# A Formal Approach for Traceability Preservation in Software Development Process

Hao Wen[1,2], Jinzhao Wu[1,2], Jianmin Jiang[3](✉), Jianqing Li[3], and Zhong Hong[4]

[1] Chengdu Institute of Computer Applications, Chinese Academy of Sciences, Chengdu, China
wenhao21@mails.ucas.ac.cn, himrwujzh@aliyun.com

[2] University of Chinese Academy of Sciences, Beijing, China

[3] College of Software Engineering, Chengdu University of Information Technology, Chengdu, China
jjm@cuit.edu.cn, 1102418305@qq.com

[4] College of Mathematics and Informatics, Fujian Normal University, Fuzhou, China
fjfzhz@fjnu.edu.cn

**Abstract.** Traceability is the ability to trace the usage of artifacts during the software lifecycle process. Though the benefits of establishing a traceability software system have been widely recognized, it is difficult to be applied well in actual development. In this paper, we propose a new method for traceability preservation which may be used in the practical software development process. A formal model for the traceability of software artifacts is first presented, which consists of variable traceability relations, classification and version number controls. We then present the composition, restriction and refinement operations in the software development process. Next, the preservation of traceability under these three operations is discussed respectively. To demonstrate the effectiveness of our approach, we finally develop a prototype tool named Formalized Software Management System (FSMS).

**Keywords:** Traceability · Preservation · Formal method · UML

## 1 Introduction

Traceability software refers to a system where artifacts at each phase of the software lifecycle process can be traced by other artifacts, and is now considered as a representation of high-quality software [21,42,50]. For example, source code in the implementation phase needs to be traced to artifacts in the maintenance and testing phases, while it can also trace artifacts from previous phases, such as requirement and design phases.

This work is supported by National Key R&D Program of China (No. 2022YFB3305104), National Natural Science Foundation of China (Nos. 61772006, 61772004 and 12261027) and Scientific Research Foundation for Advanced Talents of Chengdu University of Information Technology (No. KYTZ202009).

During the development of complex software, traceability plays an important role in reducing maintenance cost and analyzing change impact, but it is difficult to achieve in practice. There are several major challenges as follows: (1) In the software lifecycle process, it is hard to represent traceability between large number of artifacts [50] in a non-formal language (see Fig. 1); (2) Since various types of artifacts and relations are involved in different software systems [12,42], the abstraction level of a formal model is not easy to determine; (3) The scalability of traceability models may be impacted as the software becomes progressively larger [12,36].

When dealing with the above challenges, the main approaches of most studies are as followed: (1) Abstraction of artifacts and relations at different phases [19,20,26,30]; (2) Establishment of relations between artifacts in non-adjacent phases, such as the link between source code and requirements [43]; (3) Recording of traceability using different storage structures, such as matrix [25] and hierarchical tree [37]. However, these existing approaches face some problems, such as the potential loss of information about the relations between artifacts in non-adjacent phases, and the large computational burden imposed by the operations of the matrix. To solve the above problems, we propose a formal model called a *structure model* [48] to describe traceability, which follows the actual software development process to directly establish different types of relations between artifacts.

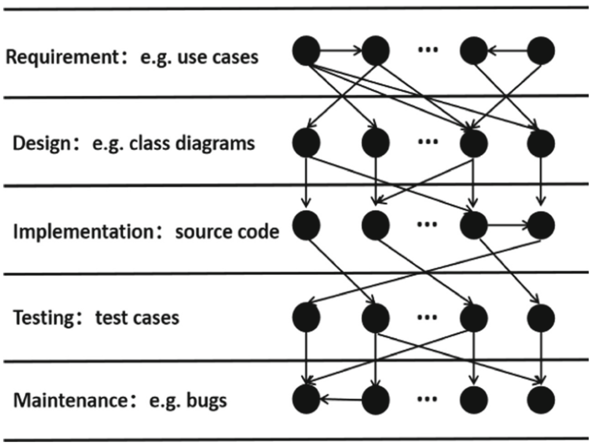
Requirement: e.g. use cases
Design: e.g. class diagrams
Implementation: source code
Testing: test cases
Maintenance: e.g. bugs
...

**Fig. 1.** Traceability links among artifacts in the software lifecycle process

In some early studies on traceability modeling (e.g. [32,42]), the types of traceability links are fixed, and then as the research progresses, most of the studies (e.g. [19,20,26]) consider abstracting similar types of links into more general relations between artifacts. However, these approaches are somewhat constrained when the types of traceability links in actual development are extended or modified. Furthermore, although some studies [4,18] achieve the extensibility and customization of traceability links, they do not support well the many-to-many relationship between two artifacts. Another interesting direction is to retrieve traceability links between source and target artifacts based on the probability [6,24]. In spite of the results showing that the majority of relevant artifacts can be retrieved, some incorrect links are also generated. Moreover, the visualization of traceability is one of the most significant aspects of modeling. Matrices, as two-dimensional structures, are commonly used in commercial tools because they can intuitively portray traceability and can be easily understood by non experts [12,33]. In addition, linear [44], hierarchical [37], graph-based [3,41] and cross-referenced [13,28] representations are also general methods. Nevertheless, the issues related to scalability have not been well addressed in previous work.

Instead of using dynamic behavior to study traceability systems, the structure model is based on the static system structure (a view similar to the structural models in UML [22] and SysML [39]), thus it avoids complex reachability algorithms when analyzing a software system. For convenience, we adopt a similar concept as in SysML [39], considering all the artifacts of different phases as *model elements* named with various labels in a structure model. Since the structure model does not limit the number of types of relations between model elements and supports the many-to-many relation, developers can automatically or semi-automatically assign relations between model elements according to the real development process using a prototyping tool called Formalized Software Management System (FSMS) where traceability can be simply visualized. Compared to the previous definition of the structure model [48], we introduce version number controls and delete the set $\lambda$ used for modeling constraints.

For a given structural model, composition and restriction are a kind of operations in horizontal direction, and refinement is a kind of operations in vertical direction [31]. However, to the best of our knowledge, only few studies [11,34,38] focus on whether traceability is preserved in different scenarios. This paper discusses in detail whether traceability is preserved under these three operations.

**Contribution.** This paper makes the following contributions.

- We propose a novel formal model named structure model to describe different types of artifacts and relations in the software development process. In contrast to classical formal models (such as Petri nets and transition systems) that describe systems from a behavioral perspective, our structure model can better represent traceability from the static structure of the system and do not need to use a complex reachability algorithm.
- We study three basic operations composition, restriction, and refinement between structure models, respectively. The results show that combination and refinement do not affect traceability, however restriction only preserves traceability in the vertical direction.
- To support our work, we develop a prototype tool which can visualize traceability.

The remainder of this paper is structured as follows. Section 2 introduces the structure model and presents the definition of traceability. Section 3 discusses the preservation of traceability based on three basic operations. Section 4 introduces our prototype tool. Section 5 is the related work. Section 6 concludes the paper and discusses the future work.

## 2 Traceability

In this section, we will introduce a formal model called a structure model [48] which is used to model and analyze traceability in the software development process. The structure model consists of model elements, variable traceability relationships between model elements, classifications and version numbers of model elements. Model elements are similar to those in SysML [39]. Based on the above concepts, developers can model and analyze traceability without diving deep into the specific implementation details of a software artifact.

In order to cope with the variable number of relation types between model elements, we choose to represent the structure model with a tuple of variable length. And the concept of version number is introduced in this model for the sake of subsequent analysis of traceability. Some assistant definitions are given below. Let **VN** be the set of version numbers such that $\forall R \subseteq \mathbf{VN} \times \mathbf{VN}$, $R$ is a strict total order. Note that $\infty$ is a special version number which means a *undetermined* version number, and $\infty \notin \mathbf{VN}$.

