Consistency analysis of UML models*

Guo-Fu Tang[1,2], Jian-Min Jiang[1,2], Hao Wen[3,4]
[1]Automatic Software Generation and Intelligence Service Key Laboratory of Sichuan Province
[2]Chengdu University of Information Technology, Chengdu 610103, China
[3]Chengdu Institute of Computer Applications, Chinese Academy of Sciences, Chengdu 610103, China
[4]University of Chinese Academy of Sciences, Beijing 101408, China
domcofu@qq.com; jjm@cuit.edu.cn; wenhao21@mails.ucas.ac.cn

Abstract

The inconsistencies of UML models (diagrams) during the software development process may cause errors in documents or programs. Massive consistency rules have been proposed by researchers to detect inconsistencies in existing works. However, the approaches of consistency analysis have not been addressed adequately in literature. In this paper, we propose a new approach for analyzing consistencies between UML models. We formally specify UML models and define the consistency relation between UML models. Based on the consistency definition, we first discuss the composition and decomposition of consistencies, and then explore the equivalence of consistencies.

Index Terms UML model, Consistency, Composition, Equivalence

1 Introduction

To develop a modern software system in the Model-Based Engineering approach, multiple perspectives of modeling the system are necessary. The UML is a semi-formal graphical modeling language [15], and is widely used in Model Driven Engineering (MDE) due to its multiple perspectives for describing software systems. Software artifacts (e.g., software architecture and implementation codes) are interrelated to UML diagrams in whatever versions, levels of abstraction, and stages. Consequently, it is often unable to avoid faults because of differences between UML diagrams in software development. Specifically, the evolution of the models or diagrams is frequently accompanied by augmenting, reducing, or modifying, which potentially results in contradictory specifications (inconsistencies). In order to classify such conflicts, there are seven UML consistency dimensions in the systematic researches [23, 8, 22, 11, 2], and a binary relation can capture these dimensions, including those that refer to *endogenous consistency* [14].

Multiple UML diagrams need to be strictly compliant in order for a software system to be accurately and completely described, especially for Safety Critical Systems (SCS). Without complete and consistent design models, programmers need to manually supplement the lack of design models with codes, which may cause faults and insecurities. Thus, we proposed to design accurate models because the models are easier to be analysed in formal methods. Corresponding to categories of UML diagrams, consistencies between these diagrams are generally divided into structural(i.e., syntactic) and behavioral(i.e., semantic) ones that has been systematically investigated in research [18, 19]. The consistency rules are sophisticated and cannot distinguish between binary and N-ary relations (i.e., the number of UML models involved in consistency rules is uncertain). For example, rule 15 from [18] involves a class diagram, a state machine diagram, and a activity diagram. Approaches to check N-ary consistencies have not yet been fully developed and the relevant theories are merely reviewed as well [18]. While using multiple rules to detect inconsistencies in a large system, it is obviously arduous to manage the duplication or absence of rules. Thus, it is necessary to present binary consistency relations for unifying N-ary consistency relations between UML models.

There are many existing efforts contributing to managing consistency relations or rules among UML models whereas ignoring the relationships between the rules [1, 17, 13]. In this paper, we propose a novel approach for analyzing

*This work is supported by National Key R&D Program of China (No. 2022YFB3305104), National Natural Science Foundation of China (No. 61772004), and Scientific Research Foundation for Advanced Talents of Chengdu University of Information Technology (No. KYTZ202009).
DOI:10.18293/DMSVIVA2023-187

consistencies between UML models. We formally specify UML models and define the consistency relation between UML models. Based on the consistency definition, we first discuss the composition and decomposition of consistencies, and then explore the equivalence of consistencies. Because of such a duality of consistency relations, our method saves complexity and makes consistency characteristics more extensible, and formal methods aid in removing ambiguities and enforcing consistency. Our work aims at describing and managing consistency in complete UML diagrams for a system. Due to the space limitation, all proofs are deleted from our paper.

2 Model Composition

In this section, we will introduce a formal model [20] to specify UML diagrams, and then show characteristics of the formal model. In UML [15] various software artifacts are all regarded as models and its constituent parts are model elements. Such consideration facilitates the analysis of and visualization representations of traceability using graph-based tools.

Definition 1. *A **unified structure**($\mathcal{US}$) is a tuple* $\langle \mathrm{ME}, \prec, \overset{1}{\hookrightarrow}, \cdots, \overset{n}{\hookrightarrow}, \lambda_m, \lambda_d, \overset{1}{\tau}, \cdots, \overset{m}{\tau} \rangle$ *with*

- ME*, a finite set of the model elements,*
- $\prec \subseteq \mathrm{ME} \times \mathrm{ME}$*, the* containment relation *such that it is an (irreflexive) partial order,*
- $\lambda_m \subseteq \mathrm{ME} \times \mathrm{ME}$*, the constraint on model elements,*
- $\lambda_d \subseteq \mathrm{ME} \times (\prec \cup \overset{1}{\hookrightarrow} \cup \cdots \cup \overset{n}{\hookrightarrow})$*, the constraint on dependencies,*
- $\forall i \in \{1, \cdots, n\}, \overset{i}{\hookrightarrow} \subseteq \mathrm{ME} \times \mathrm{ME}$*, the* dependency relation*, and*
- $\forall j \in \{1, \cdots, m\}, \overset{j}{\tau} \subseteq \mathrm{ME}$*, the* type set *of model elements such that* $\forall e \in \mathrm{ME}, \exists \tau \in \{\overset{1}{\tau}, \cdots, \overset{m}{\tau}\} : e \in \tau$.

