Supporting Online Synchronous Education for Software Engineering via Web-based Operation Record and Replay

Dejian Chen, Yanchun Sun*, Kui Wei, Zijian Qiao, Chao Xin
Institute of Software, School of Electronics Engineering & Computer Science, Peking University, Key laboratory of High Confidence Software Technologies, Ministry of Education, Beijing 100871, P.R.China
Email: {chendj12, sunyc, weikui13}@sei.pku.edu.cn, {qiaozijian, xinchao}@pku.edu.cn

Abstract—Influenced by web 2.0 and cloud computing, web applications such as online modeling tools and web-based IDE develop rapidly. They are important for Software Engineering education because practice is crucial for students to get better understanding of the concepts introduced in class. However, most of these web applications are designed for individual usage, lacking support for real-time interactions. To solve this problem, we develop and demonstrate an Online Synchronous Education Plugin (OSEP), which is based on web-based operation record and replay. On one side, OSEP supports basic synchronous and interactive education on existing non-interactive web applications by high-fidelity record on the lecturer’s terminal and self-adaptive replay on the observer’s terminals. On the other side, OSEP ensures reliable, orderly synchronization and interaction by replay smoothing and latecomer controlling. In case study, we illustrate how OSEP is used in the real-time interactive education of online UML and web-based IDE teaching, which verifies the feasibility of the online synchronous education method for Software Engineering via web-based operation record and replay.

Keywords-Web applications; Software Engineering Education; Operation Record and Replay; Online Synchronous Education

I. INTRODUCTION

With the development of web 2.0 and cloud computing, online education based on web browser develops rapidly. Software Engineering education also benefits from this trend, and more and more web-based platforms, tools and web applications are developed to aid the teaching process.

From 2012, MOOC (Massive Open Online Course) [1] has been increasing dramatically around the world such as edX, Coursera and Udacity. By connecting to the Internet with a browser, any student around the world can easily access resources on MOOCs, such as high-qualified video lectures, instant-feedback tests and interactive user forums. However, practice is a crucial factor in Software Engineering education, and MOOCs are short of web-based supporting tools for interactive practice, such as interactive designing and coding.

With the development of HTML5 and CSS3, web applications are more and more complicated due to its graphical features. The concept “graphical” here has two implications: one directly refers to web graphics such as online UML modeling, and the other refers to Graphical User Interface (GUI) of web applications such as web-based IDE. These web applications are helpful for students to practice in Software Engineering course. However, most of them are designed for individual usage, so the lack of interaction makes them difficult to serve synchronous purpose. Re-development for interactive purpose is possible but costly.

From the introduction and analysis above, we can see that current online education platforms or web applications seldom support real-time interactive synchronization for education purpose. In this paper, we develop OSEP (Online Synchronous Education Plugin) to address this problem. Rather than re-development, OSEP adds synchronous feature into existing non-interactive web applications in an innovative way by recording each web-based operation of the instructor and replaying identical operation on the browsers of students in real time. Also, the instructor can authorize different students to display their operation to realize interaction among participants. Moreover, the whole process is recorded during the synchronization and it can be reviewed afterwards.

The rest of this paper is organized as follows. Section 2 introduces the related work. Section 3 demonstrates the architecture of OSEP. Section 4 presents the implementation of browser-side and server-side of OSEP. Section 5 introduces case study by two scenarios: online UML teaching and web-based IDE teaching. Section 6 presents concluding remarks and future work.

II. RELATED WORK

Remote desktop technique, such as Virtual Network Computing (VNC) in the Linux environment and screen sharing system DisplayCast [2], enables a computer to share
screen with or remotely control another computer over a network. Although remote desktop technique supports interactions in some degree by operating on the same computer among participants, it is not suitable for education purpose because: (a) students can only observe the process but not practice on them individually since the content cannot be copied automatically to their own computer; (b) latecomers may miss previous teaching process since they can only follow the screen of instructor in real time; (c) the process is not reusable since it can be only saved as a video, and if a student wants to reproduce the process and practice on the contents after class, he can only rebuild the contents on the browser step by step by watching the video.

Most of current collaborative web applications do not support particular graphical features in Software Engineering education. Google Docs [3] is a web-based office suite platform supporting collaborative editing on documents among different users in real time, but it does not support collaboration in complicated graphical operations. Shared white boards [4] support simple graphical drawing during brain-storm, but they are not suitable for more complex graphics and environments in education.

