Development of a Software Tool to Support Traceability-Based Inspection of SOFL Specifications

Zhang, Jinghua

(出版者 / Publisher)
法政大学大学院情報科学研究科
(雑誌名 / Journal or Publication Title)
法政大学大学院紀要. 情報科学研究科編 / 法政大学大学院紀要. 情報科学研究科編
(巻 / Volume)
10
(開始ページ / Start Page)
1
(終了ページ / End Page)
6
(発行年 / Year)
2015-03-24
(URL)
https://doi.org/10.15002/00011545

Development of a Software Tool to Support Traceability-Based Inspection of SOFL Specifications

Jinghua Zhang
Graduate School of Computer and Information Sciences
Hosei University
E-mail: jinghua.zhang.6w@stu.hosei.ac.jp

***Abstract*—When developing a formal specification for a software project using the SOFL three-step modeling approach, it is essential to ensure the conformance relation between every two level specifications. Inspection is an important technique to verify the specifications. In this paper, we describe an inspection method through building traceability for rigorously verifying the conformance relation. The method consists of two steps: (1) traceability establishment and (2) inspection of the target specifications based on the built traceability. We also provide some inspection strategies such as checklists based on SOFL features to help the inspector find errors and keep the consistency. Our tool provides a convenient interface to separate components in different specifications and save their relationships to keep the consistency. We describe the design and implementation of our supporting tool in this thesis. A case study to inspect the specifications of a travel plan booking system is given to show how the proposed method can be applied in practice.**

***Index Terms*—SOFL, specification, traceability, inspection, conformance.**

I. INTRODUCTION

One of the primary problems in software projects is that the requirements documented in specifications may not be accurately and easily understood by the developers carrying out different tasks [1]. In general, requirements specifications need to be written by humans, and probably will be changed during the communication between customers and designers. Therefore, the target specifications stand a great chance to contain errors. Eliminating the errors in the early phase of a software project can produce a considerable positive effect on the overall cost of the project, and the reliability of the final software product [2]. Formal engineering methods have been recognized as an effective and efficient approach for developing large-scale software systems. One way to improve the quality of specifications and therefore the quality of the corresponding software program is to formalize specifications. We choose Structured Object-Oriented Formal Language (SOFL) for this purpose in this paper.

The SOFL method provides a three-step approach to developing formal specifications. Such a development is an evolutionary process, starting from an informal specification, to a semi-formal one, to finally a formal specification [1]. In the evolutionary process, the errors can be made because of inaccurate understanding of the requirements, incorrect uses of mathematical expressions, or wrong decisions in defining data or functions [3]. When changes take place on one level specification, it may require appropriate changes in the related specifications. However, how to keep the conformance between the specifications after the changes still remains an unaddressed problem. Our research mainly focuses on how to sustain the consistency between different level specifications and eliminate errors.

According to the IEEE standard [4], the purpose of an inspection is to detect and identify software product anomalies. An inspection is a systematic peer examination that verifies the software product conforms to applicable regulations, standards, guidelines, plans, specifications, and procedures. The inspector collects software engineering data like anomaly and effort data by using supporting documentations such as checklists to show what should be checked.

In this paper, we propose an inspection method through building traceability for rigorously verifying the conformance relation, which has been briefly presented at the Winter Workshop 2014 in Oarai [5]. This method mainly consists of two steps: (1) traceability establishment and (2) inspection of the target specifications based on the built traceability. The first step is based on the structure and syntax of SOFL three-step specifications, corresponding items will be generated together in the evolutionary process. The checklists will be provided to help the inspector confirm whether to establish the traceability between two items in different specifications or not. The second step inspects the target specifications based on the built traceability. Pair review is a useful way to check whether the traceability is correct or not by comparing the textual specifications and the Condition Data Flow Diagram (CDFD). Our supporting tool provides a convenient interface to separate components in different specifications and save their relationships to keep the consistency. Based on the correct syntax of components, our tool can get all items automatically to check whether the target specification fits user's requirements or not. We present a case study of the inspection method by describing how it is applied to inspect the specifications of a travel plan booking system to show the method's feasibility, and explore some potential challenges in using our supporting tool.