Here, for all $x, y \in \mathrm{ME}$, $x \overset{i}{\hookrightarrow} y$ ($i \in \{1, \cdots, n\}$) is called a *dependency*, read as x *depending on* y (note that i denotes that the type of dependencies). And $x \prec y$ means x is contained in y. If $x \prec z, y \prec z$, they are simplistically denoted by $x, y \prec z$ and means x and y are both contained in z. For all $w, v \in \mathrm{ME}$, the notation $v \not\prec w$ means that w does not contain v. The tuples $\overset{1}{\tau}, \cdots, \overset{m}{\tau}$ are grouping constructs for model elements and are used to classify the model elements.

We then present an example showing how UML diagrams are converted into $\mathcal{US}$ models. Figure 1 presents UML diagrams of a video-on-demand (VOD) system that allows user U to select or play movies provided by server S. Different from the original one [5], our example adds a *Composite State* in the state machine diagram and the

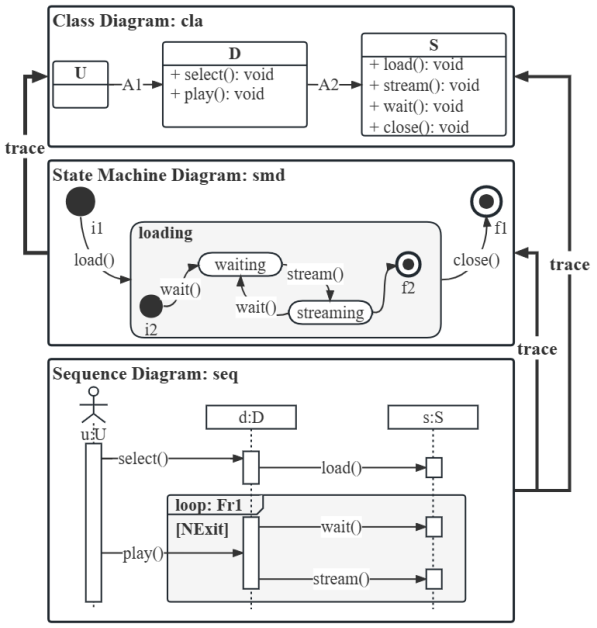

Figure 1. VOD System Example

corresponding *Combined Fragment* in the sequence diagram. The structure of the VOD system is represented in the class diagram (top), where the state machine diagram(middle) specifies the behavior of the class S in the class diagram, and the sequence diagram (bottom) depicts watching a movie.

Example 1. *By Definition 1, the entire VOD system is denoted as* $\mathcal{US}_{\mathcal{VOD}} = \langle \mathrm{ME}, \prec, \lambda_m, \lambda_d, \overset{ClaD}{\tau}, \overset{SeqD}{\tau}, \overset{StaD}{\tau}, \overset{Trace}{\hookrightarrow} \rangle$ *where* $\mathrm{ME} = \{seq, smd, cla, (cla, seq), (cla, smd), (smd, seq)\}$, $\prec = \lambda_m = \lambda_d = \emptyset$, $\overset{ClaD}{\tau} = \{cla\}$, $\overset{SeqD}{\tau} = \{seq\}$, $\overset{StaD}{\tau} = \{smd\}$, *and* $\overset{Trace}{\hookrightarrow} = \{(cla, seq), (cla, smd), (smd, seq)\}$. *$ClaD$ refers to class diagrams, $SeqD$ means sequence diagrams, $StaD$ denotes state machine diagrams, and $Trace$ is dependencies between these diagrams. Intuitively, this is constructed at a higher level which views the entire system as a diagram. The US supports not only high-level modeling but also concrete one.*

All UML diagrams in Figure 1 can be independently modeled by Definition 1. The class diagram cla, one of elements in $\overset{ClaD}{\tau}$ *of* $\mathcal{US}_{\mathcal{VOD}}$*(i.e., the* cla *element), is represented by* $\mathcal{US}_{cla} = \langle \mathrm{ME}', \prec', \lambda'_m, \lambda'_d, \overset{Class}{\tau}, \overset{Oprs}{\tau}, \overset{Asoc}{\tau} \rangle$ *where* $\mathrm{ME}' = \{U, D, S, select, play, load, stream, wait, close\}$, $\prec' = \lambda'_m = \lambda'_d = \emptyset$, $\overset{Class}{\tau} = \{U, D, S\}$, $\overset{Asoc}{\tau} = \{A1, A2\}$, *and* $\overset{Oprs}{\tau} = \{select, play, load, stream, wait, close\}$. *There are notations such as $Class$ representing classes, $Asoc$ representing associations, and $Oprs$ depict-*

ing a collection of operations of all classes.

The sequence diagram seq is denoted as $\mathcal{US}_{seq} = \langle \mathrm{ME}'', \prec'', \overset{Inst}{\hookrightarrow}, \overset{Seq}{\hookrightarrow}, \overset{Itrc}{\hookrightarrow}, \lambda_m'', \lambda_d'', \overset{Obj}{\tau}, \overset{Msg}{\tau}, \overset{Frag}{\tau}, \overset{Grd}{\tau} \rangle$ *where* $\mathrm{ME}'' = \{u, s, d, U, S, D, Fr1,\ NExit,\ select,\ load,\ wait, play, stream\}$, $\prec''= \{(NExit, Fr1), (wait, Fr1), (stream, Fr1)\}$, $\lambda_m'' = \lambda_d'' = \emptyset$, $\overset{Obj}{\tau} = \{u, d, s\}$, $\overset{Frag}{\tau} = \{ Fr1\}$, $\overset{Grd}{\tau} = \{NExit\}$, $\overset{Msg}{\tau} = \{select, load,\ wait,\ play, stream\}$, $\overset{Inst}{\hookrightarrow} = \{(d, D), (s, S), (u, U)\}$, $\overset{ObjRecv}{\hookrightarrow} = \{(select, d), (play, d), (load, s), (stream, s), (wait, s)\}$, $\overset{Seq}{\hookrightarrow} = \{(load, select), (wait, load), (play, wait), (stream, play)\}$, *and* $\overset{Itrc}{\hookrightarrow} = \{(d, u), (s, d)\}$. *The notation* $Inst$ *means the relation between objects* Obj *and corresponding classes,* Seq *represents the sequence of all messages* Msg, $ObjRecv$ *represents all receiving messages of particular object, and* $Itrc$ *represents the interactions between objects.* $\prec''$ *is non-empty due to combined fragment* $Frag$ *containing messages and guard represented by* Grd.

