JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2008**

Vol. 7, No. 7, September-October 2008

Adapting the User Interface of Integrated Development Environments (IDEs) for Novice Users

Ying Zou, Michael Lerner, Alex Leung, Scott Morisson, Matt Wringe
Department of Electrical and Computer Engineering, Queen's University
Kingston, Ontario, Canada

Abstract

The usability of a user interface is often neglected in the design and development of software applications. An Integrated Development Environment (IDE) is prone to poor usability problems due to the rich functionality offered through its User Interface (UI). Since an IDE targets a wide range of users (from novice to expert users), the usability requirement for an IDE vary considerably. Novice users, such as first year undergraduate students, often have difficulty in understanding many of the features provided in an IDE and have a hard time locating the appropriate menu elements. We propose an Adaptive User Interface (AUI) architecture which provides a simplified UI for the Eclipse IDE. The AUI assists novice users in using complex IDEs. We develop adaptive algorithms that modify the existing menu system for the Eclipse IDE based on statistical user interaction patterns. Our adaptive algorithms perform a cost-benefit analysis when modifying the menu system. The algorithms determine the optimal changes which reduce the time needed by novice users when searching for menu elements. A prototype AUI is developed as an Eclipse plug-in for novice users of the Eclipse IDE. Through an initial case study, we demonstrate the benefits of our AUI in improving the usability of the Eclipse IDE.

Keywords: Eclipse IDE, Adaptive User Interface, Adaptive Algorithm.

Ying Zou, Michael Lerner, Alex Leung, Scott Morisson, Matt Wringe: "Adapting the User Interface of Integrated Development Environments (IDEs) for Novice Users", n *Journal of Object Technology*, vol. 7, no. 7, September - October, pp. 55-74 http://www.jot.fm/issues/issue_2008_09/article1/

1 INTRODUCTION

In 1981 the Xerox 8010 computer system was introduced. It was the first computer to use windows, icons, menus and pointers (WIMPS) based user interface (UI). Since then the way users interact with software applications has not changed dramatically [1, 2]. To fulfill the growing requirements from the business world, software applications have gradually evolved to provide sophisticated functional features. To improve the usability of software applications, many approaches have been proposed to design and develop adaptive UIs. The proposed approaches react to different situations and requirements by learning the behavioral patterns and styles of individual users [12]. Commercial products, such as the Microsoft Office Suite 2003, provide adaptive UIs which make it easy for users to access the most frequently used menu elements. However with these advancements, the UI of Integrated Development Environments (IDEs) is often neglected; leaving the UI complex and difficult to learn. In particular, IDEs are designed to cater to a wide range of users by offering a variety of functions. Such functionality is made available at all times to all users, increasing UI clutter. IDEs, such as Eclipse (an open source Java development environment) [3], often allow for additional plug-ins to be integrated to increase functionality. The plug-ins have their own UI components (e.g., additional menu bars and icons) which further increase the complexity of the UI. It is challenging for novice users, such as first and second years undergraduate students to understand all the features in the UI for the IDE during their early programming courses. In most cases, students use a limited subset of the features of an IDE. For example, students may only create a project, compile, run, and debug a program. Other features such as refactoring and source control are seldom touched or taught in lower year programming courses. So students can focus more on the programming concepts instead of being bogged down by the rich features of modern IDEs. The GILD project [11] enhances the UI for the Eclipse IDE by presenting a fixed set of functions in the menu. Such functions are sufficient for students to conduct their programming assignments. However, as students become more familiar with the IDE, the UI is not able to automatically adjust in order to fit the growing needs of students. For example, students may want to use a source control system, such as CVS, for their group projects. The UI for the IDE has to be manually modified to add the UI menus for this feature.

In this paper, we propose an Adaptive User Interface (AUI) architecture. The AUI dynamically adapts the UI to the daily routines, such as coding, compiling, and debugging, of individual novice users. Our AUI is developed as a plug-in for Eclipse, and runs in the background to collect statistics on how the user interacts with the menu system. For example, for a user who focuses on the debugging of code, the AUI would hide irrelevant menus such as creating a new project in the “File” menu. In contrast to the commercial products, such as the Microsoft Office Suite 2003, which have similar functionality for hiding infrequently used menu elements, we predict a user’s next

possible selections and develop adaptive algorithms (e.g., the fade and enlighten algorithms) that perform a cost-benefit analysis when making modifications to the menu system. Especially, our adaptive algorithms can detect the usage patterns. When a menu element is a step within a detected usage pattern, the menu element can be displayed and highlighted (even if the element has not been used for a while). Our adaptive algorithms determine the optimal changes to the menu system in order to improve the user's efficiency when searching for the next element to click in a menu. For example, through the use of the fade algorithm, the menu elements that are seldom used by the user are hidden. The hiding of menu elements reduces the size of the menu and gives a user quick access to most frequently used menu elements. To further increase the speed of locating the appropriate menu elements, our plug-in highlights possible menu elements to be used. The highlighted menu elements are produced by the enlighten algorithm based on the user's usage patterns. The decisions for hiding and highlighting menu elements are evaluated by our proposed cost effectiveness formulas.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 presents and analyzes a user model that describes a user's interaction with an IDE menu system. Section 4 discusses the design of our AUI. Section 5 describes our case studies. Section 6 concludes the paper and explores future work.

