Open-World Loose Semantics of Class Diagrams as Basis for Semantic Differences

Imke Nachmann, Bernhard Rumpe, Max Stachon and Sebastian Stüber [1]

Abstract: Class Diagrams (CDs) model data structures in object-oriented systems and evolve throughout the course of the development process. Analyzing the semantic differences between consecutive versions of a CD is crucial to detect unintended changes of the modeled structures and involves comparing the sets of valid object models of both CDs. Established definitions of CD-semantics employ a closed-world assumption for the validity of object structures, which may not fit all stages of the development process. In this paper, we provide different definitions of CD-semantics and discuss their validity, relationship and limitations in the context of semantic evolution analysis. We show that the closed-world semantics of a CD is a subset of its open-world semantics and how this can be used for analyzing model-evolution. We also consider objects both as simple datastructures, as well as instances of (super-)classes and interfaces, and analyze how these approaches affect refinement and refactoring.

Keywords: Class Diagrams; Open-World Semantics; Closed-World Semantics; Semantic Differences

1 Introduction

Class Diagrams (CDs) are the most widely used Unified Modeling Language (UML)-Models in industry and research [DP06, La14, Hu11]. A CD describes the structure of an object-oriented system by means of classes, attributes and associations. In Model-Driven Engineering (MDE) CDs are used to generate code in general programming languages such as Java or C# [SN11]. Often CDs are used in combination with other UML models such as Statecharts or Sequence Diagrams [PSB11, SKM07, SS19, Sw12]. Furthermore, CDs can be used to generate graphical interfaces [Ge21].

As CDs are one of the primary artifacts in the development process, they are also frequently modified. Syntactic changes are made due to changing requirements, bug fixes, or when moving to a next design level. Effective change management is a major concern in MDE and not yet fully addressed. Nowadays, only syntactic differencing operators are widely established [AP03, KKT11, T.13, KGE09, Kü08, Ta14, TK09].

However, they do not show the impact on the model's semantics, i. e. its meaning [HR04]. The UML-Definition [Ma15] contains a syntactical description of CDs, but is vague on

[1] RWTH Aachen, Lehrstuhl für Software-Engineering, Ahornstraße 55, 52074 Aachen, Germany,
nachmann@se-rwth.de, rumpe@se-rwth.de, stachon@se-rwth.de, stueber@se-rwth.de
Gefördert durch die Deutsche Forschungsgemeinschaft (DFG) - Projektnummer 250902306

doi:10.18420/modellierung2022-016

the meaning of the diagrams. The semantics of a CD is the set of all object structures representing legal instances of the CD [MRR11a]. A semantic difference of two CDs is the set of all object structures representing a legal instance of the first and no legal instance of the second CD [MRR10, MRR11a]. The absence of such object structures implies that the first CD *refines* the latter [MRR10].

The *closed-world* assumption [Re78] requires legal instances to only instantiate classes and associations introduced in the CD (e. g. [BM13, MRR11a]). This becomes problematic during the analysis of systems, when the CDs are underspecified, as not all information is captured. Following analysis steps can add new classes or associations, which where not part of the initial analysis. In the closed-world semantics, this is not a refinement and leads to a semantic difference.

In contrast, the *open-world* assumption allows, whatever is not explicitly forbidden, i. e. legal instances may instantiate classes or associations not used in the CD as long as all CD constraints are satisfied. Adding new information (e. g. an additional class) leads to a refinement in open-world semantics.

Furthermore, in addition to the distinction between open-world and closed-world approaches, CD-semantics definitions may differ in their semantic domain, e. g., in [MRR11a], objects within an object structure are viewed simply as data structures and only instantiate a single class each. The superclasses and implemented interfaces of this instantiated class impact the semantics only indirectly via inherited attributes and associations. This definition of semantics is valid and useful, especially under a closed-world assumption, as it permits meaningful syntactical changes that do not change the semantics of a CD, as we will show in the following section. However, this approach becomes problematic when we consider the use of objects as method arguments. There, the instantiated superclasses and interfaces are relevant for type checking. In this situation, a semantic domain where objects are instances of multiple classes and interfaces is more appropriate.

This paper presents and compares open- and closed-worlds semantics definition for CDs with both single-instance objects, as well as multi-instance objects. We give examples that show the limitations of a closed-world approach and the utility of open-world CD-semantics for system-analysis in early developments stages.

In the remainder, Sect. 2 presents motivating examples, which demonstrate the limitations of closed-world semantics and motivates inheritance information in object structures. Sect. 3 introduces the abstract syntax of CDs, which is used in Sect. 4 to define multiple semantic mappings. These different semantics are compared in Sect. 5 and Sect. 6 debates related work. Finally, we conclude in Sect. 7.

2 Motivating Examples

Consider the CD $cd1$ in Fig. 1, it consists of the classes `Professor`, `Lecture` and `Student`. `Professors` hold at least one `Lecture` and each `Lecture` is held by exactly one `Professor`. A `Lecture` may be attended by multiple `Students` and a `Student` may attend multiple `Lectures`. We modify the CD by adding another class `Room`. Each `Lecture` is now required to take place in at least one $Room$ and each `Room` can host multiple $Lectures$. In early development this new CD $cd2$ would be understood as a refinement of $cd1$, since adding new elements simply adds new requirements on top of the already existing requirements regarding valid object structures. We would therefore consider the semantics of $cd2$ a subset of the semantics of $cd1$. This is the case under an open-world assumption. However, under a closed-world assumption the semantics of $cd1$ prohibits instances of any class not explicitly modeled in the CD. Thus, $om1$ is in the closed-world semantics of $cd1$ but not $cd2$, and $om2$ is in the closed-world semantics of $cd2$ but not $cd1$. We therefore say that the closed-world semantics of $cd1$ and $cd2$ are incomparable.

Now, consider the CDs $cd3.1$ and $cd3.2$ in Figure 2. If we view objects simply as data-structures that instantiate a single class, these two CDs would be semantically equivalent under a closed-world assumption. $cd3.1$ would be considered a refactoring of $cd3.2$ and vice-versa. If instead we consider a semantics definition that views objects as instances of multiple classes, we would find $cd3.1$ and $cd3.2$ to have incomparable semantics under a closed-world assumption, as $om3.1$ would be in the closed-world semantics of $cd3.1$ but not $cd3.2$ and $om3.2$ would be in the closed-world semantics of $cd3.2$ but not $cd3.1$. However, under an open-world assumption $cd3.1$ would be considered a strict refinement of $cd3.2$, as the open-world semantics of $cd3.1$ does not restrict instances of the class $Employee$.