And the state machine diagram smd *can be denoted by* $\mathcal{US}_{smd} = \langle \mathrm{ME}''', \prec''', \overset{Trans}{\hookrightarrow}, \lambda_m''', \lambda_d''', \overset{I}{\tau}, \overset{F}{\tau}, \overset{CS}{\tau}, \overset{SS}{\tau}, \overset{Acts}{\tau} \rangle$ *where* $\mathrm{ME}''' = \{i2, i1, s2, waiting, streaming, loading, f1, f2, load, wait, stream, close\}$, $\prec'''= \{(i2, loading), (waiting, loading), (streaming, loading), (f2, loading), (wait, loading), (stream, loading)\}$, $\overset{Trans}{\hookrightarrow} = \{(loading, i1), (waiting, i2), (waiting, streaming), (streaming, waiting), (f2, streaming), (f1, loading)\}$, $\lambda_m''' = \lambda_d''' = \emptyset$, $\overset{I}{\tau} = \{i1, i2\}$, $\overset{F}{\tau} = \{f1, f2\}$, $\overset{CS}{\tau} = \{\ loading\ \}$, $\overset{SS}{\tau} = \{ waiting, streaming\}$, *and* $\overset{Acts}{\tau} = \{load,\ wait,\ stream, close\}$. *We classify states of the state machine diagram into initial states* I, *final states* F, *simple states* SS, *and composite states* CS. *We named the transitions in the state machine diagram as* $Trans$. *Containment relation* $\prec'''$ *is formed between composite states and the elements inside.*

It is obvious that the unified structure model can easily specify single UML model and the composition of UML models.

Definition 2. *Let* $\mathcal{US} = \langle \mathrm{ME}, \prec, \overset{1}{\hookrightarrow}, \cdots, \overset{n}{\hookrightarrow}, \lambda_m, \lambda_d, \overset{1}{\tau}, \cdots, \overset{m}{\tau} \rangle$ *be a unified structure.*

(1) A sequence $rc = x_1 \cdots x_n$ *is called a* relation chain *in* $\mathcal{US}$ *iff* $\forall i \in \{1, \cdots, n-1\}, x_i, x_{i+1} \in \mathrm{ME}$, $(x_i, x_{i+1}) \in (\prec \cup \overset{1}{\hookrightarrow} \cup \cdots \cup \overset{n}{\hookrightarrow}) \vee (x_{i+1}, x_i) \in (\prec \cup \overset{1}{\hookrightarrow} \cup \cdots \cup \overset{n}{\hookrightarrow})$.

$\ddot{rc}$ *denotes the model elements in the relation chain* $rc = x_1 \cdots x_n$, *that is,* $\ddot{rc} = \{x_1, \cdots, x_n\}$. $RC(\mathcal{US})$ *denotes all possible relation chains in* $\mathcal{US}$.

(2) A sequence $dc = x_1 \cdots x_n$ *is called a* dependency chain *in* $\mathcal{US}$ *iff* $\forall i \in \{1, \cdots, n-1\}, x_i, x_{i+1} \in \mathrm{ME}$, $(x_i, x_{i+1}) \in (\prec \cup \overset{1}{\hookrightarrow} \cup \cdots \cup \overset{n}{\hookrightarrow})$.

$\hat{dc}$ *denotes the model elements in the dependency chain* $dc = x_1 \cdots x_n$, *that is,* $\hat{dc} = \{x_1, \cdots, x_n\}$. $DC(\mathcal{US})$ *denotes all possible dependency chains in* $\mathcal{US}$. $[dc]$ *denotes the number of model elements in the dependency chain* $dc = x_1 \cdots x_n$, *that is,* $[dc] = n$.

Obviously, a relation chain is nondirectional while a dependency chain is directional. For example, $dc_{trace} = cla\ smd\ seq$,$[dc_{trace}] = 3$ in Figure 1 is one of dependency chains and likewise a relation chain.

Proposition 1. *Let* $\mathcal{US}$ *be a unified structure and* $dc = x_1 \cdots x_n \in DC(\mathcal{US})$. *If* $\forall i \in \{1, \cdots, n-1\}, (x_i, x_{i+1}) \in \prec$, *then there does not exist a cycle in* dc.

This proposition states that a dependency chain only containing containment relations does not have a cycle.

Proposition 2. *If* $\mathcal{US}$ *is a unified structure, then* $DC(\mathcal{US}) \subseteq RC(\mathcal{US})$.

Clearly, the number of relation chains is greater than equal to that of dependency chains in a unified structure.

Complex software systems contain many UML diagrams to specify complete information about the systems. Once the whole system model is constructed, the UML diagrams (or their subparts) must be consistent with information. We then discuss the composition of models.