**Definition 1.** *A* ***structure model*** *($\mathcal{SM}$) is a tuple* $\langle \mathrm{ME}, \prec, \overset{1}{\hookrightarrow}, \cdots, \overset{n}{\hookrightarrow}, \overset{1}{\tau}, \cdots, \overset{m}{\tau}, vs, ve \rangle$ *with*

- $\mathrm{ME}$*, a finite set of the model elements,*
- $\prec \subseteq \mathrm{ME} \times \mathrm{ME}$*, the* containment relation *such that it is a (irreflexive) partial order,*
- $\forall i \in \{1, \cdots, n\}, \overset{i}{\hookrightarrow} \subseteq \mathrm{ME} \times \mathrm{ME}$*, the* dependency relation*,*
- $\forall j \in \{1, \cdots, m\}, \overset{j}{\tau} \subseteq \mathrm{ME}$*, the* type set *of model elements such that* $\forall e \in \mathrm{ME}, \exists \tau \in \{\overset{1}{\tau}, \cdots, \cdots, \overset{m}{\tau}\} : e \in \tau$.
- $vs : \mathrm{ME} \longrightarrow \mathbf{VN} \cup \{\infty\}$*, the* initial version function*, and*
- $ve : \mathrm{ME} \longrightarrow \mathbf{VN} \cup \{\infty\}$*, the* final version function*.*

A model element is assigned the initial version number when it is created, whereas it is assigned the final version number when it is inactivated. If the version number of a model element is not determined, it is assigned $\infty$.

In order to better understand the definition, we use the structure model to represent a simple student information management system in the following example.

*Example 1.* We here present the example shown in Fig. 2. In this example, the initial requirements of a student information management system are as follows.

- $R$: The system shall be able to manage students' information.
- $R1$: The system shall allow students to choose course.
- $R2$: The system shall allow students to check their course scores.

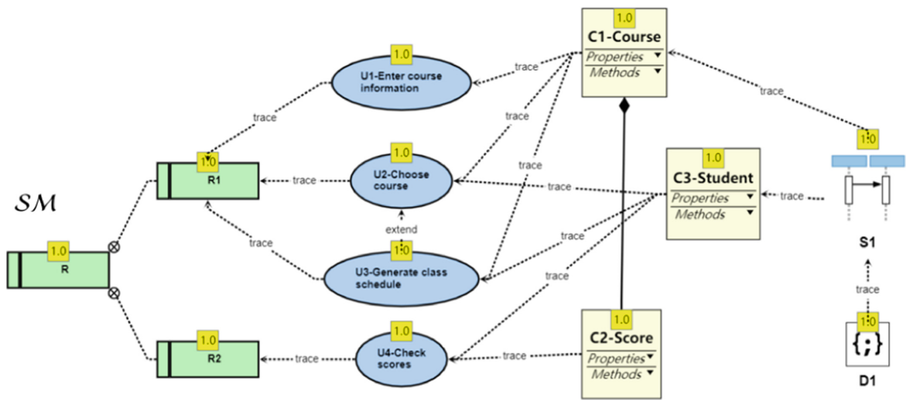

**Fig. 2.** The partial relations among software artifacts in a system

Obviously, the requirements $R1$ and $R2$ are both contained in $R$. In addition to requirements, the other model elements in Fig. 2 correspond to artifacts at different phases of software development. $U1$, $U2$, $U3$, $U4$ represent use cases, $C1$, $C2$, $C3$ denote classes, $S1$ denotes the sequence diagram, and $D1$ is the corresponding implementation code of $S1$. For convenience, the specific detail for each artifact is not given in Fig. 2.

There exist four types of relations between model elements: *containment, trace, extend, composition*. Note that the relations between these model elements are assigned by the software engineer. Thus the student information management system can be represented by a structure model $\mathcal{SM} = \langle \mathrm{ME}, \prec, \overset{trace}{\hookrightarrow}, \overset{extend}{\hookrightarrow}, \overset{composition}{\hookrightarrow}, \overset{requirements}{\mathcal{T}}, \overset{design}{\mathcal{T}}, \overset{implementation}{\mathcal{T}}, vs, ve\rangle$ with $\mathrm{ME} = \{R, R1, R2, U1, U2, U3, U4, C1, C2, C3, S1, D1\}$, $\prec = \{(R1, R), (R2, R)\}$, $\overset{trace}{\hookrightarrow} = \{(U1, R1), (U2, R1), (U3, R1), (U4, R2), (C1, U1), (C1, U2), (C1, U3), (C2, U4), (C3, U2), (C3, U3), (C3, U4), (S1, C1), (S1, C3), (D1, S1)\}$, $\overset{extend}{\hookrightarrow} = \{(U3, U2)\}$, $\overset{composition}{\hookrightarrow} = \{(C2, C1)\}$, $\overset{requirements}{\mathcal{T}} = \{R, R1, R2, U1, U2, U3, U4\}$, $\overset{design}{\mathcal{T}} = \{C1, C2, C3, S1\}$, $\overset{implementation}{\mathcal{T}} = \{D1\}$. Here, we may suppose that the current version number is 1.0, thus $\forall e \in \mathrm{ME}, vs(e) = 1.0$ and $ve(e) = \infty$.

This example shows that developers can visually represent the model elements and their relations using a structure model. Note that the version numbers are attributes of model elements.

Moreover, in order to make the paper more readable, we summarize some notations which will be used later (see Table 1).

**Definition 2.** *Let* $\mathcal{SM} = \langle \mathrm{ME}, \prec, \overset{1}{\hookrightarrow}, \cdots, \overset{n}{\hookrightarrow}, \overset{1}{\mathcal{T}}, \cdots, \overset{m}{\mathcal{T}}, vs, ve\rangle$ *be a structure model.*

*(1) A sequence* $rc = x_1 \cdots x_p (p > 1)$ *is called a* relation chain *in* $\mathcal{SM}$ *iff* $\forall i \in \{1, \cdots, p-1\}, x_i, x_{i+1} \in \mathrm{ME}$, $(x_i, x_{i+1}) \in (\prec \cup \overset{1}{\hookrightarrow} \cup \cdots \cup \overset{n}{\hookrightarrow}) \vee (x_{i+1}, x_i) \in (\prec \cup \overset{1}{\hookrightarrow} \cup \cdots \cup \overset{n}{\hookrightarrow})$. *$RC(\mathcal{SM})$ denotes all possible relation chains in $\mathcal{SM}$.*

**Table 1.** Notations.

| Symbol | Description |
|---|---|
| $\mathcal{SM}$ | A structure model |
| $rc$ | A relation chain in $\mathcal{SM}$ |
| $RC(\mathcal{SM})$ | A set contains all possible relation chains in $\mathcal{SM}$ |
| $dc$ | A dependency chain in $\mathcal{SM}$ |
| $DC(\mathcal{SM})$ | A set contains all possible dependency chains in $\mathcal{SM}$ |
| $\mathcal{SM}_1 \sqsubseteq \mathcal{SM}_2$ | $\mathcal{SM}_1$ is a substructure of $\mathcal{SM}_2$ |
| $\mathcal{SM}_1 \uplus \mathcal{SM}_2$ | The composition of $\mathcal{SM}_1$ and $\mathcal{SM}_2$ |
| $\mathcal{SM}\|_X$ | The restriction of $\mathcal{SM}$ to a given set $X$ |
| $ref(e)$ | A function that refines a model element $e$ |
| $\mathcal{SM}[e.ref(e)]$ | The refinement of $\mathcal{SM}$ under $ref(e)$ |

*(2) A sequence $dc = x_1 \cdots x_p (p > 1)$ is called a* dependency chain *in $\mathcal{SM}$ iff $\forall i \in \{1, \cdots, p-1\}, x_i, x_{i+1} \in$ 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 $dc = x_1 \cdots x_p$, that is, $\hat{dc} = \{x_1, \cdots, x_p\}$. $DC(\mathcal{SM})$ denotes all possible dependency chains in $\mathcal{SM}$. $[dc]$ denotes the number of model elements in the dependency chain dc, that is, $[dc] = p$.*

Clearly, a relation chain does not distinguish the direction of relations, while a dependency chain is a directed sequence. For instance, there exists a dependency chain $D1S1C3U3U2R1R$ in Example 1, and this dependency chain is also a relation chain.

