XTT Rules Design and Implementation with Object-Oriented Methods

Unknown Anonymous
Nowhere University
of UnApplied but Costly Research
donotwriteme@nowhere

Abstract

In the 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 rulebase is then transformed to UML behavioral diagrams, which can be considered a visual encoding. The rule implementation involves the serialization to Java (or other OOP) language using classes representing the decision tables grouping rules.

Introduction

In Artificial Intelligence (Russell and Norvig 2003) system modeling consist in 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 (KE) is often a gradual process that uses number of methods from ones close to natural language, to formalized representations such as rules or ontologies.

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 with KE problems is often of limited use. One of the issues in this integration is how to align modeling tools used in these domains. While is 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 the 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. Extended Tabular Trees (XTT for short) (**?**) is a structured knowledge representation for rules, based on some classic KE notions of decision tables and decision trees. The XTT rulebase is then transformed to UML behavioral diagrams, which can be considered a visual encoding. Representing XTT with UML artifacts encounters number of issues addressed in the paper. A new algorithm for encoding an XTT diagram using UML is. The rule implementation involves the serialization to Java or other OO language using classes representing the decision tables grouping rules. The approach presented in the paper is aimed at overcoming the semantic gap present in the classic software engineering approach.

The rest of the paper is organized as follows. The second section of the paper presents the motivation for the research, including the development of knowledge-based methods for software design; a brief introduction to the challenges of using Object-Oriented (OO) modeling with knowledge engineering methods are also outlined.

Then, in the third section the principles of the HeKatE methodology. In the nexts section the UML representation details are given. The sixth section is dedicated to the presentation of the OO serialization. The next section discusses selected issues of an on-line model verification in the given approach. The last section gives an evaluation of the approach, and outlines the future work, including bidirectional UML translation.

Motivation

Knowledge engineering and representation methods provide transparent and declarative design methods that are a powerful tool for expressing the application logic. However, today applications rely on complex and heterogeneous applications stacks and middlewares (e.g. the Java EE stack). Moreover, they require integration with number of high-level services including transparent network communications. These requirements are addressed by integrated design and prototyping environments, such as Eclipse. A low-level foundation of such environments is the use of standard design tools and implementation languages, such as UML, and OO languages, mainly Java. But when it comes to modeling the core application logic in an implementation-agnostic fash-

ion, without even considering the object-oriented perspective, these tools have some persistent problems. These issues 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) (Obj 2007) 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 I1 research project, where existing UML diagrams are used to model different type of rules.

It can be observed, that integrating knowledge representation and engineering with software engineering approach should be considered on two levels.

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 (or more recently also C#), on top of complex application stacks, e.g. Java EE (or .NET).

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 straight forward. 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 rule sets is 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 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. 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.

The paper proposes a UML representation for rules that preserves the *semantics* of the original rule representation and provides an OO encoding for rules that is *effective* (does not attempt to "translate" rule semantics to classes).

Heterogeneous Approach

The integration approach proposed in this paper is developed within the HeKatE project (`hekate.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 (**?**), a conceptual rule design method ARD+ (Attribute Relationship Diagrams (**?**; **?**)), that to some extent corresponds to requirements engineering in SE, logical system design with rules in the XTT^2 structure (**?**), 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 (**?**), 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

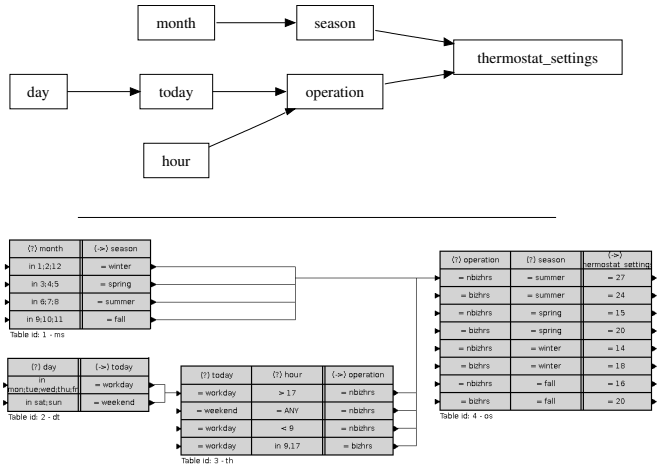

Figure 1: ARD prototye and XTT model

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 an 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 (Liebowitz 1998).