2 RELATED WORK

Improving the usability of a UI is more of an art than a science due to the involved human factors. Different users will have different usability expectations and usage patterns. However, metrics and certain methodologies can still be used to quantitatively measure, compare, and improve the usability of a UI. In general, usability is a software quality attribute which measures the extent to which an application can be used by users to achieve effectiveness, efficiency and satisfaction in a specified context of use [4]. Furthermore, usability can be decomposed into five attributes: (1) *learnability*, which measures the ease of learning the functionality of an application; (2) *efficiency*, which measures the ease of use and the level of productivity attainable by the user; (3) *memorability*, which measures the ease of remembering the functionality of an application; (4) *low error rate*, which measures how the application supports users in making less errors; and (5) *user's satisfaction*, which measures how the users enjoy using the application. We aim to improve the efficiency of a UI through the use of an AUI.

Earlier work on AUIs [5, 6] focuses on tracking the behavior of users using a Web browser and anticipating items of interest. This prior research analyzes the behavior of users by matching the keywords in Web documents. Microsoft Office Assistant uses the result of the Lumiere Project [8] which employs Bayesian network to predict future events and a goal graph to infer a user's need by considering the background, actions, and queries of a user. To date, AUIs have seldom appeared in industrial products except for the Microsoft Office suite [9] which dynamically modifies the menu elements based on the usage of the UI. When a user first opens a menu, a short list of a menu with less

content appears. If the user hovers over the menu for a few seconds, the full menu with all the content is displayed. If a user selects an element from the full menu, the element selected will appear next time in the short list. However, such AUIs do not predict possible future steps (i.e., menu elements) using a cost-benefit analysis techniques as performed in our research. [7] presents techniques which provide personalized just-in-time assistance to users and which automatically derive a user's usage patterns based on analysis of the mouse and text editing events generated from UIs. The techniques created in [7] are used to support word editing, but are not used to change menu systems.

Findlater and McGrenere [12] conducted controlled experiments to compare the AUIs with the traditional static user interfaces. In their qualitative analysis, the majority of users (55%) prefer to have control over their UIs rather than to use an adaptive AUI (i.e., AUI). However, the same users showed strong support for the AUI. A mixed initiative, where the system and the user both have some control over the UI, may be the best way to satisfy a large range of users. In our research, the users of the AUI are novice users (i.e., students). An AUI provides a limited number of menu elements and allows novice users to focus on their programming assignments without getting frustrated by the complex UI.

3 USER-MENU SYSTEM INTERACTION

To better understand how to increase the usability of the menu system, we establish quantitative measurements, and construct a simplified user model which captures how a user interacts with the menu system. Furthermore, the model tracks the usage patterns of a menu system. Using this information, optimal modifications to the menu system can be made to improve the efficiency. More specifically, we consider that the user interaction with the menu takes place in the following six cases:

1) The user selects the menu that they wish to view using either the mouse or the keyboard. The menu can be either the top level of the workspace menu or a right-click menu that is prompted by clicking on the right button of a mouse.
2) When the menu is shown, the user's focus is located at the top of the menu.
3) Using the mouse if the user is not familiar with the menu system, the user has to move the mouse from the top of the menu system downwards, passing over each menu element in a sequential order until they locate the menu element of interest.
4) When the user knows the position of the menu element of interest in the menu system, they are not likely to search through the menu system in a sequential manner. Most likely, the user moves the cursor directly to the location where they believe the menu element should be located.
5) If the user uses the keyboard to access the menu system, the user has to hit the down arrow key to move down the menu system from the top of the menu until they reach the menu element of interest.
6) When the user knows the shortcuts to access a menu element, the menu system is not used.

