

Research Article

Extension of the Function to Ensure Real-time Traceability between UML Sequence Diagram and Java Source Code on RETUSS

Kaoru Arima¹, Tetsuro Katayama¹, Yoshihiro Kita², Hisaaki Yamaba¹, Kentaro Aburada¹, Naonobu Okazaki¹

¹Department of Computer Science and Systems Engineering, Faculty of Engineering, University of Miyazaki, 1-1 Gakuen-kibanadai nishi, Miyazaki, 889-2192, Japan

²Department of Information Security, Faculty of Information Systems, Siebold Campus, University of Nagasaki, 1-1-1 Manabino, Nagayo-cho, Nishi-Sonogi-gun, Nagasaki, 851-2195, Japan

ARTICLE INFO

Article History

Received 25 November 2020

Accepted 28 May 2021

Keywords

Software quality

Traceability

UML

Sequence diagram

Java

ABSTRACT

Ensuring traceability of software deliverables is one of the methods to ensure software quality. RETUSS (Real-time Ensure Traceability between UML and Source-code System) is a tool that saves labor and time, and eliminates mistakes by human handling in ensuring traceability between UML and source code. However, RETUSS is not useful due to its limited scope of application. This paper improves the usefulness of RETUSS by extending the function to ensure real-time traceability between UML sequence diagrams and Java source code on RETUSS.

© 2022 The Author. Published by Sugisaka Masanori at ALife Robotics Corporation Ltd.

This is an open access article distributed under the CC BY-NC 4.0 license
(<http://creativecommons.org/licenses/by-nc/4.0/>).

1. Introduction

The importance of software in society is increasing, and system failures and software bugs cause significant economic and social impact. Therefore, ensuring the quality of systems and software has become more important. Ensuring traceability of software deliverables is one of the methods to ensure software quality [1]. It can specify the scope of the impact due to the modification in the requirements and remove the gap between the documents and the source code. However, it has the following two problems.

- Taking much labor and time to modify similarly other related deliverables in modifying a part of deliverables.
- Having a risk that you cannot ensure traceability because of causing mistakes to ensure traceability by human handling.

In order to solve them, our laboratory developed RETUSS (Real-time Ensure Traceability between UML and Source-code System) [2], [3]. RETUSS ensures traceability between UML (Unified Modeling Language) [4] and source code by transforming them to each other in real time. Therefore, RETUSS can save labor and time, and eliminate mistakes by human handling in ensuring traceability between UML and source code. RETUSS has the following functions.

- Description of class diagram
- Description of sequence diagram
- Description of Java source code
- Description of C++ source code
- Ensuring real-time traceability between class diagram and Java source code
- Ensuring real-time traceability between class diagram and C++ source code
- Ensuring real-time traceability between sequence diagrams and Java source code

Corresponding author's E-mail: arima@earth.cs.miyazaki-u.ac.jp, kat@cs.miyazaki-u.ac.jp, kita@sun.ac.jp, yamaba@cs.miyazaki-u.ac.jp, aburada@cs.miyazaki-u.ac.jp, oka@cs.miyazaki-u.ac.jp

However, RETUSS is not useful in ensuring traceability between sequence diagrams and Java source code due to its limited scope of application. This paper improves the usefulness of RETUSS by extending the function to ensure real-time traceability between UML sequence

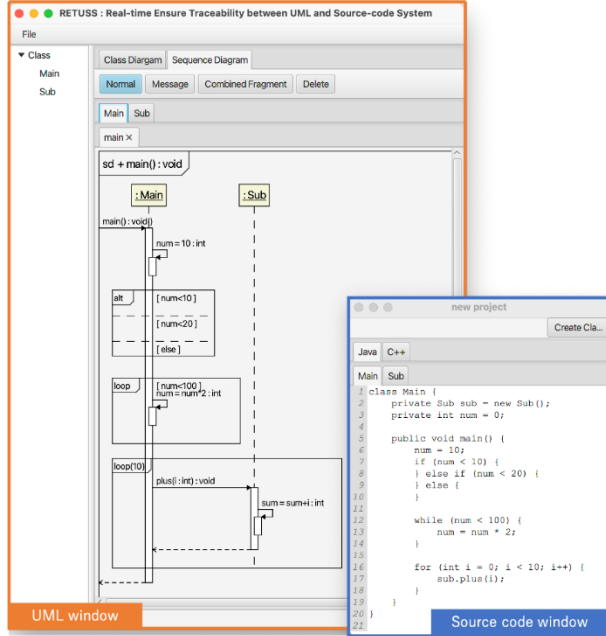


Fig. 1. The interface of the extended RETUSS diagrams and Java source code on RETUSS.

