TTExtS: A Dynamic Framework to Reverse UML Sequence Diagrams From Execution Traces

X. Zhao, A. Payne, T. Atkison
Computer Science Department, University of Alabama, Tuscaloosa, AL, USA

Abstract—Most approaches for reversing UML diagrams are static and highly dependent on the source code. Static analysis is time consuming and error prone. This paper extracts UML sequence diagrams in a dynamic fashion; reversing from system application, but not source code. To achieve this, a dynamic framework for reversing UML sequence diagrams, called TTExtS, is proposed. TTExtS contains three subsystems: trace collection, trace transformation and sequence diagram generation. Results show that TTExtS can reverse the UML sequence diagram from system execution in reasonable time, less than 2 seconds processing time. The diagrams produced are high quality, which will allow stakeholders to delve deeper into the behavior aspects of applications without manually analyzing the code.

Keywords: Reverse Engineering, Model Driven Engineering, UML Sequence Diagram, Execution Trace.

1. Introduction

Unified Modeling Language (UML) is a standard modeling language for object-oriented software design and analysis. There are several advantages when adopting UML during software system implementation. For example, many diagrams generated during UML designs, such as UML class diagrams, UML use-case diagrams, UML sequence diagrams, etc., could be employed as system testing tools and basics.

UML sequence diagrams describe interactions between different objects in chronological order. They have strong readabilities and UML specifications are more sequence diagram centric. Another important feature of UML sequence diagrams are that they have a better connection with reverse engineering. When there are discrepancies between UML sequence diagrams and system code, reverse engineering could be used as a tool for the retrieval of more accurate diagrams. Figure 1 shows a simple UML sequence diagram for a user login use-case.

There are two ways to perform the retrieval: static and dynamic. A static approach refers to the extraction of the diagram from system code, while a dynamic approach performs the process during system execution at runtime. Both of these two approaches have advantages and disadvantages. In the static approach, developers need the support of system code to achieve the process. This prerequisite brings several issues when it is unable to access the system code. Besides, when the system has a large size, analyzing the code is a time consuming process. In this paper, we are interested in dynamic reverse fashion due to the following reasons:

- Dynamic analysis is closer to the nature of sequence diagram – sequence diagram represents activities with interaction. The diagram in fact, is “dynamic”.
- Remove the restriction imposed on the reversing process by legacy systems. In dynamic reversing, sequence diagrams can be extracted from system execution, but the details behind the system executions are not needed.
- Dynamic approaches save retrieval time and reduce possibilities of bugs and errors compared with static analysis, which is mainly done manually.

Similar to previous efforts (e.g. [1] [2] [3]), this work is steered by the core issues in dynamic reversing of sequence diagrams. These issues are realized in the following questions:

- How will execution traces be obtained?
- How are process traces collected?
- How can UML sequence diagrams be generated from these traces?

An execution trace, according to the definition provided by Oracle¹, “collects information about what is happening in your program and displays it”. Though there is no unified definition of trace and what traces should look like, execution traces have to include three items: caller, method, and callee. That is to say, one execution trace needs to clarify which object

¹https://docs.oracle.com/cd/E19205-01/819-5257/blafk/index.html
invokes which method to which target. Tewfik et al. [4] defined execution trace as a triplet \(<\text{caller, method, callee}>\), which reflects the essence for execution traces.

In this paper, the above three questions are solved in different approaches. For trace collection and sequence diagram generation, existing tools – InTrace and WebSequenceDiagrams, are used. For trace process, an approach using trace transformation and trace merge is proposed. These two steps enable a bridge to be formed between trace collection and UML sequence diagram generation. To justify the framework, a prototype based on Java was implemented. The empirical case study shows that TTExTS can extract UML sequence diagram from system execution.

The rest of the paper develops as follows: Section 2 introduces the background and related work of this study, including some approaches handling execution traces for sequence diagrams, different approaches for the reverse of UML sequence diagrams and some existing reverse tools. Section 3 gives an overview of the proposed framework in this paper. Sections 4 through Section 6 are discussions of the main steps in TTExTS. In Section 7, a case study is presented to validate the framework. Finally, in Sections 8 and 9, discussions and conclusions of the paper are drawn.