**Proposition 1.** *If $\mathcal{SM}$ is a structure model, then $DC(\mathcal{SM}) \subseteq RC(\mathcal{SM})$.*

*Proof.* This proof is straightforward.

Proposition 1 states that the dependency chain is a special type of the relation chain.

Traceability is one of the significant criteria for assessing software quality [42]. Since there exist several different model elements (artifacts) for each phase of the software lifecycle process, and these model elements are largely isolated, it is necessary to correlate between various model elements through traceability [2]. Next, we will present some related concepts and give a formal definition of traceability.

The traceability of software systems can be classified as horizontal or vertical traceability [35,36,49]. The definition of traceability is as follows.

**Definition 3.** *Let* $\mathcal{SM} = \langle \mathrm{ME}, \prec, \overset{1}{\hookrightarrow}, \cdots, \overset{n}{\hookrightarrow}, \overset{1}{\tau}, \cdots, \overset{m}{\tau}, \upsilon s, \upsilon e\rangle$ *be a structure model and specify a system. Let* $\overset{requirements}{\tau} \in \{\overset{1}{\tau}, \cdots, \overset{m}{\tau}\}$ *and* $\overset{requirements}{\tau}$ *be the set of the model elements for representing requirements.*

*(1)* $\mathcal{SM}$ *is said to be* horizontally traceable *iff* $\forall e \in \mathrm{ME}, \exists dc = x_1 \cdots x_p \in DC(\mathcal{SM}) : e \in \hat{dc}$ *and* $x_p \in \overset{requirements}{\tau}$.
*(2)* $\mathcal{SM}$ *is said to be* vertically traceable *iff* $\forall e \in \mathrm{ME}, \exists dc = x_1 \cdots x_p \in DC(\mathcal{SM}) : e \in \hat{dc}$ *and* $\forall i \in \{1, \cdots, p-1\}, \upsilon s(x_{i+1}) \leq \upsilon s(x_i)$.
*(3)* $\mathcal{SM}$ *is said to be* traceable *iff* $\forall e \in \mathrm{ME}, \exists dc = x_1 \cdots x_p \in DC(\mathcal{SM}) : e \in \hat{dc}, x_p \in \overset{requirements}{\tau}$ *and* $\forall i \in \{1, \cdots, p-1\}, \upsilon s(x_{i+1}) \leq \upsilon s(x_i)$.

Here, horizontal traceability considers that all software artifacts directly or indirectly depend on requirements artifacts, whereas vertical traceability focuses on the version changes of model elements [43], that is, the version number of a model element is greater than or equal to that of its dependent model elements. Obviously, though dependency chains are directed, they can be reversely traversed based on directed graphs. Thus, forward traceability and backward traceability [5,51] are both contained in this definition. The definition considers not only the inner traceability of a model, but also the traceability among multiple models. Note that as long as there is an isolated artifact in a software system, the software system is not traceable.

*Example 2.* For each model element of the structure model $\mathcal{SM}$ in Fig. 2, there is a dependency chain containing this model element such that the elements of this chain are directly or indirectly related to requirements. Therefore according to Definition 3(1), $\mathcal{SM}$ is horizontally traceable. In addition, since all model elements in the structure model $\mathcal{SM}$ have the same version number, by Definition 3(2), $\mathcal{SM}$ is vertically traceable. Thus, by Definition 3(3), $\mathcal{SM}$ is traceable.

## 3 The Preservation of Traceability

In this section, we will introduce three common operations, and explore the preservation of traceability in the software development process.

A large-scale software project is often divided into several smaller projects which are developed concurrently in practice. A structure model is used to model and analyze the traceability of each project whichever is large or small. Thus, from the perspective of traceability, every project corresponds to a structure model. The decomposition of a complex software system into multiple subsystems can correspond to that of a structure model into multiple substructure models.

**Definition 4.** *Let* $\mathcal{SM}' = \langle \mathrm{ME}', \prec', \overset{1}{\hookrightarrow}', \cdots, \overset{n}{\hookrightarrow}', \overset{1}{\tau}', \cdots, \overset{m}{\tau}', \upsilon s', \upsilon e'\rangle$ *and* $\mathcal{SM}'' = \langle \mathrm{ME}'', \prec'', \overset{1}{\hookrightarrow}'', \cdots, \overset{n}{\hookrightarrow}'', \overset{1}{\tau}'', \cdots, \overset{m}{\tau}'', \upsilon s'', \upsilon e''\rangle$ *be two structure models.*

*A structure model* $\mathcal{SM}'$ *is called a* substructure *of* $\mathcal{SM}''$*, denoted as* $\mathcal{SM}' \sqsubseteq \mathcal{SM}''$*, iff* $\mathrm{ME}' \subseteq \mathrm{ME}'', \prec' \subseteq \prec''$*,* $\forall i \in \{1, \cdots, n\}, \overset{i}{\hookrightarrow}' \subseteq \overset{i}{\hookrightarrow}'', \forall j \in \{1, \cdots, m\} : \overset{j}{\tau}' \subseteq \overset{j}{\tau}''$ *and*

$\forall e \in \mathrm{ME}' : \upsilon s'(e) = \upsilon s''(e) \wedge \upsilon e'(e) = \upsilon e''(e)$. *A structure model $\mathcal{SM}'$ is called a* proper substructure *of $\mathcal{SM}''$, denoted as $\mathcal{SM}' \sqsubset \mathcal{SM}''$, iff $\mathcal{SM}' \sqsubseteq \mathcal{SM}''$ and $\mathcal{SM}' \neq \mathcal{SM}''$.*

**Proposition 2.** *Let $\mathcal{SM}'$ and $\mathcal{SM}''$ be two structure models. If $\mathcal{SM}' \sqsubseteq \mathcal{SM}''$, then $DC(\mathcal{SM}') \subseteq DC(\mathcal{SM}'')$.*

*Proof.* According to Definition 4, all model elements and relations of $\mathcal{SM}'$ are in $\mathcal{SM}''$. Therefore, all dependency chains of $\mathcal{SM}'$ are in $\mathcal{SM}''$. By Definition 2(2), $DC(\mathcal{SM}') \subseteq DC(\mathcal{SM}'')$.

## 3.1 Composition

Once all subsystems of a system are completed, all software artifacts in the subsystems should be composed. The model elements and relations in every subsystem should be preserved.

**Definition 5.** *Let $\mathcal{SM}' = \langle \mathrm{ME}', \prec', \overset{1}{\hookrightarrow}', \cdots, \overset{n}{\hookrightarrow}', \overset{1}{\tau}', \cdots, \overset{m}{\tau}', \upsilon s', \upsilon e' \rangle$ and $\mathcal{SM}'' = \langle \mathrm{ME}'', \prec'', \overset{1}{\hookrightarrow}'', \cdots, \overset{n}{\hookrightarrow}'', \overset{1}{\tau}'', \cdots, \overset{m}{\tau}'', \upsilon s'', \upsilon e'' \rangle$ be two structure models.*

*If $\forall e \in \mathrm{ME}' \cap \mathrm{ME}'', \upsilon s'(e) = \upsilon s''(e) \wedge \upsilon e'(e) = \upsilon e''(e)$, and $\prec' \cup \prec''$ is a (irreflexive) partial order, the composition of $\mathcal{SM}'$ and $\mathcal{SM}''$ is defined as $\mathcal{SM}' \uplus \mathcal{SM}'' = \langle \mathrm{ME}, \prec, \overset{1}{\hookrightarrow}, \cdots, \overset{n}{\hookrightarrow}, \overset{1}{\tau}, \cdots, \overset{m}{\tau}, \upsilon s, \upsilon e \rangle$ where $\mathrm{ME} = \mathrm{ME}' \cup \mathrm{ME}'', \prec = \prec' \cup \prec''$, $\forall i \in \{1, \cdots, n\}$: $\overset{i}{\hookrightarrow} = \overset{i}{\hookrightarrow}' \cup \overset{i}{\hookrightarrow}''$, $\forall j \in \{1, \cdots, m\}$ :$\overset{j}{\tau} = \overset{j}{\tau}' \cup \overset{j}{\tau}''$, $\forall e \in \mathrm{ME}' : \upsilon s(e) = \upsilon s'(e) \wedge \upsilon e(e) = \upsilon e'(e)$, and $\forall e \in \mathrm{ME}'' : \upsilon s(e) = \upsilon s''(e) \wedge \upsilon e(e) = \upsilon e''(e)$. $\mathcal{SM}'$, $\mathcal{SM}''$ are said to be* composable.

