Formal Semantics and Reasoning about UML Class Diagram

Marcin Szlenk
Warsaw University of Technology
Institute of Control & Computation Engineering
Nowowiejska 15/19, 00-665 Warsaw, Poland
M.Szlenk@ia.pw.edu.pl

Abstract

The main way of coping with the complexity of software systems is to construct and use models expressed in UML. Unfortunately, the semantics (meaning) of models written in UML is not precisely defined. It may result in the incorrect interpretation of a model and make it hard to strictly verify a model and its transformation. In this paper we formally (mathematically) define UML class diagram and its semantics. The problem of consistency of the diagram is then introduced and some examples of inconsistencies are forwarded.

1: Introduction

A modeling language is one of the fundamental tools used in the development process of a software system. Models hide irrelevant information about the system at the given stage of development, thus in a way reducing the complexity of the system. As the complexity of current systems is still increasing, the use of models in the development process becomes indispensable.

The more complex the software system is, the more difficult it becomes to ensure its quality properties like dependability or security. The use of models can help in dealing with the complexity, however questions arise about the quality of the model, itself [8]. Is the model correct? Is the model complete? Is the model consistent? It is impossible to fully answer these questions without the knowledge, what does the given model exactly mean and more generally, what the precise semantics of the modeling language expressions is.

1.1: Unified Modeling Language

Unified Modeling Language (UML) [1, 6] is a visual modeling language that is used to specify, construct and document software systems. It is important to note that it is a modeling language and *not* a method. It does not define nor advise the types of models which should be created and what steps should be taken to construct the software system. From a user's point of view, the UML can be roughly treated as a set of different types of diagram.

The UML has been adopted and standardized by the *Object Management Group* (OMG). The UML specification [1], published by OMG, is based on a *metamodeling* approach (see [2] for details about metamodeling). The metamodel (a model of UML) gives information about the abstract syntax of UML, but does not deal with semantics, this is expressed in a natural language. Furthermore, because the UML is method-independent, its specification rather sets a range of potential interpretations than an exact meaning.

1.2: Class diagram

A *class diagram* is the most fundamental and widely used UML diagram. It shows a static view of a system, consisting of classes, their interrelationships (including generalization/specialization, association, aggregation and composition), operations and attributes of the classes. The way the class diagram is drawn (the notation elements used and the level of detail) and interpreted, depends on the perspective taken. There are three different perspectives which can be used in drawing a class diagram [3, 5]:

1. The *conceptual perspective* — the diagram is interpreted as a description of concepts in the real world or domain being studied, regardless of the software that might implement them.
2. The *specification/design perspective* — the diagram is interpreted as a description of software abstractions or components with interfaces, but without commitment to a particular implementation.
3. The *implementation perspective* — the diagram is interpreted as a description of software implementation using a particular technology or language.

However, the above perspectives are not defined in the UML specification.

The class diagram which is made from the conceptual perspective is called a *conceptual class diagram*. The conceptual class diagram describes the most significant concepts (represented as classes) and relations in the problem domain (represented as relationships between classes). It is characterized by a low level of detail. The conceptual diagram does not specify operations of the classes. Although attributes of the classes may be specified, from the conceptual perspective, there is no difference between an attribute of the class and an association [3].

In this paper we formally define both the syntax and semantics of a conceptual class diagram (hereafter, the term 'class diagram' will be used) in the UML notation. The definitions which are presented here relate to the UML 2.0, which is the current official version.

2: Mathematical notation

As a language for defining the class diagram (so called *metalanguage*), we use basic mathematical notation. The advantage of this approach lies in the versatility and universality of mathematical notation. In this section the list and function notation, which may vary in different publications, is briefly outlined.

For a set A, $\mathcal{P}(A)$ denotes the set of all the subsets of A, and A^* denotes the set of all the finite lists of elements of A. The function $\mathbf{len}(l)$ returns the length of a list l. For simplicity, we add the expression $A^{*(2)}$, which denotes the set of all finite lists with a length of at least 2. The function $\pi_i(l)$ projects the i-th element of a list l, whereas the function $\overline{\pi}_i(l)$ projects all but the i-th element. The list $[a_1, \ldots, a_n]$ is formally equal to the tuple $(a_1, \ldots, a_n)$. For a finite set A, $|A|$ denotes the number of elements of A.

The partial function from A to B is denoted by $f\colon A \rightharpoonup B$, where the function $\mathbf{dom}(f)$ returns the domain of f. The expression $f\colon A \to B$ denotes the total function from A to B (in this case it holds $\mathbf{dom}(f) = A$).