2. Related Work

Execution traces are an important intermediate in our approach. In this section, some basic background on execution traces and how execution traces are developed and analyzed in order to obtain the reverse of sequence diagrams in existing literatures will be explored. Second, considering extracting execution traces, some methods use static or hybrid ways. These static/hybrid methods will be discussed after execution traces extraction. Finally, some existing tools for reverse engineering of sequence diagrams are presented.

2.1 Execution Traces for Sequence Diagram

Execution trace is an important aspect in dynamic program comprehension. In Tewfik’s work [4], the authors defined execution trace as a sequence of method invocations. In a method invocation, a caller invokes callee through certain method(s). Execution traces can be applied to many areas, such as system understanding [5], performance analysis [6] [7], debugging and testing [8], and reverse engineering [4]. There are some efforts on how to extract execution traces at system runtime [9]. For this research, the focus will be on dynamic retrieving of execution traces to the reverse of sequence diagrams. The next subsection will discuss some static and hybrid methods for the reverse of sequence diagrams.

Alalfi proposed a dynamic approach [10] for sequence diagrams from execution traces. Though this approach is dynamic, it only works in web applications. The authors did not generalize the framework to other fields. Similarly, Zaidman et al. [11] analyzed the connection between user and server and collected the traces during the communication. However, this method only applies to Ajax-based applications. In Ishio’s work [12], the authors proposed a quite novel idea to achieve this goal – it divides a long execution trace into several pieces and combine these pieces to generate the features in the sequence diagrams. However, this novel idea needs empirical evaluation on its performance, but the authors did not conduct experiments on the tool. The same issue also appears in another work [13]. Though Ziadi et al. [4] provided an evaluation of the tool, the evaluation lacks comparison. One can not determine its performance contrast with other current state-of-the-art tools.

In Ng’s work [14], the authors proposed MoDec, a dynamic approach used to the reverse of sequence diagrams. Though the authors provided evaluation of the tool and verified that the tool outperformed other tools, this method highly depends on design motifs, which are provided by design patterns. It is hard to finish the reverse when the design pattern is unclear. COSMOPEN, a tool discussed in another work [15], also suffered from the high dependency on specific situations. This tool needs the correct capture of stacks stored during the execution.

2.2 Static and Hybrid Reverse for Sequence Diagrams

Compared with execution traces for reverse of sequence diagrams, there are some works that attempted to employ static or hybrid (static and dynamic) approaches. Static approaches need the support from source code. In Imazeki’s study [16], the authors managed to extract sequence diagrams with analyzing source code. However, the approach proposed is only applicable to Web applications. Likewise, Roubstov [17] and Serebrenik [18] investigated this issue but they used the idea of interceptors. The introduction of interceptors could be helpful for reversing because it clearly separates core functions in JavaBeans. However, it is only applicable to Enterprise JavaBeans applications, which lacks the ability of generalization. Wang et al. [19] proposed a static method on sequence diagrams reverse based on Object Constraint Language (OCL) mapping. This method applied more meta-modeling thoughts, which has more consistencies, completeness and high-level abstractions. However, the authors failed to provide us a practical tool, making the idea only theoretically feasible.

Recently, there is a trend that more and more methodologies for reversing of UML sequence diagrams adopt a hybrid approach, which combine both static and dynamic analysis. The motivation for the hybrid approach is that the execution traces are sometimes voluminous and unordered [20], it is efficient if only meaningful classes and methods are analyzed during execution. This needs the support of source code comprehension, which is a static aspect of system understanding. The basic idea behind these works is to reduce workload
of execution traces by applying control information with the help of static code analysis [21] [22] [23]. Other works, focused on more narrow aspects in the reversing process. In Myers’s investigation [24], the authors studied how loops in the program could be better analyzed and transformed to the reverse of sequence diagrams. Hybrid reverse approaches have not been fully studied by researchers due to the fact that it needs both the support of source code and the support of application execution. However, it seems that it will be an interesting study direction in the future.

