A Focused Learning Environment for Eclipse

Izzet Safer and Gail C. Murphy
Department of Computer Science
University of British Columbia
201 - 2366 Main Mall
Vancouver, BC, V6T 1Z4, Canada
{isafer, murphy}@cs.ubc.ca

Julie Waterhouse and Jin Li
IBM Toronto Laboratory
8200 Warden Avenue
Markham, ON, L6G 1C7, Canada
{juliew, jinli}@ca.ibm.com

ABSTRACT

The support available in Eclipse to help software developers learn complex APIs and development tools is inadequate; the support is largely passive and the support typically operates alongside the resources associated with normal software development tasks, thereby overloading the environment with additional complexity. In this paper, we describe an approach that enables a dynamic learning process within the context of a developer's Eclipse work environment. Our approach integrates a mechanism for explicit support of a learning process flow (cheat sheets) with a mechanism for explicit representation of different work contexts (Mylar). We have implemented a working prototype of our approach. We found it relatively easy to integrate cheat sheets with Mylar through available extension points. We describe our architecture and report on some limitations and missing features in the existing plug-ins we discovered as we developed our prototype.

Categories and Subject Descriptors

H.5.2 [**Information Systems**]: User Interfaces - Training, help, and documentation—*User-centered design*

General Terms

User Assistance

Keywords

Cheat Sheet, Mylar

1. INTRODUCTION

Increasingly, software development involves writing source code with complex APIs, and using tools to generate and transform source code artifacts. For example, building an application according to a service-oriented architecture (SOA) involves the use of APIs such as Java™Server Faces (JSF) as well as prescribed processes to model, transform, construct, assemble, test, and deploy several types of programming artifacts involved in such an application.

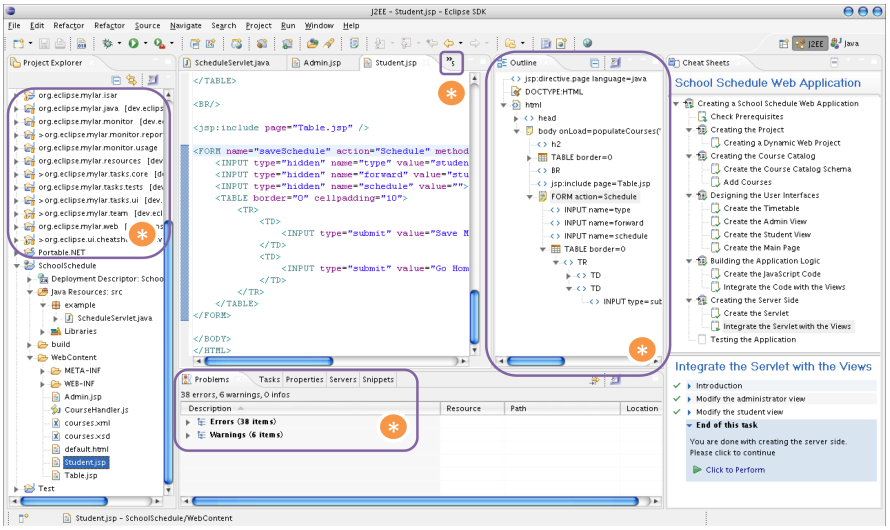
Learning to use these APIs and tools often requires a developer to painstakingly follow steps laid out in documentation. Sometimes, the developer may have computerized support to aid this process, such as wizards or online help. Although such support can make it easier for the software developer to work through the learning process, it is all passive and is inadequate to guide the developer through complex end-to-end scenarios. In some domains, it is possible to introduce specialized environments or tools to aid the learning process. For instance, many systems have been developed to aid students in learning subjects (e.g., [1]). In contrast, in software development, a user needs to learn an API or tool in the context of the same environment in which he or she normally works, as the learning requirements are driven directly from the tasks he or she is performing.

