Design and Implementation of Operation Replay for Virtual Experiment

Jiufei Tang, Xingming Ouyang*, Junqing Yu, and Liefu Ai

Computer College, Huazhong University of Science & Technology,
Wuhan, Hubei, 430074, China
Tangjiufei@hust.edu.cn, ouyangxingming@hust.edu.cn

Abstract. Although virtual experiment systems have been widely used in universities and colleges, many functions, such as experiment demonstration and experiment process analysis, need to be extended. Operation replay is the core and foundation of these 2 functions. Under MFC framework, the mechanism of operation replay is studied. A method based on message capturing and sending is proposed and the feasibility of this method is proved by using DFA theory. Experimental results demonstrate that the proposed method is effective and can be realized in existed MFC-based VESs easily. The idea of operation replay has some referential value for other e-learning system researchers and developers .

Keywords: virtual experiment system; operation replay; valid message.

1 Introduction

E-Learning is becoming more and more popular in recent years. It can improve the flexibility and quality of education effectively. Various kinds of e-learning systems have been developed and implemented. They bring a lot of benefits and conveniences to learners and educators. [1,2]

However, learners can not do experiments in these systems. As a supplement to these e-learning systems, VESs(Virtual Experiment System) have attracted lots of interest. VES is a software-created environment, in which learners can do experiments as if in a real laboratory. VESs can reduce the dependence of experiments on time, space and hardware, can cut down the cost to run and maintain laboratories. The fundamental functions of a VES are to simulate real experiment hardware and real experiment processes. Most of the current VESs fall into this category. [3,4]

Furthermore, after virtualization of experiment systems, with the powerful computing and storage abilities of computer and network, a lot of functions can be extended. Ref[5] proposed a method to monitor and diagnose experiment process, by modeling experiment steps with Petri-Net.

This paper investigates the functional requirements of operation replay in VESs, proposes the mechanism and scheme to design and implement operation replay under MFC framework. Experimental results demonstrate that the proposed method is feasible and effective.

*Corresponding author.

2 Concept of Operation Replay

Replay or execution replay is not a new concept. It is developed originally to support fault tolerance in distributed computing. It refers to the ability to reconstruct the past execution of a system or program. Ref [6] proposes a context-sensitive method to capture and replay program states for Java software. It is used to debug and analyze execution of the program. Ref [7] describes the design of a tool for capturing and replaying Java program executions. It is used for debugging and execution profiling.

Anyway, the general meaning of “replay” is too wide for VES field, restricting and clarifying need to be made for our research purpose.

Definition 2.1. Operation: Operation refers to the actions experimenter takes to construct or edit a virtual circuit in a VES, including adding, deleting, moving and modifying of virtual instruments or apparatus, by using input devices of a computer. Typical actions include keyboard strikes and mouse clicks. Several continuous operations in an experiment compose an operation sequence.

Definition 2.2. Operation Replay: Operation Replay refers the function described as below: experiment operations during an experiment are recorded and saved into a file. Sooner or later, the experimenter or others can open the file and watch the whole processes of the experiment. The former execution is called “original run”, the later is called “replay run”.

Based on operation replay, 2 useful applications can be extended in VES. The first is experiment demonstration. This can be realized by recording the operation, sending it to other computers one by one, then the receiver displaying the operation. The second is experiment process and result analysis. Experimenters save the operations to files and hand over the files to the instructor. The instructor can replay the experiment processes later in his own computer, check and analyze the experiments. He can give scores to the experimenters by watching operation replay too.

A direct and simple method of operation recording is screen recording, i.e., record the changes of computer screen during one period of time, and save as a stream media file, and then replay it by standard media players. There is much software to do this, such as Desktop Screen Recorder 5 [8]. Most of teaching supporting systems (such as Blackboard[9]) provides this function too. One advantage of this method lay in that this method is irrelevant to application programs, and can be implemented easily.

Anyway, this method captures and processes screen images, occupies a number of computing resources. The size of the stream media file is quite large too. So, screen recording is not suitable for operation replay in virtual experiment systems.

Ref[6] and Ref[7] propose designs of replay for Java software, but the replay is for debugging or execution analyzing.

In VES, the purpose of replay is to monitor or analyze the operations, to analyze the effects of experiment courses or classes. We need to analyze the particular environment of VES and propose corresponding design and implementation method for operation replay in VES.

3 Design of Operation Replay

According to the definition of operation replay, it can be implemented through 3 steps: operation capture, operation saving and operation recurring. Now we launch on the design for these 3 steps.

3.1 Operation Capture

This section need to determine what can represent user operations, where, when and how to capture them.

For a standard MFC program, the responding procedures are shown as Fig. 1.