Definition 3. *Let* $\mathcal{US}' = \langle \mathrm{ME}', \prec', \overset{1}{\hookrightarrow}', \cdots, \overset{n}{\hookrightarrow}', \lambda_m', \lambda_d', \overset{1}{\tau}', \cdots, \overset{m}{\tau}' \rangle$ *and* $\mathcal{US}'' = \langle \mathrm{ME}'', \prec'', \overset{1}{\hookrightarrow}'', \cdots, \overset{n}{\hookrightarrow}'', \lambda_m'', \lambda_d'', \overset{1}{\tau}'', \cdots, \overset{m}{\tau}'' \rangle$ *be two unified structures.*

$\mathcal{US}'$ *is called a* substructure *of* $\mathcal{US}''$, *denoted as* $\mathcal{US}' \sqsubseteq \mathcal{US}''$, *iff* $\mathrm{ME}' \subseteq \mathrm{ME}'', \prec' \subseteq \prec'', \overset{1}{\hookrightarrow}' \subseteq \overset{1}{\hookrightarrow}'', \cdots, \overset{n}{\hookrightarrow}' \subseteq \overset{n}{\hookrightarrow}''$, $\lambda_m' \subseteq \lambda_m'', \lambda_d' \subseteq \lambda_d''$ *and* $\overset{1}{\tau}' \subseteq \overset{1}{\tau}'', \cdots, \overset{m}{\tau}' \subseteq \overset{m}{\tau}''$.

Example 2. *As Example 1 illustrates, we first construct* $\mathcal{US}_{\mathcal{VOD}}$ *and then independently model each diagram in* $\mathcal{US}_{\mathcal{VOD}}$. *Thus, there exists the substructures* $\mathcal{US}_{cla} \sqsubseteq \mathcal{US}_{VOD}, \mathcal{US}_{seq} \sqsubseteq \mathcal{US}_{VOD}$, *and* $\mathcal{US}_{smd} \sqsubseteq \mathcal{US}_{VOD}$.

A substructure is included in the original unified structure, and the unified structure may be separated into multiple substructures.

Definition 4. *Let* $\mathcal{US}' = \langle \mathrm{ME}', \prec', \overset{1}{\hookrightarrow}', \cdots, \overset{n}{\hookrightarrow}', \lambda_m', \lambda_d', \overset{1}{\tau}', \cdots, \overset{m}{\tau}' \rangle$ *and* $\mathcal{US}'' = \langle \mathrm{ME}'', \prec'', \overset{1}{\hookrightarrow}'', \cdots, \overset{n}{\hookrightarrow}'', \lambda_m'', \lambda_d'', \overset{1}{\tau}'', \cdots, \overset{m}{\tau}'' \rangle$ *be two unified structures.*

If $\prec' \cup \prec''$ *is an (irreflexive) partial order, the composition of* $\mathcal{US}'$ *and* $\mathcal{US}''$ *is defined as* $\mathcal{US}' \uplus \mathcal{US}'' = \langle \mathrm{ME}, \prec, \overset{1}{\hookrightarrow}, \cdots, \overset{n}{\hookrightarrow}, \lambda_m, \lambda_d, \overset{1}{\tau}, \cdots, \overset{m}{\tau} \rangle$ *where* $\mathrm{ME} = \mathrm{ME}' \cup \mathrm{ME}''$, $\prec = \prec' \cup \prec''$, $\forall i \in \{1, \cdots, n\}$: $\overset{i}{\hookrightarrow} = \overset{i}{\hookrightarrow}' \cup \overset{i}{\hookrightarrow}''$, $\lambda_m =$

$\lambda'_m \cup \lambda''_m, \lambda_d = \lambda'_d \cup \lambda''_d$ *and* $\forall j \in \{1, \cdots, m\}$: $\overset{j}{\tau} = \overset{j}{\tau'} \cup \overset{j}{\tau''}$. $\mathcal{US}'$, $\mathcal{US}''$ *are said to be* composable.

Note that two composable unified structures may have different number of types of dependency relations and elements, we equivalently translate them into the two unified structures with the same number of dependency or element types before composition. For example, $\mathcal{US}'$ and $\mathcal{US}''$ are composable where $\mathcal{US}' = \langle \mathrm{ME}', \prec', \overset{a}{\hookrightarrow}', \overset{x}{\hookrightarrow}', \overset{n}{\hookrightarrow}', \lambda'_m, \lambda'_d, \overset{1}{\tau}', \cdots, \overset{m}{\tau}' \rangle$ and $\mathcal{US}'' = \langle \mathrm{ME}'', \prec'', \overset{x}{\hookrightarrow}'', \overset{y}{\hookrightarrow}'', \overset{z}{\hookrightarrow}'', \lambda''_m, \lambda''_d, \overset{1}{\tau}'', \cdots, \overset{m}{\tau}'' \rangle$. Obviously, we can translate $\mathcal{US}'$ and $\mathcal{US}''$ into $\mathcal{US}_1$ and $\mathcal{US}_2$, respectively:

$$\mathcal{US}_1 = \langle \mathrm{ME}', \prec', \overset{a}{\hookrightarrow}', \overset{x}{\hookrightarrow}', \overset{y}{\hookrightarrow}', \overset{z}{\hookrightarrow}', \lambda'_m, \lambda'_d, \overset{1}{\tau}', \cdots, \overset{m}{\tau}' \rangle$$

where $\overset{y}{\hookrightarrow}' = \overset{z}{\hookrightarrow}' = \emptyset$ and

$$\mathcal{US}_2 = \langle \mathrm{ME}'', \prec'', \overset{a}{\hookrightarrow}'', \overset{x}{\hookrightarrow}'', \overset{y}{\hookrightarrow}'', \overset{z}{\hookrightarrow}'', \lambda''_m, \lambda''_d, \overset{1}{\tau}'', \cdots, \overset{m}{\tau}'' \rangle$$

where $\overset{a}{\hookrightarrow}'' = \emptyset$.

Clearly, $\mathcal{US}_1 = \mathcal{US}', \mathcal{US}_2 = \mathcal{US}''$. Moreover, $\mathcal{US}_1$ and $\mathcal{US}_2$ have the same number of dependency types. Thus, $\mathcal{US}_1$ and $\mathcal{US}_2$ can be composed according to the previous definition. The composition of unified structures has the following properties.

