MOF-based Metamodeling for the XTT Knowledge Representation

Krzysztof Kluza, Grzegorz J. Nalepa

Institute of Automatics,
AGH – University of Science and Technology,
Al. Mickiewicza 30, 30-059 Kraków, Poland
Institute of Physics,
Jan Kochanowski University
ul. Żeromskiego 5, 25-369, Kielce, Poland
kluza@agh.edu.pl, gjn@agh.edu.pl

Abstract. *This paper concerns the metamodel of the XTT rule-based knowledge representation. XTT is a knowledge representation and design method which aims at combining decision trees and decision tables. As the pure XTT method required dedicated tools to design systems, the UML representation has been developed. The abstract syntax of UML is defined by the UML metamodel. This paper introduces the metamodel proposal for the developed UML representation.*

1. Introduction Nowadays, business requirements change very fast and the product lifecycle is shortening. Moreover, the complexity of developing applications is continually growing. To deal with the complex software production, visual modeling methods have been developed.

Visual modeling is already an essential part of a software engineering process [16]. In software modeling the Unified Modeling Language (UML) [15], a graphical modeling language, has become the dominant graphical notation.

UML can help to enable the model-driven approach to system development [12]. This can be an evolutionary step in the software development industry. Although there are no restrictions to use only UML models, UML has some great strengths, such as:

- providing a metamodel,
- allowing to define model extensions,
- allowing to raise the modeling abstraction.

In the model-based approach metamodels play an increasingly important role. Metamodeling is to be an essential foundation for Model Driven Development (MDD) [1]. However, there is still little consensus on the precise form it should take in the process.

The general problem considered in the article is modeling and metamodeling in the software engineering process. This article discusses UML models and MOF metamodels, provides the description of the Hybrid Knowledge Engineering (HeKatE) methodology and the UML models of the XTT Knowledge Representation in HeKatE process. The specific problem presented in the article is the metamodel for the UML representation of the XTT.

The research presented here is the continuation of work previously carried out within [9, 8].

2. Models and Metamodels

2.1. Model Modeling plays a crucial role in a system design process. The feature of a model is that it can be used to produce things existing in reality, e.g. a model describing a system can be used to produce similar systems [6]. Thus, a model is an abstraction of the reality which can be treated as a pattern or template. To model is to represent the real world's features, such as objects, systems or concepts [17].

Models are described in a particular language. In UML, a model is a set of diagrams [6]. Such diagrams describe the system (or only a part of it). The complete system can be described by a number of models. Each one describes the system from a different angle than another, often on another level of abstraction.

2.2. Metamodel A language can be defined by the description of its syntax and semantics. To describe the semantics often a textual description is used. The syntax, in turn, is defined rather in other ways, e.g. grammar.

Visual languages, like UML, have a graphical syntax. Therefore, they require a visual mechanism for defining their syntax [6]. Although the syntax of the visual language can be described by some complex graph grammar, the easier way is to use a metamodel, especially in the case of UML where the metamodel is already provided. UML provides also the notation, which is supported by the UML tools.

Not every syntactical rule can be explicitly expressed in the metamodel, e.g. the rule specifying the order of the particular artifacts. However, such a rule can be easily expressed in the constraint language such as OCL [14].

2.3. Approaches to Metamodeling Two approaches of creating a metamodel can be distinguished. The first approach consists in creating a new metamodel conforming to MOF.

The second one is based on redefining and adjusting the UML metamodel without leaving the standard of UML. This can use mechanisms extending UML metamodel such as:

- *the UML profiles* which provide a mechanism for customizing the metamodel for particular domains or platforms [3], using e.g. stereotypes, or
- *constraints* which can provide a detailed specification, written as Object Constraints Language (OCL) [14] expressions which impose on the model. OCL is a formal language developed in order to avoid ambiguous constraint expressions.