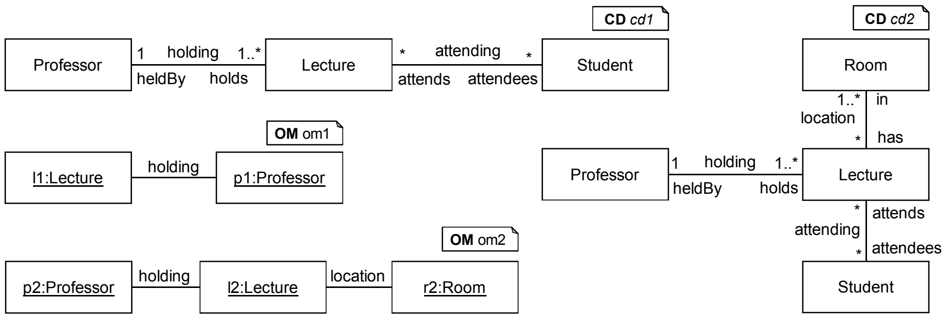

Fig. 1: Running examples of CDs and object structures

3 Class Diagrams and Object Structures

In the following, we use a variant of CDs which describes object structures in terms of classes, associations, roles, extends relations, and multiplicity constraints. An object

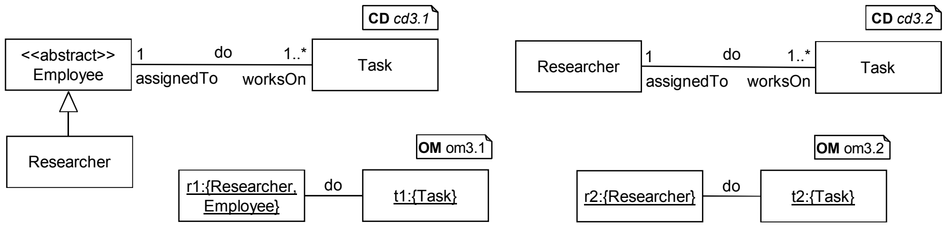

Fig. 2: Examples of CDs. Object structures include inheritance information

structure represents a possible data state of a system by means of objects and links between them [Ru16]. Those object structures that satisfy all constraints prescribed by a CD, are the elements of the CD's semantics.

Notation. Let A be a set. Then, $|A|$ denotes the cardinality of the set A. Further, $\wp_2(A)$ denotes the set of two-element subsets of A and $\wp_{\mathrm{fin}}(A)$ denotes the set containing all non-empty finite subsets of A. We denote the set of natural numbers including 0 by $\mathbb{N}_0$, and the set of natural numbers including 0 and the infinity symbol by $\mathbb{N}_\infty$. The relation $\leq$ is the natural total ordering on this set. Given a binary relation R, we denote the reflexive transitive hull of that relation by R^*.

3.1 An Abstract Syntax for Class Diagrams

From now on, let C^U be a universe of class names, $\mathcal{R}^U$ be a universe of role names, and $\mathcal{A}^U$ be a universe of association names. The following function assigns two roles to every association. We assume that associations describe exactly two roles. Roles are not shared among different associations. To this effect, $rls : \mathcal{A}^U \to \wp_2(\mathcal{R}^U)$ maps each association to its roles. Since roles are not shared, we require $rls(a_1) \cap rls(a_2) = \emptyset$ for every two associations $a_1, a_2 \in \mathcal{A}^U$ with $a_1 \neq a_2$. The assumption that role names are not shared does not imply a loss of generality, as, e. g. prepending the association's name to every role name of the association yields the required property. For example, consider the CD $cd3.1$ in Figure 2 and its association $\texttt{do} \in \mathcal{A}^U$. The two roles of $\texttt{do}$ are $rls(\texttt{do}) = \{\texttt{assignedTo}, \texttt{worksOn}\}$. A CD consists of a finite set of classes, a finite set of associations with multiplicity constraints on both association ends, and a finite set of extends relations between the classes. The CD assigns the roles of each associations a unique class. We need not consider attributes, as they can be simulated by corresponding 1-to-1 associations. Similarly, interfaces are not explicitly regarded, as they would be semantically equivalent to abstract classes in our semantics definitions. Finally, our syntax and semantics definitions omit methods, since they require behavioural analysis, and we are primarily concerned with object structures.

Definition 3.1 (Class Diagram) *A class diagram is a tuple $cd = (C, \mathcal{A}bs, \mathcal{A}, cls, \mathcal{M}, \mathcal{H})$, where **(1)** $C \subseteq C^U$ is a finite set of classes, **(2)** $\mathcal{A}bs \subseteq C$ is a set of abstract classes, **(3)** $\mathcal{A} \subseteq \mathcal{A}^U$ is a finite set of associations, **(4)** the function $cls : rls(\mathcal{A}) \to C$ assigns each role of each association a unique class, **(5)** C contains all classes of all roles of all associations, i. e. for all associations $a \in \mathcal{A}$ and all roles $r \in rls(a)$ of a, it holds that $cls(r) \in C$, **(6)** $\mathcal{M} = \{min_a^{cd}, max_a^{cd} \mid a \in \mathcal{A}\}$ is a finite set containing a minimum multiplicity constraint $min_a^{cd} : rls(a) \to \mathbb{N}_0$ and a maximum multiplicity constraint $max_a^{cd} : rls(a) \to \mathbb{N}_\infty$ for each association $a \in \mathcal{A}$, **(7)** $\mathcal{H} \subseteq \{c_1 \preccurlyeq c_2 \mid c_1, c_2 \in C\}$ is a finite set of extends relations between the classes in C such that **(8)** $\mathcal{H}^*$ is a partial ordering on C, **(9)** and for any $c_1 \preccurlyeq c_2 \in \mathcal{H}$, it holds that $c_1 \in \mathcal{A}bs \implies c_2 \in \mathcal{A}bs$.*