Our research team has put forward a method of operation record and replay technique and created an IDE-based interactive tutorial tool called SmartTutor [5, 6], which permits instructors and learners to edit tutorials on demand and create interactive tutorials. But it is based on Eclipse and does not support tutorials on web-based tools, so we further develop web-based operation record and replay techniques.

Web-based record and replay techniques are developed in some researches, but most of them are not suitable for online education purpose. CoScripter [7] is a system that allows users to capture, share, automate, and personalize business processes on the web. WaRR [8] is a tool that records and replays the interaction between users and modern web applications with high fidelity. However, these researches focus on form processing, such as buttons, check boxes and text fields, lacking support for graphical features such as capturing and automating of scalable vector graphics (SVG). Also, they do not support real-time interaction between learners in different places to finish a collaborative study.

Different from existing record and replay techniques, our approach features on “self-adaptive replay” and “real-time collaboration”. To the best of our knowledge, our approach is the first approach that supports online synchronous education for Software Engineering by using web-based record and replay technique.

III. THE ARCHITECTURE OF OSEP

OSEP is composed of two modules: the graphical web operation record and replay module ensures that operations on graphical elements can be captured and reproduced during synchronization, and the interactive synchronization control module enables interactions among participants (See Fig. 1). Lecturer and observer are two kinds of roles in the system, where the lecturer is the one who gets the display authority and the observer is the one who observes the operations from the lecturer in real time. An instructor, initially acts as the lecturer, can designate different students as lecturers in different time to realize interactive education.

A. Graphical Web Operations Record and Replay

Since graphical web applications are more complicated than normal web pages due to moving, editing and resizing of web graphics and operations on GUI, more sophisticated record and replay technology is required.

1) High-fidelity record

When recording users’ actions on normal web pages, we can only record crucial events such as “mouse click” and “keyboard input”, but for actions on graphical web pages, we need to capture more detailed events with high fidelity such as “mouse move” and “mouse over”. Raw events recorded, however, are abundant and non-semantic, so we need to encapsulate and optimize them.

There are three stages in high-fidelity record. (a) In capturing stage, raw events are captured by adding event listeners on browser. (b) In encapsulating stage, raw events are tailored to high-level operations, e.g. a combination of “mouse down”, “mouse move”, “mouse up” events on the same element can be encapsulated into a “drag and drop” operation. An operation can be described as \( \langle A, L, P \rangle \), in which \( A \) means crucial attributes set, \( L \) means location sequence and \( P \) means operation type. In the previous example, \( A = \{a \mid a = \text{coordinate attributes of target element}\} \), \( L = \{\text{DOM location sequence of target element}\} \), \( P = \text{“drag and drop”}\). (c) In optimizing stage, we filter redundant operations such as continuous duplicated “mouse move” events on the same target element.

2) Self-adaptive replay

For reusable purpose, recorded operations should be replayed on different terminals and at different time, and this feature is termed as “self-adaptive”. Due to heterogeneous environments of different terminals, the replay should adapt to the differences of screen size, resources loading speed and even versions of browsers. Due to different time points of record and replay, the website may be updated and the replay should adapt to insignificant changes of the website.

There are three stages in self-adaptive replay. (a) In positioning stage, target elements are selected from the DOM tree according to crucial attributes set \( A \) and location sequence \( L \) from \( \langle A, L, P \rangle \). (b) In resolving stage, high-level operations are reverted back into browser-level events...
according to operation type $P$ from $\langle A, L, P \rangle$. (c) In executing stage, appropriate event dispatch APIs are chosen to execute browser events according to current version of the browser.

To ensure “self-adaptive” feature, target elements are selected based on similarity metric aiming to adjust changes of website. The similarity is measured by two factors: content similarity and structural similarity. The process can be described as the following four steps.

a) Filter candidate elements

Given a recorded element description $X = \langle A, L \rangle$, element tag is obtained from attributes set $A$, and we select elements with identical tag from DOM tree as candidate elements. For each candidate element $Y$, it can be described as $Y = \langle B, M \rangle$, where $B$ and $M$ denote attributes set and location sequence respectively.

b) Calculate content similarity score

Crucial attributes set $A$ of recorded element $X$ can be described as a vector: $A = (a_1, a_2, ..., a_n)$, where $a_i (i=1,2,...,n)$ is an attribute of element. Also, for each attribute $Y_i$, its corresponding attribute vector is $B = (b_1, b_2, ..., b_m)$. To calculate similarity score of $A$ and $B$, we can use Minkowski distance to describe the difference, and then convert it to similarity. With dimension coefficient of 1, the distance is defined as

$$d(A, B) = \frac{1}{n} \sum_{i=1}^{n} |a_i - b_i|$$

The distance $d(A, B)$ is standardized, if each of the attribute pair distance $|a_i-b_i|$ is standardized with a range of $[0, 1]$. Three types of attributes are involved: nominal type, numeric type and string type.