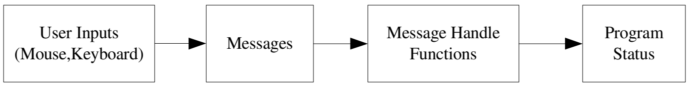


Fig. 1. Responding Procedures of a MFC Program

All operations are taken through input devices in the "User Inputs" layer. Inputs generate messages, MFC maps messages and the message handle functions, message handle functions change program status. Below, we try to determine which layer is the most suitable source for replay.

User Inputs: user's inputs represent operations directly. Operation capture can be implemented by recording all the inputs in this layer. But one problem arises when recurring. There need a virtual mouse and a virtual keyboard to recur the saved inputs. There are lots of things to develop these 2 virtual apparatuses. Therefore, direct user input is not an ideal source for replay.

Program status is the result of operations. It is static, it can not display a relatively integrate process of the operations. There would not get a good effect simply by recurring program status.

Therefore, we should find the capture source in the third layer. Let's check 2 properties of message in MFC.

The first is that message can be handled easily in MFC program. The second is that valid message has high fidelity to the operations. Below is brief analysis of this property.

In a MFC program, suppose that the operation sequence set is: *Operations* = $\{a_1, a_2, a_3, ..., a_M\}$, message sequence set is: *Messages* $=\{b_1, b_2, b_3, ..., b_N\}$. The relationships of operations, messages and message handle functions are shown as Fig. 2.

One operation could invoke several messages; 2 different operations could invoke a same message. So, we need to give out a strict definition of valid operation as below.

Definition 3.1. Suppose that the operation sequence is *Operations*, message sequence is *Messages*, if for any operation $a_i \in$ *Operations*, $\exists\, b_j \in$ *Messages*, b_j is created by a_i, and there is a message handle function corresponding to b_j, then we call a_i valid operation, all the valid operations compose a valid operation set *validOperations*, all the corresponding messages compose a valid message set *validMessages*.

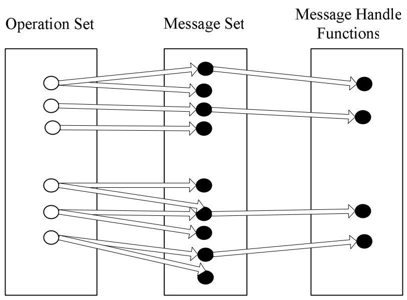


Fig. 2. Mapping of Operation, Message and Message Handle functions

According to Definition 3.1, we obtain a subset of operation sequence *Operations*, that is valid operation sequence *validOperations* = { a'_1, a'_2, a'_3 ,..., a'_N }, the number of element is N. This sequence and the valid message sequence *validMessages* compose a one-to-one map: f(a'_i) = b_i.

This is the property of valid messages: high fidelity to the operations. This property, together with the easily-to-handle property, makes valid message a good source for operation capture. Therefore, we choose the valid message set *ValidMessages* as the source of operation capture.

Operation capture is implemented in the message handle function of a valid message. The order of message handle function invoking is identical with the order of user's operations. Therefore, the message sequence captured in these functions keeps the same order of operations.

3.2 Operation Saving

From the above analysis, the saving of operations is transformed to saving of messages.

The data structure to save message is defined as below:

```
  struct Message {  int message;  int object;
            WPARAM parameter1;   LPARAM parameter2;  };
```

Here, Object refers to the program classes or objects to send messages. It is coded with integer. "parameter1" and "parameter2" are parameters send to message handle functions in Windows system, the data types are WPARAM and LPARAM respectively.

All the messages are saved to a linked list. This linked list, together with the global setting of VES, is saved into a file. We call this file "operation file" of the experiment.

3.3 Operation Recurring

Two problems need to be solved in this section: to prove that the original operations can recur from the message sequence saved in the file, and to design some methods to implement operation recurring.

3.3.1 Proving of the Feasibility

Definition 3.2. From the starting, if a MFC program reached a certain stable program environment, we call this is a status of the program, marked as S , and the status just after the starting is called initial status, marked as S_0. If a status S_i is reached after the handling of a message b_i , we call that status S_i is corresponded to message b_i , marked as $T(b_i) = S_i$.

Theorem 3.1. After a series of operations, A MFC program reaches to a certain status. Suppose that, the valid message set is *Valid-Messages* = $\{b_1, b_2, b_3... b_M\}$, correspongding status set is *Status* = $\{S_0, S_1, S_2, S_3... S_M\}$, where S_0 is the initial status, $T(bi) = S_i$, $1 \le i \le M$. Then, a DFA (deterministic finite automaton) can be constructed as below: M = {Status, UB, T, S_0, Status}.

The status diagram of this DFA is given in Fig. 3.

Now we prove the feasibility of operation replay by using Theorem 3.1.