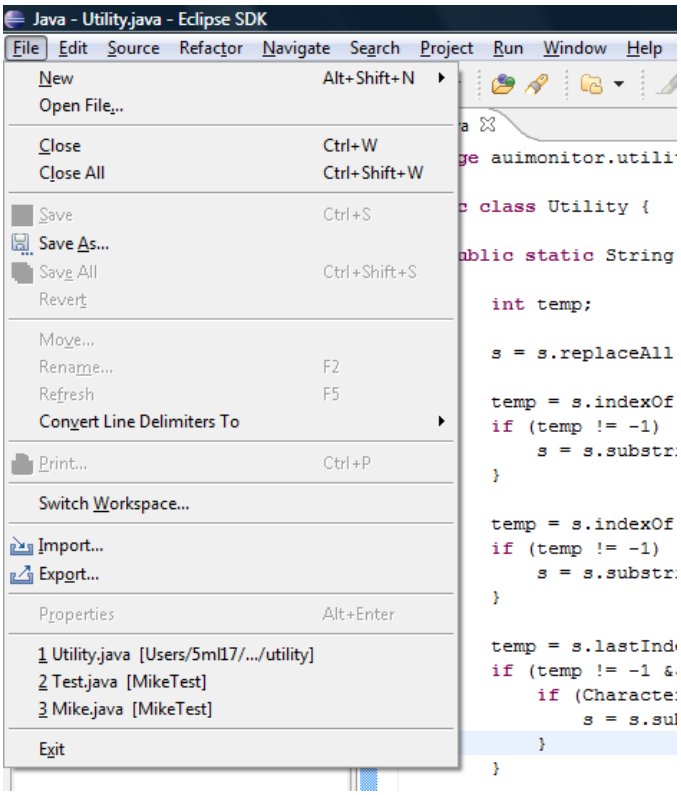


Figure 1: A Screenshot for the Initial View of the Original Menu System in the Eclipse IDE

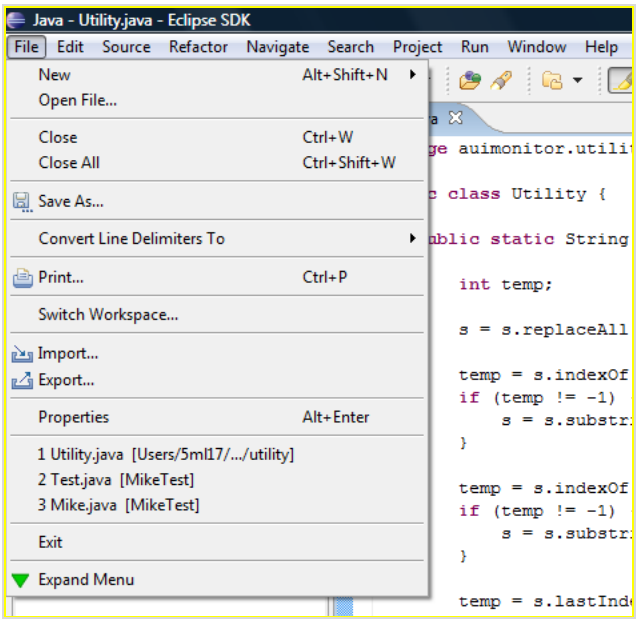


Figure 2: A Screenshot for the Initial View of Our Adapted Menu System in the Eclipse IDE

User Model Analysis

To improve the efficiency of the UI, we shorten the *average time to element* metric which measures the average time that a user takes to find an element within a menu. The *average time to element* is the number of elements that must be considered before the element of interest is selected. For a menu with its menu elements randomly selected, the average number of elements examined before finding the element of interest will equal half the number of elements in the menu (the same as a linear search). By minimizing the *average time to element*, we can maximize the efficiency of a menu system. We can minimize the metric using two ways: *1)* by hiding elements that are unlikely to be selected; and *2)* by highlighting elements with a high probability of being selected. We measure the *average time to element* by tracking the time it takes a user to locate the menu element of interest.

Initially, our AUI (AUI) plug-in does not modify the menu system except by hiding grayed out (disabled) menu elements. For example as shown in Figure 1, some menu elements, such as “Move”, “Rename” and “Refresh” are disabled because the menu system prevents the user from clicking on them. In this case, we hide all disabled menu elements. The hiding of disabled elements shortens most menus and helps a user in quickly accessing the menu elements of interest. The shortened menu is illustrated in Figure 2. The hiding and highlighting of menu elements is sensitive to the current usage context. Moreover, menu changes only begin to occur after sufficient usage data has been collected. To improve efficiency, we highlight the most likely menu element to be clicked. Therefore, the user does not have to search through the entire menu to find the element which they want. As the user continuously interacts with the menu system and more usage data is collected, the accuracy of predicting menu elements improves. When the prediction accuracy is high enough, the user no longer needs to remember where menu elements are positioned in a menu system. Our AUI plugin will predict the most likely menu elements of choice, in turn reducing the user’s need to memorize the functionality of the menu system.