In this paper, we present an approach that supports a dynamic learning process within the context of a developer's work environment. Our focused cheat sheets approach integrates a mechanism for explicit support of a learning process flow with a mechanism for explicit representation of different work contexts. This integration presents the developer with a means of stepping through and revisiting parts of the learning process, and focuses the environment by presenting only the information related to a particular step. Moreover, the integration allows the learning process to appear as if it is occurring in a separate environment since the focusing mechanism hides the full complexity of the normal work environment during the learning activity. We have built a proof-of-concept prototype for this approach within the Eclipse IDE in which we use cheat sheets[1] to support the learning process flow and Mylar[2] to support the explicit representation of work contexts.

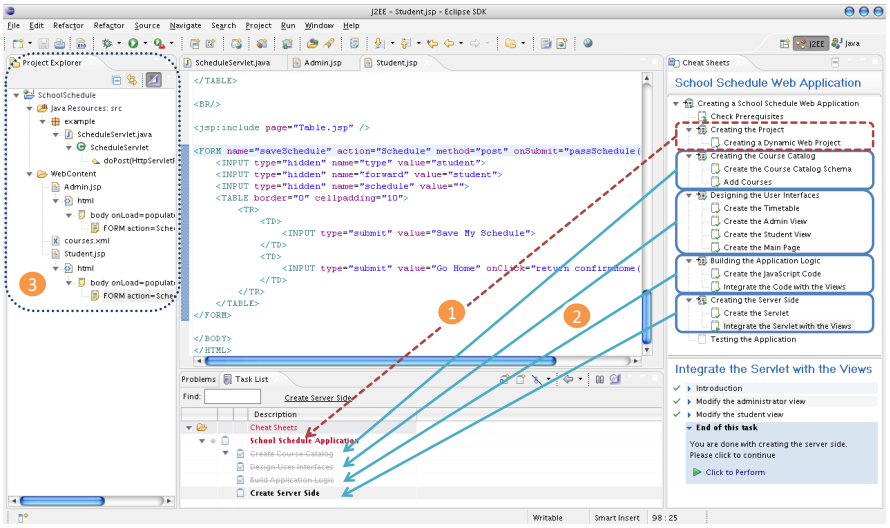
We begin by demonstrating how the learning process can be improved for a developer through the integration of these two technologies (Section 2). We then describe the architecture and mechanics behind the integration (Section 3) before discussing how these technologies both supported and hindered the integration (Section 4). We briefly discuss related efforts (Section 5) before summarizing how this work may be taken forward in the future (Section 6).

[1] `http://help.eclipse.org/help32/index.jsp?topic=/org.eclipse.platform.doc.user/reference/ref-cheatsheets.htm` verified 28/08/06

[2] `http://www.eclipse.org/mylar/` verified 28/08/06



(a) Using cheat sheets in Eclipse 3.2 leads to a cluttered learning environment



(b) Our focused learning environment makes learning tasks explicit and displays only the resources relevant to a learning task

Figure 1: Two views of a developer's working environment: a) without focus and b) with focus

2. EXAMPLE

Consider the case of a software developer who must learn how to combine a particular set of technologies to complete a development task. To help the developer (user) in this process, Eclipse provides support for a largely passive learning approach called cheat sheets. A cheat sheet consists of a number of steps[3]; steps can have sub-steps, enabling a cheat sheet to be structured as a tree. A user follows a cheat sheet by opening a step, which typically displays text about the item and either directs the user to perform commands manually or presents a link that automatically runs commands for the user. Steps can be marked complete, skipped, or redone.

Figure 1-(a) shows a user working through a cheat sheet that presents directions about creating a Web application; the particular Web application used as an example is for school scheduling.[4] The cheat sheet (shown in the upper right) walks the user through the creation of an administrative view to manage a catalog of courses offered, a student view to build a schedule, and a main page with links to the views. Notice the cluttered areas marked with * on the screen in Figure 1-(a); this clutter arises because the user's own projects and views are intermixed with the new project that is the subject of the cheat sheet. This intermixing with existing work and inclusion in the workspace of all cheat sheet steps performed to date makes it difficult for the user

[3]We use the term *step* for a cheat sheet *task* and reserve the term *task* to refer to Mylar tasks.

