value in it),
   (b) if the value occurs frequently, the flow is finished with a fork node with number of outputs equal to the number of times the value appears in XTT table.

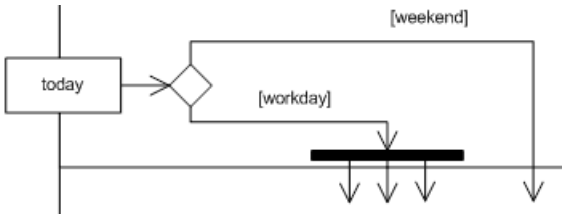


Figure 6. Example of applying of the second step of algorithm

3. For each rule (a row in XTT) a join node with the number of inputs equal to the number of input parameters is drawn and another one for output. For each join node:
   (a) inputs are connected using an adequate flow control (in accordance with the values of attributes in the rule),
   (b) outputs are connected using a flow control with the action having a value corresponding to the output attribute in the rule:
      i. directly, if the value of attribute occurred in XTT only once,
      ii. otherwise through merge node.
4. Outputs of all actions are merged in a merge node and a control flow is lead to output parameter of activity.

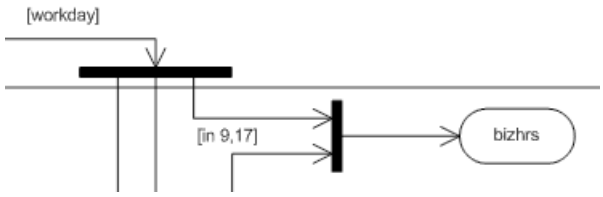


Figure 7. Example of applying of the third step of algorithm

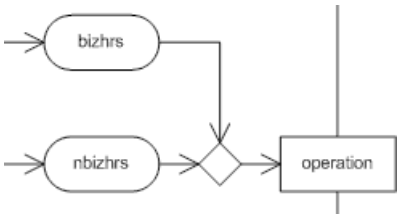


Figure 8. Example of applying of the fourth step of algorithm

7 XTT for Thermostat in UML

Activity diagrams constructed with an algorithm above for the thermostat will look as in the Tables 3, 4, 5, 6. For the sake of transparency, modeling the diagram of the entire thermostat, these activities are nested. An activity presented in a nested form refers to a number of actions of this activity. (However, it is not presented directly in the diagram). Figure 9 shows the diagram for the whole thermostat system as discussed in the Sect. 4.