3: The syntax of a class diagram

Graphical elements of a class diagram are shown in Fig. 1. In this section we formally define the

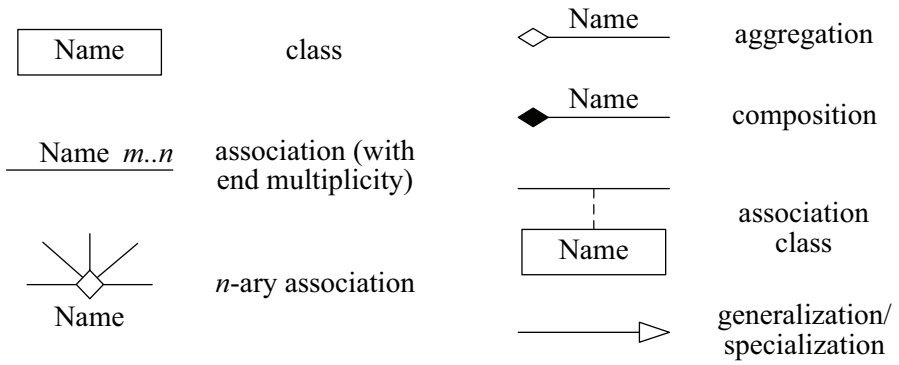

Figure 1. Elements of a class diagram

abstract syntax of the class diagram. The syntax is defined in a way which reflects the following semantic relationships between elements of the diagram: an association class is both a kind of association and a kind of class (it is a single model element [1, page 43]), an aggregation is a kind of association, and a composition is a kind of aggregation. To a large extent, it simplifies the definition of the semantics presented in Sec. 4.

Both a class, an association and an association class are called a *classifier*. If we let $\mathrm{Classifiers}$ denote the set of all classifiers (names) which may appear on a diagram, then by a class diagram, we understand a tuple

$$\mathcal{D} = (\mathrm{classes}, \mathrm{assocs}, \mathrm{ends}, \mathrm{mults}, \mathrm{assocs}_{\mathrm{agg}}, \mathrm{assocs}_{\mathrm{com}}, \mathrm{specs}) \text{ , where:}$$

1. $\mathcal{D}.\mathrm{classes}$ is a set of classes:
$$\mathcal{D}.\mathrm{classes} \subseteq \mathrm{Classifiers} \,. \tag{1}$$
2. $\mathcal{D}.\mathrm{assocs}$ is a set of associations:
$$\mathcal{D}.\mathrm{assocs} \subseteq \mathrm{Classifiers} \,. \tag{2}$$
For the diagram $\mathcal{D}$, a set of association classes and a set of all classifiers are thus respectively defined as:
$$\mathcal{D}.\mathrm{asclasses} =_{def} \mathcal{D}.\mathrm{classes} \cap \mathcal{D}.\mathrm{assocs} \,, \tag{3}$$
$$\mathcal{D}.\mathrm{classifiers} =_{def} \mathcal{D}.\mathrm{classes} \cup \mathcal{D}.\mathrm{assocs} \,. \tag{4}$$
3. $\mathcal{D}.\mathrm{ends}$ is a function of association ends. The function maps each association to a finite list of at least two, not necessarily different, classes participating in the association:
$$\mathcal{D}.\mathrm{ends}\colon \mathcal{D}.\mathrm{assocs} \to \mathcal{D}.\mathrm{classes}^{*(2)} \,. \tag{5}$$
The position on the list $\mathcal{D}.\mathrm{ends}(as)$ uniquely identifies the association end.
4. $\mathcal{D}.\mathrm{mults}$ is a function of multiplicity of association ends. Multiplicity is a non-empty set of non-negative integers with at least one value greater than zero. The default multiplicity is the set of all non-negative integers ($\mathbb{N}$). The function assigns to each association a list of multiplicity on its ends:
$$\mathcal{D}.\mathrm{mults}\colon \mathcal{D}.\mathrm{assocs} \to (\mathcal{P}(\mathbb{N}) \setminus \{\emptyset, \{0\}\})^{*(2)} \,. \tag{6}$$

As before, the position on the list $\mathcal{D}.\mathrm{mults}(as)$ identifies the association end. The multiplicity must be defined for each association end:

$$\forall as \in \mathcal{D}.\mathrm{assocs} \cdot \mathbf{len}(\mathcal{D}.\mathrm{mults}(as)) = \mathbf{len}(\mathcal{D}.\mathrm{ends}(as)) \; . \tag{7}$$