Proposition 3. *Let $\mathcal{US}$, $\mathcal{US}'$ and $\mathcal{US}''$ be three unified structures. And let every two of the three unified structures be composable. Then*

(1) $\mathcal{US}' \uplus \mathcal{US}''$ is a unified structure,

(2) $\mathcal{US}' \uplus \mathcal{US}'' = \mathcal{US}'' \uplus \mathcal{US}'$, and

(3) $(\mathcal{US} \uplus \mathcal{US}') \uplus \mathcal{US}'' = \mathcal{US} \uplus (\mathcal{US}' \uplus \mathcal{US}'')$.

This proposition shows the composition of unified structures has closure, commutativity, and associativity.

Proposition 4. *If $\mathcal{US}$, $\mathcal{US}'$ are two unified structures and composable, then $(DC(\mathcal{US}) \cup DC(\mathcal{US}')) \subseteq DC(\mathcal{US} \uplus \mathcal{US}')$.*

The composition of unified structures does not add or reduce any elements of native structures. Consequently the dependency chain of unified structures remains after composition.

3 Consistency Relation

Consistency can be treated as a relation that stores the pairs of either two elements in UML diagrams or two UML models, which satisfies the contained consistency rules. A comparison should be sought between two or more UML models with UML consistency rules. Consistency rules are systematically collected in plain English text [18]. All external and internal rules are required to ensure the consistency of a system model. But we select and extend a few *external* rules listed (See Table 1) for VOD in Figure 1. The *internal* consistency rules are not considered in this paper.

Table 1. Several Consistency Rules

ID	Description
CR1	Each class in the class diagram must be instantiated in a sequence diagram.
CR2	Each public method in a class diagram triggers a receiving message in a sequence diagram.
CR3	A *Loop* fragment matches repetition of states caused by messages.

Definition 5. *Let $\mathcal{US}' = \langle \mathrm{ME}', \prec', \overset{1}{\hookrightarrow}', \cdots, \overset{n}{\hookrightarrow}', \lambda'_m, \lambda'_d, \overset{1}{\tau}', \cdots, \overset{m}{\tau}' \rangle$ and $\mathcal{US}'' = \langle \mathrm{ME}'', \prec'', \overset{1}{\hookrightarrow}'', \cdots, \overset{n}{\hookrightarrow}'', \lambda''_m, \lambda''_d, \overset{1}{\tau}'', \cdots, \overset{m}{\tau}'' \rangle$ be two unified structures. A relation $R_C \subseteq \mathrm{ME}' \times \mathrm{ME}''$ is called a* consistency relation *between $\mathcal{US}'$ and $\mathcal{US}''$ iff there exists a consistency rule between $\mathcal{US}'$ and $\mathcal{US}''$ such that R_C satisfies the correspondence between $\mathcal{US}'$ and $\mathcal{US}''$ under such a rule. We define $R_C|_{\mathcal{US}'} = \{e \in \mathrm{ME}' \mid \exists e' \in \mathrm{ME}'' : (e, e') \in R_C\}$ and $R_C|_{\mathcal{US}''} = \{e \in \mathrm{ME}'' \mid \exists e' \in \mathrm{ME}' : (e', e) \in R_C\}$.*

Here, we formally denote the consistency rule as a binary relation.

Example 3. *According to the consistency rules CR1 and CR2 in Table 1, there exist the corresponding consistency relations $R_{C1} = \{(D, d), (S, s)\}$ and $R_{C2} = \{(select, select), (play, play), (load, load), (stream, stream), (wait, wait)\}$ between the class diagram $\mathcal{US}_{cla}$ and the sequence diagram $\mathcal{US}_{seq}$ in Figure 1.*

Proposition 5. *Let $\mathcal{US}'$ and $\mathcal{US}''$ be two unified structures. If R_{C1}, R_{C2} be two consistency relations between $\mathcal{US}'$ and $\mathcal{US}''$, then $R_{C1} \cup R_{C2}$ be a consistency relation between $\mathcal{US}'$ and $\mathcal{US}''$.*

Example 4. *As R_{C1}, R_{C2} is presented in Example 3, we have $R_C = R_{C1} \cup R_{C2} = \{(D, d), (S, s), (select, select), (play, play)\}$. Obviously, R_C is the composition of the consistency rules CR1 and CR2.*

Proposition 6. *Let $\mathcal{US}'$ and $\mathcal{US}''$ be two unified structures. And let R_{C1}, R_{C2} be two consistency relations between $\mathcal{US}'$ and $\mathcal{US}''$. Then*

(1) $R_{C1} \cup R_{C2}$ be a consistency relation between $\mathcal{US}'$ and $\mathcal{US}''$.

(2) $R_{C1} \cap R_{C2}$ be a consistency relation between $\mathcal{US}'$ and $\mathcal{US}''$.

(3) $R_{C1} \setminus R_{C2}$ be a consistency relation between $\mathcal{US}'$ and $\mathcal{US}''$.

This proposition states that the consistency relations are preserved under the union, intersection and minus operations

Theorem 1. *Let $\mathcal{US}_1$, $\mathcal{US}'_1$, $\mathcal{US}_2$, $\mathcal{US}'_2$ be four unified structures. Let $\mathcal{US}_1 \sqsubseteq \mathcal{US}'_1$ and $\mathcal{US}_2 \sqsubseteq \mathcal{US}'_2$. If R_C be a consistency relation between $\mathcal{US}_1$ and $\mathcal{US}_2$, then R_C be a consistency relation between $\mathcal{US}'_1$ and $\mathcal{US}'_2$.*

Clearly, the consistency between the submodels implies the consistency between the original models.