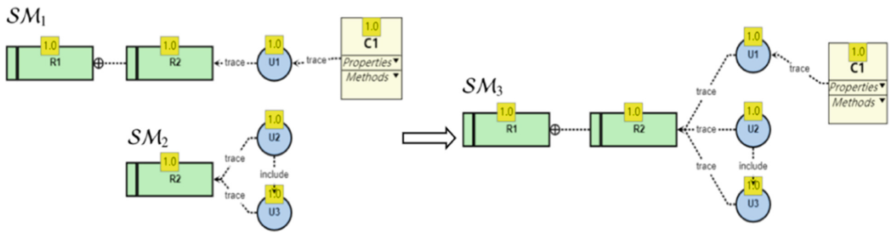

**Fig. 3.** The composition of two structure models

Here, when there are different types of relations between model elements in two composable structure models, we need to equivalently translate them into the two structure models which have the same relations before composition. For instance, in Fig. 3, there exist two types of relations in $\mathcal{SM}_1$: *containment* and *trace*, while there exist two types of relations in $\mathcal{SM}_2$: *trace* and *include*. $\mathcal{SM}_1$ and $\mathcal{SM}_2$ can be translated into the following two structure models $\mathcal{SM}'$ and $\mathcal{SM}''$, respectively.

$$\mathcal{SM}' = \langle \mathrm{ME}', \prec', \overset{trace}{\hookrightarrow'}, \overset{include}{\hookrightarrow'}, \overset{requirements}{\tau'}, \overset{design}{\tau'}, vs', ve' \rangle \text{ with } \overset{include}{\hookrightarrow'} = \emptyset,$$

$$\mathcal{SM}'' = \langle \mathrm{ME}'', \prec'', \overset{trace}{\hookrightarrow''}, \overset{include}{\hookrightarrow''}, \overset{requirements}{\tau''}, \overset{design}{\tau''}, vs'', ve'' \rangle \text{ with } \prec'' = \emptyset.$$

Obviously, $\mathcal{SM}_1 = \mathcal{SM}'$ and $\mathcal{SM}_2 = \mathcal{SM}''$. Thus $\mathcal{SM}_1$ and $\mathcal{SM}_2$ have the same type of relations after translation. Note that we need to perform a similar translation when composing two structure models with different types of classifications. By Definition 5, $\mathcal{SM}_1$ and $\mathcal{SM}_2$ are composable. The structure model $\mathcal{SM}_3$ is the composition of $\mathcal{SM}_1$ and $\mathcal{SM}_2$ in Fig. 3.

**Proposition 3.** *Let $\mathcal{SM}$, $\mathcal{SM}'$ and $\mathcal{SM}''$ be three structure models. And let every two of the three structure models be composable. Then*

*(1) $\mathcal{SM}' \uplus \mathcal{SM}''$ is a structure model.*
*(2) $\mathcal{SM}' \uplus \mathcal{SM}'' = \mathcal{SM}'' \uplus \mathcal{SM}'$.*
*(3) $(\mathcal{SM} \uplus \mathcal{SM}') \uplus \mathcal{SM}'' = \mathcal{SM} \uplus (\mathcal{SM}' \uplus \mathcal{SM}'')$.*

*Proof.* This proof is straightforward.

Proposition 3 states that the composition of structure models has closure, commutativity and associativity.

**Theorem 1.** *Let $\mathcal{SM}, \mathcal{SM}'$ be two composable structure models. If $\mathcal{SM}$ and $\mathcal{SM}'$ are traceable, then $\mathcal{SM} \uplus \mathcal{SM}'$ is traceable.*

*Proof.* Assume that $\exists \mathcal{SM}'' = \mathcal{SM} \uplus \mathcal{SM}' = \langle \mathrm{ME}'', \prec'', \overset{1}{\hookrightarrow''}, \cdots, \overset{n}{\hookrightarrow''}, \overset{1}{\tau''}, \cdots, \overset{m}{\tau''}, vs'', ve'' \rangle$ and $\overset{requirements}{\tau''} \in \{\overset{1}{\tau''}, \cdots, \overset{m}{\tau''}\}$. By Definition 5, $\forall e \in \mathrm{ME}'' : e \in \mathrm{ME} \vee e \in \mathrm{ME}'$. When $e \in \mathrm{ME}$, since $\mathcal{SM}$ is traceable, according to Definition 3 (3), $\exists dc_1 = x_1 \cdots x_p \in DC(\mathcal{SM}) : e \in \widehat{dc}_1, x_p \in \overset{requirements}{\tau}$ and $\forall i \in \{1, \cdots, p-1\}, vs(x_{i+1}) \leq vs(x_i)$. When $e \in \mathrm{ME}'$, since $\mathcal{SM}'$ is traceable, according to Definition 3 (3), $\exists dc_2 = y_1 \cdots y_q \in DC(\mathcal{SM}') : e \in \widehat{dc}_2, y_q \in \overset{requirements}{\tau}$ and $\forall i \in \{1, \cdots, q-1\}, vs'(y_{i+1}) \leq vs'(y_i)$. By Proposition 2, $\mathcal{SM} \sqsubseteq \mathcal{SM}'' \wedge \mathcal{SM}' \sqsubseteq \mathcal{SM}'' : DC(\mathcal{SM}) \subseteq DC(\mathcal{SM}'') \wedge DC(\mathcal{SM}') \subseteq DC(\mathcal{SM}'')$. Moreover, $vs''(e) = vs(e) \vee vs''(e) = vs'(e)$, thus $\exists dc_3 = z_1 \cdots z_k \in DC(\mathcal{SM}'') : e \in \widehat{dc}_3, z_k \in \overset{requirements}{\tau}$ and $\forall i \in \{1, \cdots, k-1\}, vs''(z_{i+1}) \leq vs''(z_i)$. By Definition 3 (3), $\mathcal{SM} \uplus \mathcal{SM}'$ is traceable.

This theorem shows that the composition of two traceable structure models is traceable.

## 3.2 Restriction

In Sect. 3.1, we have introduced the composition operation which describes a structure model becoming increasingly large and complex. By contrary, there exists some research on model decomposition [1,9,45], which can automatically or semi-automatically obtain sub-models. Thus, we next discuss how to decompose a structure model by restriction.

**Definition 6.** *Let* $\mathcal{SM} = \langle \mathrm{ME}, \prec, \overset{1}{\hookrightarrow}, \cdots, \overset{n}{\hookrightarrow}, \overset{1}{\tau}, \cdots, \overset{m}{\tau}, vs, ve\rangle$ *be a structure model. For* $X \subseteq \mathrm{ME}$, *the* restriction *of* $\mathcal{SM}$ *to* $X$ *is defined as a structure model* $\mathcal{SM}|_X = \langle \mathrm{ME}', \prec', \overset{1}{\hookrightarrow}', \cdots, \overset{n}{\hookrightarrow}', \overset{1}{\tau}', \cdots, \overset{m}{\tau}', vs', ve'\rangle$ *with*

- $\mathrm{ME}' = X$,
- $\prec' = \{(x, y) | \forall x, y \in X, (x, y) \in \prec\}$,
- $\forall i \in \{1, \cdots, n\} : \overset{i}{\hookrightarrow}' = \{(x, y) | \forall x, y \in X, (x, y) \in \overset{i}{\hookrightarrow}\}$ ,
- $\forall j \in \{1, \cdots, m\} : \overset{j}{\tau}' = X \cap \overset{j}{\tau}$ ,
- $\forall x \in X, vs'(x) = vs(x)$, *and*
- $\forall x \in X, ve'(x) = ve(x)$.