The rest of this paper is organized as follows. We introduce some basic concepts in Section II. Section III mainly discusses

Supervisor: Prof. Shaoying Liu

the possible way to build the traceability and how to inspect the components through the traceability. We discuss the design and implementation of our supporting tool in Section IV. In Section V, a case study is given to show how the proposed method can be applied in practice. Related work is introduced in Section VI. Finally, we give conclusions and point out future research directions in Section VII.

II. Basic Concepts

In this section, we first introduce SOFL and then some inspection strategies, such as checklists and pair reviews for inspecting SOFL specifications.

A. SOFL

SOFL is a formal engineering method that provides a formal but comprehensible language for both requirements and design specifications. A SOFL specification mainly consists of two parts: the textual specification and the Condition Data Flow Diagram. The textual specification is a written documentation and mainly built by the component called "process". A process models a transformation from input to output, which provides pre-condition and post-condition to describe the functionality and constraints of transferred data. Different processes contact with each other to handle data. A set of processes can be defined in a "module", which can achieve some independent functions of the target system. At the same time, some processes can also be decomposed into a low level "module", which can explain the complicated data manipulation more clearly. By combining the generation and decomposition of processes reasonably, we can easily achieve the system requirements in our SOFL specification. Fig. 1 shows an example of SOFL textual specification.

```
module JTB_Decom/SYSTEM_JTB
type
Customer = composed of
                    user_id : seq of nat0
                    name : FullName
                    tel : seq of nat0
                    pass_no : seq of nat0
                    sex : bool
                    status : {<Login>,<Logout>}
           end;
CustomerList = set of Customer;
var
CustomerDB : CustomerList;
inv
forall[x: RoomNo] | 1 <= x <= 100;
forall[x: dom(rlist)] | rlist(x).status = <Reserved>
or rlist(x).status = <Check In>;

process Register(register_request: Customer)
        register_signal: bool
ext wr CustomerDB
pre true
post (exists! customer inset CustomerDB |
(customer.user_id = register_request.user_id
and register_signal = false)) or CustomerDB =
~CustomerDB union {register_request} and
register_signal = true
end_process;

process UpdateProfie(update_profile_request:
        Customer)update_signal: bool
ext wr CustomerDB
pre true
post (exists! ~customer inset CustomerDB |
(~customer.user_id = update_profile_request.user_id
and ~customer.pass_no =
update_profile_request.pass_no and customer =
update_profile_request and update_signal = true)) or
update_signal = false
end_process;
end_module;
```

Fig. 1. SOFL textual specification.

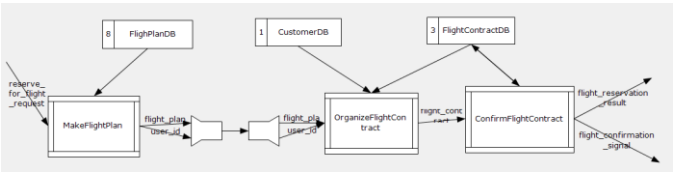

Fig. 2. CDFD describing a flight plan.

The textual specification is produced based on a three-step approach to developing formal specifications. Such a development is an evolutionary process, starting from an informal specification, to a semi-formal specification, then to a formal specification. Informal specification is the first step in SOFL method to reach user's requirements. Although it is difficult to define the concept of a well-organized specification, such a specification must clearly and concisely describe the following items:

1) the functions to be implemented in the software project;

2) the resources to be used in implementing functions;

3) the necessary constraints on both functions and resources.

The semi-formal specification derives from the informal specification. Its goal is to clarify and define all the functions, resources, and constraints, and to determine the relationships among those three parts contained in the informal specification. The most distinct feature of a semi-formal specification is that the format of the specification obeys the syntax of the formal specification, but the pre- and post-conditions of all processes in modules are defined in a natural language in an organized manner. In the formal specification, by evolving all items from the semi-formal specification in logical expressions, some processes written by the natural language will be found too complicated to transform into logical expressions. Under this situation, we need to decompose the process into some sub processes to keep them logical.