In a CD $cd = (C, \mathcal{A}bs, \mathcal{A}, cls, \mathcal{M}, \mathcal{H})$, for each association $a \in \mathcal{A}$, and each role $r \in rls(a)$ of the association, $min_a^{cd}(r) \in \mathbb{N}_0$ defines the minimum multiplicity of the class $cls(r)$ with role r required by association a in the CD cd. Similarly, $max_a^{cd}(r) \in \mathbb{N}_\infty$ defines the maximum multiplicity of the class $cls(r)$ with role r required by the association a in the CD cd. For instance, the abstract syntax of the CD $cd3.1$ depicted in Figure 2 can be formalized as $cd3.1 = (C, \mathcal{A}bs, \mathcal{A}, cls, \mathcal{M}, \mathcal{H})$, where $C = \{\texttt{Employe}, \texttt{Task}, \texttt{Researcher}\}$, $\mathcal{A}bs = \{Employee\}$, $\mathcal{A} = \{\texttt{do}\}$, $cls = \{\texttt{assignedTo} \mapsto \texttt{Employee}, \texttt{worksOn} \mapsto \texttt{Task}\}$, $\mathcal{M} = \{min_{\texttt{do}}^{cd3.1}, max_{\texttt{do}}^{cd3.1}\}$, and $\mathcal{H} = \{\texttt{Researcher} \preccurlyeq \texttt{Employee}\}$. The multiplicity constraints in $\mathcal{M}$ are given by the functions $min_{\texttt{do}}^{cd1} = \{\texttt{assignedTo} \mapsto 1, \texttt{worksOn} \mapsto 1\}$, $max_{\texttt{do}}^{cd1} = \{\texttt{assignedTo} \mapsto 1, \texttt{worksOn} \mapsto \infty\}$.

3.2 Strict Expansion of CDs

We now define the notion of a strict expansion of a class diagram that will serve as the basis for an open-world semantic definition in Sect. 4.2. Under an open-world assumption, CDs are considered underspecified. Adding a new class, association or extends relation to a CD should therefore be considered a refinement, since we are specifying previously underspecified parts of the CD. This should also be the case for moving an association to a superclass and changing a non-abstract class into an abstract class.

Definition 3.2 *We say that a class diagram $cd' = (C', \mathcal{A}bs', \mathcal{A}', cls', \mathcal{M}', \mathcal{H}')$ is a strict expansion of another class diagram $cd = (C, \mathcal{A}bs, \mathcal{A}, cls, \mathcal{M}, \mathcal{H})$ iff **(1)** $C \subseteq C'$, $\mathcal{A}bs \subseteq \mathcal{A}bs'$, $\mathcal{A} \subseteq \mathcal{A}'$, $\mathcal{M} \subseteq \mathcal{M}'$, $\mathcal{H} \subseteq \mathcal{H}'$ and **(2)** for all $a \in \mathcal{A}$ and $r \in rls(a)$, we find that $cls(r) \preccurlyeq cls'(r) \in (\mathcal{H}')^*$ Furthermore, we use $EXP(cd)$ to denote the set of all strict expansions of cd.*

3.3 Semantic Domain

Similar to [MRR11a], we define the semantics of a CD as the set of object structures, which contains all object structures permitted by the CD. The abstract syntax of object structures comprises objects and links between them. Unlike [MRR11a], we consider both a semantic domain based on single-instance objects, as well as one consisting of multi-instance objects. To this effect, let O^U denote a universe of objects. A *link* of type $a \in \mathcal{A}^U$ connects two objects via the roles $rls(a)$. For example, consider the object structure $om3.2$ depicted in Figure 2. Formally, the link connecting the `Researcher`-object `r2` to the `Task`-object `t2` is represented by the tuples $(\texttt{r2}, \texttt{worksOn}, \texttt{do}, \texttt{t2}, \texttt{assignedTo})$ and $(\texttt{t2}, \texttt{assignedTo}, \texttt{do}, \texttt{r2}, \texttt{worksOn})$. The two are necessary to obtain undirected links.

Definition 3.3 (Object Structure) *An object structure is a tuple* $om = (O, \text{class}, \mathcal{L})$*, where (1)* $O \subseteq O^U$ *is a finite set of objects, (2)* $\text{type} : O \to \wp_{\text{fin}}(C^U)$ *is a function that assigns a finite set of classes to each object. (3)* $\mathcal{L} \subseteq O \times \mathcal{R}^U \times \mathcal{A}^U \times O \times \mathcal{R}^U$ *is a finite set of links such that for all* $(o, r, a, o', r') \in \mathcal{L}$ *it holds that a)* $r, r' \in rls(a)$ *and* $r \neq r'$*, and b)* $(o', r', a, o, r) \in \mathcal{L}$*.*

The constraint (a) assures that the roles used by a link that instantiates an association $a \in \mathcal{A}^U$, correspond to the roles of the association. The constraint (b) assures undirectedness by requiring both link directions to be included in the set of links. For instance, the abstract syntax of the object structure $om3.1$ depicted in Figure 2 is given by $om3.1 = (O, \text{class}, \mathcal{L})$ where $O = \{\texttt{r1}, \texttt{t1}\}$, $\text{type} = \{\texttt{r1} \mapsto \{\texttt{Researcher}, \texttt{Employee}\}, \texttt{t1} \mapsto \{\texttt{Task}\}, \}$,
$\mathcal{L} = \{(\texttt{r1}, \texttt{worksOn}, \texttt{do}, \texttt{t1}, \texttt{assignedTo}), (\texttt{t1}, \texttt{assignedTo}, \texttt{do}, \texttt{r1}, \texttt{worksOn})\}$

Multiplicity constraints affiliated with an association restrict, for every object, the number of outgoing links of this association type. In an object structure om, the auxiliary function $\mathbb{L}^{om}$ captures these links: $\mathbb{L}^{om}(o, a, r) \overset{\text{def}}{=} \{(o, r, a, o', r') \in \mathcal{L} \mid o' \in O \land r' \in rls(a) \setminus \{r\}\}$.

Consider the object structure $om1$ in Figure 1. The set of links of association type `holding` connecting the object `l` via role `heldBy` to `Professor`-objects is $\mathbb{L}^{om2}(\texttt{l}, \texttt{holding}, \texttt{holds}) = \{(\texttt{l}, \texttt{heldBy}, \texttt{holding}, \texttt{p}, \texttt{holds})\}$. The set of `attending`-links connecting the object `l` to `Student`-objects is the empty set: $\mathbb{L}^{om2}(\texttt{l}, \texttt{attending}, \texttt{attends}) = emptyset$.

A *simple object structure om* $= (O, \text{class}, \mathcal{L})$ is an object structure with $|class(o)| = 1$ for all $o \in O$. We may use simple object structure as the semantic domain of a CD-semantics definition that does not consider the superclasses instantiated by an object. The term "simple" is in reference to simple graphs from graph theory.