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^2 formally described with ALSV(FD), and visually represented by the structured XTT^2 rulebase. In this phase a corresponding *XTT 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^2 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 *XTT 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^2-based logic is integrated with the application in the *Model-View-Controller* paradigm (Burbeck 1992), providing a *logical application model*.

This approach aims at solving two problems present in the classic SE approach: the *semantic gap* between the design and the implementation (Mellor and Balcer 2002), as well as the *analysis specification gap* (Rash et al. 2005), which regards a difficulty with the transition from a specification to the design.

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

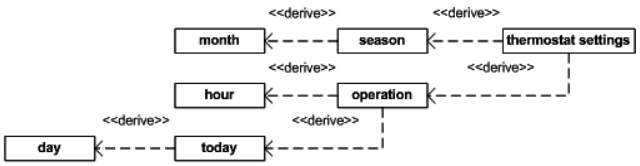

Figure 2: Representing ARD with component diagrams

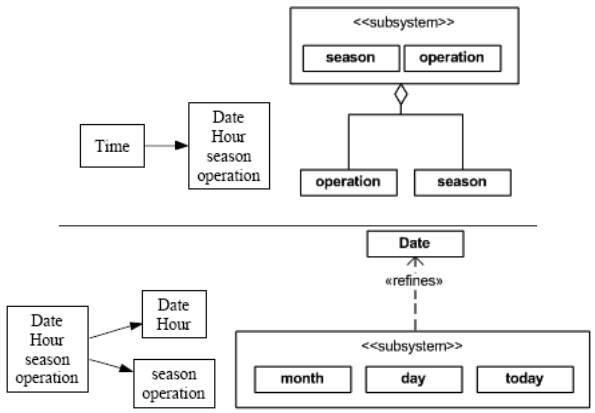

Figure 3: Types of transformation in TPH diagrams

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 shows 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.

XTT Representation in UML

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 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 a common context*. The rationale for this is, that in XTT rules are considered to model the dynamics of the XTT 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 (**?**). 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 times the value appears in 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 (**?**). 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 for 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.

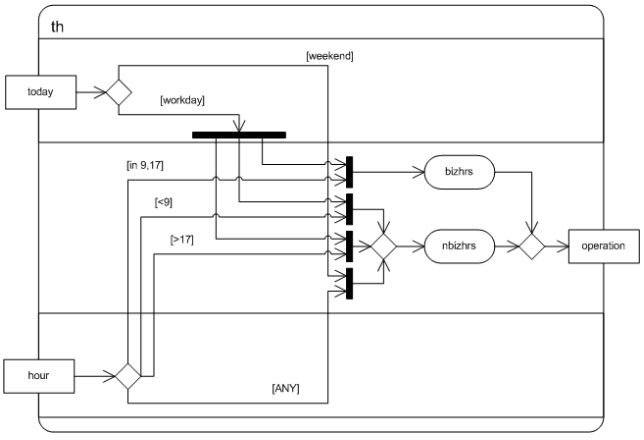

Figure 4: Activity diagram corresponding to XTT TH table

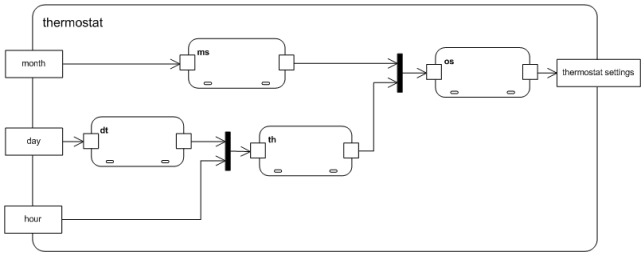

Figure 5: Activity diagram for the whole thermostat

XTT Serialization

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

In case of rule models designed with REWERSE URML, or OMG PRR 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 serialization was to optimize the translation process for simplicity and transparency (less classes). The simple idea was, to translate the XTT structure into classes on the table level. The versions of the approach are currently considered.

