SmartTutor: Creating IDE-based Interactive Tutorials via Editable Replay

Ying Zhang, Gang Huang*, Nuyun Zhang, Hong Mei
Key Laboratory of High Confidence Software Technologies, Ministry of Education
School of Electronics Engineering and Computer Science, Peking University, Beijing, 100871, China
{zhangying06, huanggang, zhangny04}@sei.pku.edu.cn meih@pku.edu.cn

Abstract

Interactive tutorials, like Eclipse's cheat sheets, are good for novice programmers to learn how to perform tasks (e.g., checking out a CVS project) in an Integrated Development Environment (IDE). Creating these tutorials often requires programming effort that is time-consuming and difficult. In this paper, we propose an approach using editable replay of user actions to help authors create interactive tutorials with little programming effort. User actions of performing a task can be recorded, edited, and presented as a tutorial. The tutorial can be replayed interactively for mentoring. We present our SmartTutor implementation in the Eclipse IDE and conduct a preliminary evaluation on it, which demonstrates efficiency gains for the tutorial authors.

1. Introduction

Novice programmers often face a steep learning curve when performing tasks in an Integrated Development Environment (IDE). These tasks commonly require them to navigate through many views and dialogs, and input various parameters in the process, which often confuse these newcomers.

Fortunately, novices can get help from tutorials that provide step-by-step instructions on IDE-based tasks. These tutorials can be generally categorized into two types: non-interactive and interactive.

1. Non-interactive tutorials usually give instructions in the form of textual documents (some may include illustrations and video/audio clips) that describe the task process. Learning them requires novice programmers to perform actions on various UI objects of the IDE manually.
2. Interactive tutorials such as cheat sheets [1] in Eclipse can do more to help programmers: locating onscreen UI objects mentioned in the text; highlighting instructions to help novices avoid missing steps during learning; automatically performing some steps while leaving some to be performed manually, etc.

Interactive tutorials are good for novice programmers, but the creation of these tutorials is laborious [6]. It often requires programming effort that is time-consuming and difficult. For instance, in order to associate the descriptive elements within the documentation with the actual application UI, cheat sheet authors have to write some predefined commands or program some special functions. Let alone the effort spent on dividing the instructional steps to form a task outline and validating these programmed functions in the actual context.

To address this problem, we propose an approach using editable replay of user actions to create interactive tutorials with little programming effort involved. A user action is an operation step users performed via input-devices such as the mouse and keyboard to operate the IDE. This approach is based on the observation that IDE-based tutorial learning requires a learner to perform step-by-step operations on various UI objects of the IDE, such as wizards or editors. If we can record and replay all user actions of performing a task, we open up the possibility of demonstrating them as a tutorial to novice programmers.

To investigate this approach, we build SmartTutor that helps authors create interactive tutorials with little programming effort. It is a plug-in based on our previous work SmartReplayer [2] in the Eclipse IDE. SmartTutor can 1) record user actions of performing a task; 2) convert the parameter-input actions into template variables for replacing with new parameters during replay; 3) display the recorded actions in a tree structure to help authors create a task outline and add textual instructions; and 4) present and replay these edited actions as an interactive tutorial for mentoring new users just like Eclipse's cheat sheets do.

The major contributions of this paper are:

- A new solution to reduce the burden of creating IDE-based interactive tutorials;
- An implementation called SmartTutor to support the creation of interactive tutorials for the Eclipse IDE;
- The results of a preliminary evaluation of SmartTutor.

2. An Illustrative Example

In this section, we take creating an interactive tutorial for the task of checking out a CVS project by Eclipse v3.3 as an Example. This task often confuses novice programmers, because there are several wizard pages to go through and various parameters to input in the process.

Cheat sheet is a kind of interactive tutorials provided by the Eclipse IDE. A cheat sheet consists of several steps (may have sub-steps) with textual instructions for guidance. Some steps can be performed automatically, and the others are described so that users can manually perform them. In order to create a cheat sheet for good mentoring, an author has to not only write detailed instructions for each step, but also program to make the learning process interactive. For instance, to help users bring up the "Add CVS Repository" dialog of Eclipse, a cheat sheet author has to write a special Java class that implements the "ICheatSheetAction" interface. If the author wants to make the textfield UIs such as "Host" and "Repository path" to be focused or highlighted, he also has to write some other classes for that purpose.