5. $\mathcal{D}.\mathrm{assocs}_{\mathrm{agg}}$ is a set of aggregations:

$$\mathcal{D}.\mathrm{assocs}_{\mathrm{agg}} \subseteq \mathcal{D}.\mathrm{assocs} \; . \tag{8}$$

Only binary associations can be aggregations [1, page 37]:

$$\forall as \in \mathcal{D}.\mathrm{assocs}_{\mathrm{agg}} \cdot \mathbf{len}(\mathcal{D}.\mathrm{ends}(as)) = 2 \; . \tag{9}$$

We assume that aggregate class (a class on the association end with a diamond adornment) is the first class on the list $\mathcal{D}.\mathrm{ends}(as)$.

6. $\mathcal{D}.\mathrm{assocs}_{\mathrm{com}}$ is a set of compositions:

$$\mathcal{D}.\mathrm{assocs}_{\mathrm{com}} \subseteq \mathcal{D}.\mathrm{assocs}_{\mathrm{agg}} \; . \tag{10}$$

7. $\mathcal{D}.\mathrm{specs}$ is a function of specializations. The function assigns to each classifier a set of all (direct or indirect) its specializations:

$$\mathcal{D}.\mathrm{specs} \colon \mathcal{D}.\mathrm{classifiers} \to \mathcal{P}(\mathcal{D}.\mathrm{classifiers}) \; . \tag{11}$$

The specialization hierarchy must be acyclical [1, page 49], what means that a classifier cannot be its own specialization:

$$\forall cf \in \mathcal{D}.\mathrm{classifiers} \cdot cf \notin \mathcal{D}.\mathrm{specs}(cf) \; . \tag{12}$$

4: The semantics of a class diagram

A classifier describes a set of *instances* that have something in common. An instance of a class is called an *object*, whereas an instance of an association is called a *link*. A link is a connection between two or more objects of the classes at corresponding positions in the association. An instance of a class association is both an object and a link, so it can both be connected by links and can connect objects.

4.1: Domain state

The existing instances of a classifier are called its *extent*. The classifier extent usually varies over time as objects and links may be created and destroyed. Thus, from a conceptual perspective, the classifiers' extents form a snapshot of the state of a problem domain at a particular point in time.

If we let $\mathrm{Instances}$ denote the *finite* set of instances that may come into existence in a problem domain, then a *domain state* (or shortly a *state*) is a pair

$$\mathcal{S} = (\mathrm{instances}, \mathrm{ends}) \; , \text{where:}$$

1. $\mathcal{S}.\mathrm{instances}$ is a partial function of extents. The function maps each classifier to a set of its instances (extent):

$$\mathcal{S}.\mathrm{instances} \colon \mathrm{Classifiers} \rightharpoonup \mathcal{P}(\mathrm{Instances}) \; . \tag{13}$$

2. $\mathcal{S}.\mathrm{ends}$ is a partial function of link ends. The function assigns to each instance of an association, i.e. link, a list of instances of classes (objects) which are connected by the link:

$$\mathcal{S}.\mathrm{ends}\colon \mathrm{Instances} \rightharpoonup \mathrm{Instances}^{*(2)} \ . \tag{14}$$

The position on the list uniquely identifies the link end, which on the other hand, corresponds to an appropriate association end.

4.2: The relation of satisfaction

The conceptual class diagram shows the structure of domain states or, from a different point of view, defines some constraints on domain states. Thus, the diagram can be interpreted as the set of all such states in which the mentioned constraints are satisfied. In this section we formally define what kind of constraints they are and what the word 'satisfied' means in this context.

If we let $\mathrm{Diagrams}$ be the set of all class diagrams as they were defined in Sec. 3 and let States be the set of all domain states as they were defined in Sec. 4.1, then for a given $\mathcal{S} \in \mathrm{States}$ and $\mathcal{D} \in \mathrm{Diagrams}$, we say that the diagram $\mathcal{D}$ is *satisfied* in the state $\mathcal{S}$ and we write

$$Sat(\mathcal{D}, \mathcal{S}) \text{ , if and only if:}$$

1. $\mathcal{S}$ specifies the extents of all classifiers in $\mathcal{D}$ (and maybe others, not depicted in the diagram $\mathcal{D}$):

$$\mathcal{D}.\mathrm{classifiers} \subseteq \mathbf{dom}(\mathcal{S}.\mathrm{instances}) \ . \tag{15}$$