For nominal type, the value is enumerable in DOM, such as “id”. Specially, a Boolean type attribute lies in this category whose value limits to “true” of “false”. The handling is straightforward that the attribute pair distance is 1 if both values are the same or 0 if they are different.

For numeric type, the value is a number with unlimited range. Distance can be measured by their absolute value of their difference, but it should be standardized within range $[0, 1]$ by exponential or reciprocal conversion.

For string type, the value is an array of characters. To calculate the distance between two strings, we use Levenshtein distance (or string distance) as measurement [9]. It is defined as the minimum number of single-character edits (insertions, deletions or substitutions) required to change one string into the other. With dynamic programming, it can be implemented with time complexity of $O(n^2)$. Also, to standardize the result, we divide the distance by the larger length of the strings to limit the range to $[0, 1]$.

Table 1 summarizes the standardized distance of three attribute types introduced above. After distance is computed for each attribute pair, we can get content distance of $X$, $Y$, and their content similarity:

$$Content-Similarity(X, Y) = 1 - d(A, B)$$

c) Calculate structural similarity score

<table>
<thead>
<tr>
<th>Attribute Type</th>
<th>Nominal</th>
<th>Numeric</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Examples</td>
<td>id, checked, readonly</td>
<td>clientX, height</td>
<td>href, src, value, innerText</td>
</tr>
<tr>
<td>Attribute Pair Distance</td>
<td>$d(ai, bi)$</td>
<td>$\begin{cases} 1 &amp; (ai=bi) \ 0 &amp; (ai\neq bi) \end{cases}$</td>
<td>$\frac{e^{-</td>
</tr>
<tr>
<td>Range</td>
<td>$[0, 1]$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If the website updates in replay stage, some structural changes of DOM tree may occur. For example, in web-based IDE, some new projects may have been established after the record stage, so the project list in replay stage may be different. In this case, it is probable that the sub-tree structure of the selected project shifts from one node to another node, and the location sequence of target element should not change dramatically.

We calculate the structural similarity by measuring how two location sequences are alike, or namely the similarity of $L$ and $M$ from $X$ and $Y$. Similar to content similarity, Levenshtein distance is used as the measurement. However, it differs that we are comparing DOM tree nodes rather than string characters. The key here is that how to judge whether two nodes are identical. Although we can use the content similarity model as above, it is time-consuming when DOM tree are large. Under insignificant changes assumption, we compare only tag names, node indices and the most important attributes such as “id” and “type”. In practice, it is much faster and the performance is very close to the content similarity model.

The structural similarity is calculated as:

$$Structural-Similarity(X, Y) = 1 - \frac{\text{LevDist}(L, M)}{\text{MaxLength}(L, M)}$$

d) Select the most similar element

For each candidate element $Y$, calculate content-similarity($X,Y$) and structural-similarity($X,Y$), which are denoted as $C$ and $S$ respectively. Since tag names of $X, Y$ are the same, we can allocate specific weights $W_C$ and $W_S$ according to different tags and calculate the weight-average similarity score:

$$Similarity(X, Y) = W_C \ast C + W_S \ast S$$

Elements such as $\langle input \rangle$, $\langle a \rangle$, $\langle img \rangle$ are more identifiable by their attributes, thus we allocate more weights on $C$ than $S$ ($W_C > W_S$). Likewise, elements such as $\langle div \rangle$, $\langle span \rangle$, $\langle td \rangle$ distinguish from others mainly based on their structural features, so we allocate more weights on $S$ than $C$ ($W_C < W_S$). The candidate element with the highest score is regarded as the identical one to the recorded element.

3) Applicable scope

The technology of graphical web operation record and replay is applicable to websites with “manifest” DOM structure such as SVG (Scalable Vector Graphics). If a website
is developed using flash or HTML5 canvas whose internal structure is not manifested as DOM, the technology is unsuitable due to lack of adaptability among different terminals. Besides, the reply is "self-adaptive" only if the changes of websites are insignificant. If a website has changed dramatically since recorded, i.e. the layout and content both update to a brand new version, it is necessary for users to start a new record stage on this "new" website.