4 Semantics of Class Diagrams

The semantics of a CD cd, denoted by $[\![cd]\!]$, is the set of all valid object structures. A constraint-based approach to semantics considers an object structure as valid with respect to a CD iff the object structure satisfies the constraints prescribed by the CD. The semantic mapping involves a definition of which constraints a CD prescribes and how an element of the semantic domain satisfies a constraint. Depending on the interpretation of what the elements of the semantic domain represent, these definitions may differ, which impacts the outcome of a semantic differencing operator, i. e. the set of valid instances of one CD that are not instances of another CD depends strongly on these definitions. This section formally defines a variety of open-world and closed-world semantics for CDs based on this notion.

4.1 Closed-World Semantics of Class Diagrams

This section introduces CD semantics that apply the closed-world assumption. Given a CD $cd = (C, \mathcal{A}bs, \mathcal{A}, cls, \mathcal{M}, \mathcal{H})$, an $om = (O, \text{type}, \mathcal{L})$ satisfies the closed-world assumption iff **CW-1** $\forall o \in O$ it holds that $\text{type}(o) \subseteq C$, **CW-2** $\forall (o, r, a, o', r') \in \mathcal{L} : a \in \mathcal{A}$ **CW-3** for all $o \in O$ it holds that $\text{type}(o) \nsubseteq \mathcal{A}bs$

That is, objects may only instantiate classes in the CD, links may only instantiate associations in the CD, and no object may instantiate only abstract classes. This section introduces two closed-world semantics of CDs that interpret object structures as simple data structures in which objects are considered instances of a single class. The other definition considers objects as instances of possibly many classes reflecting the notion that if an object instantiates a subclass, it also instantiates the superclass. Considering the semantic domain to represent simple datastructures with single-instance objects, the semantics of a CD only contains simple object structures.

Definition 4.1 (Simple Closed-World CD-Semantics) *Consider a CD in the sense above, i. e. $cd = (C, \mathcal{A}bs, \mathcal{A}, cls, \mathcal{M}, \mathcal{H})$. The* simple closed-world semantics*, denoted $[\![cd]\!]^{sc}$ is the set of all simple object structures $om = (O, \text{type}, \mathcal{L})$ such that **(1)** om satisfies the closed-world assumptions, i. e. CW-1 to CW-3. **(2)** for all $o \in O$ and $c_1, c_2 \in C$ such that $c_1 \in \text{type}(o)$, it holds that $c_1 \preccurlyeq c_2 \in \mathcal{H}^* \implies \forall a \in \mathcal{A}, (r, r') = rls(a)$, such that $cls(r') = c_2$, it holds that $min_a^{cd}(r) \leq |\mathbb{L}^{om}(o, a, r)| \leq max_a^{cd}(r)$. **(3)** for all $o \in O$ and $c_1, c_2 \in C$ such that $c_1 \in \text{type}(o)$, it holds that $c_1 \preccurlyeq c_2 \notin \mathcal{H}^* \implies \forall a \in \mathcal{A}, r \in rls(a), r' \in rls(a) \setminus \{r\}$, such that $cls(r') = c_2$, it holds that $\mathbb{L}^{om}(o, a, r) = \emptyset$*

The extends relations affect simple object structures only inderectly, i. e. objects and their links have to also satisfy the multiplicity constraints of inherited associations. The Conditions (2), and (3) formalize this: Consider an association a that a subclass c_2 inherits from its superclass c_1 in a CD. Then an object of type c_2 has to fulfill the multiplicity constraints of a, i. e. the number of outgoing links of type a is greater than or equal to $min_a^{cd}(c_2)$ and smaller than or equal to $max_a^{cd}(c_2)$, cf. Condition (2). On the other hand, an object that is not of a subclass type may not have outgoing links that instantiate associations of another class, cf. Condition (3). Simple object structures do not incorporate the notion that instances of a subclass type also instantiate the superclass type. When it comes to semantic differencing, this may yield results that are incorrect with respect to the understanding of the developer. Therefore, we define a closed-world semantics, that considers the types of an object to be a *set* of classes. Applying the closed-world semantics, the type of an object must include a class and exactly all of its superclasses as specified by the extends relations of the CD.

Definition 4.2 (Multi-Instance Closed-World CD-Semantics) *Consider the CD* $cd = (C, \mathcal{A}bs, \mathcal{A}, cls, \mathcal{M}, \mathcal{H})$. *The multi-instance closed-world semantics* $[\![cd]\!]^{mc}$ *is the set of all object structures* $om = (O, \text{type}, \mathcal{L})$ *such that* **(1)** *om satisfies the closed–world assumptions, i. e. CW-1 to CW-3.* **(2)** *for all* $c_1 \preccurlyeq c_2 \in \mathcal{H}, o \in O$ *it holds that* $c_1 \in \text{type}(o) \Rightarrow c_2 \in \text{type}(o)$, **(3)** *for all associations* $a \in \mathcal{A}$, *all roles* $r \in rls(a), r' \in rls(a) \setminus \{r\}$, *and all objects* $o \in O$ *such that* $cls(r') \in \text{type}(o)$, *it holds that* $min_a^{cd}(r) \leq |\mathbb{L}^{om}(o, a, r)| \leq max_a^{cd}(r)$. **(4)** *for all associations* $a \in \mathcal{A}$, *all roles* $r \in rls(a), r' \in rls(a) \setminus \{r\}$, *and all objects* $o \in O$ *such that* $cls(r') \notin \text{type}(o)$, *it holds that* $\mathbb{L}^{om}(o, a, r) = \emptyset$, *and* **(5)** $\forall o \in O$ *it holds that* $\forall c_1, c_2 \in \text{type}(o) : c_1 \preccurlyeq c_2 \in \mathcal{H}^*$ *or* $c_2 \preccurlyeq c_1 \in \mathcal{H}^*$.

In the above definition, we consider objects, whose type is a *set* of classes. Condition (2) states that objects whose class type includes a subclass must also include all superclasses according to the extends relation of the CD. Condition (3) assures that any object that instantiates a subclass that inherits an association from its superclass obeys the multiplicity constraints prescribed by that association, while Condition (4) assures that an object does not have outgoing links of an association that its class-type does not inherit. Condition (5) assures the correct instantiation of the class-hierarchy prescribed by the extends relations in the CD.

4.2 Open-World Semantics of Class Diagrams

The closed-world assumption reflects the notion that everything which is not modeled by the CD is not allowed. In contrast, the open-world assumption promotes that whatever is not modeled is not restricted and therefore allowed. Open-world semantics definitions therefore, do not restrict object structures to only instantiate elements from the CD. As for the closed-world, there are multiple ways to define open-world semantics. This section introduces two kinds of open-world semantics.

Since we have used a constraint-based approach for defining the closed-world semantics, we might approach the definition of an open-world semantics by simply removing/loosening some of these constraints.

