Towards Automated Synthesis of Executable Eclipse Tutorials

Nuyun ZHANG, Gang HUANG*, Ying ZHANG, Ning JIANG, Hong MEI
Key Laboratory of High Confidence Software Technologies (Ministry of Education), School of Electronics Engineering and Computer Science, Peking University, Beijing, 100871, China
*corresponding to huanggang@sei.pku.edu.cn

Abstract

Eclipse tutorials guide users step by step to perform programming tasks. However, as current executable tutorials can only guide users through one way, users’ specific requirements cannot be satisfied. In this paper, we propose an automated approach to synthesizing different tutorials to generate a tutorial that covers many ways. We use record and replay techniques to generate the original tutorials and synthesize them by mining experts’ comments and actions. First, we find the same sub-tasks of different tutorials according to their titles and synthesize the actions of a sub-task by building a hidden Markov model. Then with the synthesized tutorial, steps of a task are recommended gradually during the programming process and executed automatically by replaying some actions. In addition to presenting this approach, we finally present a tool to implement the approach and evaluate its feasibility by an experiment.

Keywords: Eclipse, tutorial, user action.

1. Introduction

Eclipse based Java programming is complex. As a result there are many tutorials to help users. A tutorial is a step by step guide to perform a task [3]. It is usually written in natural language for users easy to learn. However, users may find it inconvenient to use the text tutorials. Therefore, there are some executable tutorials that are able to automatically execute some steps of a task, such as Cheat Sheet [12], JTutor[4] and SmartTutor[1]. A Cheat Sheet is a text based tutorial with some buttons added in the text. The button can trigger the execution of some work described in the text. JTutor and SmartTutor create a tutorial by recording and replaying experts’ programming actions, with critical information highlighted and commented. Since executable Eclipse tutorials are vivid and efficient, they become popular recently.

A tutorial always has a focused mentoring-goal. People can be taught about how to achieve the goal in various ways. However, due to the reason that executable tutorials are difficult to create, an author often present only one way of achieving that goal in the corresponding tutorial. This practice may leave the novices totally unknown to the other processing ways, other than the given and fixed way in the tutorial. The other ways may be more suitable for his/her programming context. We argue that let novices know the various ways/or the best-suitable way of archiving the goal is important. Therefore, we propose an executable tutorial with more than one way of performing a task. However, knowing the requirements of different ways is challenging. Fortunately, there are many experts who perform a task in different ways in programming practice. They make different tutorials for a task. We can summarize them to create the tutorial.

In this paper, we propose an approach to automatically synthesizing executable tutorials. Usually, each task has some sub-tasks. The ways of performing a task are different from the ways of performing the sub-tasks. First, we find out the same sub-tasks of each tutorial by matching key word of their titles. Then, we compare the performing ways of the same sub-tasks and try to find a complete set of performing ways. We do this by building a math model that takes each step sequence as input and a complete step sequence as output. We build the model by an algorithm proposed by Jonathan [15] to discover process from event data.

After observing programmers’ programming processes, we find out that programmers’ specific requirements are exposed gradually along with the development of the programming process. Our tutorial recommends the steps with the development of the programming process. We implement our approach as an extension of our previous work SmartTutor. It works as follows: at the beginning of a task, the system shows users the sub-tasks of performing the task which is an overview, and with the development of task performing, the system recommends the detailed steps on users’ demand. The input of the system is a user’s finished
actions and her/his requirements about the next step. The output is a set of advice on the next steps for users to choose along with the actions that compose the step.

2. An illustrative example

To illustrate our approach, we take the “Create a Hello World application” cheat sheet of Eclipse 3.4 in Figure 1 as an example. In this cheat sheet, there are 5 steps (sub-tasks): 1) “open the Java perspective”, 2) “create a Java project”, 3) “create your hello world class”, 4) “add a print statement”, 5) “run your Java application”. Bob, a novice, opens the cheat sheet to create an application with existing source. He cannot achieve his goal with this only cheat sheet in Eclipse since step 2 does not contain how to include existing sources.

Our tutorial can help him in the following ways.