In the basic case the translation is as follows:

- a *class* `XTTable` is provided for XTT tables,
- one table is translated to one *object* of the class,
- XTT table attributes are translated as class *fields*,
- rules (table rows) are implemented as class *methods*,
- every method uses a simple `if` (or `case`) template for rule condition checking.

In XTT rules different attribute types are used. On the lower level they all map to two simple types called *numeric* (with optional scale to model floats)and *symbolic*. These are directly mapped to Java *Integer*, *Float*, and *String*.

In general, the XTT rulebase is composed of different types of tables, including special tables with no decision part (fact tables) and no condition parts (action tables). The decision part of the tables modifies attribute values, thus changing system state. In this part it is also possible to perform calculation, or call external methods that interface with the application View.

In order to simplify the implementation of this structure the extended case the translation differs:

- separate classes for conditional and decision parts of the XTT tables are considered,
- this separates the conditional part from state changing decision part, and
- simplifies implementation of different XTT table types (conditional, fact, etc.)

The serializer prototype is being implemented as a module for the XTT meta interpreter written in Prolog. The interpreter has a HMR parser, so it is easy to generate additional symbolic representation on the fly. The resulting code may be compiled with another classes providing the View and Controller. A more complex deployment solution with a Java based application server based on Tomcat is also planned in the future.

Model Refinement and Verification

In the proposed approach the rule-based control logic is designed with XTT^2 rules. XTT has a formal description, with formal verification framework, see (**?**). 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 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 (OMG br2003 09 03) 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 the 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 interoperation 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 the paper enables 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.

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 embeddable 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.

Warning Some self-citations have been removed, so they do not appear in the reference list.

References

Brachman, R., and Levesque, H. 2004. *Knowledge Representation and Reasoning*. Morgan Kaufmann, 1st edition.

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.

Connolly, T.; Begg, C.; and Strechan, A. 1999. *Database Systems, A Practical Approach to Design, Implementation, and Management*. Addison-Wesley, 2nd edition.

Kluza, K. 2008. Metody inżynierii wiedzy – projekt: Uml and ard/xtt. AGH UST. Supervised by G. J. Nalepa, Ph. D.

Liebowitz, J., ed. 1998. *The Handbook of Applied Expert Systems*. Boca Raton: CRC Press.

Lukichev, S., and Wagner, G. 2005. Visual rules modeling. In *Sixth International Andrei Ershov Memorial Conference PERSPECTIVES OF SYSTEM INFORMATICS, Novosibirsk, Russia, June 2006*, LNCS. Springer.

Mellor, S. J., and Balcer, M. 2002. *Executable UML: A Foundation for Model-Driven Architectures*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Miller, J., and Mukerji, J. 2003. *MDA Guide Version 1.0.1*. OMG.

Negnevitsky, M. 2002. *Artificial Intelligence. A Guide to Intelligent Systems*. Harlow, England; London; New York: Addison-Wesley. ISBN 0-201-71159-1.

Object Management Group. 2007. *Production Rule Representation (PRR), Beta 1*.

OMG. 1997. Uml notation guide version 1.1. Technical report, Object Management Group.

OMG. br/2003-09-03. Production rule representation. Technical report, Object Management Group.

Rash, J. L.; Hinchey, M. G.; Rouff, C. A.; Gracanin, D.; and Erickson, J. 2005. A tool for requirements-based programming. In *Integrated Design and Process Technology, IDPT-2005*. Society for Design and Process Science.

Ross, R. G. 2003. *Principles of the Business Rule Approach*. Addison-Wesley Professional, 1 edition.

Russell, S., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 2nd edition.

Sommerville, I. 2004. *Software Engineering*. International Computer Science. Pearson Education Limited, 7th edition.

Suzuki, J., and Yamamoto, Y. 1999. Extending uml with aspects: Aspect support in the design phase. In Moreira, A. M. D., and Demeyer, S., eds., *ECOOP Workshops*, volume 1743 of *Lecture Notes in Computer Science*, 299–300. Springer.

van Harmelen, F.; Lifschitz, V.; and Porter, B., eds. 2007. *Handbook of Knowledge Representation*. Elsevier Science.