Definition 4.3 (Loose CD-Semantics) *Let* $cd = (C, \mathcal{A}bs, \mathcal{A}, cls, \mathcal{M}, \mathcal{H})$ *be a CD in the above sense. The loose semantics* $[\![cd]\!]^{l}$ *is the set of all object structures* $om = (O, \text{type}, \mathcal{L})$

*such that **(1)** for all $o \in O$ it holds that* $\text{type}(o) \not\subseteq \mathcal{A}bs$ ***(2)** for all $c_1 \preccurlyeq c_2 \in \mathcal{H}, o \in O$ it holds that $c_1 \in$* $\text{type}(o) \Rightarrow c_2 \in \text{type}(o)$*, **(3)** for all associations $a \in \mathcal{A}$, all roles $r \in rls(a), r' \in rls(a) \setminus \{r\}$, and all objects $o \in O$ such that $cls(r') \in$* $\text{type}(o)$*, it holds that $min_a^{cd}(r) \leq |\mathbb{L}^{om}(o, a, r)| \leq max_a^{cd}(r)$.*

The loose semantics does not require objects and links to be of class, or association types that are elements of the CD. However, instantiating only abstract classes is not allowed (cf. Condition (1)). We consider objects to instantiate multiple classes, therefore the Condition (2) assures that objects of subclass type are also of superclass type. Condition (3) requires valid object structures to obey the multiplicity constraints prescribed by the CD.

For instance, the object structure $om3.1$ in Figure 2 is an element of the loose open-world semantics of $cd3.2$ as it obeys all extends relations and all multiplicity constraints of $cd3.2$. Because $cd3.2$ does not restrict the class hierarchy of the class `Researcher`, by the open-world assumption, objects of this type may also instantiate other classes. Since the class `Employee` is not an element of $cd3.2$, the CD does not pose any restrictions on the instantiation of this class. Therefore, objects whose class type includes `Employee` may exist and have outgoing links of the association type `do`.

An issue regarding the previous definition of open-world semantics is that, given a CD cd, there may be object structures in $[\![cd]\!]$ that cannot occur in the closed-world semantics of any CD. Consider, for example, the CD $cd4$ and the object structure $om4$ in Fig. 3: we find that $om \in [\![cd4]\!]^l$. However, there exists no CD $cd' = (C, \mathcal{A}bs, \mathcal{A}, cls, \mathcal{M}, \mathcal{H})$ such that $om \in [\![cd']\!]^{mc}$. This is because the object $o3$ instantiates both classes A and B, thus either $A \preccurlyeq B \in H^*$ or $A \preccurlyeq B \in H^*$ would have to hold, but since $o1$ and $o2$ only instantiate A and B respectively, $A \preccurlyeq B \notin H^*$ and $A \preccurlyeq B \notin H^*$ would have to hold, as well, which is impossible.

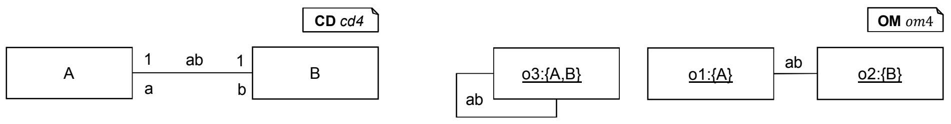

Fig. 3: object structure $om4$ in the loose semantics of CD $cd4$ is not in the closed-world semantics of any CD.

Under the open-world assumption a CD is considered underspecified. This notion implies that any object structure within the open-world semantics of a CD cd, should also be in the closed-world semantics of some CD cd' that is an (open-world) refinement of cd. We define a new open-world semantics accordingly:

Definition 4.4 (Expansion-Based Open-World CD-Semantics) *We define the simple* expansion-based *open-world semantics as*

$$[\![cd]\!]^{so} := \bigcup_{cd' \in EXP(cd)} [\![cd']\!]^{sc}.$$

Analogously, the multi-instance expansion-based *open-world semantics is defined as*

$$[\![cd]\!]^{mo} := \bigcup_{cd' \in EXP(cd)} [\![cd']\!]^{mc}.$$

An object structure that is an element of the expansion-based open-world semantics is therefore valid, iff there exists a strict expansion of the CD such that the object structure is a valid closed-world instance of this strict expansion. The notion of refinement requires that the properties of the elements of a CD also hold for every refining CD. The cd' in Definition 4.4 always refines the original cd, because for any $cd'' \in EXP(cd')$, it holds that $cd'' \in EXP(cd)$, as well, and thus any object structure in the semantics of cd' must also be in the semantics of cd''.

5 Comparison of Semantics with Regards to Semantic Differencing

CDs evolve naturally during the development process, and semantic evolution analyses, such as semantic differencing, allow a systematic tracking of these changes with respect to the modeled system. A successor version containing all previously defined classes and associations can be interpreted as a refinement of the predecessor version even if new classes or associations are added, as long as the constraints prescribed by the previous version are not contradicted. In this case, the addition corresponds to enhancing the model by further information about the system. During specification, semantic evolution analysis should interpret the addition of classes and associations as well as extends relations as the process of removing underspecification. The definition of an open-world CD semantics reflects this interpretation. In contrast, closed-world CD semantics are not (and do not need to be) compatible with the above interpretation. Using a closed-world semantics from above, each instance of a CD must not contain instances of classes and associations that are not used in the CD. Therefore, adding a class, an association, or an extends relation to a CD, results in a CD with a closed-world semantics that is incomparable to the closed-world semantics of the original CD. While this is a limitation in the context of semantic evolution analyses, it is no limitation, e. g. in the context of finite satisfiability, which is not concerned with comparing the semantics of the two CDs. In fact, a class that is used in a CD is satisfiable using the closed-world semantics of [LN94, BM13] iff it is satisfiable in the CD using the open-world definitions given above.

As seen above, open- and closed-world are not defined independently, but use a paradigm to define the set of valid object structures. In general, the multi-instance closed-world semantics of a CD is a subset of both multi-instance open-world semantics (cf. Definition 4.3 and Definition 4.4), and the simple closed-world semantics of a CD is a subset of the expansion-based simple open-world semantics. In the context of semantic differencing, the semantics-defining paradigm also influences the output of a semantic difference. This section outlines the relations, differences and application scenarios of the above definitions of CD semantics.