2. An instance of a given association only connects instances of classes participating in this association (on the appropriate ends):

$$\begin{aligned} &\forall as \in \mathcal{D}.\mathrm{assocs} \cdot \forall ln \in \mathcal{S}.\mathrm{instances}(as) \cdot \\ &\quad \mathbf{len}(\mathcal{D}.\mathrm{ends}(as)) = \mathbf{len}(\mathcal{S}.\mathrm{ends}(ln)) \wedge \forall i \in \{1, \ldots, \mathbf{len}(\mathcal{D}.\mathrm{ends}(as))\} \cdot \\ &\quad\quad \pi_i(\mathcal{S}.\mathrm{ends}(ln)) \in \mathcal{S}.\mathrm{instances}(\pi_i(\mathcal{D}.\mathrm{ends}(as))) \ . \end{aligned} \tag{16}$$

3. Instances of an association satisfy the specification of multiplicity on all association ends[1]. For any $n-1$ ends of n-ary association ($n \geq 2$) and $n-1$ instances of classes on those ends, the number of links they form with instances of the class on the remaining end belong to the multiplicity of this end [1, pages 37–38]:

$$\begin{aligned} &\forall as \in \mathcal{D}.\mathrm{assocs} \cdot \forall i \in \{1, \ldots, \mathbf{len}(\mathcal{D}.\mathrm{ends}(as))\} \cdot \forall p \in \mathrm{product}(as, i) \cdot \\ &\quad |\{\ ln \in \mathcal{S}.\mathrm{instances}(as) : \overline{\pi}_i(\mathcal{S}.\mathrm{ends}(ln)) = p\ \}| \in \pi_i(\mathcal{D}.\mathrm{mults}(as)) \ , \end{aligned} \tag{17}$$

where:

$$\mathrm{product}(as, i) =_{def} \bigtimes_{j=1,\ j \neq i}^{\mathbf{len}(\mathcal{D}.\mathrm{ends}(as))} \mathcal{S}.\mathrm{instances}(\pi_j(\mathcal{D}.\mathrm{ends}(as))) \ . \tag{18}$$

4. An extent of an association includes, at most, one link connecting a given set of class instances (on given link ends)[2]:

$$\begin{aligned} &\forall as \in \mathcal{D}.\mathrm{assocs} \cdot \forall ln_1, ln_2 \in \mathcal{S}.\mathrm{instances}(as) \cdot ln_1 \neq ln_2 \Rightarrow \\ &\quad \exists i \in \{1, \ldots, \mathbf{len}(\mathcal{D}.\mathrm{ends}(as))\} \cdot \pi_i(\mathcal{S}.\mathrm{ends}(ln_1)) \neq \pi_i(\mathcal{S}.\mathrm{ends}(ln_2)) \ . \end{aligned} \tag{19}$$

[1] The meaning of multiplicity for an association with more than two ends was not precisely defined in UML prior to version 2.0. Possible interpretations are discussed in detail in [4].

[2] This condition does not have to be true for an association with a {bag} adornment. See [7] for details.

5. An aggregation relationship is transitive and asymmetric across all aggregation links, even from different aggregations [6]. That is, an object may not be directly or indirectly part of itself:

$$\forall ob \in \text{Instances} \cdot ob \notin \text{parts}(ob) \,, \tag{20}$$

where $\text{parts}(ob)$ determine the set of all parts of an object. Formally:

$$ob_2 \in_{def} \text{parts}(ob_1) \,, \tag{21}$$

if ob_2 is a direct part of ob_1:

$$\begin{aligned} &\exists as \in \mathcal{D}.\text{assocs}_{\text{agg}} \cdot \exists ln \in \mathcal{S}.\text{instances}(as) \cdot \\ &\quad ob_1 = \pi_1(\mathcal{S}.\text{ends}(ln)) \wedge ob_2 = \pi_2(\mathcal{S}.\text{ends}(ln)) \end{aligned}$$

or an indirect one, i.e. for a certain $n \geq 2$, it holds that:

$$\begin{aligned} &\exists as_1, \ldots, as_n \in \mathcal{D}.\text{assocs}_{\text{agg}} \cdot \\ &\quad \exists ln_1 \in \mathcal{S}.\text{instances}(as_1), \ldots, \exists ln_n \in \mathcal{S}.\text{instances}(as_n) \cdot \\ &\quad\quad ob_1 = \pi_1(\mathcal{S}.\text{ends}(ln_1)) \wedge ob_2 = \pi_2(\mathcal{S}.\text{ends}(ln_n)) \wedge \\ &\quad\quad \forall i \in \{1, \ldots, n-1\} \cdot \pi_2(\mathcal{S}.\text{ends}(ln_i)) = \pi_1(\mathcal{S}.\text{ends}(ln_{i+1})) \,. \end{aligned}$$