Here, we can obtain various substructures of a structure model by restriction.

**Proposition 4.** *Let* $\mathcal{SM} = \langle \mathrm{ME}, \prec, \overset{1}{\hookrightarrow}, \cdots, \overset{n}{\hookrightarrow}, \overset{1}{\tau}, \cdots, \overset{m}{\tau}, vs, ve\rangle$ *be a structure model. Then* $\forall X \subseteq \mathrm{ME}, \mathcal{SM}|_X \sqsubseteq \mathcal{SM}$.

*Proof.* According to Definition 4 and Definition 6, the result obviously holds.

The restriction of a structure model must be the substructure of the original structure model, but the substructure of a structure model may not be the restriction of the original model. For example, both $\mathcal{SM}_2$ and $\mathcal{SM}_3$ are obviously proper substructures of $\mathcal{SM}_1$ (see Fig. 4) and thus $\mathcal{SM}_2 \sqsubset \mathcal{SM}_1$, $\mathcal{SM}_3 \sqsubset \mathcal{SM}_1$. Then $\mathcal{SM}_2$ is obviously a restriction of $\mathcal{SM}_1$ to $X = \{R, U1, U2\}$, but $\mathcal{SM}_3$ is not $\mathcal{SM}_1|_X$.

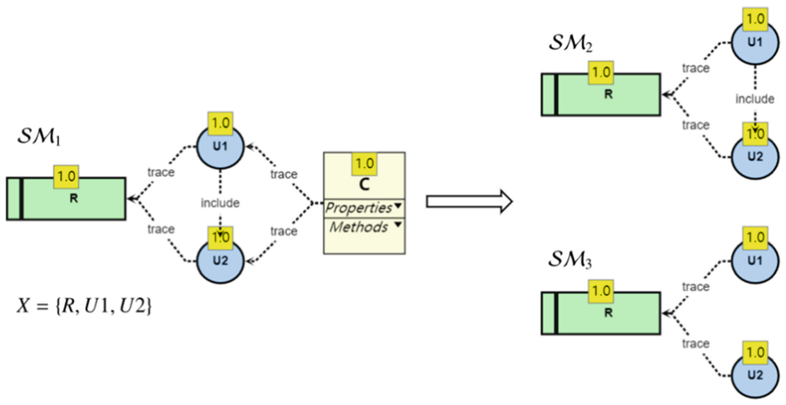

**Fig. 4.** An example for explaining the restriction of a structure model

**Proposition 5.** *Let* $\mathcal{SM} = \langle \mathrm{ME}, \prec, \overset{1}{\hookrightarrow}, \cdots, \overset{n}{\hookrightarrow}, vs, ve\rangle$ *be a structure model. If* $\forall X, Y \subseteq \mathrm{ME} : \mathcal{SM}|_X \uplus \mathcal{SM}|_Y = \mathcal{SM}$, *then* $X \cup Y = \mathrm{ME}$.

*Proof.* According to Definition 5 and Definition 6, the result clearly holds.

This proposition shows that when two substructures are composed to obtain the original structure model, the two substructures necessarily contain all the model elements of the original structure model.

**Theorem 2.** *Let* $\mathcal{SM} = \langle \mathrm{ME}, \prec, \overset{1}{\hookrightarrow}, \cdots, \overset{n}{\hookrightarrow}, \overset{1}{\tau}, \cdots, \overset{m}{\tau}, \upsilon s, \upsilon e\rangle$ *be a structure model and* $\mathcal{SM}$ *be traceable. Then* $\forall X \subseteq \mathrm{ME}$*,* $\mathcal{SM}|_X$ *is vertically traceable.*

*Proof.* This proof is straightforward.

Theorem 2 states that vertical traceability can be preserved after the restriction. However, when the restricted structure model contains isolated elements or does not contain requirements, horizontal traceability is obviously influenced.

## 3.3 Refinement

Refinement is another important operation during the software development process. It means that a model at higher abstraction level is transformed into the corresponding concrete one at lower abstraction level. Many researchers have studied the refinement operation based on different models [1,14,46]. We here investigate the refinement operation between structure models in this subsection.

We assume a fixed set **ME** of model elements. Let **SM** denotes the set of all structure models. The empty structure model $\langle \emptyset, \emptyset, \emptyset, \cdots, \emptyset, \emptyset, \emptyset\rangle$ is denoted by $\varnothing$.

**Definition 7.** *(1) A refinement function for structure models is a total function* $ref$ : $\boldsymbol{ME} \rightarrow \boldsymbol{SM}\backslash\{\varnothing\}$ *such that* $\forall e \in \boldsymbol{ME}, ref(e) = \langle \mathrm{ME}, \prec, \overset{1}{\hookrightarrow}, \cdots, \overset{n}{\hookrightarrow}, \overset{1}{\tau}, \cdots, \overset{m}{\tau}, \upsilon s, \upsilon e\rangle$ *where*

- $\exists e' \in \mathrm{ME}$, $e'$ *is a copy of* $e$, $\prec = \{(x, e') \mid x \in \mathrm{ME} \setminus \{e'\}\}$*, and*
- $\forall x, y \in \mathrm{ME}, \upsilon s(x) = \upsilon s(y) \wedge \upsilon e(x) = \upsilon e(y)$.

*(2) Let* $\mathcal{SM} = \langle \mathrm{ME}, \prec, \overset{1}{\hookrightarrow}, \cdots, \overset{n}{\hookrightarrow}, \overset{1}{\tau}, \cdots, \overset{m}{\tau}, \upsilon s, \upsilon e\rangle$ *be a structure model. Let* $e \in \mathrm{ME}$*,* $ref(e) = \langle \mathrm{ME}', \prec', \overset{1}{\hookrightarrow}', \cdots, \overset{n}{\hookrightarrow}', \overset{1}{\tau}', \cdots, \overset{m}{\tau}', \upsilon s', \upsilon e'\rangle$ *such that* $\forall x \in \mathrm{ME}', \upsilon s'(x) \geq \upsilon s(e) \wedge \upsilon e'(x) \geq \upsilon e(e)$*. Moreover, let* $\mathrm{ME} \cap \mathrm{ME}' = \emptyset$*. The* refinement *of* $\mathcal{SM}$ *under* $ref(e)$ *is the structure model* $\mathcal{SM}[e.ref(e)] = \langle \mathrm{ME}'', \prec'', \overset{1}{\hookrightarrow}'', \cdots, \overset{n}{\hookrightarrow}'', \overset{1}{\tau}'', \cdots, \overset{m}{\tau}'', \upsilon s'', \upsilon e''\rangle$ *with*

- $\mathrm{ME}'' = \mathrm{ME} \cup \mathrm{ME}'$,
- $\prec'' = \prec \cup \prec'$,
- $\exists i \in \{1, \cdots, n\} : \overset{i}{\hookrightarrow}'' = \overset{refine}{\hookrightarrow}'' = \overset{refine}{\hookrightarrow} \cup \overset{refine}{\hookrightarrow}' \cup \{(e', e)\}$,
- $\forall j \in \{1, \cdots, n\}\backslash\{i\} : \overset{j}{\hookrightarrow}'' = \overset{j}{\hookrightarrow} \cup \overset{j}{\hookrightarrow}'$,
- $\forall k \in \{1, \cdots, m\} : \overset{k}{\tau}'' = \overset{k}{\tau} \cup \overset{k}{\tau}'$,
- $\forall p \in \mathrm{ME}' : \upsilon s''(p) = \upsilon s'(p) \wedge \upsilon e''(p) = \upsilon e'(p)$*, and*

- $\forall q \in \mathrm{ME} : vs''(q) = vs(q) \wedge ve''(q) = ve(q)$.