B. Interactive Synchronization Control

In the process of interactive teaching, synchronous control mechanism is required to ensure that normal participants as well as latecomers can take part in the online class regardless of heterogeneous network environment.

Two issues are crucial to interactive synchronization control: server-push and disconnection.

Server-push techniques [10], such as polling, Comet, Server-sent event or HTML5 WebSocket, are required to ensure timely synchronization. There are some open-source frameworks that support these techniques, such as Pushlet for Servlet and Socket.IO for Node.js.

Disconnection occurs when networks or systems are unstable. Although it may be partially solved by some server-push frameworks using the store-and-forward strategy, some problems are unsolved in the particular education scenario. Based on graphical web operation record and replay techniques, two related disconnection issues must be addressed.

1) Replay Smoothing

Ideally, whenever an instructor acts on the website, the operation is recorded on the record-side, broadcast via the server and replayed on the browsers of the replay-side in real time. However, delay or disconnection may block the broadcast and when it resumes, the recorded operations crowd in the replay-side. Without data buffering and replay speed control, problems of disordered replaying, unclear displaying or other unexpected errors (such as resources loading speed is slow in replay-side, and operations are executed before resources are loaded) may occur. Therefore, we should adjust differences in server-push speed and replay speed, and the process is termed as "replay smoothing".

The problem is abstracted similar to the classical "producer-consumer" model in the Operating System field, where the producer is network-receiving module, the consumer is replay-executing module, and products are operations. However, there are four differences in our case: (a) operations (products) are ordinal and should be replayed in correct order; (b) network-receiving module (producer) should confirm the operations and send back requests if operations are missing; (c) the buffer is unlimited in size since JavaScript objects are extendable; (d) most importantly, producer and consumer cannot block each other in context of multithreading because JavaScript is non-blocking.

Although HTML5 WebWorker enables thread features in JavaScript, it is still in development and some browsers may not support it. Therefore, we develop a modified "Producer-Consumer" model under non-blocking and event-driven assumptions of JavaScript. The producer and the consumer are independent, and they communicate by global variables. Global variables include next, denoting the index of next operation to be replayed, buffer of operations array, and interval, denoting polling interval of consumer which is introduced in the following.

The receiving module (producer) is driven by server-push events. When new operations are received, they are stored in buffer. If buffer[next] is still empty, a request is sent for fetching the missing operations due to disconnection or delay.

The replaying module (consumer) is driven by "setTimeout" events, where "setTimeout" is a mechanism provided by JavaScript that a callback function is triggered after a given time interval on non-blocking basis. By triggering itself, the function is actually under a polling process. To increasing efficiency, in practice, we start by setting the polling interval as 100 millisecond and scales up to a maximum of 1000 millisecond if there are no new operations. Otherwise, it resets to 100 millisecond and starts the polling process again.

The schematic diagram of modified "producer-consumer" model is shown as Fig. 2, and the algorithm is shown as algorithm 1.

Algorithm 1: Modified "Producer-Consumer" Model

```
// Global Variables
next ← 0
buffer [] ← ∅
interval ← 100 ms

// Producer procedure
PROCEDURE onReceive (opt, bgnInx, endInx)
    INPUT: Received operations opt, Beginning index bgnInx
    and Ending index endInx
    put_buffer(opt, bgnInx, endInx)
    IF buffer[next] is empty AND next < bgnInx
    THEN request_server(next, bgnInx -1)

// Consumer procedure
PROCEDURE tryReplay()
    IF Page is ready AND buffer[next] is not empty
    THEN execute_replay(buffer[next])
        next ← next +1
```

Figure 2. Schematic Diagram of Modified “Producer-Consumer” Model
2) Latecomer Problem

As mentioned in [11, 12], a latecomer can be treated as an observer that has been disconnected since the beginning of the session, and it can be handled by store-and-forward strategy. Two modes can be chosen by the latecomer to catch up with current progress: the fast replay mode enables latecomers to review all historical operations in an accelerated speed, while the content replication mode skips non-crucial operations by replicating content directly in forms of DOM sub-tree in order to catch up the teaching process as soon as possible. An algorithm of HTML tree-matching based on editing distance is introduced in [13], which is adopted to address the sub-tree matching issue. At present, the function “Latecomer control” is still under construction.

