XTT Rules Design and Implementation with Object-Oriented Methods

Grzegorz J. Nalepa
Institute of Automatics,
AGH University of Science and Technology,
Al. Mickiewicza 30, 30-059 Kraków, Poland
gjn@agh.edu.pl

Abstract
In this paper certain knowledge and software engineering methods integration issues are discussed. The principal idea is to consider an effective design and implementation framework for rule design with UML, and implementation with Java. The solution proposed in the paper consists of using a custom knowledge engineering design method for rules in the design stage. The rule base is then transformed to UML behavioral diagrams, which can be considered a visual encoding. The rule implementation involves the serialization to Java language using classes representing the decision tables grouping rules sharing the same attributes.

Introduction
In Artificial Intelligence (Russell and Norvig 2003) system modeling consist in providing a practical knowledge representation (van Harmelen, Lifschitz, and Porter 2007) about the system. It is a generic approach, where some specific concepts such as the structure of the system, or representation artifacts have not to be determined from the start. Modeling in knowledge engineering is often a gradual process that uses number of methods from ones close to the natural language, to formalized representations such as rules.

In recent years there has been a continuous research on integrating Knowledge Engineering (KE) methods in practical Software Engineering (Sommerville 2004) (SE). One of the examples is the business rules approach (Ross 2003), another one is the Semantic Web initiative. The fact is that KE is being 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 to solve KE problems is often of limited use. One of the issues in this integration is how to align modeling methods used in these domains. While in SE UML is a de facto standard modeling tool, in KE number of methods exist (Brachman and Levesque 2004; van Harmelen, Lifschitz, and Porter 2007).

In this paper certain knowledge and software engineering methods integration issues are discussed. The principal idea is to consider an effective design and implementation frame-

Copyright © 2009, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.
are tackled with the attempts to introduce rule-based representations to the UML-centric solutions family. Some of the proposed standards include the Production Rule Representation (PRR) from OMG, where some ideas of extending existing semantics of UML were contained. Another one is the Unified Rule Modelling Language (URML), (see (Lukichev and Wagner 2005)) from the REWERSE II research project, where existing UML diagrams are used to model different types of rules.

It can be observed, that integrating knowledge representation and engineering with software engineering approach could be considered on two levels. So a complete integration solution should address these two perspectives. In both cases it is not an easy task.

**Firstly, on the modeling or design level**, where KE uses number of knowledge representation methods with decision rules, tables and trees being some of the most important ones, and SE exclusively uses UML, and other MOF-based solutions (MetaObject Facility).

**Secondly, on the implementation or runtime level**, where KE uses multiple solutions and languages (both AI ones, such as Lisp or Prolog), including the so-called expert systems shells, or rule engines, with CLIPS, Jess or Drools, being the best examples. In the case of practical SE, the implementation is conducted in a OO language, usually Java, on top of complex application stacks, e.g. Java EE.

So a complete integration solution should address these two perspectives. In both cases it is not an easy task.

Applying UML as a Knowledge Engineering method is not straightforward. Existing diagrams are not suitable for rule modeling since they lack the rule semantics. 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 rule sets might be tricky and inefficient.

There are several possible approaches when it comes to practical UML application for knowledge engineering. In fact, they need to somehow explicitly tackle both of the integration levels. The first solution is the “classic” and definitely the easiest one. It consists in modeling the system with a knowledge-based approach, that uses some classic knowledge representation methods, 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. 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 (see PRR and URML).

In this paper another approach is proposed. It is a hybrid approach, where the application logic is modeled with rules, and remaining interfaces use classic SE solutions. This relies on the rule-based knowledge representation with UML, and knowledge implementation with OO languages.

More specifically, the idea is to model the application logic with a custom KE method, based on decision tables, grouping decision rules, with the rulebase explicitly structured in a tree-like fashion, then providing a UML representation for SE designers, and then possibly integrate the rule interpreter with the OO application stack (as in the Model-View-Controller design pattern (Burbeck 1992)), or generate OO code serializing the designed rule knowledge.

This paper proposes a UML representation for rules that preserves the semantics of the original rule representation and provides an effective OO encoding for rules.