2. The extended RETUSS

This paper extends the following two functions of the existing RETUSS.

- Description of sequence diagram
- Ensuring real-time traceability between sequence diagrams and Java source code

Fig. 1 shows the interface of the extended RETUSS. The extended RETUSS has the UML window and the source code window.

The UML window is a window for describing UML. It has a class diagram tab and a sequence diagram tab. In the class diagram tab, there are class diagram drawing buttons and a class diagram description area. In the sequence diagram tab, there are sequence diagram drawing buttons and a sequence diagram description area.

The source code window is a window for describing source code. It has language tabs. Within each language tabs, there are class tabs. In each class tabs, there is a text editor.

2.1. Extending the function of the description of sequence diagram

We add the following three functions to the function of the description of the sequence diagram.

- Adding a message
- Adding a combined fragment
- Deleting elements

However, the user can only add operation invocation messages among the messages. In addition, the user can add only three types of combined fragments: opt, alt, and loop.

In order to add these three functions, we added three buttons to the UML window: Message button, Combined Fragment button, and Delete button. In addition, we add description dialogs that correspond to these buttons. By adding these functions, a user can edit the sequence diagram directly on RETUSS.

2.2. Extending the function to ensure real-time traceability between sequence diagrams and Java source code

RETUSS ensures traceability between sequence diagrams and Java source code in real time by transforming them in real time. In this paper, we define four new transformation rules to extend the scope of the real-time transformation between sequence diagrams and Java source code. Table 1 shows the four new transformation rules. The following shows the correspondence of the elements in the transformation rules.

- Message
 - The message name of the message signature corresponds to the method name of the method invocation expression.
 - The parameter name of the message signature corresponds to the parameter name of the method invocation expression.
 - The parameter type of the message signature corresponds to the parameter type of the method declaration.
 - The return type of the message signature corresponds to the return type of the method declaration.
- Combined fragment opt
 - The guard of combined fragment opt corresponds to the conditional expression of the if-then statement.
- Combined fragment alt
 - The guards of combined fragment alt correspond to the conditional expressions of the if-then-else statement.
- Combined fragment loop

- The guard of combined fragment loop corresponds to the conditional expression of the while statement.

- The number of loops corresponds the number of loops calculated from X and Y in the for statement.

Table 1. The four new transformation rules for the real-time transformation between sequence diagrams and Java source code

Name in sequence diagram	Notation in sequence diagram	Name in Java	Syntax in Java
Message	<pre>messageName (parameterName : parameterType, ...) : returnType</pre>	Method invocation expression	methodName(parameter, ...);
		Method declaration	accessModifier returnType methodName(parameterType parameterName, ...) { ... }
Combined fragment opt	<pre>opt [guard]</pre>	if-then statement	if (expression1) { ... }
Combined fragment alt	<pre>alt [guard 1] [guard 2] :</pre>	if-then-else statement	if (expression1) { ... } else if (expression2) { ... } ...
Combined fragment loop	<pre>loop [guard]</pre>	while statement	while (expression) { ... }
		for statement	for(int i=X; i<Y; i++) { ... }
			for(int i=X; i<=Y; i++) { ... }
			for(int i=X; i>Y; i--) { ... }
			for(int i=X; i>=Y; i--) { ... }

Fig. 2 shows the data flow of real-time transformation between sequence diagrams and Java source code. The sequence diagram information is the data structure to store sequence diagram elements in RETUSS. The Java information is the data structure to store Java syntaxes in RETUSS. The following shows the flow of transformation from sequence diagrams to Java source code.