2.3 Existing Tools

There are several tools used for reversing sequence diagrams (While there are some tools used for the reverse of class diagrams, such as MoDisco, ModelMaker, they will not be discuss here). Among these tools, some of them depend on source code and adopt static analysis, such as RSA and MagicDraw. Other tools, such as MaintainJ\(^1\), JSonde, UML2 trace Interaction View, javaCallTracer\(^2\) and CodeLogic\(^3\), employ dynamic approaches by analyzing execution traces. There are still no reversing tools that are popular, or well-known tools that are in a hybrid fashion for reversing sequence diagrams. Compared with purely static or purely dynamic approaches, hybrid reverse is still immature and needs more development. Besides, in practical situations, hybrid reverse may introduce much more analyzing time due to the time consumption from both source code analysis and application execution. However, this may be an interesting direction for future research.

Among all the tools, MaintainJ is a popular one because of the popularity, ease of access, and good performance. Tools like JSonde, is out-dated and not available through the official website. Even if some tools are still available on the Internet, they have not been updated for many years (For example, the latest version of UML2 trace Interaction View, the tool developed by IBM, still stays in 2009). CodeLogic is a good tool for reverse sequence diagrams, but it is not free (limited features for the free version). Though MaintainJ is also a licensed tool, the free version provides enough functions for users.

3. TTExTS Overview

TTExTS stands for “Textual Transformations from Execution Traces to Sequence Diagram.” As the name implies, the core part of TTExTS is the transformation of traces. Figure 2 shows the overview of this framework.

From Figure 2, the three steps in TTExTS can be seen: (1) Trace Collection; (2) Trace Transformation; (3) Diagram Generation. The framework starts with system executions and ends at UML sequence diagram generations.

In Step 1, the framework collects execution traces during system execution at runtime. The output are several execution traces. Then in Step 2, these traces are transformed and merged in order to be recognizable for Step 3. This step acts as a bridge to connect Step 1 and Step 3. Finally, in Step 3, TTExTS takes merged traces as input and produces a UML sequence diagram as the final result, finishing the whole process. In Sections 4 to Section 6, these three steps will be discussed in detail, respectively.

4. Trace Collection

There are several trace collection approaches and tools [25] [26]. In TTExTS, InTrace\(^4\) is employed to collect system execution traces. InTrace is a tool for tracing different classes (such as system class, library class, self-defined class) and collecting execution traces during system execution. The reasons InTrace was chosen are as follows:

- InTrace is integrated into Eclipse IDE. It is easy to access and free to use.

\(^1\)http://maintainj.com/
\(^2\)https://sourceforge.net/projects/javacalltracer/
\(^3\)http://www.codelogicinc.com/
\(^4\)http://mchr3k.github.io/org.intrace/
• Traces collected by InTrace contain all the information needed for further analyses.
• InTrace provides filter functions with include and exclude settings. Traces useful for analysis can be kept, and unneeded traces can be eliminated.

In order to better explain execution traces collected by InTrace, the execution of the following code snippet is introduced as an example:

```java
public class Test{
    public static class A{
        public void m1(){
            B b = new B();
            b.m2();
        }
    }
    public static class B{
        public void m2(){
            /* TO DO */
        }
    }
    public static void main(String args[]){
        A a = new A();
        a.m1();
    }
}
```

Figure 3 shows the results obtained from InTrace for trace collections after executing the code snippet above. As seen, InTrace collected all the execution traces during system execution. The signature of one execution trace collected from InTrace is:

```
[Timestamp] : [Thread ID] : 
Class being called:
Method being called:
Entrance/Exit Point
```

From the signature, the following information is obtained: (1) When the method is called in the system and the intervals for different executions by subtracting timestamps; (2) Which thread is being executed (for multiple-thread systems); (3) Which class and which method in this class is being executed at which place in the source code. There are many forms of execution traces. InTrace may not provide the most information compared with other tools, but it already provides enough information for the future analysis to be done in this work. Readers could refer to its website for more information about the meaning for each token in one execution trace.