### Heterogeneous Approach

The integration approach proposed in this paper is developed within the HeKatE project (hekat.e ia.agh.edu.pl). In the following subsection main goals of the project are presented, and the outline of the approach is given.

#### Rule Modeling in HeKatE

HeKatE aims at developing formalized knowledge representation methods based on rules, and providing an effective integration framework with software engineering tools. Main concepts include an extended formal logical system description using the ALSV(FD) formalism (Attributive Logic with Set Values over Finite Domains (Nalepa and Ligeza 2008), a conceptual rule design method ARD+ (Attribute Relationship Diagrams (Nalepa and Ligeza 2005b; Nalepa and Wojnicki 2008)), that to some extent corresponds to requirements engineering in SE, logical system design with rules in the XTT structure (Nalepa and Ligeza 2005a), with an on-line formal verification, and finally an automated implementation with prototype generation in a meta representation, that can be executed by a meta-interpreter.

HeKatE methodology goals also include delivering rule-based design methods for knowledge-based systems, that could be effectively integrated into business applications, and UML-based integration of these methods at the design level, with OO interface at the runtime level, while providing a formal verification of application logic with continuous quality control during the application development cycle.

The HeKatE design process begins with the conceptual ARD model, which is the prototype for the logical XTT model. The key underlying assumption in the ARD design with knowledge specification in attributive logics is that, similarly as in the case of Relational Databases (Connolly, Begg, and Strechan 1999), 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. At the most detailed level, XTT diagrams are added to define dependencies among attributes and to describe how to calculate attribute values. The ARD design process is similar, in terms of its goals, to UML Structure Diagrams. However, while the Structure Diagrams tend to describe what elements the software consists of, ARD describes what is known about it.

The XTT (Extended Tabular Trees) knowledge representation (Nalepa and Ligeza 2005a), has been proposed in order to solve some common design, analysis and implementation problems present in rule-based systems. 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 meta representation and a custom interpreter. 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 a rule in another table.

In Fig. 1 an ARD rule prototype and the corresponding XTT structure for the Thermostat rule-based system (Negnevitsky 2002) are shown. Using them the UML representation for both aspects will be introduced.

Outline of the Approach

In the proposed approach, the important phases of the classic SE process (Sommerville 2004) are matched by the corresponding phases of the HeKatE design process, in a way similar to classic KE process or expert system design.

The requirements engineering in SE corresponds to the conceptual design or rules prototyping in KE, which in HeKatE is supported by the attribute formalization in ALSV(FD) and ARD+ design. In this phase a corresponding custom UML representation for ARD is provided.

The main design phase UML diagrams corresponds to the logical design, which in the case of HeKatE is provided by the XTT formally described with ALSV(FD), and visually represented by the structured XTT rulebase. In this phase a corresponding XTTL representation for XTT is provided.

The implementation phase which in SE includes partial and semi-automatic OO code generation from the UML model, corresponds to the physical design of expert systems or RDB systems (Connolly, Begg, and Strechan 1999). In HeKatE two scenarios are considered in this phase:

- representing XTT in the textual algebraic Hekate Meta Representation (HMR), and executing it using a meta interpreter integrated with a OO Java runtime level,
- generating OO Java code for rules that logically corresponds to the XTT, in the process called XTTL serialization; the code can be easily integrated with other Java-based parts of the application at the source code level.

In both cases the XTT-based logic is integrated with the application in the Model-View-Controller paradigm, providing a logical application model.

The focus of this paper is on the UML representation for ARD and XTT, as well as the description of the serialization procedure. These are described in the following sections.

ARD Representation in UML

The ARD method aims at capturing relations between attributes describing system properties. Attribute relationships are a certain structure in the attribute space. Therefore, the proposed UML model uses static structure diagram to show these relations. Elements in proposed UML model corresponds one-to-one with the attributes and properties in ARD (it is a bijective transformation) (Kluza 2008).

A UML dependency in structure diagrams “indicates a semantic relationship between model elements. It relates the model elements themselves and does not require a set of instances for its meaning. It indicates a situation in which a change to the target element may require a change to the source element in the dependency” (OMG 1997).