Another important part of SOFL is Condition Data Flow Diagram. Different from the textual specification, the CDFD is a directed graph that specifies how processes work together to provide functional behaviors. The process specification mainly focuses on the internal logical relationships and data constraints, while the CDFD mainly represents the relation between different processes by transferring different data. Fig. 2 shows an example of CDFD describing a flight plan.

From Fig. 2, we can see that each module generated by a set of processes has a corresponding CDFD. A process in the CDFD is presented as a rectangle pane and connects each other by arrows called data flows. A data flow represents input or output data in the textual specification. Also there is another kind of rectangles starting with a number called data stores. A data store is a variable holding data in rest. By using these rectangles, data can convert into the expected situation.

B. Checklists

When inspecting specifications, we need strategies to help inspectors check SOFL specifications easily. One strategy is Checklist. Checklists are a well-established reading support mechanism often used by individual inspectors for the purposes of preparation. Checklists are based upon a series of

specific questions that are intended to focus the inspector's attention on common sources of defects.

The format of the checklist follows what is used by Laitenberger and DeBaud [6] and suggested by Chernak [7]. The schema consists of two components, "where to look" and "how to detect". The first component is a list of potential "problem spots" that may appear in the work product, and the second component is a list of hints on how to identify a defect in the case of each problem spot.

Finally, the questions are ordered to support the inspector in achieving a thorough understanding of the specifications. As the inspector moves through the different groups of questions (invariant, process, etc.), he successively proceeds from a higher-level and general perspective toward a more detailed and fine-grained one. Each group of questions requires more and more understanding of each item in three-step specifications, and the final question in the evolutionary specifications section, "does the target specification match the corresponding specification?" will be easier to answer once all the other questions have been applied.

C. Pair Review

Pair review is a group way of inspecting requirements specifications like a software development technique called pair programming. In pair programming, two programmers work together at a single keyboard, one is coding while the other observes and reviews. The roles are switched at regular intervals. Based on characters of SOFL language we mentioned above, pair review is very helpful when inspecting the textual specification and the corresponding CDFD. By reviewing the textual specification, we can see whether the input and output data in the CDFD are correct or not, and data should be stored in the right data stores. By reviewing the CDFD, we can check whether the set of processes in the corresponding textual specification are generated in the right order or not. At the same time, the types of data flow and the logical constraints about pre- and post-condition will be confirmed in the textual specifications.

III. Building Traceability and Inspection

Requirements traceability refers to the ability to describe and follow the life of a requirement, in both forward and backward directions. For example, from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases. Pinheiro and Goguen [8] think that requirements traceability refers to the ability to define, capture and follow the traces left by requirements on other elements of the software development environment and the trace left by those elements on requirements. Some traceability definitions emphasize the use of traceability to document the transformation of a requirement into successively concrete design and development artifacts. Hull, Elizabeth et al. [9] explain that in the requirements engineering field, traceability is about understanding how high-level requirements – objectives, goals, aims, aspirations, expectations, needs – are transformed into low-level requirements. It is therefore primarily concerned with the relationships between layers of information. From this definition, we can find that the SOFL method is from high-level requirements included in informal specification to low-level requirements included in formal specification. In this process, the requirements traceability is clear in corresponding items.

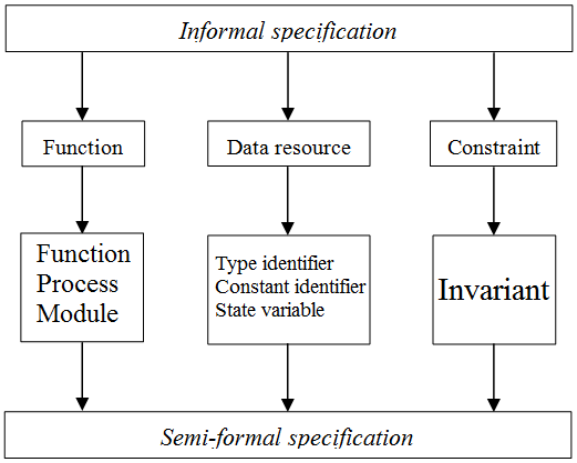

Fig. 3. Corresponding components between informal and semi-formal specification.