[4]This cheat sheet is an extension of a WTP Tutorial *Creating a School Schedule Web Application* `http://www.eclipse.org/webtools/community/tutorials/SchoolSchedule/BuildingAScheduleWebApp.html` verified 28/08/06

to see the effects of a particular step and to review a step once it is completed.

Figure 1-(b) shows a user working with our enhanced cheat sheet environment. In this environment, the steps of the cheat sheet have been overlaid with (Mylar) tasks, groups of steps that share a common context of resources and relationships. For this application, we have overlaid five tasks: creating and configuring the Web project, creating the course catalog, designing user interfaces, building application logic, and creating server side logic. When the user starts working with the cheat sheet, the task associated with the first step is created automatically (Figure 1-(b), #1) and the Eclipse user interface is focused on the resources associated with that step. In this case, the `SchoolSchedule` project and `WebContent` folder are displayed in the Project Explorer, instead of other resources created with the project associated with this cheat sheet. Also, the standard J2EE perspective is modified to show only the useful views.

As the user moves through the steps of the cheat sheet, new subtasks are created (Figure 1-(b), #2[5]) and the Eclipse user interface is refocused accordingly (Figure 1-(b), #3[6]). If a user wishes to revisit a step, it is sufficient to select the step and click on *Review*. This action reactivates the corresponding Mylar task, focusing the UI again on the resources relevant to the step. Furthermore, the user can inspect views provided by Mylar to reflect on their learning; for instance, the user can see the time spent on each step.

By integrating cheat sheets and Mylar functionality, we are able to provide a dynamic learning environment that allows users to focus on their learning while completing an interactive tutorial within their existing environment.

3. THE FOCUSED LEARNING ENVIRONMENT

Our learning environment is a plug-in that includes three components (the UI Manager, the Context Augmentor, and the Event Pattern Matcher) and that integrates two existing plug-ins within Eclipse: cheat sheets and Mylar. The UI Manager shows and hides views so that only those views relevant to the current cheat sheet task are visible. The Context Augmentor fills in a Mylar task context with relevant resources. Finally, the Event Pattern Matcher matches sequences of events in the user interaction so that the tool can respond with suggested next steps.

Figure 2 portrays the architecture of the environment, specifying the components, the role each component plays in the environment, and their interactions[7]. Our contribution is presented in solid lines whereas the existing technologies are shown in dashed lines.

3.1 Producing an Interactive Tutorial

Creating a cheat sheet is time intensive for two reasons. First, conceptualizing stories with a clear-cut division of steps requires expertise in user assistance. Second, even though the cheat sheets are simple XML documents, it is a painstaking job to place the content in between the abundant number of XML tags. We leave the resolution of the first issue to cheat sheet designers. However, the second problem can be mitigated through tool support. Eclipse 3.3 is likely to contain cheat sheet authoring tools[8] to create the stories easily.

Designing a focused cheat sheet requires a small amount of extra work on top of creating a cheat sheet. The cheat sheet designers have to make the tutorial communicate with Mylar, UI Manager, Context Augmentor and Event Pattern Matcher in the appropriate cheat sheet steps (Figure 2-A, B).

It is important to distinguish cheat sheet steps and Mylar tasks as two different concepts; one cheat sheet task may or may not map directly to a Mylar task. Currently, there is no mechanism to determine automatically how Mylar tasks should span cheat sheet steps; the designer of the focused cheat sheet must determine this mapping. To complete the focused learning support, the designer may have to prepare a list of views that should be shown or hidden during different tasks, figure out which resources cheat sheets are interested in to augment a task's context, and finally create patterns of events to compare with what the user is performing.