The following UML dependencies where selected to specify the diagram semantics:

- derive specifies a derivation relationship among model elements, where one of them can be computed from the another.
- refine specifies a refinement relationship among model elements at different levels of development. Refinement can be used to model transformation from one to another phase of a sequential model development.
- trace specifies a trace relationship among model elements that represent the same concept in different models. It is mainly used for tracking changes across models.

ARD is based on the concept of the functional dependency among properties. A simple property is described by a single attribute, while a complex property is described by multiple attributes. ARD supports two kinds of attributes: conceptual (describing some general, abstract aspect of the system to be specified and refined) and physical (describing a well-defined, atomic aspect of the system). To show the dependency relation among properties in ARD, the corresponding UML component diagram uses the UML derive dependency. The UML representation of the ARD model presented in Fig. 1 is shown in Fig. 2.

In the hierarchical ARD design process two types of transformations are used: finalization and split. Finalization introduces a more specific knowledge about the given property through transforming a simple property described by a conceptual attribute into a property described by one or more conceptual or physical attributes. In Fig. 3 an example of finalization (in ARD) and corresponding UML component diagrams for this transformation. In the same figure the complex property representation using a subsystem artifact can be observed.
Finding an appropriate UML representation for XTT is not a trivial task. Both the semantical aspect, as well as the visual scalability needs to be considered. In contrast to the PRR or URML, where a single rule is represented by number of classes corresponding to elements of the rule vocabulary (in a loose sense attributes in XTT), in the XTT representation in UML behavior (activity) diagrams are used, where single diagram corresponds to a single XTT table, that is a set of rules working in the same context. The rationale for this is that in XTT rules are considered to model the dynamics of the system structure prototyped with ARD.

UML activity diagrams are related to flow diagrams and can illustrate the activities taking place in the system, they include artifacts such as: Action, Decision node, Merge node, Fork node, Join node, Partitions (swimlanes), and Parameter of activity.

Several attempts to find an optimal representation where evaluated. Finally an optimal translation has been proposed, as presented below. For an in-depth discussion see (Nalepa and Kluza 2008). 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 sake of transparency the diagram can be divided into the partitions with a swimlane).
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 occurrences of the value in the XTT table.
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 a merge node.
4. Outputs of all actions are merged in a merge node and a control flow is lead to output parameter of activity.

The complete transformation of the Thermostat example has been described in detail in (Nalepa and Kluza 2008). Here only the diagram for the TH table (the second on the right in Fig. 1) is presented in Fig. 4 and the complete model in Fig. 5. In the second figure the activity diagrams for tables are nested to provide a visual scalability.

It is worth noting, that in general a counter-wise transformation could be considered. This would allow for UML-based XTT rule design in any standard-compliant UML editor. However, this is not possible without introducing some kind of special annotations in the UML model. Ultimately, an UML profile for XTT is considered as a solution.

**XTT Serialization**

In this phase the XTT rule base is automatically translated into the corresponding Java code. This translation allows for the source code level integration with other application components.

In case of rule models designed with REWERSE URML, or OMG PRR the number of classes may by much higher than the number of rules, with possibly multiple classes being generated for a single rule. The goal of the XTT serial-
Figure 5: Activity diagram for the whole thermostat

The design approach introduced in this paper, including the rule-modeling in UML is not a new problem, it is worth noting how the HeKatE approach compares to the existing solutions. Currently, two most important representations include OMG PRR and REWERSE URML (Lukichev and Wagner 2005). 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. Neither it aims at redefining some of the UML semantics by using a custom profile (such a profile could be considered for the means of bidirectional translation though). Instead, it tries to explore and efficiently use existing diagrams.

It also worth emphasizing, that while the XTT representation scales well in larger examples than the one presented here, its UML representation is not as efficient. In general from a modeling point of view, the XTT table provides a more compact representation than the activity diagram. The UML representation is considered in order to allow interoperability with UML modeling tools, as well as MOF-based description of XTT.

A custom method of XTT rules serialization to Java was also proposed. It allows to translate a rule-based logic designed and analyzed with XTT to a OO code that can be easily integrated on the source level with Java applications.