Definition 7 shows that if the model element of a structure model is refined, then the newly obtained structure model is the refinement of the original one. In order not to pollute the relations between model elements in the original structure model [7,40], we choose to first copy the model element to be refined and then create new relations based on the replica. Thus the refinement operation includes the following steps:

(1) copy $e$ as the new model element $e'$.
(2) assign $e'$ a new version number, then create the *containment* relation between $e'$ and all model elements of $ref(e)$ respectively (although the model elements of $ref(e)$ are a more concrete representation of $e$, they may be regarded as descendants of $e$).
(3) establish the *refine* relation between $e$ and $e'$.

The refinement operation between structure models can be clearly represented graphically. For distinguishing between different version numbers, once a model element is assigned to a new version number, the old version number will become grey in our tool.

For example, we assume that the model element $U1$ of $\mathcal{SM}_1$ needs to be refined in Fig. 5 and there exists a structure model $\mathcal{SM}_2 = ref(U1)$, then the refinement of $\mathcal{SM}_1$ under $ref(U1)$ can be represented by a new structure model $\mathcal{SM}_3 = \mathcal{SM}_1[U1.ref(U1)]$.

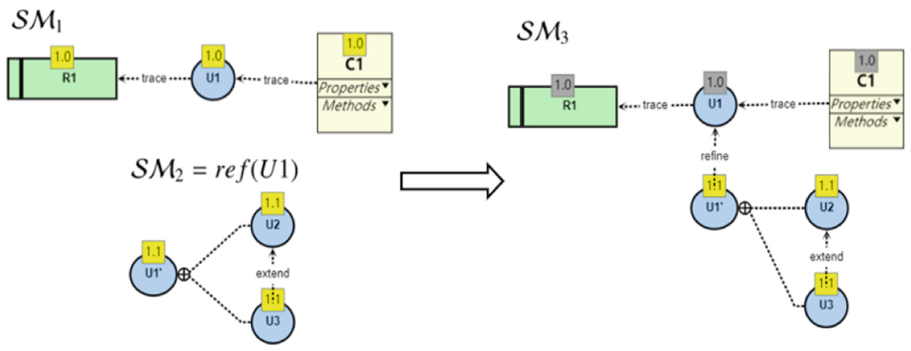

**Fig. 5.** An example for interpreting the refinement of a structure model

**Proposition 6.** *Let* $\mathcal{SM}_1 = \langle \mathrm{ME}_1, \prec_1, \overset{1}{\hookrightarrow}_1, \cdots, \overset{n}{\hookrightarrow}_1, \overset{1}{\tau}_1, \cdots, \overset{m}{\tau}_1, vs_1, ve_1 \rangle$ *be a structure model. Moreover, let* $ref$ *be a refinement function for structure models. If* $\forall e \in \mathrm{ME}_1 : \mathcal{SM}_2 = \mathcal{SM}_1[e.ref(e)]$, *then* $\mathcal{SM}_1 = \mathcal{SM}_2|_{\mathrm{ME}_1}$.

*Proof.* Since $\mathcal{SM}_2$ is a refinement of $\mathcal{SM}_1$ under $ref(e)$, by Definition 7 (2), all model elements and relations in $\mathcal{SM}_1$ are not change. Assume $X = \mathrm{ME}_1$, since $X \subseteq \mathrm{ME}_2$, by Definition 6, $\mathcal{SM}_1$ is a restriction of $\mathcal{SM}_2$ to $X$. Thus $\mathcal{SM}_1 = \mathcal{SM}_2|_{\mathrm{ME}_1}$.

This proposition states that if a structure model is refined, then the newly obtained structure model preserves all the model elements of the original structure model.

**Theorem 3.** *Let* $\mathcal{SM}_1 = \langle \mathrm{ME}_1, \prec_1, \overset{1}{\hookrightarrow}_1, \cdots, \overset{n}{\hookrightarrow}_1, \overset{1}{\tau}_1, \cdots, \overset{m}{\tau}_1, vs_1, ve_1 \rangle$ *be a structure model and* $\mathcal{SM}_1$ *be traceable. Moreover, let* $ref$ *be a refinement function for structure models. If* $\forall e \in \mathrm{ME}_1 : \mathcal{SM}_2 = \mathcal{SM}_1[e.ref(e)]$, *then* $\mathcal{SM}_2$ *is traceable.*

*Proof.* Assume that $\mathcal{SM}_2 = \langle \mathrm{ME}_2, \prec_2, \overset{1}{\hookrightarrow}_2, \cdots, \overset{n}{\hookrightarrow}_2, \overset{1}{\tau}_2, \cdots, \overset{m}{\tau}_2, vs_2, ve_2 \rangle$. Since $\mathcal{SM}_2$ is a refinement of $\mathcal{SM}_1$ under $ref(e)$, by Definition 7, $\forall x \in \mathrm{ME}_2 : x \in \mathrm{ME}_1 \vee x \in \mathrm{ME}_2 \setminus \mathrm{ME}_1$. When $x \in \mathrm{ME}_1$, since $\mathcal{SM}_1$ is traceable, by Definition 3 (3), $\exists dc_1 = x_1 \cdots x_p \in DC(\mathcal{SM}_1) : x \in \hat{dc_1}, x_p \in \overset{requirements}{\tau}$ and $\forall i \in \{1, \cdots, p-1\}, vs_1(x_{i+1}) \leq vs_1(x_i)$. Moreover, by Proposition 4 and Proposition 6, $\mathcal{SM}_1 \sqsubseteq \mathcal{SM}_2$. According to Proposition 2, $DC(\mathcal{SM}_1) \subseteq DC(\mathcal{SM}_2)$, thus $dc_1 \in DC(\mathcal{SM}_2)$. When $x \in \mathrm{ME}_2 \setminus \mathrm{ME}_1$, by Definition 7, $\exists dc_2 = y_1 \cdots y_q \in DC(\mathcal{SM}_2) : y_q = e \wedge y_1 = x$ and $vs_2(e) \leq vs_2(x)$. Thus $\exists dc_3 = z_1 \cdots z_k \in DC(\mathcal{SM}_2) : x \in \hat{dc_3}, z_k \in \overset{requirements}{\tau}$ and $\forall i \in \{1, \cdots, k-1\}, vs_2(z_{i+1}) \leq vs_2(z_i)$. By Definition 3 (3), $\mathcal{SM}_2$ is traceable.

This theorem states that the refinement operation does not change traceability in the software development process.

## 4 Tool: Formalized Software Management System

In order to facilitate the investigation of traceability, a prototype tool named Formalized Software Management System (FSMS) was developed with JavaScript. This tool mainly consists of two independently developed modules (See Fig. 6 for the main interface). One is a draggable drawing modules with traceability management, the other is the consistency management module. This paper is dedicated to the former.

Developers can use different components in the toolbox to visualize the artifacts and the relations between them in the software system, and then the graphic representation of this system can be automatically converted to JSON format for storage. Moreover, this tool implements several analysis functions based on the structure model, such as change impact analysis. The tool has been deployed on the website http://124.220.63.75/ now. More functions will be added to this tool step by step.

## 5 Related Work

Software traceability has been studied for many years (see References [2,36,51] for surveys). Most of the studies (see References [4,8,15] for surveys) involving traceability do not apply formal methods, but our work is based on formal methods to rigorously model traceability and to analysis traceability. Therefore, we only provide some comparisons between our work and the most related work on traceability modeling and change impact analysis.