5.1 Simple vs Multi-Instance Semantics

First, let us compare simple and multi-instance semantics under a closed-world approach. We find that two CDs may be semantically equivalent under the simple closed-world semantics, but not under the multi-instance closed-world semantics and vice versa. Consider the CDs in Figure 4 and the object structures in Figure 5: it holds that $[\![cd5.1]\!]^{sc} = [\![cd5.2]\!]^{sc}$, but $om5 \in [\![cd5.1]\!]^{mc} \setminus [\![cd5.2]\!]^{mc}$. On the other hand, $[\![cd5.2]\!]^{mc} = [\![cd5.3]\!]^{mc}$, but $om5 \in [\![cd5.2]\!]^{sc} \setminus [\![cd5.3]\!]^{sc}$. For all three of our open-world approaches we find that $om7$ is in the semantic difference of $cd5.1$ to $cd5.2$ and $cd5.2$ is a strict refinement of $cd5.1$. It makes sense that adding an additional abstract super-class would further restrict the semantics and thus refine an under-specified CD. Note also that $om6 \in [\![cd5.2]\!]^{mo} \setminus [\![cd5.3]\!]^{mo}$, as well as $om6 \in [\![cd5.2]\!]^{l} \setminus [\![cd5.3]\!]^{l}$.

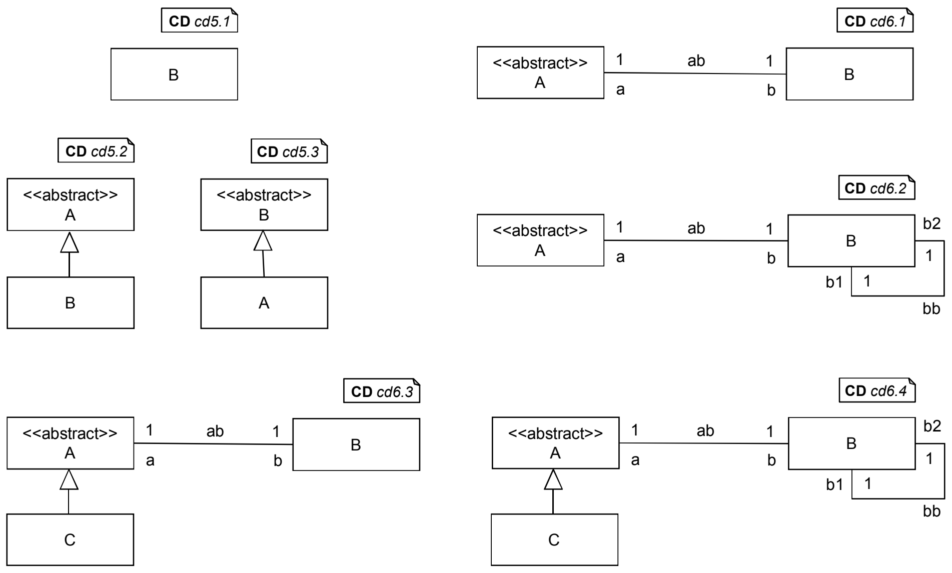

Fig. 4: CDs to illustrate the difference between the semantics definitions.

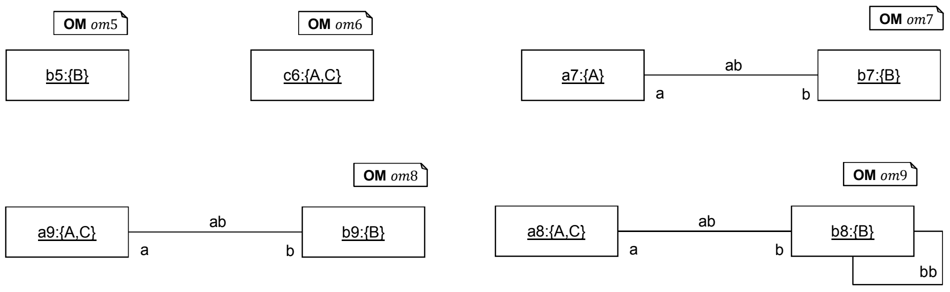

Fig. 5: Object structures to illustrate the difference between the semantics definitions.

5.2 Closed-World vs Open-World Semantics

To outline further differences between closed-world and open-world semantics, consider the CDs $cd6.1$ and $cd6.2$: under both simple and multi-instance closed-world semantics these two CDs are semantically equivalent, as none of their classes can be instantiated. By adding an additional class C as a subclass to A in both CDs, we get $cd6.3$ and $cd6.4$, which are no longer semantically equivalent. In fact, they are semantically incomparable: $om8$ is in the closed-world semantics of $cd6.3$ but not $cd6.4$ and $om9$ is in the closed-world semantics of $cd6.4$ but not $cd6.3$. This situation in which two previously semantically equivalent, models become semantically incomparable by adding the same new element to both, calls into question the well-formedness of these close-world CD-semantics. The expansion-based open-world semantics, by their very Definition 4.4, do not suffer from this flaw, and regarding the previous example, we find that $cd6.2$ is a strict refinement of $cd6.1$ under open-world semantics.

5.3 Relation Between Constraint-Based Open- and Closed-World Semantics

Compared, to the open-world constraint-semantics Definition 4.3, the constraint-based closed-world Definition 4.2 additionally requires instances of cd to solely contain objects and links, typed with classes and associations used in the CD cd. These definitions are related in the sense that removing all objects and links from an instance of the CD, which are not typed with classes and associations of the CD, yields an object structure that instantiates the CD considering either definition of semantics.

To this effect, let $cd = (C, \mathcal{A}bs, \mathcal{A}, cls, \mathcal{M}, \mathcal{H})$ be a CD and let $om = (O, \text{class}, \mathcal{L})$ be an object structure. We denote by $om \downarrow cd \overset{\text{def}}{=} (O', \text{type}', \mathcal{L}')$ the restriction of om to instances of classes and associations used in cd, i. e. $O' = \{o \in O \mid \text{type}(o) \cap C \neq \emptyset\}$, $\mathcal{L}' = \{(o, r, a, o', r') \in \mathcal{L} \mid a \in \mathcal{A}\}$, $\text{type}' : O' \to \wp_{\text{fin}}(C), o \mapsto \text{type}(o) \cap C \setminus (\{c_2 \in C \mid \nexists c_1 \in \text{type}(o) : c_1 \preccurlyeq c_2 \in \mathcal{H}^*\}) \cup \{c_1 \in C \mid \nexists c_2 : c_1 \preccurlyeq c_2 \in \mathcal{H}^*\}$.

The following lemma formalizes the relation between the two semantic mappings: If an object structure is an element of the semantics of a CD, then its restriction to the elements of the CD is an element of the closed-world CD semantics.