* Corresponding author

ICSE'09, May 16–24, 2009, Vancouver, Canada
978-1-4244-3452-7/09/$25.00 © 2009 IEEE

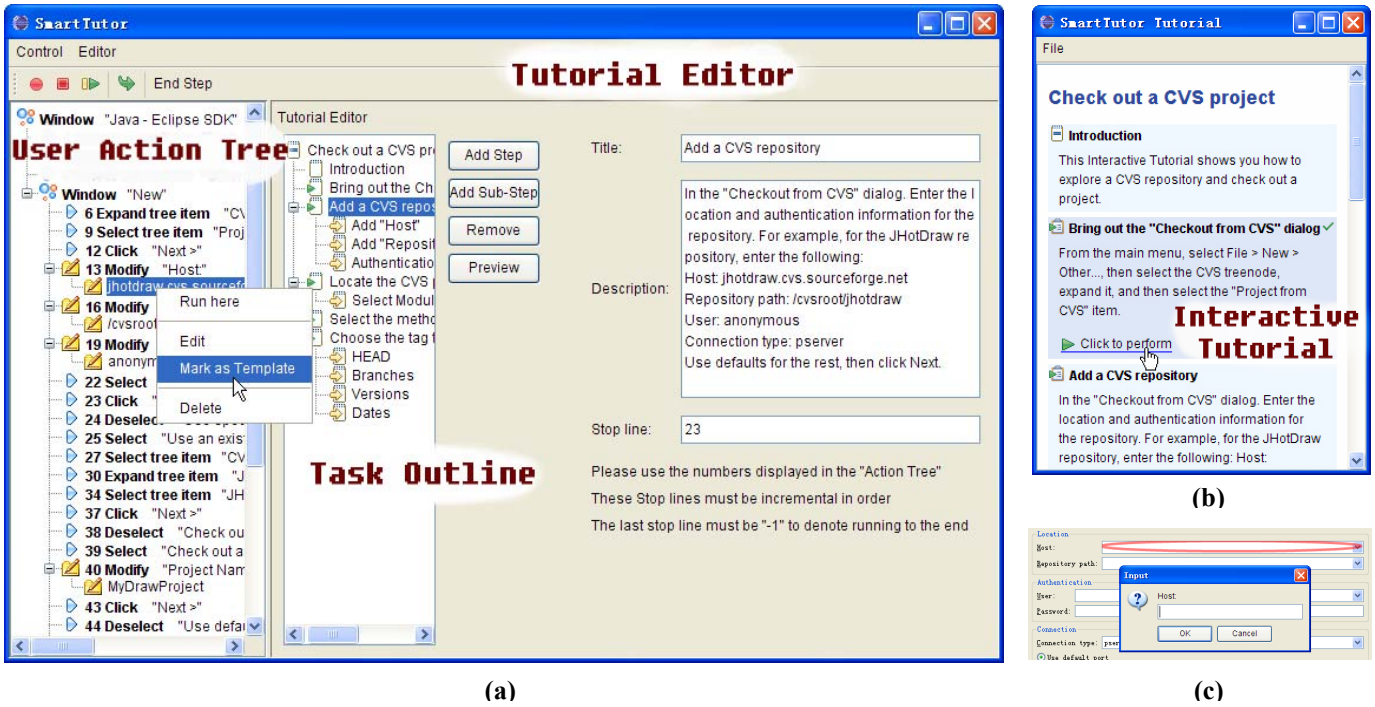

Figure 1. The SmartTutor System. (a) Tutorial Editor: editing the actions of checking out a CVS project to be an interactive tutorial. (b) Tutorial Viewer: presenting the newly created tutorial for the CVS task. (c) A dialog of SmartTutor for inputting a new parameter to replace the old one. The original input-UI is highlighted by drawing a red circle over it.

By using SmartTutor shown in Figure 1 (a), authors will have little programming work to do. To create a CVS tutorial, an author 1) first turns on the action-recording function of SmartTutor before performing the task of checking out a CVS project manually in Eclipse. After finishing the task and turning off recording, all the performed actions will be displayed in the *User Action Tree* of SmartTutor. 2) Then by using the "Mark as Template" function, he can convert parameter-input actions (e.g., "inputting CVS host") into template variables for replacing with new parameters during replay. 3) The author continues to use the *Interactive Tutorial Editor* of SmartTutor to edit these recorded user actions. He adds steps/sub-steps to outline the CVS checking-out task by grouping these actions. (A step/sub-step can contain one or more user actions.) Then writes a title and a description for each step, and then goes on to assign the "stop line" of each step to denote where to stop when replaying these step-actions as an interactive tutorial. (The "stop line" numbers must be sourced from the sequence numbers displayed in front of each action in the *User Action Tree*.) 4) Finally, the author can click the "Preview" button to check the newly created interactive tutorial as shown in Figure 1 (b). After that, an interactive tutorial looking like cheat sheet will be generated.