There are two steps in our inspection method through building traceability. First, we generate the traceability between informal and semi-formal, then semi-formal and formal specifications. The traceability means the congruent relationships of elements which represent the same users' requirements in different specifications. For example, a function in the informal specification may be correlated to a process in the corresponding semi-formal specification. Second, by comparing with the built traceability and CDFD, we inspect corresponding items in different specifications together.

Because there are three specifications, we separate the traceability into two parts to make it more clearly: (1) traceability between informal and semi-formal specifications, (2) traceability between semi-formal and formal specifications.

A. Building traceability between informal and semi-formal specifications

During the first part, user's requirements will be refined and described more precisely. To cover as many user's requirements as possible, the structures in the informal specifications are rough. They contain only three components: functions, data resources and constraints. Because of the partition in informal specification, the conversion to semi-formal specification is quite flexible and mainly depends on user's experience. However, we can still compare corresponding components based on structures in different specifications as shown in Fig. 3. By making a signal between the corresponding items – examples of components – in different specifications, every item will get the relationship with one or many items. We can make a checklist as shown in Table I to build the traceability about all items clearly between informal specification and semi-formal specification. If an item has no traceability with other items, the item should be removed or some items need to be added in the corresponding specification by comparing with the same kind of items.

TABLE I. CHECKLIST ABOUT TRACEABILITY BETWEEN THREE SPECIFICATIONS

	Component	*Question*
Informal Specification(S1):		
1	Function	Is the function decomposed into sub functions properly?
2		Is the function has the same name process in S2?
3	Data resource	Is the data resource used in the corresponding function?
4		Is the data resource evolved into the data type in S2?
5	Constraint	Is the constraint associated with a function or a data resource?
6		Does the constraint have the similar invariant in S2?
7		Is the constraint achieved in the pre- or post-condition of a process in S2?
Semi-formal Specification(S2):		
8	Constant identifier	Is the constant identifier available and can be found in data resources in S1?
9	Type identifier	Is the type identifier needed from data resources in S1?
10	State variable	Is the state variable defined based on the type identifier?
11	Invariant	Does the invariant have the corresponding relation with the constraint in S1?
12	Process	Is the process named by the function in S1?
13		Is the process treated as a sub function in S1?
Formal Specification(S3):		
14	Constant identifier	Is the constant identifier available and can be found in constant identifier declarations in S2?
15	Type identifier	Does the type identifier exist in S2?
16	State variable	Does the state variable exist in S2?
17	Invariant	Does the invariant have the same meaning of invariants in natural language in S2?
18	Process	Does the process have the same name, input data, output data in S2?
19		Does the pre- and post-condition of the process fit the natural expression in S2?
20		Is the process treated as a sub process and is a series of processes equal to the process in S2?

B. Building traceability between semi-formal and formal specifications

For the second part, structures are almost the same between semi-formal and formal specifications. We should pay more attention about the invariants and processes. As shown in Table I, corresponding invariant definitions will be compared with one another to check whether their logical meanings are the same or not. Also we need to focus on the pre- and post-condition of processes to make the logical expression fit the requirements written in natural language.

After generating two parts of traceability, we can inspect all items throughout the whole requirements specifications.

C. Inspection based on traceability

To inspect errors and defects, firstly we should provide the standard format of all data types. We can get them from the existing publication in [1], especially about the syntax of Set type, Sequence type, Composite type, Product type, Map type, Union type, Process type. The key words of these types will influence building traceability between different specifications.

For making pair reviews by two inspectors, the textual specification and corresponding CDFD should be inspected together. We can inspect the traceability based on building functional scenarios provided in [10]. Let P(Piv, Pov)[Ppre, Ppost] denote the formal specification of an operation P, where Piv and Pov are the sets of all input and output variables. Ppre and Ppost are the pre-condition and post-condition of operation P, respectively. Let Ppost=C1∧D1∨C2∧D2∨⋯∨Cn∧Dn, where Ci (i∈{1, 2, ..., n}) is a guard condition and Di is a defining condition. Then, a conjunction ~Ppre∧Ci∧Di is called a functional scenario.