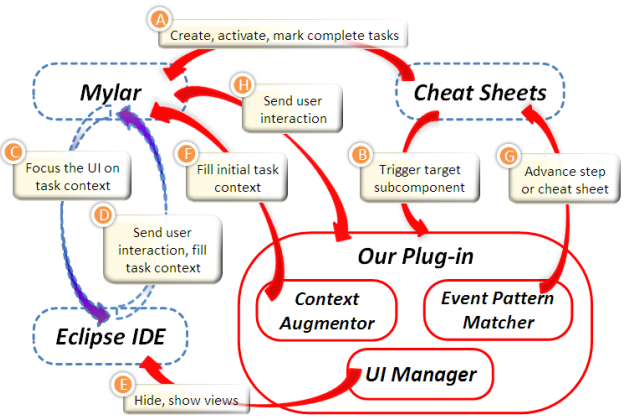


Figure 2: Architecture of the Learning Environment

3.2 Architecture

We have integrated cheat sheets, Mylar, and our plug-in through commands and command handlers. Eclipse commands are function pointers to the handlers, which are the implementation for the commands. The following sections present the components in the architecture of our focused learning environment and how each component can be reached using commands.

3.2.1 Mylar

Mylar is a task-focused UI for Eclipse that reduces the information shown in Eclipse to just the information relevant to the task at hand [4] (Figure 2-C). Mylar determines what is relevant for a task based on the interactions performed by a user as part of the task [5] (Figure 2-D). Mylar supports task switching: when a task is switched, the contents of Eclipse views are updated according to the new task context.

[5]The tasks that are completed have their titles struck out as per standard Mylar functionality.

[6]The children of `Admin.jsp` and `Student.jsp` in the Project Explorer tree are mock-ups as if there existed Mylar JSP structure and UI bridges [5].

[7]Boxes in the figure describe the roles, and the arrows point the direction of the message flow.

[8]Personal communications, Eclipse Bugzilla Bug IDs 146988 and 123921.

Mylar tasks and subtasks can be created, activated, deactivated, marked complete, and marked incomplete through commands we have created. For example, the following command may be placed in a cheat sheet step to create a new Mylar task:

```
<command autorun = "true" serialization = "org.
 eclipse.mylar.tasklist.commands.addNewLocalTask
  (taskName=School Schedule Application,
   taskHandle=sschedapp-1,categoryName=Cheat Sheets,
   resetContext=false,activateTask=true)"/>
```

Similarly, a Mylar subtask can be created by specifying a parent task handle instead of a task category name:

```
<command autorun = "true" serialization = "org.
 eclipse.mylar.tasklist.commands.addNewLocalSubTask
  (taskName=Create Course Catalog,taskHandle=
   sschedcatalog-2,parentHandle=sschedapp-1,
   resetContext=false,activateTask=true)"/>
```

Note that in order to avoid inconsistencies, we actually use placeholders instead of handles and task names.

3.2.2 UI Manager

We took the idea of Mylar one step further by filtering the views themselves. Even though Eclipse perspectives enable a set of views to be easily switched, some of the views within a perspective may still may be unrelated to the current task, thus taking up valuable screen real estate. Our UI Manager enables views within a perspective to be shown or hidden depending upon the context of the work being performed (Figure 2-E). At present, the UI Manager performs this functionality statically, reading a configuration file to determine the appropriate views to show. Future work may involve making this adaptive user interface [2] dynamic, more in the spirit of Mylar.

The UI Manager can be invoked using a command by supplying the path of a view configuration file as the parameter. The reason we are not using the existing `Show View` command registered in Eclipse Workbench is that the complementary command to hide views does not exist. In addition, reading a configuration file makes the design simpler and reusable.

3.2.3 Context Augmentor

Mylar creates a context — a set of resources and relationships relevant to the user's task — based on how the user interacts with the system's resources and with Eclipse. A context for a new Mylar task typically starts empty. To support a learning environment, we needed the ability to seed a task context, adding resources and relationships known to be of interest for the learning step being presented.