2.4. MOF metamodel Meta Object Facility (MOF) [13] is an OMG standard that defines the language to define modeling languages. It is a universal way of describing modeling constructs [5]. MOF is defined using MOF itself. So, MOF is at the highest abstraction level [4]. A fragment of the MOF metamodel is shown in Fig. 1, see [2].

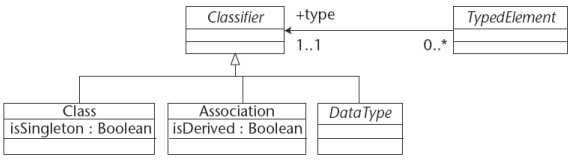

Fig. 1. A fragment of the MOF metamodel abstract syntax (for MOF 1.4) [2]

Since the MOF syntax is based on UML class diagrams, artifacts can be modeled as classes and their properties as attributes of the class. The relationships between artifacts can be modeled as associations between classes representing these artifacts. Furthermore, to increase precision of the metamodel one can use OCL expressions [5].

OMG defined a four-layered architecture for MOF: a meta-metamodel layer, a metamodel layer, a model layer, and an information layer. Fig. 2 captures relationships between these layers [1].

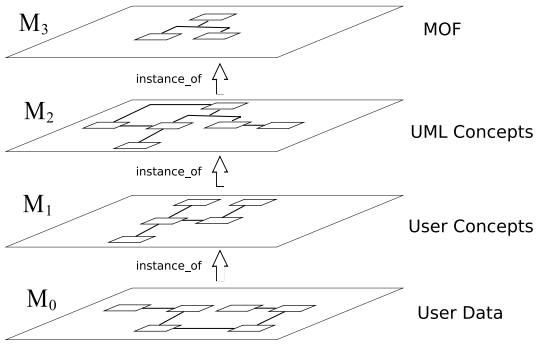

Fig. 2. Traditional OMG Modeling Infrastructure [1]

The MOF meta-metamodel is an abstract language used to define metamodels. For instance, to define the UML metamodel. A metamodel can be described as an instance of the MOF meta-metamodel. It is a language used to define models. Models, in turn, are comprised of metadata that describe data in the information layer. Each element in a certain layer describes elements in the layer below.

3. MOF-based Metamodeling for the XTT Knowledge Representation

3.1. XTT EXtended Tabular Trees (XTT) [7] is a hybrid knowledge representation and design method aims at combining decision trees and decision tables. It is developed in HeKatE research project (`hekate.ia.agh.edu.pl`) that regards Software Engineering based on Knowledge Engineering.

XTT uses a hierarchical visual representation of the decision tables linked into a tree-like structure. It is used to model, represent, and store the business logic of designed systems. Moreover, the XTT method is supported by a Prolog-based interpretation.

XTT method requires dedicated tools to design systems. However, in Software Engineering UML is de facto the standard for modeling software applications. The UML representation of XTT have been proposed to provide users a wider choice of tools and integrate XTT method with the most popular visual notation. As XTT methodology is a knowledge-based approach and uses a declarative model specification, UML is also a declarative language, so this provides compatibility for translations.

The HeKatE process consist of three design phases:

1. *Conceptual design*. Conceptual design is the most abstract phase. During this phase both system attributes and their functional relationships are identified. This phase uses ARD (Attribute-Relationship) [11] diagrams as the modeling tool. ARD method uses a visual representation to specify relationships between attributes. It allows to design the logical XTT structure.
2. *Logical design*. In this phase system structure is represented as a XTT hierarchy. The preliminary model of XTT can be obtained as a result of the previous phase. This phase uses the XTT representation as the design tool. The logical design phase allows on-line analysis, verification, and also revision and optimization (if necessary) of the designed system properties, using Prolog.
3. *Physical design*. During the physical design phase the preliminary Prolog-based implementation of the system is generated from the XTT model. The generated code can be compiled, executed and debugged.

Fig. 3 shows this top-down hierarchical design methodology of the HeKatE project, Knowledge Representation Diagrams for each phase and corresponding UML-based Knowledge Representations which are described in next section.