To manage consistencies between models, it is necessary to handle how consistencies influence mutually. We introduce *atomic* consistencies, i.e., the consistencies that cannot be further decomposed. The combination of atomic consistencies can express those complicated consistencies. Thus, we discuss how the combination works in the following.

Definition 6. *Let $\mathcal{US}' = \langle \mathrm{ME}', \prec', \overset{1}{\hookrightarrow}', \cdots, \overset{n}{\hookrightarrow}', \lambda'_m, \lambda'_d, \overset{1}{\tau}', \cdots, \overset{m}{\tau}' \rangle$ and $\mathcal{US}'' = \langle \mathrm{ME}'', \prec'', \overset{1}{\hookrightarrow}'', \cdots, \overset{n}{\hookrightarrow}'', \lambda''_m, \lambda''_d, \overset{1}{\tau}'', \cdots, \overset{m}{\tau}'' \rangle$ be two unified structures. Let R_C be a consistency relation between $\mathcal{US}'$ and $\mathcal{US}''$. R_C is said to be* atomic *iff $\exists\ \overset{x}{\tau}' \in \{\overset{1}{\tau}', \cdots, \overset{m}{\tau}'\} : R_C|_{\mathcal{US}'} \subseteq \overset{x}{\tau}'$ and $\exists\ \overset{y}{\tau}'' \in \{\overset{1}{\tau}'', \cdots, \overset{m}{\tau}''\} : R_C|_{\mathcal{US}''} \subseteq \overset{y}{\tau}''$.*

This states that an atomic consistency is only connected to the type sets of model elements when the model is translated into corresponding unified structures.

Example 5. *In Figure 1, there exist consistency relations $R_C = \{(D, d), (S, s)\}$ and $R_{C4} = \{(D, d), (select, select), (play, play), (S, s), (load, load), (wait, wait), (stream, stream)\}$ between $\mathcal{US}_{cla}$ and $\mathcal{US}_{seq}$. R_{C4} is under the consistency rule that a lifeline(including receiving messages) in sequence diagram conforms to its class(including operations). R_C expresses consistency between the class of $\mathcal{US}_{cla}$ and the object of $\mathcal{US}_{seq}$. There are one single type of elements in $\mathcal{US}_{cla}$ and $\mathcal{US}_{seq}$. Nevertheless, R_{C4} describes consistency between the class and operation of $\mathcal{US}_{cla}$ and the object and receiving message of $\mathcal{US}_{seq}$. Two distinct types of elements are involved in R_{C4}. R_{C4} is therefore not* atomic.

Theorem 2. *Let $\mathcal{US}' = \langle \mathrm{ME}', \prec', \overset{1}{\hookrightarrow}', \cdots, \overset{n}{\hookrightarrow}', \lambda'_m, \lambda'_d, \overset{1}{\tau}', \cdots, \overset{m}{\tau}' \rangle$ and $\mathcal{US}'' = \langle \mathrm{ME}'', \prec'', \overset{1}{\hookrightarrow}'', \cdots, \overset{n}{\hookrightarrow}'', \lambda''_m, \lambda''_d, \overset{1}{\tau}'', \cdots, \overset{m}{\tau}'' \rangle$ be two unified structures. Let R_C be a consistency relation between $\mathcal{US}'$ and $\mathcal{US}''$.*

If R_C is not atomic, then there exists the k atomic consistency relations $R_{C_1}, \cdots, R_{C_k}$ such that $R_C = R_{C_1} \cup \cdots \cup R_{C_k}$.

Example 6. *According to Example 5, we have consistency relation R_{C4} between $\mathcal{US}_{cla}$ and $\mathcal{US}_{seq}$ in Figure 1. R_{C4} is able to be the union of R_{C1} and R_{C2} between $\mathcal{US}_{cla}$ and $\mathcal{US}_{seq}$ by Example 3.*

This theorem considers that the $atomic$ consistency relations are the most fundamental ones as they can compose new consistency relations.

There are several ways to illustrate containment relations in UML diagrams. The sequence diagram, for example, employs *Combined Fragment* to explicitly confine a series of interactions, but the class diagram uses relations (e.g., composition, aggregation, realization, and inheritance) to represent implicit containment between elements. The consistency of these containment relations has yet to be widely debated. The *unified structure* is capable of modeling these containment relationships and it is then used to discuss the consistency of containment relations.

Theorem 3. *Let $\mathcal{US}' = \langle \mathrm{ME}', \prec', \overset{1}{\hookrightarrow}', \cdots, \overset{n}{\hookrightarrow}', \lambda'_m, \lambda'_d, \overset{1}{\tau}', \cdots, \overset{m}{\tau}' \rangle$ and $\mathcal{US}'' = \langle \mathrm{ME}'', \prec'', \overset{1}{\hookrightarrow}'', \cdots, \overset{n}{\hookrightarrow}'', \lambda''_m, \lambda''_d, \overset{1}{\tau}'', \cdots, \overset{m}{\tau}'' \rangle$ be two unified structures. Let R_C be a consistency relation between $\mathcal{US}'$ and $\mathcal{US}''$.*

If $(a, b) \in R_C$ and there exists a relation $R_x \subseteq \mathrm{ME}' \times \mathrm{ME}''$ such that $\forall (e', e'') \in R_x, e' \prec' a \wedge e'' \prec'' b$,then R_x is a consistency relation between $\mathcal{US}'$ and $\mathcal{US}''$.

The theorem shows that the consistency relation is preserved under containment relations.