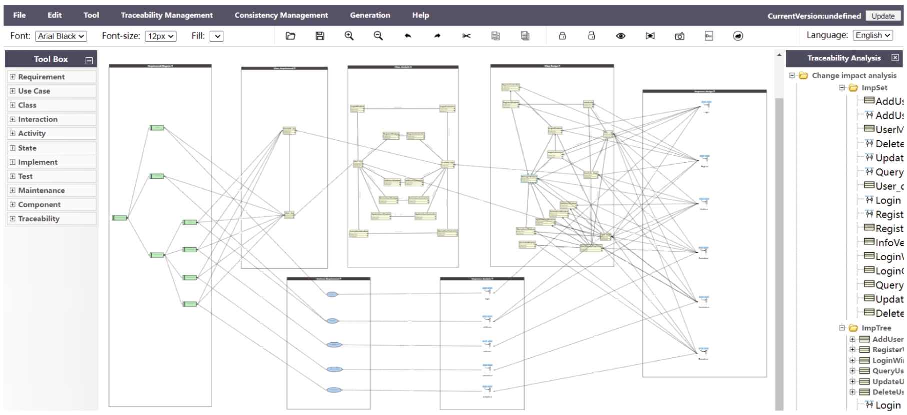

**Fig. 6.** The main interface of the prototype tool

Goknil et al. [19,20] mainly focus on the well-defined traceability between the requirement (R) and architecture (A), and they have investigated the reasoning, consistency checking and automatic generation of trace relations based on the system's specifications. Rempel and Mader [43] focus only on the relations between the requirement and implementation (code) phases. However, we hold that the artifacts in the following phases depend on those in the preceding phases, so traceability between non-adjacent or abstract phases may be less practical to lose some implicit details in software lifecycle process.

In [26], six types of traceability relations, which are used to analyze the evolution of the requirement and architecture, are introduced: goal dependency, temporal dependency, service dependency, conditional dependency, task dependency and infrastructure dependency. Lago et al. [30] give a scoped approach to identify core traceability paths and distinguish four relations in R&A: drive, depend-on, modify and influence. Based on the above two studies, Goknil et al. [20] generalize these trace types into within-model traces (refines, requires) and between-model traces (satisfies, allocatedTo). Spanoudakis et al. [47] propose a rule-based method for establishing traceability. This method contains two rules: requirements-to-object-model (RTOM) and inter-requirement traceability (IREQ). Then based on RTOM and IREQ, four different types of traceability relations can be generated between requirement statements and use cases. Cleland-Huang et al. [10] present a novel concept. They consider that the relations between various software artifacts as a direct or indirect link which can be dynamically modified by event notifications. Obviously, neither the different number of relations nor event-based links can be applied to a large-scale development with a very large variety of traceability relations between artifacts. However, our structure model does not limit the number of relation types.

Antoniol et al. [6] propose a seminal approach to retrieve traceability relations between documentation and code based on probability. Their work uses the information retrieval algorithm to calculate the similarity between two artifacts to establish links between them. However, the precision is not high enough, resulting in the possibility of generating incorrect links. In order to establish high quality links, De Lucia et al. [16] introduce a method to understand the semantic context between artifacts using Latent Semantic Indexing, and in [23,47], how to connect heuristic rules with link retrieval is discussed. Obviously, these methods can reduce the number of incorrect links, but cannot avoid the error. In [17], a platform called Tarskil is used to build the semantics of interactive traceability, which is based on first-order relational logic, thus simplifying the reasoning of relations and consistency checking. Santos et al. [44] use a tool named TIRT to show data with list, which is a kind of one-dimensional approach. However, these two researches mentioned above do not well support the many-to-many relation between artifacts. In Holten's work [25], he propose to store adjacency relations by a two-dimensional approach, such as traceability matrix. Although this approach is easily understood by the stakeholder and saves storage space, the change of traceability matrix becomes complicated when the structure of the system is changed.

Laghouaouta et al. [29] use the softgoal tree to manage the traceability of the composition of multi-view models in the Model Driven Engineering (MDE). In [7,27], some approaches to merging trackable links on-demanded are discussed. Obviously, these studies are completely different from our work.

## 6 Conclusion and Future Work

In this paper, we have investigated how to ensure traceability under the operations such as composition, restriction and refinement in software development process. To demonstrate the availability of these results, we have developed a prototype tool called FSMS that enables traceability visualization.

In future work, we will explore how to make the visualization of our tool better. We also wish to integrate our tool with other software development platforms. On the other hand, automated code generation is another important field of our research based on formal methods. Since automated code generation techniques heavily rely on the traceability, we will focus on how to automatically generate high quality code in the software development process.

**Acknowledgements.** We would like to thank the anonymous reviewers for their very valuable comments and very helpful suggestions.

## References

1. Abrial, J.R., Hallerstede, S.: Refinement, decomposition, and instantiation of discrete models: application to event-B. Fund. Inform. **77**(1–2), 1–28 (2007)
2. Aizenbud-Reshef, N., Nolan, B.T., Rubin, J., Shaham-Gafni, Y.: Model traceability. IBM Syst. J. **45**(3), 515–526 (2006)
3. van Amstel, M.F., van den Brand, M.G.J., Serebrenik, A.: Traceability visualization in model transformations with TraceVis. In: Hu, Z., de Lara, J. (eds.) ICMT 2012. LNCS, vol. 7307, pp. 152–159. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30476-7_10
4. Anquetil, N., et al.: A model-driven traceability framework for software product lines. Softw. Syst. Model. **9**(4), 427–451 (2010)

5. ANSI/IEEE: IEEE guide to software requirements specification, ANSI/IEEE std 830-1984 (1984)
6. Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., Merlo, E.: Recovering traceability links between code and documentation. IEEE Trans. Softw. Eng. **28**(10), 970–983 (2002)
7. Barbero, M., Fabro, M., Bézivin, J.: Traceability and provenance issues in global model management. In: 3rd ECMDA-Traceability Workshop (2007)
8. Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B., och Dag, J.N.: An industrial survey of requirements interdependencies in software product release planning. In: Proceedings Fifth IEEE International Symposium on Requirements Engineering. IEEE (2001). https://doi.org/10.1109/isre.2001.948547
9. Chen, H., Jiang, J., Hong, Z., Lin, L.: Decomposition of UML activity diagrams. Softw. Pract. Exp. **48**(1), 105–122 (2018)
10. Cleland-Huang, J., Chang, C., Christensen, M.: Event-based traceability for managing evolutionary change. IEEE Trans. Softw. Eng. **29**(9), 796–810 (2003). https://doi.org/10.1109/tse.2003.1232285
11. Cleland-Huang, J., Chang, C.K., Ge, Y.: Supporting event based traceability through high-level recognition of change events. In: Proceedings 26th Annual International Computer Software and Applications, pp. 595–600. IEEE (2002)
12. Cleland-Huang, J., Gotel, O.C., Huffman Hayes, J., Mäder, P., Zisman, A.: Software traceability: trends and future directions. In: Future of Software Engineering Proceedings, pp. 55–69 (2014)
13. Cuadrado, J.S., Molina, J.G., Tortosa, M.M.: RubyTL: a practical, extensible transformation language. In: Rensink, A., Warmer, J. (eds.) ECMDA-FA 2006. LNCS, vol. 4066, pp. 158–172. Springer, Heidelberg (2006). https://doi.org/10.1007/11787044_13
14. Cusack, E.: Refinement, conformance and inheritance. Form. Asp. Comput. **3**(2), 129–141 (1991). https://doi.org/10.1007/bf01898400
15. Dahlstedt, Å.G., Persson, A.: Requirements interdependencies: state of the art and future challenges. In: Aurum, A., Wohlin, C. (eds.) Engineering and Managing Software Requirements, pp. 95–116. Springer-Verlag, Heidelberg (2005). https://doi.org/10.1007/3-540-28244-0_5
16. De Lucia, A., Fasano, F., Oliveto, R., Tortora, G.: Enhancing an artefact management system with traceability recovery features. In: 2004 Proceedings of 20th IEEE International Conference on Software Maintenance, pp. 306–315. IEEE (2004)
17. Erata, F., Challenger, M., Tekinerdogan, B., Monceaux, A., Tüzün, E., Kardas, G.: Tarski: a platform for automated analysis of dynamically configurable traceability semantics. In: Proceedings of the Symposium on Applied Computing, pp. 1607–1614 (2017)
18. Espinoza, A., Garbajosa, J.: A study to support agile methods more effectively through traceability. Innov. Syst. Softw. Eng. **7**(1), 53–69 (2011)
19. Goknil, A., Kurtev, I., van den Berg, K., Veldhuis, J.W.: Semantics of trace relations in requirements models for consistency checking and inferencing. Softw. Syst. Model. **10**(1), 31–54 (2009). https://doi.org/10.1007/s10270-009-0142-3
20. Goknil, A., Kurtev, I., Berg, K.V.D.: Generation and validation of traces between requirements and architecture based on formal trace semantics. J. Syst. Softw. **88**, 112–137 (2014). https://doi.org/10.1016/j.jss.2013.10.006
21. Gotel, O., Finkelstein, C.: An analysis of the requirements traceability problem. In: Proceedings of IEEE International Conference on Requirements Engineering. IEEE Computer Society Press (1994). https://doi.org/10.1109/icre.1994.292398
22. Group, O.M.: Omg unified modeling language tm (OMG UML): Version 2.5. Needham: OMG (2015)