3.2. UML-based Knowledge Representation for XTT The models of ARD and XTT use the subsets of UML artifacts and their relationships. The reason of this is to be able to use existing UML tools.

The key assumption of ARD design is that the attributes are functionally dependent. ARD allows for specification of dependencies using a visual representation [11]. The UML model of ARD is very similar to the original ARD. The representation uses class diagrams and stereotyped dependencies from Standard Profile L2 to convey ARD.

As the decision tables of XTT are processed sequentially, their representation uses UML activity diagrams, which shows not so much the structure of the system, but its behavior. In general, activity diagrams are related to flow diagrams and can illustrate the activities taking place in the system. Every XTT attribute is represented by the Activity Parameter. There are input and output parameters. The whole Activity acts like a logical gates system, where Merge and Join Nodes function as logical *or* and *and* gates. Values of attributes are checked in guard conditions by the Decision Nodes, and values of output parameters are set by Actions. Although the model is not similar to the original XTT, it is intuitive when knowing the logical gates.

3.3. MOF Metamodel of the UML Model of XTT The UML representation of ARD and XTT uses the second approach of creating a metamodel – redefining and adjusting the UML metamodel. Therefore, the proposed metamodel of XTT is a part of the UML metamodel.

Fig. 4 shows the metamodel of the UML representation for ARD diagrams. The representation is very simple and uses only UML classes (with or without attributes) and the «derive» dependencies. The presented metamodel itself is not so pre-