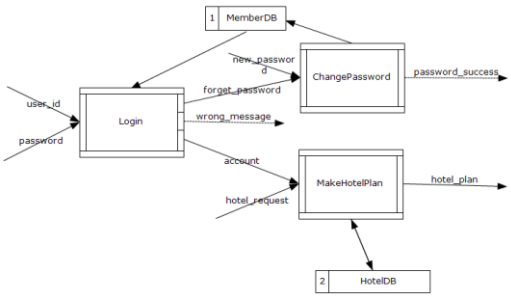

Fig. 4. CDFD of making a hotel plan.

To make inspectors to check corresponding items easily, we provide functional scenarios from CDFD as a standard to compare with both two specifications. In this situation, define the format (input_1, input_2,…, input_n){process_1, process_2,…, process_n}(output_1, output_2,…, output_n) as a functional scenario from CDFD. Fig. 4 shows a CDFD about making a hotel plan.

In this CDFD, we can get 3 functional scenarios:

(1)(user_id, password, new_password){Login, ChangePassword} (password_success);

(2)(user_id, password){Login}(wrong_message);

(3)(user_id, password, hotel_request){Login, MakeHotelPlan}(hotel_plan).

These three functional scenarios show three different conditions with submitting different data. When inspecting traceability between semi-formal and formal specification, a set of processes with same functional scenario (1), (2), (3) will be generated, and errors about wrong data flows should be removed.

IV. SUPPORTING TOOL

We have built a supporting tool, called the *Traceability-based Specifications Inspection Supporting Tool* (TSIST), to support our inspection method through building traceability. The goal of building the tool is to help inspectors check specifications more precisely, and save the traceability information made by them for iterative inspections. The supporting tool is implemented using Visual Studio 2012 with language C#. Fig. 5 gives the CDFD of TSIST functions. As Fig. 5 shows, our tool can search key words from specifications, divide all items, then build traceability between different items in corresponding specifications, and finally inspect specifications by comparing traceability and CDFD. The whole process in using TSIST summarizes into three main functions below:

(1) Searching key words and inspecting syntax errors in three specifications;

(2) Selecting items from corresponding specifications manually or automatically;

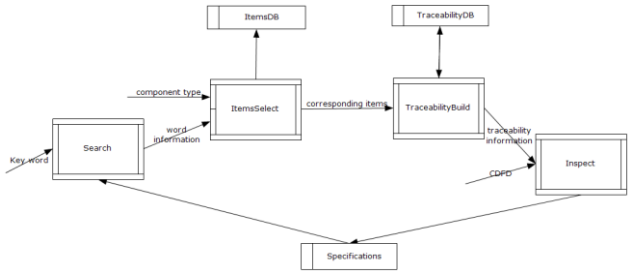

Fig. 5. CDFD of TSIST functions.

(3) Building traceability between corresponding items in two specifications and saving traceability information for comparisons and iterative inspections.

A. Searching key words

TSIST provides the function to search key words in the specification documentation. When a key word is entered in the search column, our tool will match the key word in the target specification and find out all eligible elements. In this way, the user can check the syntax of target items quickly.

B. Selecting items automatically

To build traceability between two specifications, the inspector should get all items in textual specifications first. Obviously, we provide the manual way to add items directly. Based on the standard format of items, the inspector can also traverse the specification to get all items of the same component (such as Function, Data resource, Process).

To make the component "*process*" as an example. Setting the *targetComponent = "process", endFlag = "(", breakFlag = ";"*, when reading the specifications from the beginning, the scanner gets the *targetComponent*, it will repeatedly read the next character until finding the key word *endFlag*. By using this way, the inspector can get names of all processes easily. From the structure and syntax of the component "process", we can find when we get the *breakFlag* before meeting the *endFlag*, it means the process ends with the syntax "*end_process*;" then the scanner will skip and try to find the next *targetComponent*. By changing *endFlag*, we can get input data, output data, pre- and post-condition of the target process respectively.

C. Building traceability

After generating all items in specifications, TSIST provide an interface to select corresponding items to build traceability between two specifications as shown in Fig. 6. At the same time, the traceability between informal and semi-formal specification and the traceability between semi-formal and formal specifications can be generated together to keep the consistency through the whole requirements specifications.