Moreover, InTrace also provides filter functions with exclude and include options. To see how class A and class B from Figure 3 interact with each other, other traces can be filtered out by adding filter conditions, such as excluding instructions, execution traces with initiation traces - <init>, and main function traces with the key word “main”. After filtering, the simplified execution traces for the code snippet above are shown in Figure 4.

5. Trace Transformation

Trace transformation acts as a connector in TTEXTS. As the name TTEXTS implies, this framework processes execution traces mainly through textual transformation. The reasons for adopting textual transformations are:

• In the Trace Collection phase, execution traces collected by InTrace could be stored in .txt files easily. InTrace provides an interface for users to store the traces in a textual format. The contents in these .txt files are the same as Figure 3 shows.
• In the Sequence Diagram Generation phase, the tool used needs some textual contents as input. The textual input conforms to some specific rules, each specific rule corresponds to certain structure in UML sequence diagrams.

However, the output of InTrace cannot be used directly as input for WebSequenceDiagrams because there are format discrepancies. In order to fill this gap, the trace transformation step was proposed.

5.1 Valid Textual Input for Diagram Generation

In WebSequenceDiagrams, there are several valid inputs for generating sequence diagrams. Step 2 addresses the question: What should execution traces look like after transformation to make them valid for WebSequenceDiagrams? In the context of this paper, only three situations were focused on: normal message passings, message passings with loop structure, and message passings with alternative structure.

Normal message passaging are the most simple while common situations. From Figure 1, one can see that the most important
components in UML sequence diagrams are objects and methods as passing messages. In WebSequenceDiagrams, a basic passing message has the signature of:

**Caller** → **Callee** : method

In the sequence diagram shown in Figure 1, the first message – user attempts to log in – can be expressed as:

user → loginPage : login()

Besides basic expressions for normal message passings, two other important features in UML sequence diagrams are of interest to this research: “alt” and “opt”. “Alt” means alternative. It is similar to the “if...else...” structure in a high level programming language. In WebSequenceDiagrams, the expression for an alt structure is:

```
alt condition 1
  A → B: method1
else condition 2
  A → B: method2
end
```

“Opt” means optional. It is similar to “alt”, but “opt” only provides the situation with “if...” but no “else...” The expression format for “opt” is:

```
opt condition
  A → B: method
end
```

To conclude, the goal of this process is to transform execution traces from the format shown in Figure 4 to the same format shown in the above expressions.

### 5.2 Loop Structure Identification

In a trace, there may exist loop structures. The character of loops shown in execution traces is that one execution trace or several execution traces binding as a group repeat at least two times continuously among all the execution traces. In this paper, loop structures are identified through Longest Repeated Substring (LRS). The implementation for finding the LRS is based on suffix trees [27]. For execution traces in one system execution, LRS must first be found. If these traces contain one LRS, then it must be determined whether the occurrences of this LRS are continuous for all the positions it appears. If these are such LRSes, their occurrences are recorded and all these LRSes are transformed to loop structure. At last, these LRSes are placed with empty substring. The above process is iterated to find the second-longest repeated string, third-longest repeated string, so on and so forth. The search stops when there are no LRS in the execution traces or all the execution traces are replaced by empty substrings.

### 5.3 Alternative Structure Identification

If alternative structures are to be identified, at least two execution trace files are needed. When execution traces merge from different trace files, common traces must first be found according to the order they appear. Then these common traces must be fixed, and the different traces replaced with “alt”. If one of them is null, then the structure can be refined as “opt” in the results.

### 6. Diagram Generation

In this section, the last step in the proposed framework will be discussed. The last step is UML sequence diagram generation. In this step, another external tool – WebSequenceDiagram\(^1\) is deployed. WebSequenceDiagrams is an online tool that generates UML sequence diagrams. This tool was chosen for the following reasons:

- Like InTrace, WebSequenceDiagrams is easy to access and free to use. Besides, it provides the plug-ins and APIs, which is convenient for users to import the tool as an external resource.
- The syntax for generating sequence diagrams is very simple. It is straight-forward and users could grasp the grammar in a short time.
- The product produced by WebSequenceDiagrams is high-quality and the generation time is short.