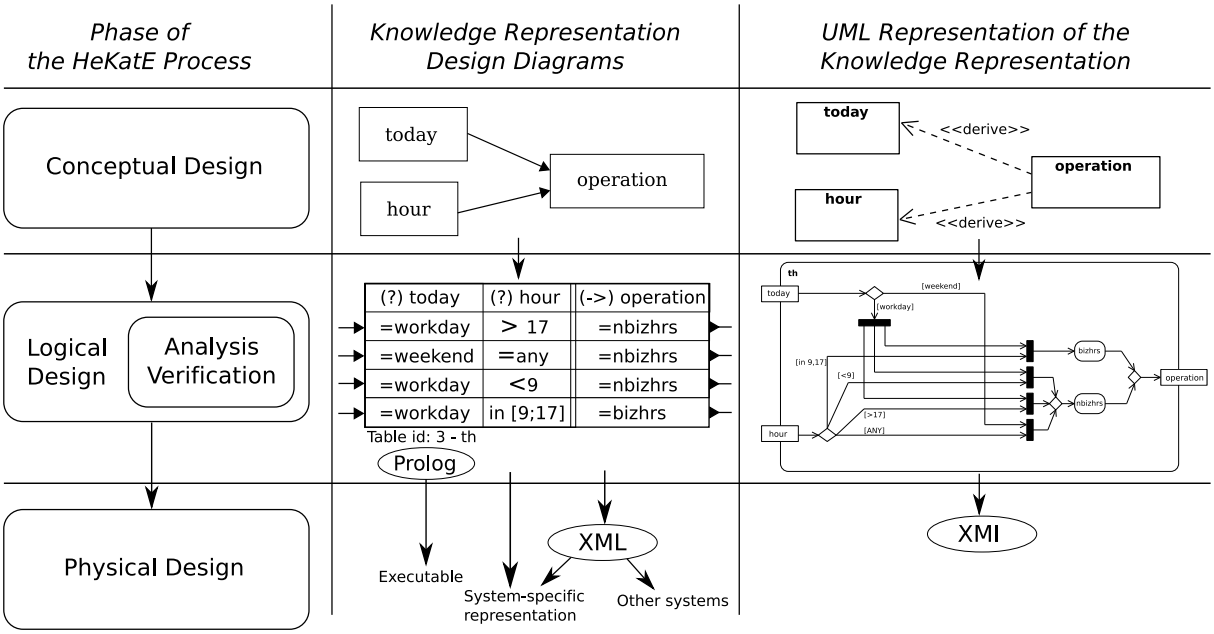

Fig. 3. HeKatE Methodology Phases and Knowledge Representation

cisely stated as required by the models. UML diagrams are typically not detailed enough to provide for every aspect of a specification. In particular, not every relevant aspect can be expressed in pure UML. To ensure the accuracy of the models, some constraints for the metamodel are provided, e.g. *every ARD simple property should have only one attribute* (that means that every class which is not abstract has no attribute). This constraint can be written in OCL as follows:

```
context Class inv:
 self.isAbstract = false implies
  self.ownedAttribute->size() = 0
```

As in the case of ARD, the XTT model uses a subset of UML artifacts and their relationships as well. There are two model levels of XTT:

- *the lower level model* conforming to a single XTT table, and
- *the higher level model* conforming to a tree of XTT tables.

As both these models are based on UML Activity Diagrams, their metamodel is created as a subset of UML Activity Diagram metamodel [15] with OCL constraints imposed. The proposed metamodel is shown in Fig. 5.

As can be observed in the metamodel, it also generates models which do not match to the XTT models. It is worth noting that the problem of this metamodel is that it does not require the order of nodes, despite the fact that the required order of nodes in XTT models is known. For example, the order for XTT model at the lower abstraction level is as follows (nodes which can occur optional are in brackets):

`Activity Parameter Node → Decision Node → (Fork Node) → Join Node`[1] `→ Fork Node`[2] `→ (Merge Node) → Action → (Merge Node) → Activity Parameter Node.`

The solution for the enforcement of the order of nodes is to use constraints expressions in OCL, e.g. the constraints for the activity parameter nodes can be described in natural language:

- *Incoming edge to every parameter node goes from either a merge node or an action.*
- *Outgoing edge from every parameter node goes to a decision node.*

These two rules can be written in OCL as follows:

```
context ActivityParameterNode inv:
```

[1] Element occurs when there is more than one input activity parameter node.

[2] Element occurs when there is more than one output activity parameter node or there is the transition parameter node (which enables the transition to different tables).