When using this tutorial, a user can collapse the steps to view the task outline and expand them to get detailed instructions for each step. He can select the "Click to perform" hyperlinks presented in some steps (sub-steps) to forward the mentoring process by driving SmartTutor to replay the recorded actions. When replaying a parameter-input action that has already been converted into a template variable, SmartTutor will bring up a dialog to receive a new input parameter. Meanwhile, it will highlight the original input-UI in Eclipse by drawing a red circle over it as shown in Figure 1 (c). In this way, SmartTutor can replace these parameters dynamically and make the user focus on these steps in the task-learning process. In addition, if all the actions in a step have been replayed, a checkmark icon will appear left to this step. That can help the user avoid missing steps during learning.

Creating these IDE-based interactive tutorials requires little programming effort. Additionally, maintaining and updating them are relatively easy, which can further benefit the author.

3. An Editable Replay Approach

We discuss our editable replay approach in this section. We first present how SmartTutor records and replays user actions; then describe how to edit these actions and convert them into interactive tutorials.

3.1. Recording and Replaying User Actions

IDE-based tutorial learning requires a learner to perform operations on various UI objects of the IDE. If we can record and replay all user actions of performing a task, we open up the possibility of demonstrating them as a tutorial.

3.1.1. Recording User Actions. Most IDEs are Graphical User Interface (GUI) applications that use events to interact with users. We can record actions by catching events. However, the information in events is relatively low-level. For instance, when clicking a button that locates in the center of the screen, we can get the following event: "type = mouse click, position = (512, 384)". Such information is not easy to edit and cannot help to locate the button if layout is changed. To address these problems, we extract relatively high-level information from events to represent user actions. SmartTutor records four parts of high-level information as a user action:

1. UI object features. For instance, the features of a button object include class type, caption, size, etc.

2. UI object context. Given a UI object, SmartTutor will record the name and the class type of its parent UI.
3. User operation. SmartTutor categorizes the events, which are triggered by mouse and keyboard strokes, into relatively high-level operations such as "*Select*", "*Click*", and "*Modify*". In this way, it can reduce the amount of information to be recorded. Additionally, these operations have more semantics than events (e.g., "*KeyDown*" and "*KeyUp*"). Therefore, they are more comprehensible and easier to edit.
4. Time. It includes the time to complete a user action and the interval between two successive actions.

All the four parts are indispensable. The features and the context can be used to locate the corresponding UI objects when replaying a task; the user operation can be used to drive the located UI object to work; the time can be used to adjust the replaying speed.

SmartTutor collects the low-level events by listener [10] mechanism. It registers listeners to the "Display" class that is designated for event dispatching and forwarding in Eclipse. It then analyzes the received events to extract the four parts of information by reflection [10].

3.1.2. Replaying User Actions. In order to replay the recorded actions, SmartTutor first transforms them back to events, then dispatches these events to their source UI objects. These objects will then perform the corresponding operations. This process is repeated until all the actions have been handled in sequence.

As there are no Universal Unique IDs (UUID) can be used to tag and help find the source UI object of an event, SmartTutor uses the recorded features and context information to locate the source UIs. For instance, in a context containing only button A ("Next") and button B ("Back"), to find the button with "Next" as its caption, SmartTutor can quickly locate button A by comparing the caption feature. After that, SmartTutor will create a new event with all its properties set, and then dispatch it. This way, the user actions are replayed automatically.

3.2. Editing Actions and Creating Tutorials

User actions of a task should be edited to 1) represent a tutorial for the task; 2) and make this tutorial interactive. SmartTutor supplies a function to enable users to replace the recorded input parameters of a task dynamically in the learning process; and supplies an editor to create tutorials for the task.

3.2.1. The "Mark as Template" Function. To be a good interactive tutorial, it should allow users to change some input parameters in the task-learning process for better understanding. For instance, tutorials for the CVS checking-out task should suggest and allow users to try other host locations other than "sample.cvs.host". Additionally, to guarantee the consistency of dynamic substitution, all the information related to the old parameter should also be changed.