Proving: From the linear DFA in Fig. 3., we can make a statement: for 2 runs of a MFC program,if both the initial status and the input sequence are identical, the 2 status sequences of the 2 runs will be identical. Because we record the whole valid message sequence of the original run. The message sequence of the replay run is read from the file, it is identical with the original one. And, we can take some methods to assure the initial status of the 2 runs to be identical.Thus, the staus sequences of the original run and the replay run will be identical.And status squence is the result of user operations. So, operation replay can be realize by message recording and recuring.

Prove finished.

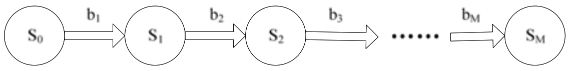


Fig. 3. Status Diagram of the DFA

The core of operation replay is to recreate the status sequence a program responses to operations, because the ultimate purpose of operations is to get expected status changes.

3.3.2 Implementation of Operation Recurring

From the analysis and design upwards, operation recurring can be implemented by steps as below.

Step1. Open the operation file, and initialize program status;
Step2. Read a record and send the message to corresponding program objects;
Step3. Wait until the end of the message handling;
Step4. Repeat step2 and 2 until the end of the operation file;
Step5. Recurring ends.

4 Realization of Operation Replay in IVDLEP

Operation replay has been realized in an actual experiment system: IVDLEP (Interactive Digital Logic Virtual Experiment Platform).

4.1 Introduction of IVDLEP

IVDLEP is a virtual experiment platform developed for experiment teaching of the course “Digital Logic”. The platform provides the virtual apparatus of all the physical apparatus used in real experiments, including virtual experimental board, virtual power, virtual chips, virtual sockets, virtual switches, virtual LEDs or lamps, virtual oscilloscope, virtual wires and so on. User can choose a virtual apparatus according to his own design, and place it on the virtual experimental board. Various kinds of combinational logic circuit and sequential logic circuit can be constructed and simulated on this platform. Fig.4 is the virtual circuit of full-adder, a typical combinational logic circuit.

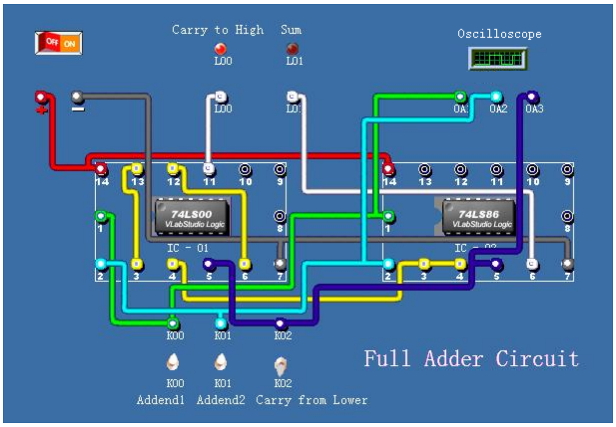


Fig. 4. Virtual Circuit of a Full-Adder

4.2 Realization of Operation Replay

According the analysis and design in section 3, the steps to realize operation replay in an existed system are listed as below.

Step 1.Add definition and declaration of data structures and variables;

Step 2.Read and analyze the design document and code of the system, find out the objects or classes which are related with user inputs, then code these objects or classes;

Step 3.Search for the message handle functions related to user inputs; then add code to capture the corresponding message into the linked list;

Step 4.Add menu items, dialogs and corresponding codes;

Step 5.Compile, link, and test and release the system.

We realized operation replay in IVDLEP by these 7 steps. Some notes are given out as below.

(1) In IVDLEP system, we found 4 classes related to user inputs: CMainFrame, CView, CTreeCtrl, and CMiniPlatform. CMainFrame responds to menu item clicks, toolbar clicks; CView class is corresponding to virtual experimental board, it handles actions of virtual apparatuses on the board, such as placing and moving; Ctreectrl is corresponding to virtual apparatus library, it handles virtual apparatuses choosing; the

last one, CMiniPlatform is a full-view of the virtual board, it handles virtual board navigating. We code these 4 classes as 1,2,3,4, respectively.

(2) In the source codes, we find about 91 message handle functions related to user inputs, and add codes to capture the operations.

(3) Operation replay control panel is designed as shown in Fig.5.