Lemma 5.1 *Let cd be a CD and let om be an object structure. If $om \in [\![cd]\!]^{l}$, then $om \downarrow cd \in [\![cd]\!]^{mc}$.*

Proof sketch The definitions of O', type$'$, and $\mathcal{L}'$ rule out the existence of objects in $om \downarrow cd$ of class types or links of association-types that are not defined in the CD. Since an om in the loose open-world semantics does not include objects of abstract types, $om \downarrow cd$ does not either. Therefore the closed-world assumptions hold for $om \downarrow cd$. Loose open-world instances do not include objects whose class type does not include all super-classes of included sub-classes. Therefore Condition (2) of Theorem 4.2 holds. Instances of the loose open-world semantics obey the multiplicity constraints defined in the CD. Since, the restriction removes links that instantiate associations which are not defined in the CD and since instances of the loose open-world semantics already obey the multiplicity constraints, the restriction $om \downarrow cd$ does not include objects with outgoing links that instantiate associations which the object's class type does not inherit and all objects continue to be consistent with the extends relations prescribed in the CD. Therefore, also Conditions (3) and (4) of Theorem 4.2 hold. The definition of type$'$ assures that the classes in the type of an object in $om \downarrow cd$ are related by extends relations defined in cd which implies Condition (5) of Theorem 4.2.

By Lemma 5.1, the closed-world semantics is a restriction of the open-world semantics, i. e. every closed-world instance of a CD is also an open-world instance of the CD. This does not necessarily hold vice-versa. Further, from every open-world instance om of cd , we can construct a closed-world instance by the transformation $om \downarrow cd$.

6 Related Work

While UML does not define formal semantics of CDs, every tool working with CDs uses a semantic definition. Commonly, the semantics for generators [SN11, PSB11, SKM07, SS19, Sw12] is not explicitly defined and the behavior for certain UML-constructs such as "composition" might vary. Generators use a closed-world semantics and only translate modeled elements.

An important analysis on CDs are *consistency* checks. Especially when "nontrivial multiplicity constraints (different than 0, 1, *)" [BM13] occur, there might not exist a valid object structure. In [Sz06] proves the inconsistency of an exemplary CD. [ACIG10] shows that the complexity of full-satisfiability problems is ExpTime-complete. With restrictions on used CD concepts, the complexity can be significantly decreased.

Consistency analysis is often performed using a translation to Alloy [MRR11b, MGB04, Ka17, SAB09, CGR15] or SMT [Wu17, Pr16, PWD16]. Because the closed-world semantics is always a subset of the open-world semantics, consistency checks on closed-world semantics can lead to useful results in open-world semantics.

Like consistency analysis, *semantic differencing*[FLW11, LMK14, Ka17, MRR11a] searches for a valid witness object structure. In previous work, we have presented a semantic differencing tool using a translation to Alloy [Ka17, MRR11a].

Refinement is an interesting property in many domains. For example, the refinement calculus [BW12] can prove the correctness of program modifications. [BS01] proves refinement properties over distributed systems. Open-world models are useful in combination with a merge operator [Br06, LWD11], which reduces underspecification.

7 Conclusion

In this paper we have defined multiple closed-world and open-world semantics for CDs that consider an object either as an instance of a single class or additionally as an instance of all superclasses. We have analyzed and compared these CD-semantics regarding semantic differencing, as well as model-evolution.

We found that an open-world semantics definition based on strict expansions adheres to the notion of underspecification in CDs better than a semantic based on relaxed constraints, and that it is therefore well-suited for semantic differencing and model evolution, especially during analysis in the early stages of development, where the addition of new elements and features should be seen as a refinement. In later stages of the development or when considering code generation, a closed-world approach might be more appropriate.

Moreover, we consider both the simple as well as the multi-instance semantics to be valid and useful depending on the circumstances. If one is, for instance, primarily concerned with development and evolution of data structures, the simple semantics might be the better choice. However, if the usage of objects as method arguments has to be considered (e. g. code generation) a multi-instance semantics would seems more appropriate.

Finally, we found that for our semantics definitions a closed-world instance is always an open-world instance, as well, and that any open-world instance can be transformed into a corresponding closed-world instance by removing not explicitly modeled elements. However, we have yet to develop a transformation that reliably produces non-empty instances for certain edge cases. Nevertheless, we do believe that a reduction of multi-instance expansion-based open-world semantic differencing to a closed-world semantic differencing should be possible. Furthermore, it might be interesting to consider a multi-instance semantics that distinguishes sub- and super-classes instantiated by an object.

Bibliography

[ACIG10] Artale, Alessandro; Calvanese, Diego; Ibáñez-García, Angélica: Full satisfiability of UML class diagrams. In: International Conference on Conceptual Modeling. Springer, pp. 317–331, 2010.

[AP03] Alanen, Marcus; Porres, Ivan: Difference and Union of Models. In: «UML» 2003 - The Unified Modeling Language. Modeling Languages and Applications. 2003.

[BM13] Balaban, Mira; Maraee, Azzam: Finite Satisfiability of UML Class Diagrams with Constrained Class Hierarchy. ACM Trans. Softw. Eng. Methodol., 22(3), 2013.

[Br06] Brunet, Greg; Chechik, Marsha; Easterbrook, Steve; Nejati, Shiva; Niu, Nan; Sabetzadeh, Mehrdad: A manifesto for model merging. In: Proceedings of the 2006 international workshop on Global integrated model management. pp. 5–12, 2006.

[BS01] Broy, Manfred; Stølen, Ketil: Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement. Springer Science & Business Media, 2001.

[BW12] Back, Ralph-Johan; Wright, Joakim: Refinement calculus: a systematic introduction. Springer Science & Business Media, 2012.

[CGR15] Cunha, Alcino; Garis, Ana; Riesco, Daniel: Translating between Alloy specifications and UML class diagrams annotated with OCL. Software & Systems Modeling, 14(1):5–25, 2015.

[DP06] Dobing, Brian; Parsons, Jeffrey: How UML is used. Communications of the ACM, 49(5):109–113, 2006.

[FLW11] Fahrenberg, Uli; Legay, Axel; Wąsowski, Andrzej: Vision Paper: Make a Difference! (Semantically). In: Model Driven Engineering Languages and Systems. 2011.

[Ge21] Gerasimov, Arkadii; Michael, Judith; Netz, Lukas; Rumpe, Bernhard: Agile Generator-Based GUI Modeling for Information Systems. In (Dahanayake, Ajantha; Pastor, Oscar; Thalheim, Bernhard, eds): Modelling to Program (M2P). Springer, pp. 113–126, March 2021.