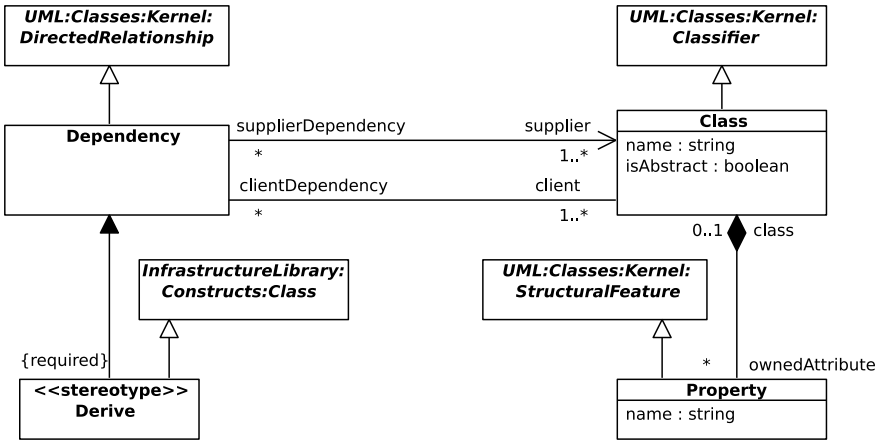

Fig. 4. Metamodel for ARD diagrams

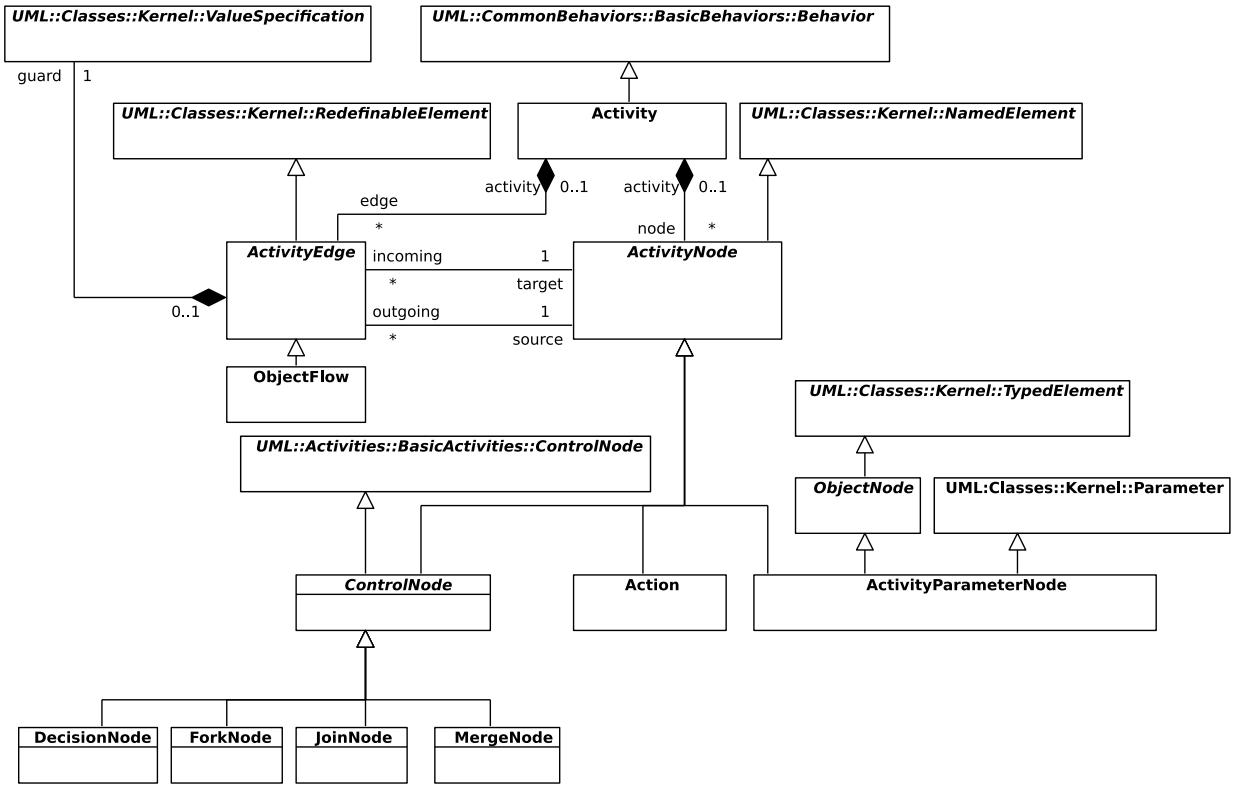

Fig. 5. Metamodel for XTT diagrams

```
self.incoming->forAll( edge |
 edge.source.oclIsTypeOf(MergeNode)
 xor edge.source.oclIsKindOf(Action)
)
self.outgoing->forAll( edge |
 edge.target.
     oclIsTypeOf(DecisionNode))
```

4. Summary In the paper the research in the field of knowledge and software engineering is presented. Metamodeling is considered as a challenging field for the future software development. Moreover, the OMG introduced their own MOF standard of defining metamodels. Therefore, the research presented in this paper aims at creating of MOF metamodels.

The original contribution of the paper consists in the creation of the metamodel for the UML representation of the XTT rule-based knowledge representation. The paper presents an approach based

on redefining and adjusting the UML metamodel. In this approach, the model is based on the original UML syntax and the metamodel is a subset of UML metamodel. In addition, some detailed specification is provided by using OCL expressions which impose on the model. Thus, the proposed metamodel conforms to the UML standard.