- The user specifies a class and an operation, and then modifies the sequence diagram.
- RETUSS searches for the class and the operation, and then modifies the sequence diagram information.
- RETUSS redraws the sequence diagram based on the sequence diagram information.
- RETUSS transforms the sequence diagram information into Java information based on the transformation rules (Table. 1). In this case, RETUSS transforms only the class to be modified.
- RETUSS generates Java source code from Java information.
- RETUSS displays the Java source code in a source code window.

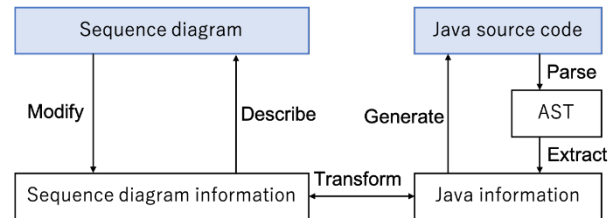


Fig. 2. The data flow of real-time transformation between sequence diagrams and Java source code

The following shows the flow of transformation from Java source code to sequence diagrams.

- The user modifies Java source code.
- RETUSS parses the Java source code of the modified class.
- RETUSS extracts the Java information from the AST and integrates it with the existing Java information.
- RETUSS transforms the Java information into sequence diagram information based on the transformation rules (Table. 1). In this case, RETUSS transforms only the class to be modified.
- RETUSS redraws the sequence diagram based on the sequence diagram information.

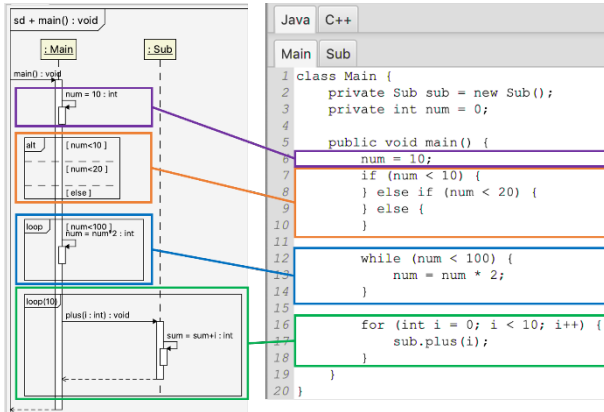


Fig. 3. The screenshot when the Java source code are written in the extended RETUSS

3. Application example

Fig. 3 shows the screenshot when the Java source code are written in the extended RETUSS. It shows that the extended RETUSS can ensure traceability between sequence diagram and Java source code in writing if-then-else statement, while statement, and for statement of Java. In addition, we confirmed that the extended RETUSS can ensure traceability between sequence diagram and Java source code in describing sequence diagram.

4. Evaluation

To evaluate the usefulness of the extended RETUSS, we experiment with four students of the University of Miyazaki. The steps of the experiment are shown below.

- (i) The experimenter prepares traceable sequence diagrams and Java source code. We call these deliverables.
- (ii) The experimenter instructs the participants to change the deliverables.
- (iii) The participants change the deliverables as instructed.

There are two types in the changes: sequence diagrams changes, Java source code changes. There are two cases below for participants to change the deliverables.

- Case A: using the extended RETUSS.
- Case B: using EA (Enterprise Architect) [5] and a text editor.

Table 2 shows the times it took the participants to change in the two cases and the two changes. From Table 2,

Table 2. The times it took the participants to change (seconds)

Participants	Sequence diagrams changes		Java source code changes	
	Case A	Case B	Case A	Case B
1	59	205	134	369
2	62	221	95	346
3	38	216	102	361
4	60	289	126	398
Average	54.75	232.75	114.25	368.50

the time in case A was about 76.5% shorter than the time in case B, when the sequence diagrams changes was instructed. In addition, the time in case A was about 69.0% shorter than the time in case B, when the Java source code changes were instructed.

In summary, the extended RETUSS can save labor and time in ensuring traceability between sequence diagrams and Java source code. Therefore, the usefulness of RETUSS has improved by extending the function to ensure real-time traceability between UML sequence diagrams and Java source code while retaining the benefits of the existing RETUSS.