6. An object may be a direct part of only one composite object at a time. Precisely, only one composition link (across all composition links, even from different compositions) may exist at one time for a one part-object [6]:

$$\begin{aligned} &\forall as_1, as_2 \in \mathcal{D}.\text{assocs}_{\text{com}} \cdot \forall ln_1 \in \mathcal{S}.\text{instances}(as_1) \cdot \\ &\quad \forall ln_2 \in \mathcal{S}.\text{instances}(as_2) \cdot ln_1 \neq ln_2 \Rightarrow \pi_2(\mathcal{S}.\text{ends}(ln_1)) \neq \pi_2(\mathcal{S}.\text{ends}(ln_2)) \,. \end{aligned} \tag{22}$$

7. An instance of a specializing classifier is also an instance of the specialized classifier:

$$\begin{aligned} &\forall cf_1, cf_2 \in \mathcal{D}.\text{classifiers} \cdot \\ &\quad cf_2 \in \mathcal{D}.\text{specs}(cf_1) \Rightarrow \mathcal{S}.\text{instances}(cf_2) \subseteq \mathcal{S}.\text{instances}(cf_1) \,. \end{aligned} \tag{23}$$

4.3: The diagram's meaning

Now, using the satisfaction relation, the semantics of a class diagram can be formally defined. As stated earlier, the meaning of a class diagram from the conceptual perspective is the set of all domain states in which the diagram is satisfied. Let $\mathcal{M}\colon \text{Diagrams} \rightarrow \mathcal{P}(\text{States})$ be the function which is defined as:

$$\mathcal{M}(\mathcal{D}) =_{def} \{\, \mathcal{S} \in \text{States} : \mathit{Sat}(\mathcal{D}, \mathcal{S}) \,\} \,.$$

The value $\mathcal{M}(\mathcal{D})$ we call the *meaning* or *interpretation* of the diagram $\mathcal{D}$.

5: Consistency

As far as a model quality is concerned, *consistency* is one of the main criteria to be examined [8]. Generally, consistency is a measure of whether there are contradictions among the various diagrams within the model or between models produced at various stages of development. In the case of a class diagram, checking the consistency can also determine whether or not there are any internal conflicts within a single diagram. In this section, we outline the above problem by introducing a formal definition of classifier consistency.

5.1: The consistency of a classifier

If we let $\mathcal{D} \in \mathrm{Diagrams}$ and $cf \in \mathrm{Classifiers}$, then we can say that the classifier cf is *consistent* in the context of the diagram $\mathcal{D}$, if and only if:

$$\exists \mathcal{S} \in \mathcal{M}(\mathcal{D}) \cdot \mathcal{S}.\mathrm{instances}(cf) \neq \emptyset \ .$$

In other words, the diagram admits a domain state in which at least one instance of that classifier exists. Otherwise, the classifier is deemed to be inconsistent. Two examples of inconsistent classifiers are presented below.

5.2: Examples of inconsistencies

Let $\mathcal{D} \in \mathrm{Diagrams}$ includes the construction shown in Fig. 2a:

$$\begin{aligned} &AG \in \mathcal{D}.\mathrm{assocs}_{\mathrm{agg}} \ , \\ &\mathcal{D}.\mathrm{ends}(AG) = [A, A] \ , \\ &\mathcal{D}.\mathrm{mults}(AG) = [\mathbb{N}, 1] \ . \end{aligned}$$

In [7] it is proven that class A is inconsistent in the context of the diagram $\mathcal{D}$. The reason for the inconsistency is the multiplicity '1' on one of the aggregation ends. This multiplicity means that every object of A has exactly one part (which is also an object of A), thus from the transitivity and asymmetricity of the aggregation relationship, the objects of A must form the infinite *whole-part* chain, which is contrary to the fact that the extent of A is finite. The formal proof, however, is too extensive to be presented here in detail.