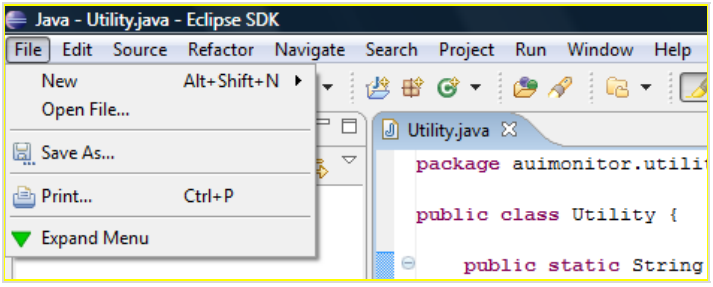


Figure 3: A Screenshot for the Adapted “File” Menu in Eclipse IDE

4 ADAPTIVE USER INTERFACE

To improve the efficiency of the Eclipse IDE for novice users, we design and develop the AUI as a plug-in for the Eclipse IDE. The user interacts with the menu system in the same manner as using the original Eclipse menus system. The menu system is enhanced using the adaptive functionality of the plug-in. While a user is using Eclipse, the plug-in collects usage patterns in the background. The AUI plug-in dynamically modifies the menu system of the Eclipse IDE. For example, in the original menu system, as shown in Figure 1, each top level menu (e.g., File, Edit, and Navigate) has a number of menu elements and sub-menus. After sufficient usage data is collected, the plug-in would hide infrequently used menu elements. For instance, if a user continuously creates new projects by selecting the "New" menu element, the AUI plug-in dynamically exposes the "New" menu element and hides the rest of the menu elements, as illustrated in Figure 3. Moreover, our AUI plug-in adds an "Expand Menu" button at the bottom of each top level menu. By clicking this button, the user can revert back to the original menu where no menu elements are hidden.

The Architecture for Our Adaptive User Interface Plug-in

The overall architecture for the AUI plug-in is depicted in Figure 4. Essentially, the user interacts with three major components: the Adaptive Menu System (AMS), the AUI wizard, and the AUI preferences.

- The AMS is the main form of interaction which the user has with the menu system.
- The AUI wizard provides guidance and notification to the user when the UI is started for the very first time after changes are made to the existing UI.
- The AUI preferences are preferences set by the user about how the AUI should act. These preferences will be accessed in the same fashion as other Eclipse preferences. In our work, we set the AUI preference with respect to the novice users.

When the user interacts with the UI, events are generated for every clicked menu element and short cuts input from keyboard. The events are captured by the event engine. The event engine has several roles. First, if a menu has been opened by the user, the event engine is responsible for notifying the adaptive engine of this event. The adaptive engine determines the changes that should be applied to the menu system. It then calls the AMS handler, which is a part of the event engine, to perform these changes in the UI. The other task of the event engine is to interact with the data storage engine to store statistical data about the events generated by the user for future use by the adaptive algorithms.

The adaptive engine is responsible for the logic of the plug-in. When the event engine captures an event from the UI, it takes all relevant information such as the time and the menu element that was clicked, and passes this information to the adaptive engine for

analysis. Using the AUI preferences, the AUI core determines whether any of the adaptive algorithms should be invoked. The algorithms use the usage pattern data stored by the data storage engine to determine if any changes to the interface are required. Any such changes are sent back to the AUI core, which invokes the utility methods of the AMS handler in order to perform the required tasks. A further description of each component in the architecture from Figure 4 is included in Table 1.