Future work will be focused on using the proposed metamodel. The metamodel defines the syntax of the visual language. However, it is not only a visual syntax presentation. The metamodel can be used for developing tools for validation of the model syntax. Validated models, serialized to XMI, can be transformed into a custom XTT representation. Then, such models can be interpret by using Prolog-based engine [10] and a prototype implementation of the system can be generated and run.

Moreover, the application of the metamodel can be wider when having different models which conform to the same metamodel. In this case, the metamodel can be used for translation from one model to another.

Currently, UML-based model of XTT and its metamodel has been proposed. The research in the field of metamodeling should be considered and possible metamodel applications should be investigated.

Acknowledgements The paper is supported by the HeKatE Project funded from 2007–2009 resources for science as a research project. The paper is carried out within the AGH UST Project No. 10.10.120.105.

References.

[1] C. Atkinson and T. Kühne. Model-driven development: A metamodeling foundation. *IEEE SOFTWARE*, 20:36–41, September/October 2003.

[2] D. S. Frankel. *Model Driven Architecture. Applying MDA to Enterprise Computing*. Wiley Publishing, Indianapolis, 2003.

[3] L. Fuentes and A. Vallecillo. An introduction to uml profiles. *UPGRADE*, 2:6–13, 2004.

[4] B. Henderson-Sellers, C. Atkinson, T. Kühne, and C. Gonzalez-Perez. Understanding metamodelling. Technical Report ER2003, October 2003.

[5] J. P. Ignizio. *An Introduction To Model Driven Architecture. Applying MDA to Enterprise Computing*. McGraw-Hill, 1991.

[6] A. Kleppe, J. Warmer, and W. Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison Wesley, 2003.

[7] G. J. Nalepa and A. Ligęza. A graphical tabular model for rule-based logic programming and verification. *Systems Science*, 31(2):89–95, 2005.

[8] Grzegorz J. Nalepa. Xtt rules design and implementation with object-oriented methods. In H. Chad Lane and Hans W. Guesgen, editors, *FLAIRS-22: Proceedings of the twenty-second international Florida Artificial Intelligence Research Society conference: 19–21 May 2009, Sanibel Island, Florida, USA*, 2009.

[9] Grzegorz J. Nalepa and Krzysztof Kluza. Uml representation proposal for xtt rule design method. In Grzegorz J. Nalepa and Joachim Baumeister, editors, *4th Workshop on Knowledge Engineering and Software Engineering (KESE2008) at the 32nd German conference on Artificial Intelligence: September 23, 2008, Kaiserslautern, Germany*, pages 31–42, Kaiserslautern, Germany, 2008.

[10] Grzegorz J. Nalepa, Antoni Ligęza, Krzysztof Kaczor, and Weronika T. Furmańska. Hekate rule runtime and design framework. In Gerd Wagner Adrian Giurca, Grzegorz J. Nalepa, editor, *Proceedings of the 3rd East European Workshop on Rule-Based Applications (RuleApps 2009) Cottbus, Germany, September 21, 2009*, pages 21–30, Cottbus, Germany, 2009.

[11] Grzegorz J. Nalepa and Igor Wojnicki. Towards formalization of ard+ conceptual design and refinement method. In *FLAIRS-21: Proceedings of the twenty-first international Florida Artificial Intelligence Research Society conference: 15–17 may 2008, Coconut Grove, Florida, USA*, pages 353–358, Menlo Park, California, 2008. AAAI Press.

[12] Object Management Group. *OMG: MDA Guide version 1.0.1*, 2003.

[13] Object Management Group. *OMG: Meta Object Facility (MOF) version 2.0, Core Specification*, 2006.

[14] Object Management Group. *OMG: Object Constraint Language version 2.0. Specification*, 2006.

[15] Object Management Group. *OMG: Unified Modeling Language (OMG UML) version 2.2. Superstructure*, 2009.

[16] I. Sommerville. *Software Engineering. 7th edn. International Computer Science*. Pearson Education Limited, 2004.

[17] J. P. van Gigch. *System Design Modeling and Metamodeling*. Plenum Press, New York, 1991.