23. Guo, J., Cleland-Huang, J., Berenbach, B.: Foundations for an expert system in domain-specific traceability. In: 2013 21st IEEE International Requirements Engineering Conference (RE), pp. 42–51. IEEE (2013)
24. Hayes, J.H., Dekhtyar, A., Sundaram, S.K.: Advancing candidate link generation for requirements tracing: the study of methods. IEEE Trans. Softw. Eng. **32**(1), 4–19 (2006)
25. Holten, D.: Hierarchical edge bundles: visualization of adjacency relations in hierarchical data. IEEE Trans. Visual Comput. Graph. **12**(5), 741–748 (2006)
26. Shakil Khan, S., Greenwood, P., Garcia, A., Rashid, A.: On the impact of evolving requirements-architecture dependencies: an exploratory study. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 243–257. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69534-9_19
27. Kolovos, D.S., Paige, R.F., Polack, F.A.: On-demand merging of traceability links with models. In: 3rd ECMDA Traceability Workshop, pp. 47–55 (2006)
28. Kolovos, D.S., Paige, R.F., Polack, F.A.C.: The epsilon transformation language. In: Vallecillo, A., Gray, J., Pierantonio, A. (eds.) ICMT 2008. LNCS, vol. 5063, pp. 46–60. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69927-9_4
29. Laghouaouta, Y., Anwar, A., Nassar, M., Coulette, B.: A dedicated approach for model composition traceability. Inf. Softw. Technol. **91**, 142–159 (2017). https://doi.org/10.1016/j.infsof.2017.07.002
30. Lago, P., Muccini, H., van Vliet, H.: A scoped approach to traceability management. J. Syst. Softw. **82**(1), 168–182 (2009). https://doi.org/10.1016/j.jss.2008.08.026
31. Lambolais, T., Courbis, A.L., Luong, H.V., Percebois, C.: IDF: a framework for the incremental development and conformance verification of UML active primitive components. J. Syst. Softw. **113**, 275–295 (2016). https://doi.org/10.1016/j.jss.2015.11.020
32. Letelier, P.: A framework for requirements traceability in UML-based projects. In: Proceedings of the 1st International Workshop on Traceability in Emerging Forms of Software Engineering, pp. 30–41 (2002)
33. Li, Y., Maalej, W.: Which traceability visualization is suitable in this context? A comparative study. In: Regnell, B., Damian, D. (eds.) REFSQ 2012. LNCS, vol. 7195, pp. 194–210. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28714-5_17
34. Mäder, P., Gotel, O.: Towards automated traceability maintenance. J. Syst. Softw. **85**(10), 2205–2227 (2012)
35. Mäder, P., Gotel, O., Kuschke, T., Philippow, I.: Tracemaintainer-automated traceability maintenance. In: 2008 16th IEEE International Requirements Engineering Conference. pp. 329–330. IEEE (2008)
36. Meedeniya, D., Rubasinghe, I., Perera, I.: Traceability establishment and visualization of software artefacts in devops practice: a survey. Int. J. Adv. Comput. Sci. Appl. **10**(7), 66–76 (2019)
37. Merten, T., Jüppner, D., Delater, A.: Improved representation of traceability links in requirements engineering knowledge using sunburst and netmap visualizations. In: 2011 4th International Workshop on Managing Requirements Knowledge, pp. 17–21. IEEE (2011)
38. Murta, L.G., Van Der Hoek, A., Werner, C.M.: Archtrace: policy-based support for managing evolving architecture-to-implementation traceability links. In: 21st IEEE/ACM International Conference on Automated Software Engineering (ASE 2006), pp. 135–144. IEEE (2006)
39. OMG: Omg systems modeling language tm version 1.5. An OMG Systems Modeling Language TM Publication, May 2017. http://www.omg.org/spec/SysML/1.5/
40. Pavalkis, S., Nemuraitė, L., Butkienė, R.: Derived properties: a user friendly approach to improving model traceability. Inf. Technol. Control **42**(1), 48–60 (2013)
41. von Pilgrim, J., Vanhooff, B., Schulz-Gerlach, I., Berbers, Y.: Constructing and visualizing transformation chains. In: Schieferdecker, I., Hartman, A. (eds.) ECMDA-FA 2008.

LNCS, vol. 5095, pp. 17–32. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69100-6_2
42. Ramesh, B., Jarke, M.: Toward reference models for requirements traceability. IEEE Trans. Softw. Eng. **27**(1), 58–93 (2001)
43. Rempel, P., Mäder, P.: Preventing defects: the impact of requirements traceability completeness on software quality. IEEE Trans. Softw. Eng. **43**(8), 777–797 (2016)
44. Santos, W.B., de Almeida, E.S., Meira, S.R.: TIRT: a traceability information retrieval tool for software product lines projects. In: 2012 38th Euromicro Conference on Software Engineering and Advanced Applications, pp. 93–100. IEEE (2012)
45. Silva, R., Pascal, C., Hoang, T.S., Butler, M.: Decomposition tool for event-B. Softw. Pract. Exp. **41**(2), 199–208 (2011)
46. Smith, G., Derrick, J.: Specification, refinement and verification of concurrent systems-an integration of object-Z and CSP. Form. Methods Syst. Des. **18**(3), 249–284 (2001)
47. Spanoudakis, G., Zisman, A., Pérez-Miñana, E., Krause, P.: Rule-based generation of requirements traceability relations. J. Syst. Softw. **72**(2), 105–127 (2004). https://doi.org/10.1016/s0164-1212(03)00242-5
48. Wen, H., Wu, J., Jiang, J., Tang, G., Hong, Z.: A formal approach for consistency management in UML models. Int. J. Softw. Eng. Knowl. Eng. **33**(5), 733–763 (2023)
49. Wen, L., Tuffley, D., Dromey, R.G.: Formalizing the transition from requirements' change to design change using an evolutionary traceability model. Innov. Syst. Softw. Eng. **10**(3), 181–202 (2014). https://doi.org/10.1007/s11334-014-0230-6
50. Wiederseiner, C., Garousi, V., Smith, M.: Tool support for automated traceability of test/code artifacts in embedded software systems. In: 2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications. IEEE (2011). https://doi.org/10.1109/trustcom.2011.151
51. Winkler, S., von Pilgrim, J.: A survey of traceability in requirements engineering and model-driven development. Softw. Syst. Model. **9**(4), 529–565 (2010). https://doi.org/10.1007/s10270-009-0145-0