Figure 4: The Architecture for Our AUI Plug-in

Table 1: Description of the Components of the Prototype AUI Plug-in

Component	Description
Adaptive Menu System (AMS)	The AMS refers to the existing UI and the menu modifications made by the AUI plug-in.
AUI Wizard	The AUI wizard informs the user at the start of UI that the AUI plug-in is running and that it will be collecting user data.
AUI Preferences	Using the existing Eclipse preferences system, the AUI preferences components allows the user to set global options for the AUI plug-in. The user may turn the plug-in on or off, as well as set preferences for hiding infrequently used menu items, hiding disabled menu items, and highlighting predicted menu items.
Event Capture	Every time the user interacts with UI using the mouse and keyboard, an event is generated. This component captures these events and passes them to the AUI core for processing. In addition, it invoked methods in the menu tree and sequence handler components, which store event information for future use by the adaptive algorithms as the user's usage patterns.
AMS Handler	This component performs modifications to the UI. For example, AMS handler hides a menu element, highlights a menu element, and adds an "Expand" button at the end of a specified menu.
AUI Core	The AUI core has most of the logic of the plugin. Once triggered by the event capture component, it uses the preferences set by the user to invoke the appropriate adaptive algorithms. The algorithms calculate the required modifications to the menu system, and the AUI core interacts with the AMS handler to perform the required changes.
Enlighten and Fade Algorithms	The fade algorithm calculates the optimal number of infrequently used menu elements to hide. The enlighten algorithms predicts the next menu element to be clicked by the user using previous usage patterns. Both of these algorithms use usage patterns stored by the menu tree and sequence handler component. The algorithms return the results to the AUI core in the form of lists of elements to be highlighted or hidden.
Menu Tree	The menu tree component stores information about the menu system in Eclipse, such as the number of times the UI has been clicked by the user. This is used in the fade algorithm in order to determine the infrequently used menu items to hide. The menu tree uses a number of data structures: Menu Bar, Menu Node, and Menu Element Node.
Sequence Handler	The sequence handler stores information about sequences of element selections by the user. It is called by the event capture component when a menu element is selected. The sequence handler then builds a data tree storing all previous sequences by the user, and the frequency of their occurrence. The sequence handler provides this information to the enlighten algorithm, which then predicts the user's next menu element selection.
Data Handler	At the termination of the UI, all usage pattern data in active memory is stored persistently on disk using the data handler component. The data is formatted in XML, and is parsed back into active memory once the UI starts again.

The components, which directly interact with the UI of the Eclipse and process the events captured from the UI, are platform dependent. However, the AUI plug-in can be generally applied in the applications developed using Eclipse rich client platform for the UIs. The components in the data storage engine subsystem and the adaptive engine subsystem are independent of the platform or the operating system. If the AUI plug-in were to be used for non-Eclipse based application, the data storage engine and adaptive algorithms can be reused without changes.

Menu Element Prediction

To predict which menu element the user is most likely to select next, it is necessary to develop a method that can efficiently track usage patterns in the menu system and which applies the collected data for effectively predicting future events. For example, if the user frequently selects the “Copy” menu element in the “Edit” menu, then clicks the “Paste” menu element in the “Edit” menu. Based on previous usage data, the AUI plug-in can anticipate with high confidence that the user is likely to select the “Paste” menu element in the “Edit” menu after the user selects the “Copy” menu element under the “Edit” menu. To reduce access time and make predicted menu elements more visible, a predicted menu element (i.e., the “Paste” menu element) is highlighted in the menu (i.e., the “Edit” menu). We also capture menu selections using keyboard shortcuts. We treat keyboard shortcuts, as if their corresponding menu item was picked. For example, a “Ctrl+C” shortcut is considered equivalent to a user picking the “Copy” menu element under the “Edit” menu. We capture the various user interactions with the menu system during a period of time, and store these events into a sequence. More specifically, some actions (e.g., “Copy” and “Paste”) tend to be triggered in sequence following a specific ordering. For example, a “Paste” action is followed by a “Copy” action. We identify such usage patterns, and call such usage patterns as an action sequence.

One of the challenges in building an AUI is to transparently determine the boundaries of several action sequences when the user interacts with the UI over a period of time. It is impractical to require the user to manually mark the starting and end point of their usage pattern for two reasons. First, such an approach requires additional effort from the user who must remember the beginning and the end of each action sequence. The additional effort would undoubtedly be cumbersome and would be hard to remember doing. Second, the user might not always be aware of his or her usage patterns and might not recognize sequence of menu selections as a common usage pattern. In our research, we consider that different action sequences are separated by moment of no action. For example, if a user stops interacting with the menu system (or using keyboard shortcuts) for one hour then we consider all previous actions as an action sequence and all future actions as a new action sequence. In addition, when predicting the next element to be selected by the user, the menu element will only be highlighted if the user has already performed that sequence at least twice in the past. This is necessary as a result of the possible inaccuracy when dynamically determining the boundaries of action sequences. Further study on methods for determining when an action sequence begins and ends would be beneficial. It

may be possible to make the time intervals dynamic so the system would dynamically determine the average time between actions, and use this average time to improve the detection of the beginning and end of a sequence of actions.

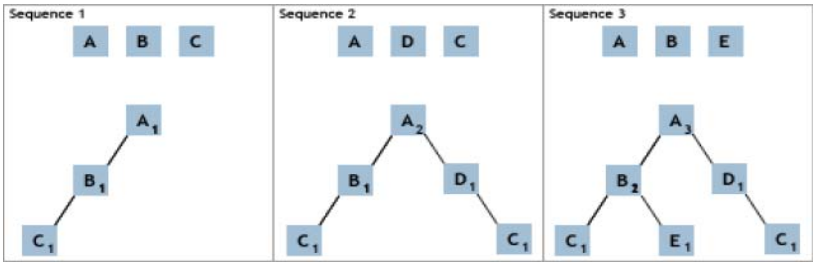


Figure 5: Action Tree Structure Building

To capture the frequency of occurrence of each action sequences, an action tree structure is used. Each node, in the tree, represents an action (e.g., create a new project) triggered by different UI events (e.g., mouse click, and keyboard presses) and each node has a number of children that represent possible actions followed when the action for the current node is triggered. Using this tree structure, each path from the root node to a leaf node represents an action sequence. Moreover, each node contains a counter that represents the frequency of the action being triggered in different action sequences. Figure 5 shows an example tree being generated from the given sequences.