Compared with the checklists, the inspectors can build traceability by using our inspection supporting tool, the traceability between corresponding specifications can be saved and removed in iterative inspections. Through building traceability, the pair reviews based on the textual specification and CDFD will help inspectors check specifications and detect defects more precisely.

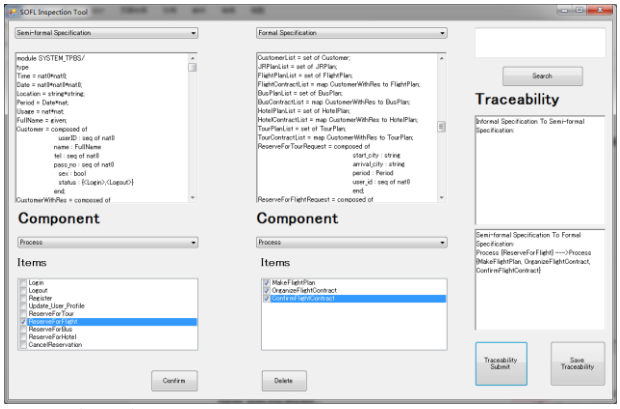

Fig. 6. Interface of TSIST.

V. Case Study

We have conducted a case study applying our inspection method to the inspection of the specifications of a *travel plan booking system* (TPBS). The purpose of this case study is to show how our inspection method works through building traceability, to learn about its performance in terms of usability and capability of finding errors, and to investigate how the inspection method can be well supported by TSIST.

A. Background

TPBS specifications describe a travel plan booking system, which allows the customer to search travel information, design his personal travel plan, and book target services (flights, hotels, etc.). TPBS mainly includes four functions:

(1) designing the tour plan;
(2) reserving flights;
(3) making bus arrangement;
(4) booking hotels.

Fig. 7 shows the textual specifications and CDFD of TPBS. From the informal specification to semi-formal specification, the functions of TPBS (such as Update_User_Profile, Reserve_for_Hotel) are listed in details, showing how the input and output data flow between different processes. Data resources (such as Tour_Plan_Information, Bus_Plan_Information) will be checked to fit the types in the semi-formal specification. And constraints are used to show the range of data in the process. From the semi-formal specification to formal specification, the natural language used in pre- and post-condition evolves into logical expressions.

B. Building traceability

As shown in Fig. 6, we can get all items such as "Process" in the target specification. Based on the checklist shown in Table I, we decide where to build traceability between corresponding items. For example, in Fig. 6, we build the traceability between the process {ReserveForFlight} in the semi-formal specification and a set of the processes {MakeFlightPlan, OrganizeFlightContract, ConfirmFlightContract} in the formal specification because they realize the same functions. Table II gives the number of items and traceability in corresponding specifications. From this table, we can know every item has the traceability with others and the item in the

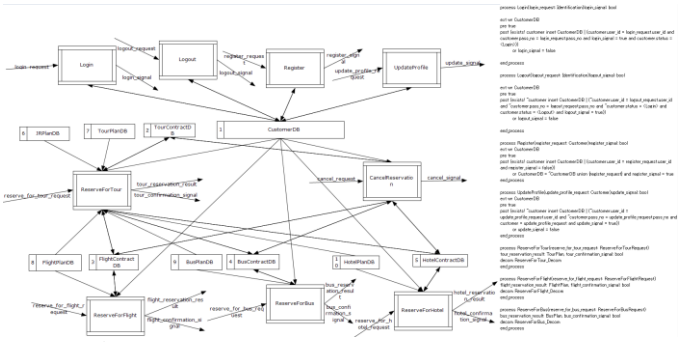

Fig. 7. CDFD of TPBS.

TABLE II. NUMBER OF ITEMS AND TRACEABILITY

	Component	*Number*
Informal Specification:		
1	Function	11
2	Data resource	9
3	Constraint	3
Traceability between informal to semi-formal specification		23
Semi-formal Specification:		
4	Constant identifier	2
5	Type identifier	29
6	State variable	10
7	Invariant	3
8	Process	11
Traceability between semi-formal to formal specification		55
Formal Specification:		
9	Constant identifier	2
10	Type identifier	29
11	State variable	10
12	Invariant	3
13	Process	24

high level may trace to a set of items in the more formal specification because they are more precise and smaller.