Example 7. *In Figure 1, there exists the consistency relation $R_C = \{(loading, Fr1)\}$ between $\mathcal{US}_{seq}$ and $\mathcal{US}_{smd}$ by consistency rule CR3 in Table 1.*

In the sequence diagram $\mathcal{US}_{seq}$, there are the messages $wait, stream$ satisfying $wait \prec'' Fr1, stream \prec'' Fr1$. Similarly, the state machine diagram $\mathcal{US}_{smd}$ has two actions $wait, stream$ that satisfy $wait \prec''' loading, stream \prec''' loading$. By Theorem 3, there exists the consistency relation $R_x = \{(wait, wait), (stream, stream)\}$ between $\mathcal{US}_{seq}$ and $\mathcal{US}_{smd}$.

Next, we discuss the composition of consistencies.

Theorem 4. *Let $\mathcal{US}_1$, $\mathcal{US}'_1$, $\mathcal{US}_2$, $\mathcal{US}'_2$ be four unified structures. Let $\mathcal{US}_1, \mathcal{US}'_1$ be composable and $\mathcal{US}_2, \mathcal{US}'_2$ be composable. If R_C be a consistency relation between $\mathcal{US}_1$ and $\mathcal{US}_2$, then*

(1) R_C be a consistency relation between $\mathcal{US}_1 \uplus \mathcal{US}'_1$ and $\mathcal{US}_2 \uplus \mathcal{US}'_2$,

(2) R_C be a consistency relation between $\mathcal{US}_1$ and $\mathcal{US}_2 \uplus \mathcal{US}'_2$, and

(3) R_C be a consistency relation between $\mathcal{US}_1 \uplus \mathcal{US}'_1$ and $\mathcal{US}_2$.

Proposition 3 shows that types of dependency relations and elements during composition between unified structures are incremental. Thus, the consistency relation between models before composition remains.

Example 8. *In Figure 1, there is a consistency relation* $R_{C1} = \{(D, d), (S, s)\}$ *between* $\mathcal{US}_{cla}$ *and* $\mathcal{US}_{seq}$ *according to rule CR1 in Table 1. As is stated by Proposition 3, the composition of unified structures remains unified structure. Let a unified structure* $\mathcal{US}' = \mathcal{US}_{seq} \uplus \mathcal{US}_{smd}$. *By Definition 4,* R_{C1} *is consistency relation between* $\mathcal{US}_{cla}$ *and* $\mathcal{US}'$ *as well. Analogously, Theorem 4 (3) is shown.*

In our method, the consistency relations are equivalent only if both ends of the consistency relation are model elements. Those non-type sets of the unified structure express implicit and explicit relationships between model elements.

Definition 7. *Let* R_{C1}, R_{C2} *be two consistency relations.* R_{C1}, R_{C2} *are* equivalent*, denoted by* $R_{C1} \leftrightsquigarrow R_{C2}$ *iff both* R_{C1} *and* R_{C2} *can be decomposed into the same* atomic *consistency relations.*

The equivalent consistency relations share identical *atomic* consistency relations.

Proposition 7. *Let* R_{C1}, R_{C2}*, and* R_{C3} *be three consistency relations.*

(1) $R_{C1} \leftrightsquigarrow R_{C1}$.

(2) $R_{C1} \leftrightsquigarrow R_{C2} \Rightarrow R_{C2} \leftrightsquigarrow R_{C1}$.

(3) $R_{C1} \leftrightsquigarrow R_{C2} \Rightarrow R_{C1} \cup R_{C3} \leftrightsquigarrow R_{C2} \cup R_{C3}$.

This proposition states that the equivalent consistency relations have idempotence and commutativity. Furthermore, the equivalence between the consistency relations preserves if they have the same operations.

4 Tool

We have developed an experimental tool(See Figure 2). All diagrams are stored in the JSON document form and can be exported in the PNG file form. The tool allows consistency management concerning two main phases: inconsistency detection and inconsistency repair. The outcomes of each detection trigger the generation of repairs for the corresponding model. Moreover, we have implemented the composition of unified structures and equivalence of consistency relations. More functions will be added to the tool step by step.

5 Related Work

A precise semantics for overall UML diagrams have drawn great attentions. Most of the efforts focus on formalizing commonly used UML diagrams. For instance, Lu et al. [12] analyse the behavior aspects of the UML sequence diagram by a trace of properties to check and reason consistency. Based on the transformation into Labeled Transition Systems (LTS), Lambolais et.al [10] propose a framework to combine refinement and extension developments, and then they check for consistency in the incremental process using the accept set (as a special failure trace semantics [16]). The semantic models mentioned above are behavior ones with not good semantics. Instead, we present a new formal model called unified structure which can characterize not only all the UML behavioral diagrams, but also all the UML structural diagrams. The conventional construction for the semantics of behavioral UML diagrams mostly resorts to the path formed by the transformations. In contrast, a unified structure views the behavior (i.e., the path) as a set of *execution steps* which are graphically represented by a transformation between two elements in the diagram.

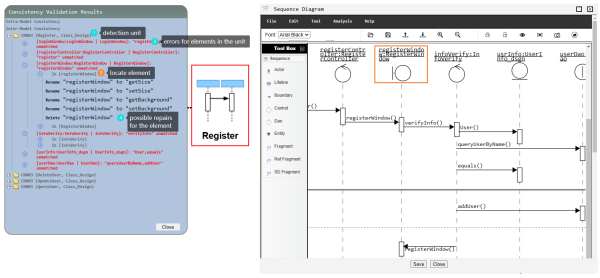

Figure 2. Inconsistencies detection

Consistency checking is promising research work in UML, and the techniques are widely discussed. Egyed [7] proposes an informal way of using rule instances to check consistency instantly and efficiently. He further discusses an incremental method to perform consistency checking in [6]. Formal approaches to check consistency are extensively reviewed. For example, Xu et al. [21] check the context's incremental consistency when constraints are represented using a First Order Logic formula, which addresses stopping the system's aberrant behavior. Based on the transformation of UML class diagram detailed in Object Constraint Language (OCL) into a Constraint Satisfaction Problem, Cabot et al. [3] resort the automatic solver to check properties. OCL extends UML diagrams by textual constraints but it limits the expressiveness of UML. And Campbell et al. [4] proposed an intermediate step for integrating UML diagrams into a formal framework to recognize discrepancies between the system's structure and behavior. Therefore, transforming UML diagrams into formal semantics differentiates among existing literature, which leads to different consistency checking methods.