When an action sequence begins, we scan the existing action trees to find if any of the current trees has the initial action as a root node. If such a tree exists, we increase the counter for all the tree nodes (i.e., actions) that appear in the action sequence. Otherwise, a new tree is created with the root node being the first action in the action sequence. In Figure 5, when the first action sequence occurs, no tree with a root node "A" exists. Hence, a new tree is created the action "A" as the root. When the second action sequence occurs in Figure 5, a tree with action "A" as the root already exists so the counter to the root node (i.e., action A) is incremented. One advantage of this structure is that it can give predictions while it is being built. The adaptive algorithms can efficiently check a list of children nodes and their occurrence counts in order to predict the next action.

Adaptive Algorithms

To dynamically personalize the menu system based on usage patterns, we propose two adaptive algorithms: the fade algorithm which temporarily hides menu elements and the enlighten algorithm which highlights menu elements in a menu. Both adaptive algorithms gather statistical data stored about the action sequences, determine which menu elements should be changed, and notify the AUI Core (as shown in Figure 4) to make the appropriate changes to the menu system. No changes should be done to a menu where its elements are uniformly selected as there is an equal probability of any elements being selected. A menu, where a few elements have a high likelihood of being selected, and a very low likelihood for the rest of the elements being selected, is a good candidate that needs to adapt to a user's behavior. The underlying variable for both adaptive algorithms

is the probability of a menu element being selected to trigger an action, and its variance relative to the rest of the menu elements in the same menu. We obtain this probability from the stored action sequences in the tree structure.

To determine if a menu element needs to be changed, we need to analyze the interdependencies among elements in the menu and the cost-benefit on the entire menu systems. We use the user model, as discussed in Section 3, to keep track of the changes made in the user interface as a whole. The goal of the adaptive algorithms is to determine the number of elements to hide or highlight in order to minimize the *average time to element*. By reducing the average time to element, we reduce the searching time for the next menu elements and can improve the efficiency of the UI.

Fade Algorithm

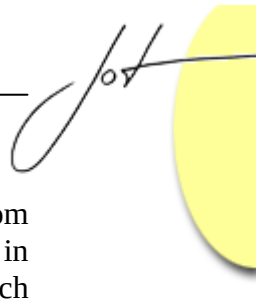
As aforementioned, the fade algorithm determines which elements to hide from the menu. An outline of the functionality of the fade algorithm is listed below:

1) Obtain the stored statistical data from the menu tree structure.
2) Determine the *average time to element* by considering all possible cases that hide different numbers of menu elements
3) Select the optimal number of elements to hide, and
4) Determine if there is a speedup after hiding the menu elements or displaying additional menu elements

To calculate the *average time to element* for a menu element, we propose the fade equation which is defined in Equation 1. The menu elements in the AUI are ordered in the same fashion as the original menu system. We consider that the probability for an element selected in the menu is equal (i.e., 1/N where N is the number of visible elements in the menu). Moreover, we select the elements to be hidden according to the probability of them being selected. Elements with the lowest probability of being selected are hidden.

$$T_{avg}(N) = P_{visible} \cdot T_{visible}(N) + P_{hidden} \cdot (T_{search}(N) + T_{expanded}(M+1)) \quad \text{.....(1)}$$

$T_{avg}(N)$:	The *average time to element* for a menu element that has N visible elements in the menu
N:	The number of visible elements in the menu
M:	The number of the elements in the original menu
$P_{visible}$:	The probability of selecting the element of choice from the visible elements
$T_{visible}$:	The *average time* for selecting an element appearing in the visible elements
P_{hidden}:	The probability of selecting an element that is hidden
$T_{search}(N)$:	The *average time to element* that it takes to search for a hidden element among the visible elements
$T_{expanded}(M+1)$:	The *average time* for selecting an element from the expanded menu



We also consider the case where a user fails to find an element because it is hidden from the menu. In this case, the user has to click the "Expand" menu element as illustrated in Figure 3. Therefore, the time to element includes two cases: 1) it takes the user to search an element among the N visible elements and find that the element of interest is not displayed in the visible elements; 2) the user will click the "Expand" menu to see the full menu, and then search the element of interest in the expanded menu. The expand menu contains M + 1 elements. M is the number of elements in the original menu and the additional one element is the "Expand" menu element. Therefore, the total number of elements in the expanded menus is M+1.