Now, let us consider the diagram in Fig. 2b. The inconsistency of the class A in this diagram can by shown in a more sophisticated way, using a simple property of consistency. If we let $\mathcal{D}_1, \mathcal{D}_2 \in \mathrm{Diagrams}$, then we say that the diagram $\mathcal{D}_2$ is a *consequence* of the diagram $\mathcal{D}_1$ and we write:

$$\mathcal{D}_1 \Rightarrow \mathcal{D}_2 \text{ , if and only if } \mathcal{M}(\mathcal{D}_1) \subseteq \mathcal{M}(\mathcal{D}_2) \ .$$

For the above definition, it is easy to prove the following theorem:

If the classifier cf is consistent in the context of the diagram $\mathcal{D}_1$ and it holds $\mathcal{D}_1 \Rightarrow \mathcal{D}_2$, then cf is also consistent in the context of $\mathcal{D}_2$.

In [7] we prove a set of transformation rules from one diagram into its consequences. By virtue of one of these rules, the implication shown in Fig. 3a-b holds. As stated earlier, the class A in

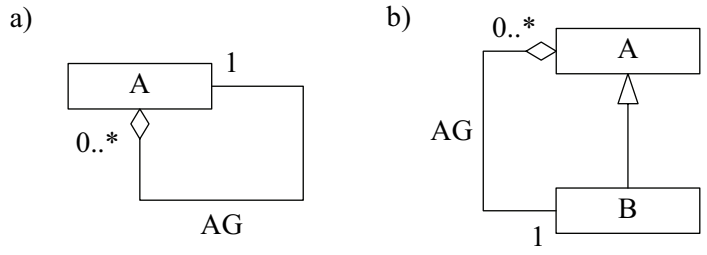

Figure 2. Examples of inconsistencies

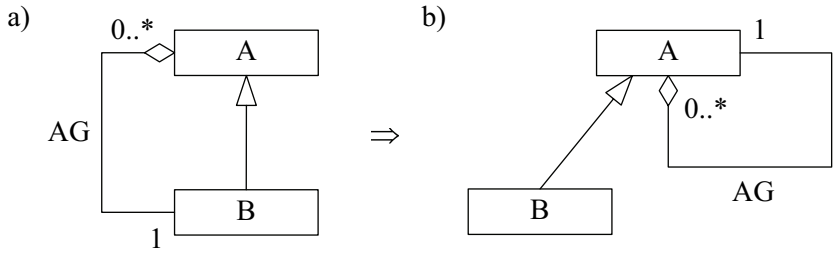

Figure 3. An example of a consequence

Fig. 2a (or Fig. 3b) is inconsistent, thus by virtue of the above theorem, A in Fig. 3a (or Fig. 2b) must be inconsistent, too. Note, that in all cases, due to the fact that the instances of class A cannot exist, neither the instances of the aggregation AG nor class B, itself, can exist. Thus, they are inconsistent as well.

6: Conclusion

The work presented here forms the formal foundation for the verification of a class diagram. The interpretation of particular elements of the diagram, as well as the interpretation of the whole diagram, have been precisely defined. The presented definitions take into a consideration the concepts which often cause interpretative difficulties like an aggregation/composition relationship or an n-ary association, thus allowing a better understanding of these concepts. Using the proposed diagram formalization, we have outlined the subject of reasoning about a class diagram, highlighting the possibility of the occurrence of internal inconsistencies in the diagram. Some interesting issues related to reasoning about a class diagram have only been briefly touched upon here (e.g. the problem of diagram transformations) and are presented in detail in [7], and the others are the subject of further investigation (e.g. the automatization of reasoning).

References

[1] *UML 2.0 Superstructure Specification (formal/05-07-04)*. Object Management Group, 2005.

[2] Tony Clark, Andy Evans, Paul Sammut, and James Willans. *Applied Metamodelling: A Foundation for Language Driven Development*. Xactium, 2004.

[3] Martin Fowler and Kendall Scott. *UML Distilled: A Brief Guide to the Standard Object Modeling Language, Second Edition*. Addison-Wesley, Boston, 2000.

[4] Gonzalo Génova, Juan Llorens, and Paloma Martínez. The meaning of multiplicity of n-ary association in UML. *Software and Systems Modeling*, 2(2):86–97, 2002.

[5] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process, Second Edition*. Prentice Hall, Englewood Cliffs, NJ, 2001.

[6] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual, Second Edition*. Addison-Wesley, 2004.

[7] Marcin Szlenk. *Formal Semantics and Reasoning about UML Conceptual Class Diagram (in Polish)*. PhD Thesis, Warsaw University of Technology, 2005.

[8] Bhuvan Unhelkar. *Verification and Validation for Quality of UML 2.0 Models*. Wiley-Interscience, 2005.