Our Context Augmentor plays this role, adding resources specified by the creator of the tutorial to the specified task's context as if the resources had been explicitly selected by the user (Figure 2-F). This imitation of an interaction event increases the degree-of-interest of the resources, making them pass beyond the threshold required for Mylar to display them in the views.

When the Context Augmentor executes, it is provided as parameters a Mylar task handle and a configuration file path. Specifying a task handle in the configuration file allows resources of interest to be inferred, rather than specifying them one by one; the resources that are displayed in the views when that task is active are automatically added to the target task's context. This idea is similar to cloning task contexts, but discards the interaction history of the original tasks.

The Context Augmentor can be applied to a Mylar task at any time. Applying the augment more than once does not reset the existing context, but continues to augment the context with new resources.

3.2.4 Event Pattern Matcher

Pattern matching is used to detect when a user has performed a recognized sequence of steps. When a pattern is detected, the learning environment can react, for example by suggesting that a user advances to the next step or even to the next cheat sheet (Figure 2-G).

We use sequences of interaction events performed by the user (selections, edits, and commands) to check whether a predefined event pattern occurs or not (Figure 2-H). We designed the event pattern configuration file with the same externalization as a task context. As a result, an event pattern can be created easily by performing the desired actions on a temporary Mylar task, and then the required interaction information can be copied to a separate event pattern configuration file.

The Event Pattern Matcher command takes three parameters: the path to the event pattern file, the qualified name of the class that will be invoked via reflection when there is a match, and the handle of Mylar task that must be active during pattern matching. The pattern matcher can also span multiple tasks; if the handle to a Mylar task is omitted as a parameter, the matcher will be valid for any active task.

4. DISCUSSION

The most significant factor in facilitating the integration of the two existing plug-ins was the ability to use modular, model-view-controller style commands contributed through Eclipse's `org.eclipse.ui.commands` extension point. By using commands, we obtained a loose coupling between cheat sheets and Mylar. However, we also encountered a few issues in cheat sheets and the Eclipse platform that required special attention.

4.1 Command support in cheat sheets

By design, cheat sheets can invoke only one command or action per step. However, to support our approach, we needed the ability to execute multiple commands. For example, to support the focusing of the UI on the current task, three commands must execute: one to modify the views, one to create a Mylar task, and one to augment the task's context.

Invoking multiple commands per cheat sheet step revealed a missing feature in the cheat sheet API: the ability to invoke commands automatically. Filling a step with multiple commands is not a solution by itself; the commands should run without requiring a selection by the user. In our current solution, we modified the cheat sheet parser and item structure to allow more than one command per step, and introduced the `autorun` attribute to be able to invoke these commands automatically as soon as the user reaches that step. On the other hand, we preserved the regular structure; there can be only one command assigned to the *Click to Perform* link.

4.2 Consistency

We noticed two features lacking in the wizard structures of the Eclipse platform. First, when wizards are opened using the `org.eclipse.ui.newWizard` command, there is no parameter to set the current project as the container for the newly created file. Instead, users have to manually choose the project each time. Second, new file creation wizards do not permit the name of the file to be assigned to a predefined value, nor return the path to the newly created file. For these reasons, cheat sheet developers may not be sure whether a user entered the right names or not. In the case of a mistake, it is likely for the story to be inconsistent and for the context augmentors and event pattern matchers to fail, causing our focused environment to become inconsistent.

We tried to cover a part of this problem by having consistency in the presentation and usage of resource names. When the cheat sheet creator advises the user to create and fill the contents of a file named `Admin.jsp`, the exact same handle should appear in the story of the cheat sheet, context augmentor and event pattern description files. To achieve this situation, we introduced placeholders and a property file reader. While parsing the cheat sheet, reading the context augmentor and event pattern configuration files, a property file is used and the placeholders are replaced with their actual values. Beyond our approach, we can only advise cheat sheet creators to trust the users until the features lacking in the wizards are implemented.

5. RELATED WORK

A substantial amount of work has been carried out on learning environments in the fields of artificial intelligence and human computer interaction. For instance, COACH [8] is a learning environment for Lisp that watches user actions to create an adaptive user model. Then, the model is analyzed to reason about how to provide personalized comments without interfering with the user's actions.