IV. IMPLEMENTATION

A. Browser Side

The browser side of OSEP is developed as a Chrome Extension (http://developer.chrome.com/extensions/), which allows developers to add some functions into Chrome without diving deeply into native code. The OSEP extension is composed of two parts: content-script and console. The content-script part, embedded in the original Chrome window, captures raw events in record stage and plays back reverted content in replay stage. The console part, shown as a separate window, provides user interface and connects to server. Both parts are implemented in JavaScript and jQuery (http://jquery.com/), and they communicate with each other by message channel APIs of chrome extension.

B. Server Side

The server side of OSEP is developed as a HTTP server built on Node.js (http://nodejs.org/), a platform built on Chrome's JavaScript runtime for building fast, scalable network applications. The event-driven, non-blocking I/O features makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices. To implement server-push and client-pull techniques, we use Socket.IO (http://socket.io/) which aims to make real-time web apps possible, blurring the differences between the different transport mechanisms. As built-in audio chatting tool is also required for better communication purpose, we use WebRTC (http://www.webrtc.org/) to support Real-Time Communications (RTC) capabilities on web browser via simple JavaScript APIs.

V. CASE STUDY

In case study, we will present how OSEP is used to implement synchronous education in Software Engineering class. GUI of OSEP is shown in the left part of Fig. 3 and figure 3. Students can participate in synchronous education with teachers in the “Online Classes” page, access and load historical recorded classes in the “Resources” page, and replay the loaded class and reproduce teaching process in the “Scripts” page.

Imagine that a teacher is giving an online course to students in different branch schools of the same university, as well as students around the world. When confronted with some complicated and practical problems in software engineering class, the teacher may use OSEP to interact with students online. Two scenarios of software engineering teaching are illustrated as follows.

A. Online UML Teaching

When teaching object-oriented method, the teacher may feel more straightforward and distinct if he uses online UML applications to display the modeling process. He chooses GWT UML (https://code.google.com/p/gwtuml/), an open source online UML modeling tool on Google Code. As GWT UML is implemented in HTML5 SVG techniques, OSEP is capable of supporting the synchronization on it.

The teacher starts by creating a class diagram, creates and renames several classes, and then adds relationships among them (see Fig. 3). While the teacher is operating on the graphics, students can observe identical graphical status by automatically replaying the same operations on their browsers. Some students are late when they join the class, and they can choose to synchronize the teaching process with: the fast replay mode where teachers operations are recurred step by step; or the content replication mode where the teacher’s current graphics are sent and replaced in the terminals of latecomers.

If the teacher wants to interact with students, he can pause and authorize a student to play the role of “lecturer”, and the student’s operations will be synchronized and displayed on other terminals simultaneously. Also, the teacher can designate another student to operate and display on the existing model to support interaction. Finally, when the synchronous teaching ends, a student can continue to practice on the online UML modeling application, which is not supported by remote desktop techniques.

B. Web-based IDE Teaching

When teaching Computer-Aided Software Engineering (CASE), the teacher may use more concrete examples to illustrate the advantages of using IDE to develop software. He chooses Cloud 9 (https://c9.io/), one of the most popular free

![Figure 3. Online UML Teaching Scenario](image-url)
online IDEs. Although cloud9 itself supports real-time collaborative editing in the editor, among users just like Google Docs, the operations on other GUI elements such as menus and menu items cannot be synchronized and displayed. One of the similar work focusing on collaboration in a web IDE is introduced in [14], but it is customized and not extendable. Instead, OSEP enables the record and replay of GUI elements, and is suitable for education.

The teacher starts by showing how to develop a project using various tools on the IDE by simply operating on GUI. In the process of configuring, coding, compiling, testing and running the software, some operations can be complicated and time consuming. When the teacher clicks the menu, opens a new dialogue, edits parameters, submits the modification, and so on, students can observe all these operations so that they can learn how to develop software step by step (See Fig. 4).

When the teaching process is finishing, the teacher can save the recorded operations as a tutorial and share it to students as after-class study material. Also, students can use the tutorial to configure a new project automatically by replaying operations on GUI, which is not supported by remote desktop.

VI. CONCLUSION

We can conclude that OSEP supports synchronous and interactive education for Software Engineering on graphical web applications, which verifies the feasibility of the synchronous education method based on graphical web operation record and replay. In future work, we would further improve latecomer control to make OSEP more suitable for usage of synchronous education.

ACKNOWLEDGMENTS

This effort is sponsored by the National Basic Research Program of China (973) under Grant No. 2011CBB302604, the Joint Fund of the National Natural Science Foundation of China under Grant No. U1201252, and the Science Fund for Creative Research Groups of China under Grant No. 61121063.

REFERENCES