SmartTutor supplies the "Mark as Template" function to do such work: 1) It replaces the recorded input parameters of "*Modify*" actions with placeholders that take the form of "%{VARn}%" (n≥0). In the tutorial demonstration process, a user can input new parameters to replace these placeholders as shown in Figure 1 (c). 2) All the related actions of the old "*Modify*" action are also changed. SmartTutor uses an algorithm to find these related actions. The following describes this *correlation* algorithm.

Algorithm Input: two actions A_i ($1 \le i \le n$) and A_j ($i \le j \le n$) in an action sequence: $\{A_1, A_2, A_3 \ldots A_n\}$. A_i is a "*Modify*" action for inputting a parameter.

Algorithm Output: return A_j for later examination if it is correlated with A_i; else return null.

Algorithm:

1. Assign a correlation threshold t ($t > 0$) to A_i. Its value is based on A_i's recorded features and context.
2. Let $\alpha + \beta + \lambda = 1$. They are all regulators with values in (0, 1).
3. The correlation between A_i and its following action A_j is determined by the three parts below:
   3.1. Sequence correlation: s. s will get bigger as long as A_j gets closer to A_i in this action sequence.
   $$s = \frac{(n-j+1)}{(n-i+1)} \times t \times \alpha$$
   3.2. Context correlation: c. If A_j and A_i have the same context, then $c = t \times \beta$; else $c = t \times \beta / 2$.
   3.3. Operation-Context correlation: o. If the operation of A_j is appeared in the context of A_i, then $o = t \times \lambda$; else $o = 0$.
4. If $s + c + o > t / 2$, then A_j is thought to be correlated with A_i. Return A_j for later examination.

3.2.2. Creating and Presenting Tutorials. SmartTutor supplies an editor shown in Figure 1 (a) to create and present these recorded user actions as interactive tutorials.

By using the "Add Step/Sub-step" function, an author can group the actions of performing a task into steps to form the task-outline of a tutorial. Then he can add a title and a description to each step to act as instructions. Additionally, the author should assign an integer to the "stop line" of a step, if this step is created to automatically perform some operations such as bringing up a wizard dialog, inputting some parameters, and navigating through some views. The assigned "stop line" must be sourced from the sequence numbers displayed in front of each action in the *User Action Tree*, so that the actions between two successive stop lines can be replayed when this tutorial is demonstrated for mentoring. Finally, the author can preview this newly created tutorial by clicking the "Preview" button.

These tutorials, with the outward appearance looks like Eclipse's cheat sheets, can be displayed in a separate window for users as shown in Figure 1 (b). SmartTutor stores them in XML-format files, so they are relatively easy to distribute, edit, and update.

4. Evaluation and Discussion

We conducted an initial evaluation of SmartTutor as a substitute for traditional tutorial authoring. The evaluation had two basic goals:

- Determining how well the author operates with SmartTutor compared to Eclipse's cheat-sheet editor.
- Determining whether the user would like to use tutorials created by SmartTutor.

We asked two researchers in our group to help evaluate SmartTutor. Both had experience using the Eclipse IDE for code development for four years. One is familiar with Eclipse's cheat sheets and the other is not. We asked the former to create a cheat sheet that can bring up the CVS wizard of Eclipse and highlight the "CVS host" textfield, and can help users to navigate through

the CVS wizard pages. We then asked the latter to use SmartTutor for creating a tutorial to do the same thing. The result is shown in Table 1.

Table 1 Time spent on creating the CVS tutorials

Time (minute)	Programming	Recording actions	Creating outline and Writing descriptions	Validating tutorial	Total
Cheat Sheet	42	0	18	5	65
SmartTutor	0	6	21	7	34

The workload of creating outlines, writing descriptions, and validating tutorials is almost the same no matter using the cheat-sheet editor or SmartTutor. Therefore, when creating a similar interactive tutorial, the time difference between these two tools is determined by the programming time required by cheat sheet and the recording time required by SmartTutor.

We can see from Table 1 that the person using SmartTutor spent less time finishing this work. He remarked that SmartTutor was easy to use and good at tutorial authoring. We then sent the two newly created tutorials to twelve novice programmers who had never used the CVS functions of Eclipse before and were anxious to learn. After using each tutorial, they said that both were helpful and there was almost no difference between the two tutorials, except one said that SmartTutor could not redo the have-done steps in the mentoring process. That is because the recorded user actions are in sequence, so they should also be represented in sequence when replaying. However, we plan to address this deficiency in future SmartTutor by storing some in-process states of a given task and restoring them when having to do a certain step again.