5. Conclusion

This paper improved the usefulness of RETUSS by extending the function to ensure real-time traceability between UML sequence diagrams and Java source code on RETUSS. The extended RETUSS allows you to edit sequence diagram directly on RETUSS, and also supports four new transformation rules for sequence diagram and Java source code.

The experimental results showed that the extended RETUSS can save the time to ensure traceability between sequence diagrams and Java source code by about 76.5% for sequence diagram changes, and about 69.0% for Java source code changes. Therefore, the usefulness of RETUSS has improved by extending the function to ensure real-time traceability between UML sequence diagrams and Java source code while retaining the benefits of the existing RETUSS.

The future works are as follows.

- Corresponding to other sequence diagram elements
- Corresponding to other Java source code syntaxes
- Corresponding to other UML diagrams
- Corresponding to other programming languages

References

1. SQuBOK Sakutei Bukai, *Guide to the Software Quality Body of Knowledge*, 2nd edn. Ohmsha, 2014 (in Japanese).

2. Tetsuro Katayama, Keisuke Mori, Yoshihiro Kita, Hisaaki Yamaba, Kentaro Aburada, Naonobu Okazaki: *RETUSS: Ensuring Traceability System between Class Diagram in UML and Java Source Code in Real Time*, Journal of Robotics, Networking and Artificial Life, Vol. 5(2), pp. 114–117, 2018.
3. GitHub, *RETUSS: Real-time Ensure Traceability between UML and Source-code System*, <https://github.com/Morichan/Retuss> (Accessed 2021-05-26)
4. The Object Management Group, *Welcome to UML Web Site!*, <https://www.uml.org/> (Accessed 2021-05-26)
5. Sparx Systems, *UML modeling tools for Business, Software, Systems and Architecture*, <https://www.sparxsystems.com/> (Accessed 2021-05-26)

Authors Introduction

Kaoru Arima



He received the Bachelor's degree in engineering (computer science and systems engineering) from the University of Miyazaki, Japan in 2020. He is currently a Master's student in Graduate School of Engineering at the University of Miyazaki, Japan. His research interests software quality, software modeling, and software maintenance.

Tetsuro Katayama



He received a Ph.D. degree in engineering from Kyushu University, Fukuoka, Japan, in 1996. From 1996 to 2000, he has been a Research Associate at the Graduate School of Information Science, Nara Institute of Science and Technology, Japan. Since 2000 he has been an Associate Professor at the Faculty of Engineering, Miyazaki University, Japan. He is currently a Professor with the Faculty of Engineering, University of Miyazaki, Japan. His research interests include software testing and quality. He is a member of the IPSJ, IEICE, and JSSST.

Yoshihiro Kita



He received a Ph.D. degree in systems engineering from the University of Miyazaki, Japan, in 2011. He is currently an Associate Professor with the Faculty of Information Systems, University of Nagasaki, Japan. His research interests include software testing and biometrics authentication.

Hisaaki Yamaba



He received the B.S. and M.S. degrees in chemical engineering from the Tokyo Institute of Technology, Japan, in 1988 and 1990, respectively, and the Ph.D. degree in systems engineering from the University of Miyazaki, Japan in 2011. He is currently an Assistant Professor with the Faculty of Engineering, University of Miyazaki, Japan. His research interests include network security and user authentication. He is a member of SICE and SCEJ.

Kentaro Aburada



He received the B.S., M.S, and Ph.D. degrees in computer science and system engineering from the University of Miyazaki, Japan, in 2003, 2005, and 2009, respectively. He is currently an Associate Professor with the Faculty of Engineering, University of Miyazaki, Japan. His research interests include computer networks and security. He is a member of IPSJ and IEICE.

Naonobu Okazaki



He received his B.S, M.S., and Ph.D. degrees in electrical and communication engineering from Tohoku University, Japan, in 1986, 1988 and 1992, respectively. He joined the Information Technology Research and Development Center, Mitsubishi Electric Corporation in 1991. He is currently a Professor with the Faculty of Engineering, University of Miyazaki since 2002. His research interests include mobile network and network security. He is a member of IPSJ, IEICE and IEEE.