First, a tutorial is generated as follows. In original tutorial generation process, we ask experts to perform the task and comment on their actions. The comments include three parts as shown in Figure 2(a). The first one is “Stop line” that divides actions into steps (sub-tasks). The second one is “Description”, which explains each step. The third one is “Title” to entitle each step. The “preview” button is for generating the cheat-sheet-like tutorial. The generated cheat-sheet-like tutorial is shown in Figure 2(b). The architecture of steps (sub-tasks) can be viewed in tutorial editor in Figure 2(a). The detailed actions are shown in action viewer.

Second, Bob’s task can be guided as follows.

a) Bob selects the tutorial “Create a Hello World application”. The tool shows him the tutorial in Figure 2(b).

b) He clicks to begin and the tool opens eclipse “New Project Wizard” in the first step as shown in Figure 2(c). This current step is high-lighted in tutorial editor and step viewer in Figure 2(a). He switches to our tutorial shown in Figure 2(b) to see the instruction to fill in the project name. He fills in a project name in a pop-up window controlled by our tool as shown in Figure 2(d).

c) He clicks “finish” to notify our tool that he has finished the name. Then it pops out a menu for him to select the next step, as shown in Figure 2(e). There are 6 options: “Create project form existing source”, “Use a project specific JRE”, “Add Library”, “Create a new class”, “Go to the default next step”, and “I will do it myself”. Bob chooses “Create project from existing source” as he wishes and clicks “finish”. The tool automatically clicks “Create project from existing source” item in “New Project Wizard” window, and clicks “Browse” button to wait for him to choose source files from the file system, as shown in Figure 2(f).

The corresponding step is always high-lighted in tutorial editor and step viewer in Figure 2(a).

d) He clicks “finish”. The tool pops out a window to recommend the next steps as shown Figure 2(g). Bob chooses “Add Library” this time. The tool clicks “Next” button of “New Project Wizard” window to go to “Java setting” page and clicks “Add Library” button of “Libraries” tab automatically, only letting him to add his favorite class files.

e) Finally, Bob creates his application within Eclipse. It takes him less than 20 minutes.

3. Approach overview

The whole approach works as shown in Figure 3. We capture experts’ actions for a task on Eclipse and their comments which describe the meaning of the actions. After a period of time, we can get many different action sequences from different experts for a same task. We mine the sequences to generate a more complete tutorial that covers many users’ requirements of a task. The generation of such tutorial includes not only steps but also descriptions of each step. Then, we can guide a user through the task. By monitoring user actions, the system recommends the subsequent steps with the development of programming process. According to the user’s choice, the system automates the steps by replaying the actions.

The work is based on our previous work SmartTutor, which generates tutorials by recording and replaying an expert’s actions with comments. The record and replay console is called SmartReplayer. By our approach, SmartTutor adds tutorial synthesis and action recommendation function to it.
Figure 2. An illustrate example of creating a hello world Java application
4. Action capture and replay

First, we describe SmartTutor and action capture/replay as the foundation of our approach. A task is executed as a user performs a sequence of actions on Eclipse. A user action is an operation a user performs via input devices such as the mouse and keyboard to operate Eclipse. We capture the low level events of Eclipse GUI system by adding a listener at the base class of all GUI objects, and turn the events into high level actions.

If an action will be used for replaying, three kinds of information should be recorded: action type, action target and action content. Action type means the operation type of using the input devices such as right click, left click, double click, text input, etc. Action target is the GUI object that the action performs on, such as a node in a project tree and a button in a 'New project' window, etc. Action content is useful when the target of an action is of some kinds. For example, to a check box, the contents are 'selected or not', and to a text box, the contents are the texts in it.

When replaying, we first transform a user action back to event, then dispatch the event to Eclipse which will perform the corresponding operation after receiving an event. The implementation details can be found in work [2] which includes some algorithms for UI matching and action editing.

5. Tutorial generation

5.1 Tutorial definition

First, we give a definition of tutorial. By observing the existing tutorials, we define a tutorial containing sequential steps as a line, as shown in Figure 5. A node represents a step (i.e. operations to finish a sub-task) with its description and a directed edge represents a transition between two steps with possible users’ input about the parameters. This kind of tutorial has only one possible next step. Since it can be generated by direct capturing and replaying, we call it original tutorial (OT) in this paper.