On the human computer interaction side, Kelleher et al. developed a stencil-based interaction technique on procedural tutorials to draw users' attention to the correct component in the interface, in order to prevent users from interacting with unrelated components [3]. According to their study, users completed the tutorials with fewer errors and 26% faster. In our approach, Mylar serves as a stencil to the UI by focusing on the related parts of the project and of the resources.

There is also a significant amount of related research within the Eclipse community. GILD [9] and Penumbra [7] transform Eclipse into a simple collaborative learning environment for novice Java programmers, by adding their own perspectives. GILD and Penumbra perspectives just show enhanced versions of common views, but none of them focuses the UI adaptively, like Mylar does. Kojouharov et al. created JTutor [6], a set of Eclipse plug-ins to create and replay cheat-sheet-like tutorials. JTutor focuses the UI on the tutorial steps and on the editor containing initial code prepared by a developer. The code is then gradually modified for the users as they advance through the steps manually. By automating the replay process, JTutor aims to represent the relevant and important parts of the code for the users.

6. CONCLUSIONS AND FUTURE WORK

We have developed an approach to link the flow of a complex task, in this case a learning task, with task context information. This approach allows a user to step through an end-to-end scenario and focuses the user's learning environment by presenting just the information relevant to the particular step.

Our future work consists of determining how our approach can be generalized to task context and task flow integration problems encountered in fields other than software development. Moreover, in order to prove the improvement in the learning process, we need to validate our approach with field studies.

7. ACKNOWLEDGMENTS AND TRADEMARKS

This research was funded by an IBM CAS fellowship. The views expressed in this paper are those of the authors and not necessarily of IBM Canada Ltd. or IBM Corporation.

8. REFERENCES

[1] C. Conati and X. Zhao. Building and evaluating an intelligent pedagogical agent to improve the effectiveness of an educational game. In *Proc. of the 9th Int'l Conf. on Intelligent User Interface*, pages 6–13, New York, NY, USA, 2004. ACM Press.

[2] E. A. Edmonds. Adaptive man-computer interfaces. In M. J. Coombs and J. L. Alty, editors, *Computing Skills and the User Interface*, pages 389–426, 1981.

[3] C. Kelleher and R. Pausch. Stencils-based tutorials: design and evaluation. In *Proc. of the SIGCHI Conf. on Human Factors in Computing Systems*, pages 541–550, New York, NY, USA, 2005. ACM Press.

[4] M. Kersten and G. C. Murphy. Mylar: a degree-of-interest model for ides. In *Proc. of the 4th Int'l Conf. on Aspect-Oriented Software Development*, pages 159–168, New York, NY, USA, 2005. ACM Press.

[5] M. Kersten and G. C. Murphy. Using task context to improve programmer productivity. In *FSE '06*, 2006.

[6] C. Kojouharov, A. Solodovnik, and G. Naumovich. Jtutor: an eclipse plug-in suite for creation and replay of code-based tutorials. In *Proc. of the 2004 OOPSLA Workshop on Eclipse Technology eXchange*, pages 27–31, New York, NY, USA, 2004. ACM Press.

[7] F. Mueller and A. L. Hosking. Penumbra: an eclipse plugin for introductory programming. In *Proc. of the 2003 OOPSLA Workshop on Eclipse Tech. eXchange*, pages 65–68, New York, NY, USA, 2003. ACM Press.

[8] T. Selker. Coach: a teaching agent that learns. *Commun. ACM*, 37(7):92–99, 1994.

[9] M.-A. Storey, D. Damian, J. Michaud, D. Myers, M. Mindel, D. German, M. Sanseverino, and E. Hargreaves. Improving the usability of eclipse for novice programmers. In *Proc. of the 2003 OOPSLA Workshop on Eclipse Technology eXchange*, pages 35–39, New York, NY, USA, 2003. ACM Press.