Table 3. Activity diagram corresponding to XTT MS table

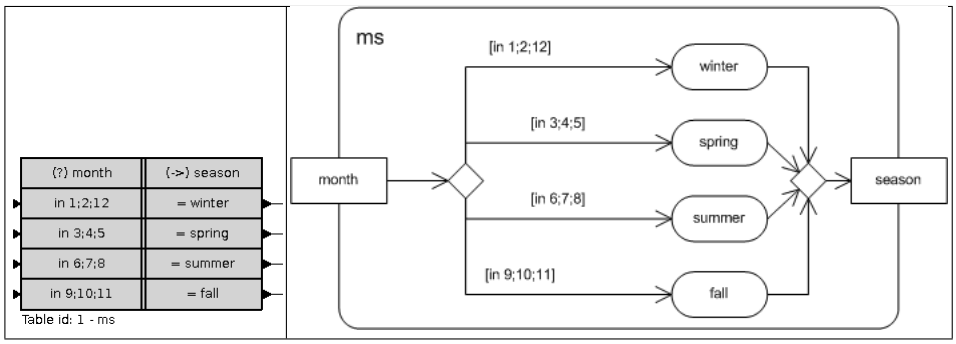


Table 4. Activity diagram corresponding to XTT DT table

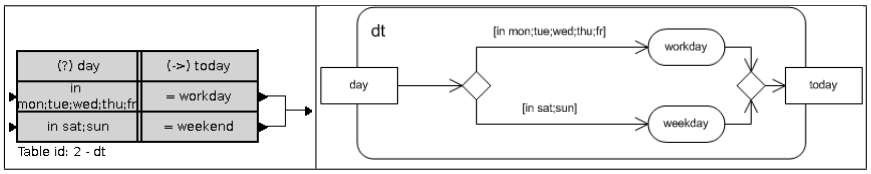
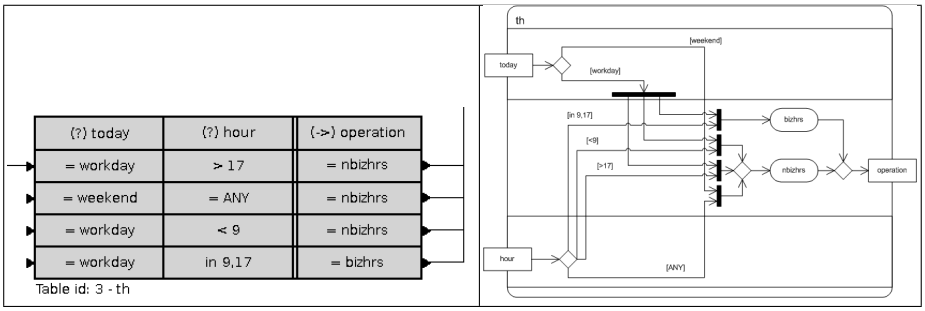


Table 5. Activity diagram corresponding to XTT TH table



8 Evaluation and Future Work

In this paper the use of UML as knowledge representation for rule-based systems has been considered. Several possibilities have been described in order to expose some non-trivial issues concerning this representation. The original contribution of the paper is an UML-based representation of XTT diagrams describing a rule-based system.

Since the problem is not new, some other approaches exist, so it is worth noting how the HeKatE approach compares to the existing solutions. Currently, two most important representations include OMG PRR [7] and REWERSE

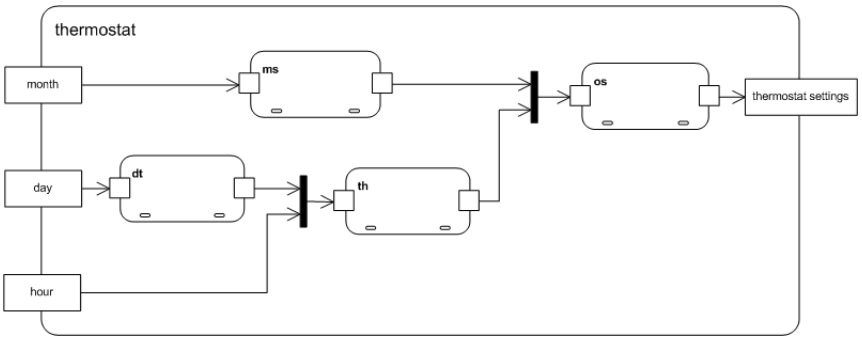
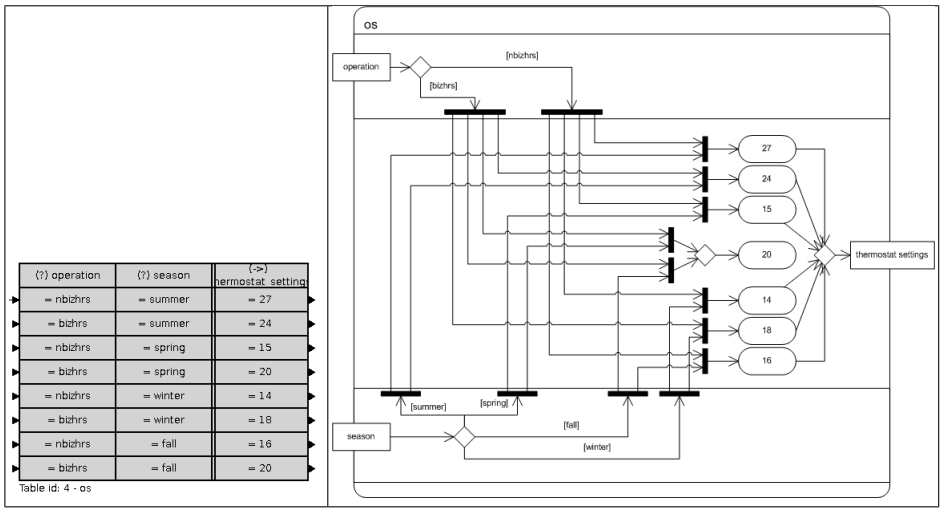