When multiple next steps are available, it is the exact tutorial that can adapt to more applications. Since this kind of tutorial can be generated by synergizing OTs, it is called synergy tutorial. We call synergy tutorials as tutorial for short later in this paper. We define a tutorial as a directed graph $G(V, E)$ as shown in Figure 6. Each node $V_i$ represents a step of the task. Each edge $E_{ij}$ between two nodes $V_i$ and $V_j$ represents the transition between the two steps. A user makes his/her decision by his/her input $A_i$.

The execution process of a tutorial is as follows. (1) At each step, there are one or more next steps. The transition ways between two steps can be multiple either. The tool recommends the possible next steps to the user according to his/her current step and the past actions. (2) On receiving some user input, the tool automatically executes and moves to some next step. It recommends and waits again.
5.2 What is tutorial synthesizing

The tutorial generation problem is to mine many OTs of a same task to get the common steps to generate the graph that covers all the OTs. It can be described in Figure 7. We call it sequence alignment to find out the common steps. For example, in Figure 7 the first few actions in Alice’s sequence are: Click ‘File’ Menu → Click ‘New’ submenu → Click ‘Java project’ menu item, while in Bob’s are: Right click on blank package explore → Click ‘New’ menu item → Focus on project → Click Project → Click Java project. Though these action sequences are different, both of them are for opening a “New Java Project Wizard” and can be aligned into one step. We assume it as the first step in Figure 7. For the third step, Alice performs some actions to add source for the project while Bob configures the library. Since the works they do are different, their actions should be aligned onto different nodes. For the last step, they perform a same set of actions and their actions can be aligned onto the same last node.

Figure 7. Aligning different sequences onto a graph

5.3. Alignment approach

We have surveyed all the tutorials in Eclipse 3.3.1 Help system. The 76 tutorials have 276 sub-tasks in total. Their titles are almost (270 of 276) composed of a verb phrase and a noun phrase. Only a few (6) titles have only a noun phrase. The most used 18 verbs and their frequencies are shown in Table 1.

Table 1. The most used verbs of sub-task titles of Eclipse tutorials

<table>
<thead>
<tr>
<th>Verb</th>
<th>Create</th>
<th>Open</th>
<th>Viewing</th>
<th>Edit</th>
<th>Define</th>
<th>Run</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>41</td>
<td>20</td>
<td>20</td>
<td>16</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>Verb</td>
<td>Add</td>
<td>Use</td>
<td>Configure</td>
<td>Set</td>
<td>Test</td>
<td>Layout</td>
</tr>
<tr>
<td>Frequency</td>
<td>15</td>
<td>13</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>Verb</td>
<td>Search</td>
<td>Stop</td>
<td>Import</td>
<td>Debug</td>
<td>Evaluate</td>
<td>More</td>
</tr>
<tr>
<td>Frequency</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

According to the above analysis, we give a word table of verbs from the survey result that has 74 words, a table of noun that has 698 words. The title should contain the two parts and the verb part is allowed to be neglected.

The sub-tasks may have dependency on each other. Their order is usually fixed. As a result, we can align the sub-tasks simply by title and do not have to reorganize their sequence. By observing the 76 tutorials, we find that all experts perform the sub-tasks of a same task in the same order, which proves the reasonability of our alignment approach.

After the alignment of sub-tasks, we compose them together on to one map. We adopt the composition algorithm proposed by Jonathan [15]. The Jonathan algorithm discovers process from event data. It takes the event data as input and output a graph as the process containing all the data. Since combining the event into a graph may produce more paths than actual process, the algorithm use graph transition to avoid redundant paths. It splits a step node into two or more nodes. Then it converts the graph into its dual graph where two or more nodes of a step are integrated into a single node again. The sub-tasks in our approach are considered as the events in Jonathan algorithm. Then we combine them to generate a graph of all the sub-tasks with the Jonathan algorithm.

5.4 Description generation

For a sub-step, there may be several titles and descriptions. We choose the title that is used most as the title for the sub-step and the longest corresponding description as the description for the sub-step.
5.5 Path validation and action recommending

Since the synergy algorithm may bring some new execution path of a task, its validity should be checked. We do this by replaying them all. If the system throws exceptions, the current execution path is wrong and we eliminate it from the tutorial.