Besides these advantages in WebSequenceDiagrams, another important reason why WebSequenceDiagrams was chosen is the fact that the input of WebSequenceDiagrams is textual-based, which is consistent with the first tool chosen for TTEXTS. The nature of textual trace collection and textual trace used for diagram generation makes WebSequenceDiagrams a good choice for this work.

---

\(^1\)https://www.websequencediagrams.com/
By employing existing API’s provided by WebSequenceDiagrams, the diagram generation function can be embed in TTExTS. In this step, the input is execution traces after transformation and merging, and the output is a UML sequence diagram.

7. Case Study

In order to test the feasibility of the proposed framework, a case study was conducted. The case selected for this study is a simple Java program (2 classes interact with each other through four different methods) with a structure like this:

```java
if Situation 1
  A
else Situation 2
  B
  C
  D
```

By checking processing time, TTExTS was found to spend ~700ms on trace collection, ~70ms on trace transformation, ~80ms on trace merging and ~300ms on generation of the UML sequence diagram. The result of this case study is shown in Figure 5.

The result shows that the proposed framework can accomplish the reversing of UML sequence diagrams from system execution. The processing time is reasonable and the diagram quality is high. However, the size of the testing case in this study is limited and it just proves the feasibility of the proposed framework. In order to mitigate this weakness, the framework is currently undergoing more complicated system testing.

8. Discussion

8.1 Limitations

In this section, some limitations associated with the major steps in this study will be discussed.

8.1.1 Tool Selection

The external tools selected for this study, InTrace and WebSequenceDiagrams, prove to be able to finish the task of the reversing of UML sequence diagrams. However, there may still exist better tools to accomplish the goal. The principal criteria for tool selection in this study are easy of access and free to use. These standards may sacrifice the performance and quality. The case study conducted for this proposed framework shows that the trace collection process occupies over 50% of all the processing time. However, this limitation is hard to mitigate in the context of this study because TTExTS needs the support of these external tools.

8.1.2 System Testing

This is the main threat to the validity of this work. In this paper, TTExTS is evaluated by case studies with limited size. However, for more realistic scenarios, the cases may be more complicated. The test cases in this study are only showing the feasibility of the prototype. To mitigate this, more complicated system testings are currently being evaluated.

8.1.3 Performance Evaluation

In this paper, although it was proved that TTExTS can accomplish the reversing of UML sequence diagrams dynamically, no comparisons are made between the proposed prototype and other existing tools, such as MaintainJ and JavaCallTracer. Therefore, the performance of the TTExTS framework cannot be evaluated. To mitigate this limitation, performance comparisons with similar state-of-the-art techniques are planned for future work.

8.2 Future Work

Future work will span three directions – system improvement, system evaluation, and system integration.

8.2.1 System improvement

System improvement consists of two aspects: system testing and algorithm optimization. In this paper, testing work was done to TTExTS but with limited case size. In the future, more complicated testing cases will be employed to test the proposed framework. Additionally, algorithm optimization will be explored in future work. For this paper, string iterations are widely used in the prototype implementation. Though it can achieve the processing goals, it may suffer from low efficiency. In the future, better algorithms will be investigated to optimize the prototype.

8.2.2 System evaluation

A very necessary future work may be the evaluation of the proposed system compared with similar tools (such as MaintainJ, JavaCallTracer) when testing the same programs. Comparisons between different tools could provide a better understandings of the advantages and weaknesses of the proposed framework. Comparisons may include indicators such as processing time and product quality, etc.

8.2.3 System integration

In order to be more accessible for developers, it is intended that the proposed system will be integrated into a plug-in for Eclipse IDE in the future. Plug-ins are convenient for users to employ external tools into their own systems, especially for starters. Besides, Eclipse provides a powerful plug-in
management system which allows users to download and use different kinds of plug-ins easily and free.

9. Conclusion

In this paper, a framework for dynamic reversing of UML sequence diagrams was proposed. By adopting existing tools, InTrace and WebSequenceDiagrams, a novel approach to extracting UML sequence diagrams from system execution was built. The case study shows that the proposed framework could solve the reversing process in an acceptable time and provide high quality sequence diagrams as final results.

References