Figure 9. Activity diagram for the whole thermostat

Table 6. Activity diagram corresponding to XTT OS table



URML [8]. The fact is, that both of these aim at detailed modelling of single rules. On the other hand, by definition, in the XTT approach the design is focused on the tree like structure of decision tables. So the representation introduced in this paper aims at translating the whole structure of extended decision tables into UML. Another difference is, that the HeKatE approach does not introduce new UML artifacts. It also does not aim at redefining some of the UML semantics by using a custom profile. Instead it tries to explore and efficiently use the existing diagrams.

The work presented in the paper is a work in progress. The proposed algorithm is being implemented and tested. Several methods are considered, including an XSLT translation to the XMI format. Ultimately the model designed with this method should be embeddable into any business application using the MVC pattern. In the future, the whole HeKatE design process including rule prototyping with ARD and design with XTT should be put into the context of the MDA [14].

References

1. Sommerville, I.: Software Engineering. 7th edn. International Computer Science. Pearson Education Limited (2004)
2. Ross, R.G.: Principles of the Business Rule Approach. 1 edn. Addison-Wesley Professional (2003)
3. Nalepa, G.J., Ligęza, A.: A graphical tabular model for rule-based logic programming and verification. Systems Science **31**(2) (2005) 89–95
4. Mellor, S.J., Balcer, M.: Executable UML: A Foundation for Model-Driven Architectures. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2002)

5. Rash, J.L., Hinchey, M.G., Rouff, C.A., Gracanin, D., Erickson, J.: A tool for requirements-based programming. In: Integrated Design and Process Technology, IDPT-2005, Society for Design and Process Science (2005)
6. Nalepa, G.J., Wojnicki, I.: Using UML for knowledge engineering – a critical overview. In Baumeister, J., Seipel, D., eds.: 3rd Workshop on Knowledge Engineering and Software Engineering (KESE 2007) at the 30th annual German conference on Artificial intelligence : [September 10, 2007, Osnabrück, Germany]. (september 2007) 37–46
7. OMG: Production rule representation. Technical report, Object Management Group (br/2003-09-03)
8. Lukichev, S., Wagner, G.: Visual rules modeling. In: Sixth International Andrei Ershov Memorial Conference Perspectives Of System Informatics, Novosibirsk, Russia, June 2006. LNCS, Springer (2005)
9. Burbeck, S.: Applications programming in smalltalk-80(tm): How to use model-view-controller (mvc). Technical report, Department of Computer Science, University of Illinois, Urbana-Champaign (1992)
10. Connolly, T., Begg, C., Strechan, A.: Database Systems, A Practical Approach to Design, Implementation, and Management. 2nd edn. Addison-Wesley (1999)
11. Negnevitsky, M.: Artificial Intelligence. A Guide to Intelligent Systems. Addison-Wesley, Harlow, England; London; New York (2002) ISBN 0-201-71159-1.
12. Miller, R.: Practical UML: A Hands-On Introduction for Developers, http://dn.codegear.com/article/31863#activity-diagrams. (2003)
13. Pilone, D., Pitman, N.: UML 2.0 in a Nutshell. O'Reilly (2005)
14. Miller, J., Mukerji, J.: MDA Guide Version 1.0.1. OMG. (2003)