To minimize the *average time to element*, we consider the number of visible elements in the menu in all possible combinations. We calculate the optimal number of elements that need to be hidden. More specifically, we hide the elements with the lowest probability of being selected.

Enlighten Algorithm

The enlighten algorithm determines which menu elements should be highlighted. The enlighten algorithm gathers the stored statistical data in the action tree structure, determines the optimal number of elements to highlight, and determines if there is a speedup to warrant highlighting elements. The enlighten equation, listed in Equation 2, is used to determine the *average time to element* by varying the number of elements being highlighted. The elements under consideration for highlighting are the elements with the highest probability of being selected based on the current action sequence.

$$T_{avg}(N) = \begin{cases} P_{highlighted} \cdot T_{search}(N) + P_{non-highlighted} \cdot (T(N) + T_{search}(M-N)) & N > 0 \\ T_{search}(M) & N = 0 \end{cases} \quad \text{.....(2)}$$

$T_{avg}(N)$: The *average time to element* when N elements are highlighted

N: The number of elements to be highlighted

M: The number of elements in the original menu

$P_{highlight}$: The probability of the selecting an element from the N highlighted elements

$P_{non\text{-}highlighted}$: The probability of not selecting an element from the N highlighted elements

$T_{search}(N)$: The average time for selecting an element from N elements

$T(N)$: The cost for checking all of the N highlighted elements

When N elements are highlighted in the menu, the highlighted elements are intended to draw a user's attention. Hence the highlighted elements are the first few elements that a user would notice when searching for the element of interest. The probability for finding the element of interest in the highlighted elements is equal to 1/N (N is the number of highlighted elements). When the element of interest is a non-highlighted element, the

user has to examine all of the N highlighted elements, and then find the element of interest in the non highlighted elements (i.e., M-N). When no elements are highlighted, the *average time to element* is searching among the elements in the original menu where M is the total number of elements in the original menu.

5 CASE STUDIES

As a proof-of-concept, we developed a prototype plug-in that highlights and hides menu elements according to the usage patterns of a user. Figure 1 and 2 demonstrates the differences between the original menu system and the adapted menu system. To examine the effectiveness of our proposed adaptive algorithms, we recruited five graduate students to use our adaptive UI as their software development environment. The graduate students use the original Eclipse IDE in their daily basis over three years. The menu system was monitored while the users were working in Eclipse. Our prototype collected the usage data in the background while the users interacted with the UI. We also evaluated the flexibility of the current implementation of the menu system for Eclipse IDE based on our experience in developing the prototype in our case study.

Adaptive Algorithms

The adaptive algorithms (the fade and enlighten algorithm) increase the efficiency of the user who is using the menu system. These algorithms are independent so that the AUI plug-in can choose to use one or both. We collect the times taken by a user to select menu elements and we calculate the probability of selecting the menu element of choice in order to measure the value of our adaptive algorithms.

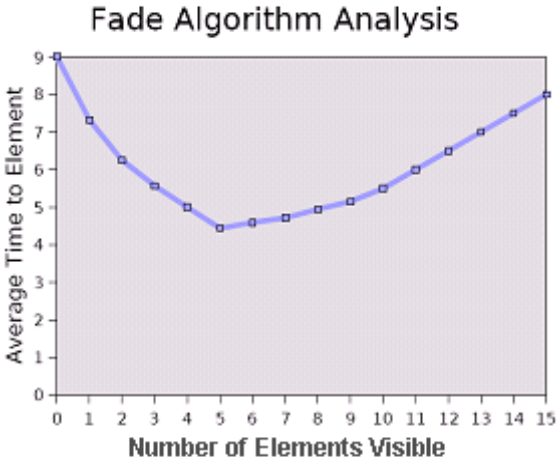
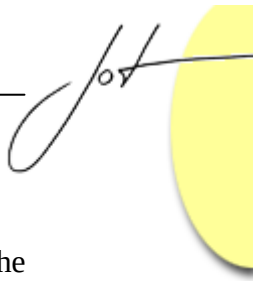


Figure 6: Sample Data for the Fade Algorithm



Test Cases for Fade Algorithm

Sample data for the fade algorithm are listed in Figure 6. The Figure illustrates the *average time to element* relative to the number of visible menu elements. As can be seen in the Figure, the number of visible elements for the lowest *average time to element* is 5 elements. With 5 elements shown the *average time to element* is 4.44 seconds. With all elements shown, the *average time to element* is 8.00 seconds. By showing only 5 elements, a speedup of 44% can be achieved. A large speedup is achieved using the fade algorithm. The probability of selecting the menu element of interest from the list of visible elements is illustrated in Table 2. The probability varies based on the number of the elements in the menu. As the number of elements increases, the probability of finding the menu element of interest decreases.