At the end of each step, if there is more than one next step available, the system pops out a window and recommends the steps by listing their titles. Some user may not know what to do at a branch point or have preferences in task performing paths. We offer an automatic approach to selecting path. For users who do not know what to do next, we recommend the default parameters and path which is the shortest path for finishing a task. For users who have special requirements, we can mine the path that meets their requirements.

The path selecting depends on a set of user-specific criterions. For example, if users prefer automated generation of the most complete code, productive paths are selected. If they prefer succinct tutorial, the shortest paths in the graph is mined and selected. If their preferences are an integration of several goals, the multi-objective programming is able to solve it.

Consider the example in Figure 9. There are 5 sequences for creating a Java Project. The dotted one has the least steps. The software artifacts they bring are the same. So, the dotted one can be customized as the most succinct tutorial. However, some people prefer the most used one. Then, the fourth (top-down) sequence in Figure 9 will be chosen. The reason is that according to our statistic, 6 people in 11 choose this way to create a project.

6. Case Study

In this section, we show how to create a Java HelloWorld application by our approach. The 11 volunteers as experts are Java programmers who use Eclipse every day in the past two years from computer science department of Peking University. We asked them to build the tutorial by SmartTutor. Surprisingly, none of them made the same tutorial. Parts of the sequences are shown in Table 4. For the convenience of representation, the actions are denoted by numbers. The mapping of number and actual actions can be found in http://code.google.com/p/smarttutor/.

The 11 experts name the sub-tasks all the same according to our word table of the sub-task titles. They perform the sub-tasks in an exactly same sequence that can be simply stringed together. The task is divided into 9 sub-tasks at most: 1) open the “New Java Project Wizard”, 2) create a new project, 3) configure the project, 4) open the “New Package Wizard”, 5) create a new package, 6) open the “New Java Class Wizard”, 7) create a new class, 8) configure the class, 9) write code. The 4th and 5th sub-tasks appear in sequence No.10. The 3rd sub-task appears in sequences No.4, 5, 6, 9, 10, and 11. The rest sub-tasks appear in all sequences.

<table>
<thead>
<tr>
<th>No.</th>
<th>Action Sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 2 3 4 5 -- 6 7 -- 8 1 2 9 -- 10 18 12-22</td>
</tr>
<tr>
<td>2</td>
<td>13-- 6 7-- 14 15 16 17-- 18 19 10 12-22</td>
</tr>
<tr>
<td>3</td>
<td>1 2 23-- 6 7-- 14 15 16 1 2 9-- 10 12-22</td>
</tr>
<tr>
<td>4</td>
<td>1 2 23-- 6 24 7-- 14 15 16 1 2 9-- 18 10 12-22</td>
</tr>
<tr>
<td>5</td>
<td>1 2 23-- 6 24 7-- 14 15 16 1 2 9-- 10 18 12-22</td>
</tr>
<tr>
<td>6</td>
<td>1 2 3 4 5-- 6 24 7-- 14 15 16 1 2 9-- 10 12-22</td>
</tr>
<tr>
<td>7</td>
<td>1 2 23-- 6 3 7-- 14 15 16 1 2 9-- 10 18 12-22</td>
</tr>
<tr>
<td>8</td>
<td>25 26 27-- 6 7-- 14 15 16 1 2 9-- 10 18 12-22</td>
</tr>
<tr>
<td>9</td>
<td>25 26 27-- 6 24 7-- 16 1 2 9-- 12-22</td>
</tr>
<tr>
<td>10</td>
<td>1 2 23-- 6 24 28 29 30 31 32 39 41 33 34 35 37 7-- 14 15 16 1 2 35 36 37-- 14 15 16 38 1 2 9-- 10 18 11 12-22</td>
</tr>
<tr>
<td>11</td>
<td>25 26 27-- 6 24 7-- 14 15 16 1 2 9 10 18 12-22</td>
</tr>
</tbody>
</table>

Each sub-task can be aligned. We take the “3) configure the project” sub-task as an example to illustrate the algorithm. The sub-task is divided into 5 sub-tasks by the 11 experts. The 10th expert gives the most complete partition: 1) configure project name, 2) configure project library, 3) configure output folder, 4) configure project source. Using the Jonathan algorithm, the generated graph of this sub-task is shown in Figure 8. The whole tutorial graph is shown in Figure 9.