We then went on to test the storage efficiency of SmartTutor. As our tutorial contains only action-related information and textual instructional information (titles and descriptions), this efficiency value can be calculated by computing the size ratio of the two and is the smaller the better. We created 50 interactive tutorials such as building a Swing/SWT project, creating an Eclipse plug-in, and using the JUnit testing framework. The sizes of these tutorials ranged from 12.1KB to 128.4KB. On average, there were 89 actions for each tutorial and the size of 100 actions was 14.6KB. The storage-efficiency values ranged from 18.4% to 58.2%. This suggested that our interactive tutorials were effectively stored.

5. Related Work

There are several work using editable record-replay approach to help programmers. We focus our comparison to these related efforts.

One of them is SCARPE [3]. It captures the runtime interactions between an application and its subsystem. It can replay them to help users to generate test cases, perform offline analysis, etc. SCARPE focuses on capturing program executions, while SmartTutor focuses on recording UI interactions. These two approaches are complementary and can be combined in GUI applications.

The Docwizards system [7] is similar to SmartTutor that uses record and replay approach to generate tutorials. However, Docwizards needs to retain a "world model" of all the UI objects in the runtime, whereas SmartTutor records only the information related to each user action. Thus, the latter's runtime cost is relatively lower. Additionally, the documents of Docwizards contain no task-outlines and few step instructions, so users may have difficult to understand a task when presented with only the recorded action sequence.

Mismar [8] is a toolset in Eclipse, which is used for creating interactive documentation in the form of guides. It can record relevant implementation examples and contribute to the documentation when a programmer using such a guide. Mismar focuses on software artifacts and their relationships, while SmartTutor focuses on task process.

Other tool like JTutor [9] is designed to create and replay code-based tutorials in Eclipse. SmartTutor can also teach novice programmers how to use this IDE.

Safer and Murphy [4] created a focused learning environment for Eclipse. It integrates Mylar [5] and cheat sheets for giving users step-by-step instructions and presenting just the information related to each task-step. It needs to record and analyze the user actions of performing a task to determine the related information. SmartTutor focuses more on tutorial authoring and presentation.

6. Conclusion and Future Work

In this paper, we have proposed an editable replay approach to help authors create IDE-based interactive tutorials with little programming effort. We have also presented SmartTutor, an implementation of our approach that can record and replay user actions and convert them into tutorials after editing.

Future work on SmartTutor includes improving it and extending it to be available in more IDEs. We also plan to use it for collaborative learning: programmers in different workplaces can easily create interactive tutorials and use them to help each other.

Acknowledgements

This work is supported by the National Basic Research Program of China (973) under Grant No. 2009CB320703；the High-Tech Research and Development Program of China under Grant No. 2007AA010301, the Science Fund for Creative Research Groups of China under Grant No. 60821003.

References

[1] http://www.ibm.com/developerworks/library/os-ecl-cheatsheets/

[2] Zhang, et al. Editable Replay of IDE-based Repetitive Tasks. In *Proc. of COMPSAC*, pages 473-480, 2008.

[3] A. Orso and B. Kennedy. Selective Capture and Replay of Program Executions. *ACM SIGSOFT Software Engineering Notes.* Session: *Workshop on Dynamic Analysis*, pages 1-7, 2005.

[4] I. Safer, G. Murphy, J. Waterhouse, J. Li. A focused learning environment for Eclipse. In *Proc. of OOPSLA workshop on eclipse technology eXchange*, pages 75-79, 2006

[5] M. Kersten and G. Murphy. Using Task Context to Improve Programmer Productivity. In *Proc. of FSE*. Session: *Empirical methods and program understanding*, pages 1-11, 2006.

[6] B. Dagenais and H. Ossher. Guidance through Active Concerns. In *Proc. of OOPSLA workshop on eclipse technology eXchange*, pages 60-64, 2006

[7] Berman, et al. DocWizards: A System for Authoring Follow-me Documentation Wizards. In *Proc. of UIST*, pages 191-200, 2005.

[8] B. Dagenais and H. Ossher. Mismar: a new approach to developer documentation. In *Proc. of ICSE*, pages 47-48, 2007

[9] Kojouharov, et al. JTutor: an Eclipse plug-in suite for creation and replay of code-based tutorials. In *Proc. of OOPSLA workshop on eclipse technology eXchange*, pages 27-31, 2004

[10] http://java.sun.com/docs/books/tutorial/