The design approach introduced in this paper, including the UML representation for rules enables a selective use of SE tools, e.g. UML editors, to design rules, so it simplifies the communication with SE engineers on the conceptual

Model Refinement and Verification

In the proposed approach the rule-based control logic is designed with XTT\textsuperscript{2} rules. XTT has a formal description, with formal verification framework, see (Ligeza and Nalepa 2005). Since the proposed OO XTT encoding does not introduce any semantic gap in the hierarchical HeKatE design process, the logic model can be refined using the ARD/XTT representation. This includes formal analysis of the rule base, including completeness and inconsistency checks.

Currently there are no plans to directly allow for the refinement of the UML representation. However, a full MOF definition of the ARD/XTT UML representations is being prepared. When these are ready, it should be possible to control the UML model refinement at the syntax level, possibly with additional OCL constraints. Then, using the backward translation from UML to XTT it would be possible to transform the refined UML model to the rule-based one.

### Evaluation and Future Work

The paper tackles problems of practical integration between software and knowledge engineering. The integration is addressed on two levels: design with UML, and implementation with a OO language. An attempt for developing a hybrid SE/KE methodology is presented. A custom, semantically equivalent UML representation, for ARD/XTT rules was discussed. The use of UML as possible knowledge representation for rule-based systems has been presented. The original contribution of the paper includes a UML-based representation of ARD diagrams that provide rule prototypes, and XTT diagrams describing a rule-based system.

Since the rule-modeling in UML is not a new problem, it is worth noting how the HeKatE approach compares to the existing solutions. Currently, two most important representations include OMG PRR and REWERSE URML (Lukichev and Wagner 2005). 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. Neither it aims at redefining some of the UML semantics by using a custom profile (such a profile could be considered for the means of bidirectional translation though). Instead, it tries to explore and efficiently use existing diagrams.

It also worth emphasizing, that while the XTT representation scales well in larger examples than the one presented here, its UML representation is not as efficient. In general from a modeling point of view, the XTT table provides a more compact representation than the activity diagram. The UML representation is considered in order to allow interoperability with UML modeling tools, as well as MOF-based description of XTT.

A custom method of XTT rules serialization to Java was also proposed. It allows to translate a rule-based logic designed and analyzed with XTT to a OO code that can be easily integrated on the source level with Java applications.

The design approach introduced in this paper, including the UML representation for rules enables a selective use of SE tools, e.g. UML editors, to design rules, so it simplifies the communication with SE engineers on the conceptual
level. Simultaneously, the logic is designed with KE methods with possible formal verification of the model. These are in fact the main reasons to provide such a UML-based representation. On the other hand, this is a non-standard approach, and uses non-standard methods, that are not directly compatible with the industry adopted OMG standards.

As for the related research it is worth noting that there are many efforts to extend and formalize the UML and the SE process. These include complex standards such as the MDA (Miller and Mukerji 2003), as well as some semi-formal process such as the Aspect-Oriented Programming, which tries to insert more process-related information into the UML model, e.g. (Suzuki and Yamamoto 1999).

The work presented in the paper is in progress. The proposed transformation and serialization algorithms are being implemented and tested. As for the UML representation several methods are considered, including an XSLT translation to the XMI format. In order to fully evaluate the algorithm a formalized description will be ultimately provided. The current UML transformation closely follows both syntax and extended semantics of XTT, so it is not directly aimed at other rule formalisms. However, the approach for providing the transformation is a generic one, so in the future its application to different rule formats may be considered. Ultimately, the logical rule-based model designed with this method should be embedable into any business application using the MVC pattern with interfaces (View) with a hybrid Controller (Burbeck 1992). This would provide a bridging methodology for the classic software and knowledge engineering methods and tools.

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

References
Burbeck, S. 1992. Applications programming in smalltalk-

80(tm): How to use model-view-controller (mvc). Technical report, Department of Computer Science, University of Illinois, Urbana-Champaign.

Perspectives of System Informatics, Novosibirsk, Russia, June 2006, LNCS. Springer.