![Figure 8. “Configure the project” sub-task generated by our approach](image-url)

We assume that some programmers would like a tutorial with the least actions. Among the paths, we use Dijkstra algorithm to find the shortest path which is numbered as “1” in the graph. It has only 8 actions. For a step following which there will be more than one ways to perform, we generate a selection window as shown in Figure 2(e) in the illustrative example section. The window recommends the next steps. In Figure 9, we can see that following “New Project Wizard” there are 3 steps. As a result, there are some options based on these steps shown in Figure 2(e). The complete generated tutorial is shown in Figure 2 in the illustrative example section.
7. Discussions

There are some conflicts in task automation and programming skill teaching. For full automation, we should offer users as little information as possible. For better teaching, we should offer users as much information as possible. What we do to compromise them is to customize the execution and exhibition of tutorials according to users’ requirement.

First, we discuss the limitations of our approach.

1. It depends on many various OTs made by experts. If there are few OTs, our approach cannot generate a complete tutorial. On the other side, the more programmers’ using our approach to perform a task, the better our tutorial will be.

2. Though we can build user-specific tutorials, too much user participation may cause some other problems, such as privacy, which is an open issue.

3. We ask the experts to use our word table to entitle the sub-tasks. It may not as convenient as the unlimited natural language. What is more, we conclude the word table from the current Eclipse Help system which may not be complete enough. The word matching problem is beyond the scope of this paper.

Second, we discuss the efficiency of our approach. It is mentioned in the Eclipse Cheat sheet developers’ guide [7] that “In a tutorial the goal is to learn how to perform a specific task. Cheat sheets will usually contain up to 10 steps and can be completed in a half an hour or less. For larger tasks consider using a composite cheat sheet.” A sub-task usually has less than ten steps so that our synergy can be efficient enough.

8. Related work

The first area of related works is on software example recommendation systems [11][13][14]. It mines all kinds of software repositories to learn from users’ usage history and give advice to them. The purpose and method of this field is mainly for API learning.

The second area of related work is on action capture and replay. Related works are numerous including SCRAPE [8], Eclipse TPTP [9]. But we replay the actions in a different way, which offers more paths than the traditional replay technique.

The third area of related work is on executable tutorial generation. As mentioned in the introduction section, Cheat Sheet uses hard coding while JTutor and SmartTutor use action record and replay. JTutor is mainly used for demonstration like a video while SmartTutor allows users to modify the recorded steps to adapt to some little change in the programming task. However, these approaches cannot automatically generate tutorials with many paths.

The fourth area of related work is on task automation, including Sheepdog [10]. It extracts a procedure from experts’ action sequences for technical support such as network configuration. The task is highly automated without human invention. Sheepdog adopts the IOHMM algorithm to train the whole process while we adopt the graph transition algorithm to train the steps of a sub-task. We have tried to use HMM training approach. It turns out that the generated models are difficult for human to understand and the results of sub-task alignment are especially terrible. But the Jonathan algorithm performs well.

9. Conclusion

Existing Eclipse tutorials usually provide one way of performing a task that cannot meet users’ specific requirements, mainly because the generation of a synthesized tutorial is very challenging. In this paper, we propose an automated approach for tutorial synthesizing so that Eclipse tutorials can guide users in many ways. We generate the OTs by action capture and replay. We synthesize them by title matching and action alignment. We can mine the many paths to find a path that has some special quality or suits users’ special requirements.

The contributions of our work are mainly two folds. First, we propose a definition for tutorial and divide them into two categories by their ways of performing a task. Second, we propose an approach to automatically synthesizing OTs, which frees human from building a complete tutorial.
10. Acknowledgement

This work is supported by the National Basic Research Program of China (973) under Grant No. 2009CB320703; the National Natural Science Foundation of China under Grant No. 60821003, 60933003; the National S&T Major Project under Grant No. 2009ZX01043-002-002; and the New Century Excellent Talent Program of MOE.

11. References