Using Real Data to Animate SOFL Formal Specifications Automatically

Mo Li and Shaoying Liu
Faculty of Computer and Information Sciences, Hosei University
mo.li.3e@stu.hosei.ac.jp and sliu@hosei.ac.jp

Abstract
To ensure the quality of final software products, it is very important to verify and validate the formal specifications before their implementation. Specification animation is realized as a very useful technique for verification and validation. It provides the end user with an intuitive way to observe the behavior of the software system described in specification. In this paper, we propose an approach to animate the specifications written in Structure Object-oriented Formal Language (SOFL). The animation strategy underlying this approach is using system functional scenario as a basic animating unit and using data as connection to connect each independent operation involved in one system functional scenario. We describe this strategy and the process of using it in practice. And a prototype that support this approach is shown at the end of the paper.

1. Introduction

To ensure the quality of final software products, it is very important to verify and validate the corresponding formal specifications before their implementation. Formal specification animation is realized as an effective technique for specification verification and validation. It provides an intuitive way to the end user to monitor behaviors described in the specification, and it is helpful for the end user to understand the system. Several tools have been built to support animating formal specifications written in different formal languages, such as ANGOR[1], and B-Model animator[2]. Most of them require a translation from a formal language to an executable programming language in order to achieve a full automation. But the translation may impose many restrictions to the style of the specifications written in a formal notation and this would bring inconvenience to the developer.

In this paper, we describe an automatic approach for SOFL[3] specification animation. There is no restrictions to the language or style of specifications. In this approach, the end user can animate the specification on the Conditional Data Flow Diagram (CDFD, a part of SOFL specification) directly. The animation strategy underlying this approach is using system functional scenario as basic animating unit and using data as connection to connect each independent operation involved in one system functional scenario. System functional scenario is used as the basic unit of an animation since it describes a specific behavior of the system. To ensure that the entire specification will be animated, all of the system functional scenarios defined in the specification will be animated.

Formally, a system functional scenario can be defined as a sequence of operations that processes a group of input data to a group of output data, and each operation in scenario is connected by intermediate data. In an animation, real data are used to connect all operations in the scenario instead of the intermediate data. The user can select to provide the data themselves or let the data be generated automatically. If the user selects to provide the data for animation, they usually provide the most typical data of the system. Meanwhile, the user need to guarantee that the provided data satisfy the pre- and post-conditions of the corresponding operations. If the user wants to let the data be generated automatically, a data generation method would generate the data that satisfy the pre- and post-conditions. But the generated data may not present the specific circumstance the user wants to animate.

2. Animation Strategy and Process

As mentioned previously, we use the system functional scenario as the basic animating unit. Theoretically, a system scenario defines a specific operational behavior through a sequential executions of operations, which is usually presented to end users as a pair of input and output. That is, given an input, the result of a behavior of the system results in a certain output. For example, Fig. 1 shows the CDFD of a simplified ATM with only two functions. Totally five scenarios are defined in the specification, and one of them is listed as follow. Given the input "*withdraw*", the output "*cash*" is processed by consequentially executing three operations.

- ✓ *{withdraw_com}[Receive_Command, Check_Password, Withdraw]{cash}*

Each independent operation involved in the scenario is connected by intermediate data. To animate a specific system scenario, the real data are used to connect all operations instead of the intermediate data. Since the data are restricted by the pre- and post-conditions of each operation, the data present a real environment of the behavior. The end user can observe the behavior of system by monitoring the data, and the data provide a concrete point of views of the behavior.

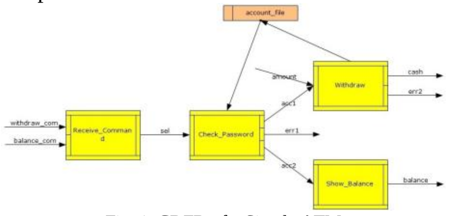


Fig. 1. CDFD of a Simple ATM

The data can be collected in two ways. One way is to let the user provide the data. And the other way is to generate data automatically. The generation method does not require translating the formal specification to any executable program, but the generated data may not present the typical circumstance of the system. The data generation method is first introduced in [4], and we will not explain it further for the sake of space.

According to the strategy described above, we suggest the following process for animation in practice.

Step 1: Derive all possible *system scenarios* from the formal specification.

Since more than one system scenarios are usually defined in the specification, it is necessary to derive all possible system scenarios for animating the entire specification.

To help the reader to understand the second step, we first define the term *operation functional scenario* as follow.