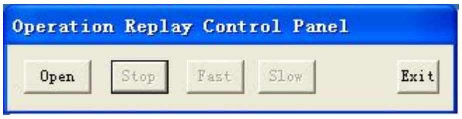


Fig. 5. Operation Replay Control Panel

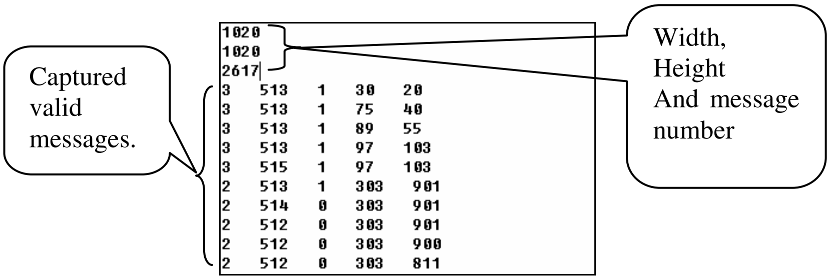
4.3 Testing Result

Using the new released version of IVDLEP, we re-construct the virtual circuit of full-adder in Fig.4. The message list is saved into a file. Some operations are listed in Table.1. The content of the operation file is shown in Fig.6.

Table 1. Some Operations to construct a full-adder

No.	Operation description	Object to send messages
1	Add a Power switch	3, CTreectrl; 2,CView
2	Add power pins “+”, “-”	3, CTreectrl; 2,CView
3	Add “L00”,”L01” LED	3, CTreectrl; 2,CView
4	Add 2 14-pin sockets	3, CTreectrl; 2,CView
5	Add 3 switches: K00,K01,K02	3, CTreectrl; 2,CView
6	Add a “74LS00” chip	3, CTreectrl; 2,CView

It is easy to understand that the operation sequences to construct a same virtualcircuit are usually different for different experimenters or different times of a same experimenter.



```
1020
1020
2617
3    513   1   30    20
3    513   1   75    40
3    513   1   89    55
3    513   1   97    103
3    515   1   97    103
2    513   1   303   901
2    514   0   303   901
2    512   0   303   901
2    512   0   303   900
2    512   0   303   811
```

Fig. 6. Content of the Operation file

For coding convenience, a flag with data type “Int” is inserted into the third field of the struct to store the captured messages.

We open the saved file and replay the operations. The experiment processes are recurring with the exact order as the original run. At the end of the replay run, the expected virtual circuit of full-adder as shown in Fig.4 is constructed.

5 Conclusion

This paper investigates the mechanism of operation replay in virtual experiment system, proposes a message-based method to implement. The mechanism and method has been realized in an actual experiment system: Interactive Digital Logic Virtual Experiment Platform. The mechanism and method is applicable for other MFC-frame software. The idea of input capture and message sending might have some referential value for other e-learning system researchers and developers too.

Anyway, there are shortcomings for this method. It is applicable strictly to MFC-based applications. The initial status of original run and replay run must be identical.

For future work, more flexible mechanism or method for operation replay should be investigated. More applications of operation replay in VES or other e-learning system need to be found.

References

1. Zhou, D., Zhang, Z., Zhong, S., Xie, P.: The Design of Software Architecture for E-Learning Platforms. In: Pan, Z., Zhang, X., El Rhalibi, A., Woo, W., Li, Y. (eds.) Edutainment 2008. LNCS, vol. 5093, pp. 32–40. Springer, Heidelberg (2008)
2. Yun, R., Pan, Z., Li, Y.: An Educational Virtual Environment for Studying Physics Concept in High Schools. In: Lau, R., Li, Q., Cheung, R., Liu, W. (eds.) ICWL 2005. LNCS, vol. 3583, pp. 326–331. Springer, Heidelberg (2005)
3. Ma, J., Nickerson, J.V.: Hands-on, Simulated, and Remote Laboratories: A comparative Literature Review. ACM Computing Surveys 38(3), Article 7 (2006)
4. Xingming, O., Xiaolong, Y., Xinrong, X.: Design and Implementation of Virtual Lab Based on Internet. Computer Engineering (in Chinese) 30(4), 185–186 (2004)
5. Chang, J.-C., Li, S.-C.: Monitoring the Experiment Process and Diagnosing the Experiment Mistakes Made by Students with Petri Net Modeling. In: Pan, Z., Aylett, R.S., Diener, H., Jin, X., Göbel, S., Li, L. (eds.) Edutainment 2006. LNCS, vol. 3942, pp. 108–115. Springer, Heidelberg (2006)
6. Xu, G., Rountev, A., Tang, Y., Qin, F.: Efficient Checkpointing of Java Software Using Context-Sensitive Capture and Replay. In: Proceedings of the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, pp. 85–94 (2007)
7. Steven, J., Chandra, P., Fleck, B., Podgurski, A.: jRapture: A Capture/Replay Tool for Observation-Based Testing. In: Proceedings of the 2000 ACM SIGSOFT international symposium on Software testing and analysis, pp. 158–167 (2000)
8. Desktop Screen Record5, `http://www.recordscreen.com/`
9. Blackboard (2005), `http://www.blackboard.com/`