C. Inspection

After building traceability, we can check corresponding items such as the process {ReserveForFlight} and the set of processes {MakeFlightPlan, OrganizeFlightContract, Confirm-FlightContract} together in the textual specifications. They will be checked whether they are equal not only in the syntax domain but also in the logical domain. Pair Review based on textual documentations and CDFD helps inspectors understand requirements easily and find errors through data flows. The incorrect data or missing processes in TPBS can be corrected in our supporting tool. The traceability through three specifications is saved for the iterative inspection.

VI. RELATED WORK

Many publications have affirmed that the requirements traceability plays an essential role in software inspection.

Pinherio, Goguen et al. [8] introduced a cited tool called TOOR to trace requirements considering both technical and social factors. TOOR can link requirements to design documents, specifications, code, and other artifacts in the life cycle through user-definable relations that are meaningful for the kind of connection being made by using both browsing and regular-expression search.

Patricio Letelier [11] presented a traceability metamodel integrating textual specifications with standard UML specifications, using the UML context itself. The metamodel offers a core framework for types of entities and types of traceability links that can be adapted to a particular UML project. Additionally, a configuration process for requirements traceability based on the corresponding UML profile was sketched.

VII. CONCLUSION AND FUTURE WORK

In this paper, we propose an inspection method through building traceability for rigorously verifying the conformance relationship. Inspection strategies such as Checklist and Pair review are used to help the inspectors build traceability and check textual specifications and CDFD together for finding errors.

Our supporting tool can divide all items based on the component types, build traceability and check specifications through traceability and CDFD. We present a case study applying our inspection method to inspect the specifications of a travel plan booking system, to show the method's feasibility and capability of finding errors, and to investigate how the inspection method can be well supported by our tool.

In the future, we will improve our supporting tool to make it more user-friendly and support more ways of building traceability and inspecting specifications.

ACKNOWLEDGMENT

I extend my most sincere thanks to my supervisor, Prof. Shaoying Liu, who gives careful guidance and endless support from the topic selection to the thesis final completion. His serious and responsible spirits encourage me to work hard with high motivation.

REFERENCES

[1] Shaoying Liu, *Formal Engineering for Industrial Software Development Using the SOFL Method.* : Springer-Verlag, 2004.
[2] B. W. Boehm, *Software Engineering Economics.* : Prentice Hall, 1981.
[3] Shaoying Liu, John McDermid, and Yuting Chen, "A Rigorous Method for Inspection of Model-Based Formal Specifications", *IEEE Transactions on Reliability*, vol. 59, no. 4, pp. 667-684, Dec. 2010.
[4] IEEE, 1028-2008 IEEE Standard for Software Reviews and Audits, IEEE Computer Society, 2008.
[5] Jinghua Zhang, Shaoying Liu, "Inspection of SOFL Specifications through Building Traceability", *Winter Workshop 2014 in Oarai*, 2014.
[6] Laitenberger, Oliver, and Jean-Marc DeBaud, "An Encompassing Life Cycle Centric Survey of Software Inspection", *Journal of Systems and Software*, vol. 50, no. 1, pp. 5-31, Jan. 2000.
[7] Yuri Chernak, "A Statistical Approach to the Inspection Checklist Formal Synthesis and Improvement", *IEEE Transactions on Software Engineering*, vol. 22, no. 12, pp. 866-874, Dec. 1996.
[8] Pinheiro, Francisco AC, and Joseph A. Goguen, "An Object-Oriented Tool for Tracing Requirements", IEEE Software, pp. 52-64, Mar. 1996.
[9] Hull, Elizabeth, Ken Jackson, and Jeremy Dick, *Requirements Engineering.* : Springer, 2005.
[10] Mo Li, Shaoying Liu, "Automated Functional Scenario-Based Formal Specification Animation", in *Proceedings of the 19th Asia-Pacific Software Engineering Conference*, IEEE CS press, pp. 107-115, 2012.
[11] Patricio Letelier, "A Framework for Requirements Traceability in UML-based Projects", *1st International Workshop on Traceability in Emerging Forms of Software Engineering*, 2002.