Let $P(P_{iv}, P_{ov})[P_{pre}, P_{post}]$ denote the formal specification of an operation P, where P_{iv} and P_{ov} are the sets of all input and output variables. P_{pre} and P_{post} are the pre and post-condition of operation P, respectively. Let $P_{post} \equiv (C_1 \wedge D_1) \vee ... \vee (C_n \wedge D_n)$, where each C_i is a predicate that contains no output variable and $\forall_{i,\ j} \in \{1, ..., n\} \cdot i \neq j \Rightarrow C_i \wedge C_j = false$; D_i contains at least one output variable. Then, a specification of an operation can be expressed as $(\sim P_{pre} \wedge C_1 \wedge D_1) \vee ... \vee (\sim P_{pre} \wedge C_n \wedge D_n)$. A conjunction $\sim P_{pre} \wedge C_i \wedge D_i$ is called an *operation functional scenario*.

Step 2: Let $d_i[P_1, ..., P_n]d_o$ be a selected *system scenario*. Derive related *operation scenarios* of each $P_i(i \in \{1, ..., n\})$ from its specification and get a set of *operation scenarios* $\{S_1, ..., S_n\}$, where S_i is the related *operation scenario* of P_i.

In animation, the operation scenario of each operation involved the animated system scenario should be derived from the specification for collecting data. As the start point, the input data of the first process in the system scenario should be collected first.

Step 3: Let $\sim P^1_{pre} \wedge C^1_i \wedge D^1_i$ be the related operation scenario of the first operation P_1 in the selected system scenario. The input data should be collected and satisfy the predicate expression $\sim P^1_{pre} \wedge C^1_i$.

The input data collected in *Step 3* is actually the input of the selected system scenario. It can be used as the basis to collect output data of P_1, which is actually the input data of P_2. Repeating this procedure, the output data of the entire system scenario can be collected eventually. This idea is reflected in *Step 4*.

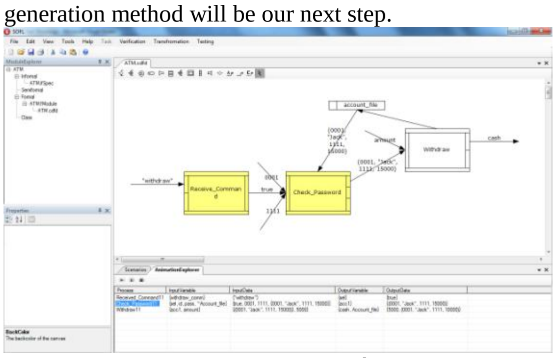
Step 4: Use the input data generated in *Step 3* and the operation scenarios derived in *Step 2* to generate the output data for each operation and entire system scenario.

Animating all possible behaviors is required in our animation strategy. The process of animating one behavior should be repeated until all of the behaviors have been animated.

Step 5: Repeat *Step 2* to *Step 4* until all the *system scenarios* derived in *Step 1* are animated.

3. Prototype

The prototype is implemented on the basis of a framework that is built to help developers specifying SOFL specification. The framework includes the editors of specification and CDFD. Fig. 2 shows the snapshot of animation board. For now, this prototype can only accept the data provided by user. Implementing the data generation method will be our next step.



Process	InputVariable	InputData	OutputVariable	OutputData
Received_Command11	{withdraw_comm}	{"withdraw"}	{sel}	{true}
Check_Password11	{sel, id, pass, "Account_file}	{true, 0001, 1111, {0001, "Jack", 1111, 15000}}	{acc1}	{{0001, "Jack", 1111, 15000}}
Withdraw11	{acc1, amount}	{{0001, "Jack", 1111, 15000}, 5000}	{cash, Account_file}	{5000, {0001, "Jack", 1111, 10000}}

Fig. 2. Animation Board

References

[1] Pierre Combes, Fabrice Dubois, and Beatrice Renard. An open animation tool: application to telecommunication systems. The International Journal of Computer and Telecommunications Networking 40(5). pages 599-620. 2002

[2] Waeselynck, H., and Behnia, S.. B model animation for external verification. Proceedings of the Second IEEE International Conference on Formal Engineering Methods. pages 36-45. 1998

[3] Shaoying Liu.: Formal Engineering for Industrial Software Development Using the SOFL Method. Springer-Verlag, ISBN 3-540-20602-7. 2004

[4] Mo Li, Shaoying Liu. Automated Functional Scenarios-based Formal Specification Animation. Proceedings of the 19th Asia-Pacific Software Engineering Conference (APSEC 2012). IEEE CS press, pages 107-115. 2012