Motivated by Egyed [7, 6], we apply consistency rules for efficient and accurate inconsistency detection. The work

of collecting UML consistency rules is ongoing, which comes from the work of Torre et al [18]. A total of 116 consistency rules were systematically documented among 10 of the 14 types of UML diagrams. The network of consistencies brings the complexity of formal verification. Klare and Heiko [9] decompose consistency relations on model transformation by extracting a tree from a reduced consistency relation graph. Our strategy, in contrast, is to allow the decomposition of consistency relations between particular models. This decomposition offers flexibility to depict consistencies. We further introduce a consistency equivalence for simplification and reduction.

6 Conclusion

In this paper, we have introduced a new formal model called unified structure. It represents UML diagrams and enables tooling and analysis for consistencies between UML diagrams. Based on the unified structure model, we have discussed the composition, decomposition, and equivalence of consistency between UML models. This paper aims to define formal consistency relations between UML diagrams. Such consideration provides a foundation for conflict checking of consistency rules.

Threats to validity come from two main aspects. On the one hand, it is vital to observe that constraints and models in our case are limited. On the other hand, consistency relations require the professional to comprehend so that the relations can be used correctly.

In future work, we will first improve scalability of the tool to analyze UML consistency relations and their equivalence. Then we will conduct experiments in complicated scenarios to update and optimize our tool.

References

[1] J. Abualdenien and A. Borrmann. A meta-model approach for formal specification and consistent management of multi-LOD building models. *Adv. Eng. Informatics*, 40:135–153, 2019.

[2] M. N. Alanazi and D. A. Gustafson. Super state analysis for uml state diagrams. In *WRI CSIE*, volume 7, pages 560–565. IEEE, 2009.

[3] J. Cabot, R. Clarisó, and D. Riera. On the verification of UML/OCL class diagrams using constraint programming. *JSS*, 93:1–23, 2014.

[4] L. A. Campbell, B. H. C. Cheng, W. E. McUmber, and R. E. K. Stirewalt. Automatically detecting and visualising errors in UML diagrams. *Requirements Engineering*, 7(4):264–287, 2002.

[5] A. Egyed. Instant consistency checking for the uml. In *ICSE*, pages 381–390, 2006.

[6] A. Egyed. UML/Analyzer: A Tool for the Instant Consistency Checking of UML Models. In *ICSE*, pages 793–796. IEEE, 2007.

[7] A. Egyed. Automatically detecting and tracking inconsistencies in software design models. *IEEE TSE*, 37(2):188–204, 2010.

[8] Y. Hammal. A modular state exploration and compatibility checking of uml dynamic diagrams. In *AICCSA*, pages 793–800. IEEE, 2008.

[9] H. Klare. Multi-model consistency preservation. In *MoDELS*, pages 156–161. ACM, 2018.

[10] T. Lambolais, A.-L. Courbis, H.-V. Luong, and C. Percebois. IDF: A framework for the incremental development and conformance verification of UML active primitive components. *JSS*, 113:275–295, 2016.

[11] K. Lano. Formal specification using interaction diagrams. In *SEFM*, pages 293–304. IEEE, 2007.

[12] L. Lu and D.-K. Kim. Required behavior of sequence diagrams. *ACM TOSEM*, 23(2):1–28, 2014.

[13] D. A. Meedeniya, I. D., and I. Perera. Software Artefacts Consistency Management towards Continuous Integration: A Roadmap. *IJACSA*, 10(4), 2019.

[14] F. u. Muram, H. Tran, and U. Zdun. Systematic review of software behavioral model consistency checking. *CSUR*, 50(2):1–39, 2017.

[15] OMG. Unified Modeling Language, v2.5.1. https://www.omg.org/spec/UML/2.5.1/PDF. [Online; accessed 2017-12].

[16] H. Ponce de León, S. Haar, and D. Longuet. Conformance relations for labeled event structures. In *Tests and Proofs: 6th International Conference*, pages 83–98. Springer, 2012.

[17] P. Stünkel, H. König, Y. Lamo, and A. Rutle. Comprehensive Systems: A formal foundation for Multi-Model Consistency Management. *FAC*, 33(6):1067–1114, Dec. 2021.

[18] D. Torre, Y. Labiche, M. Genero, and M. Elaasar. A systematic identification of consistency rules for UML diagrams. *JSS*, 144:121–142, oct 2018.

[19] D. Torre, Y. Labiche, M. Genero, and et al. Uml consistency rules: a case study with open-source uml models. In *SEFM*, pages 130–140, 2020.

[20] H. Wen, J. Wu, J. Jiang, G. Tang, and Z. Hong. A Formal Approach for Consistency Management in UML Model. *IJSEKE*, 2023.

[21] C. Xu, S. C. Cheung, and W. K. Chan. Incremental consistency checking for pervasive context. In *ICSE*, pages 292–301. ACM, 2006.

[22] J. Yang, Q. Long, Z. Liu, and X. Li. A predicative semantic model for integrating uml models. In *ICTAC*, pages 170–186. Springer, 2005.

[23] X. Zhao, Q. Long, and Z. Qiu. Model checking dynamic uml consistency. In *ICFEM*, pages 440–459. Springer, 2006.