[HR04] Harel, David; Rumpe, Bernhard: Meaningful Modeling: What's the Semantics of "Semantics"? IEEE Computer, 37(10):64–72, October 2004.

[Hu11] Hutchinson, John; Whittle, Jon; Rouncefield, Mark; Kristoffersen, Steinar: Empirical assessment of MDE in industry. In: Proceedings of the 33rd international conference on software engineering. pp. 471–480, 2011.

[Ka17] Kautz, Oliver; Maoz, Shahar; Ringert, Jan Oliver; Rumpe, Bernhard: CD2Alloy: A Translation of Class Diagrams to Alloy. Technical Report AIB-2017-06, RWTH Aachen University, July 2017.

[KGE09] Küster, Jochen M.; Gerth, Christian; Engels, Gregor: Dependent and Conflicting Change Operations of Process Models. In: Model Driven Architecture - Foundations and Applications. 2009.

[KKT11] Kehrer, Timo; Kelter, Udo; Taentzer, Gabriele: A Rule-Based Approach to the Semantic Lifting of Model Differences in the Context of Model Versioning. In: International Conference on Automated Software Engineering (ASE'11). 2011.

[Kü08] Küster, Jochen M.; Gerth, Christian; Förster, Alexander; Engels, Gregor: Detecting and Resolving Process Model Differences in the Absence of a Change Log. In: Business Process Management. 2008.

[La14] Langer, Philip; Mayerhofer, Tanja; Wimmer, Manuel; Kappel, Gerti: On the usage of UML: Initial results of analyzing open UML models. Modellierung 2014, 2014.

[LMK14] Langer, Philip; Mayerhofer, Tanja; Kappel, Gerti: A Generic Framework for Realizing Semantic Model Differencing Operators. In: PSRC@MoDELs. CEUR Workshop Proceedings, 2014.

[LN94] Lenzerini, Maurizio; Nobili, Paolo: On the interaction between ISA and cardinality constraints. In: Proceedings of 1994 IEEE 10th International Conference on Data Engineering. 1994.

[LWD11] Lutz, Rainer; Wurfel, David; Diehl, Stephan: How humans merge UML-models. In: 2011 International Symposium on Empirical Software Engineering and Measurement. IEEE, pp. 177–186, 2011.

[Ma15] Management Group, Object: , OMG Unified Modeling Language (OMG UML) Version 2.5, 2015.

[MGB04] Massoni, Tiago; Gheyi, Rohit; Borba, Paulo: A UML class diagram analyzer. TUM, p. 100, 2004.

[MRR10] Maoz, Shahar; Ringert, Jan Oliver; Rumpe, Bernhard: A Manifesto for Semantic Model Differencing. In: Proceedings Int. Workshop on Models and Evolution (ME'10). LNCS 6627. Springer, pp. 194–203, 2010.

[MRR11a] Maoz, Shahar; Ringert, Jan Oliver; Rumpe, Bernhard: CDDiff: Semantic Differencing for Class Diagrams. In (Mezini, Mira, ed.): ECOOP 2011 - Object-Oriented Programming. Springer Berlin Heidelberg, pp. 230–254, 2011.

[MRR11b] Maoz, Shahar; Ringert, Jan Oliver; Rumpe, Bernhard: Semantically Configurable Consistency Analysis for Class and Object Diagrams. In: Conference on Model Driven Engineering Languages and Systems (MODELS'11). LNCS 6981. Springer, pp. 153–167, 2011.

[Pr16] Przigoda, Nils; Hilken, Frank; Peters, Judith; Wille, Robert; Gogolla, Martin; Drechsler, Rolf: Integrating an SMT-Based ModelFinder into USE. In: MoDeVVa@ MoDELS. pp. 40–45, 2016.

[PSB11] Parada, Abilio G.; Siegert, Eliane; Brisolara, Lisane B. de: Generating Java Code from UML Class and Sequence Diagrams. In: 2011 Brazilian Symposium on Computing System Engineering. pp. 99–101, 2011.

[PWD16] Przigoda, Nils; Wille, Robert; Drechsler, Rolf: Ground setting properties for an efficient translation of OCL in SMT-based model finding. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems. pp. 261–271, 2016.

[Re78] Reiter, Raymond: On Closed World Data Bases. In: Logic and Data Bases. 1978.

[Ru16] Rumpe, Bernhard: Modeling with UML: Language, Concepts, Methods. Springer International, July 2016.

[SAB09] Shah, Seyyed; Anastasakis, Kyriakos; Bordbar, Behzad: From UML to Alloy and back again. In: International Conference on Model Driven Engineering Languages and Systems. Springer, pp. 158–171, 2009.

[SKM07] Sarma, Monalisa; Kundu, Debasish; Mall, Rajib: Automatic test case generation from UML sequence diagram. In: 15th International Conference on Advanced Computing and Communications (ADCOM 2007). IEEE, pp. 60–67, 2007.

[SN11] Sejans, Janis; Nikiforova, Oksana: Practical Experiments with Code Generation from the UML Class Diagram. In: MDA/MDSD. pp. 57–67, 2011.

[SS19] Sunitha, EV; Samuel, Philip: Automatic code generation from UML state chart diagrams. IEEE Access, 7:8591–8608, 2019.

[Sw12] Swain, Ranjita; Panthi, Vikas; Behera, Prafulla Kumar; Mohapatra, Durga Prasad: Automatic test case generation from UML state chart diagram. International Journal of Computer Applications, 42(7):26–36, 2012.

[Sz06] Szlenk, Marcin: Formal semantics and reasoning about uml class diagram. In: 2006 International Conference on Dependability of Computer Systems. IEEE, pp. 51–59, 2006.

[T.13] T. Kehrer, U. Kelter, and G. Taentzer: Consistency-Preserving Edit Scripts in Model Versioning. In: International Conference on Automated Software Engineering (ASE). 2013.

[Ta14] Taentzer, Gabriele; Ermel, Claudia; Langer, Philip; Wimmer, Manuel: A fundamental approach to model versioning based on graph modifications: from theory to implementation. Software & Systems Modeling, 13(1), 2014.

[TK09] Thüm, Thomas; , Don; Kästner, Christian: Reasoning about Edits to Feature Models. In: Proceedings of the 31st International Conference on Software Engineering. 2009.

[Wu17] Wu, Hao: MaxUSE: a tool for finding achievable constraints and conflicts for inconsistent UML class diagrams. In: International Conference on Integrated Formal Methods. Springer, pp. 348–356, 2017.