Table 2: Sample Probabilities for the Fade Algorithm

# of Elements Visible	Average Time to Element (sec)	Probability of Selection (%)
0	9.00	0
1	7.32	30
2	6.25	20
3	5.57	14
4	5.00	12
5	4.44	11
6	4.60	4
7	4.71	4
8	4.95	2
9	5.15	2
10	5.5	0
11	6.00	0
12	6.50	0
13	7.00	0
14	7.50	0
15	8.00	0

Test Cases for Enlighten Algorithm

Sample data for the enlighten algorithm are illustrated in Figure 7. The Figure depicts the *average time to element* relative to the number of highlighted menu elements. As shown in the figure, to achieve the lowest *average time to element*, the number of elements to highlight is only one element. With highlighting one element, the *average time to element*

is 6.97 seconds, whereas the *average time to element* is 8.00 seconds without highlighted elements. By highlighting one element, a speedup of 13% can be achieved. The probability of finding the menu element of interest from a list of visible ones is illustrated in Table 3. The probability varies based on the number of the menu elements highlighted in the menu. As the number of menu elements highlighted increases, the probability of finding the menu element of interest decreases.

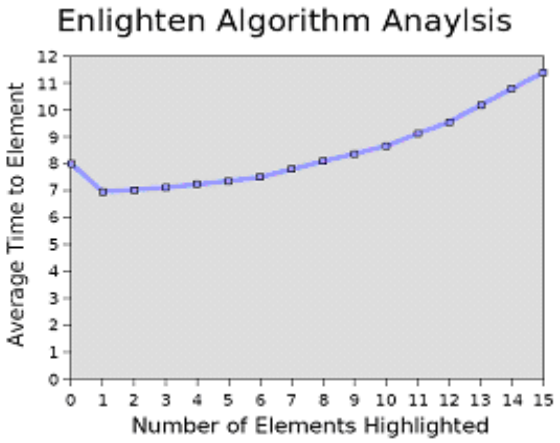


Figure 7: Sample Data for the Enlighten Algorithm

Table 3: Sample Probabilities for the Enlighten Algorithm

# of Elements Highlighted	Average Time to Element (sec)	Probability of Selection (%)
0	8.00	0
1	6.97	20
2	7.03	8
3	7.13	8
4	7.24	8
5	7.36	8
6	7.50	8
7	7.80	6
8	8.09	6
9	8.38	6
10	8.67	6
11	9.12	4
12	9.56	4

13	10.18	2
14	10.8	2
15	11.40	2

Precision of the Enlighten Algorithm

The result for evaluating the precision of the enlighten algorithm is summarized in Table 4. The events for triggering menu elements were recorded for the statistical analysis. The enlighten algorithm detects the usage sequences of the menu elements during the testing period and stores the usage patterns as sequences. The average length of the detected sequences is three menu elements. The number of predictions made is the times that the enlighten algorithm highlights the next menu element the user might click. The overall precision is the ratio of the number of times that the algorithm can predict correctly out of the total number of predictions made.

Table 4: Testing Results of Enlighten Algorithm

User	Testing Duration	Number of Sequences Detected	Number of Predictions Made	Overall Precision
User 1	5 hours	18	12	11/12=91.7%
User 2	8 hours	36	30	29/30=96.7%
User 3	10.5 hours	49	30	27/30=90.0%
User 4	11.5 hours	11	1	1/1=100%
User 5	51.5 hours	1471	1199	1167/1199=97.3%

As can be seen from Table 4, the precision of the enlighten algorithm is very high, over 90% in general. When the algorithm makes a prediction by highlighting a menu element, the prediction is accurate almost all of the time. It is also important to note that different users work with the menu system very differently. Some users prefer to use the toolbar for a lot of their menu element selections. In that situation, even if the menu element prediction is correct, it will not speed up the user's selection since they do not use the menu bar. Other users have learned a lot of the keyboard shortcuts used to access different actions. Similarly, when these users select a menu element using a shortcut, the prediction of a menu element will not help them. However, all the five users do use the menu system at least once in a while. From our observation especially when they do not use the menu system often, it is very likely that they have difficulty to find menu elements when they have to use. In this case, the enlighten algorithm can help them to quickly locate the menu element they want. Therefore, it is safe to conclude that the prediction algorithm improves the efficiency of the user interface for users in general.

Discussions and Limitations

The adaptive menu system (AMS) portion (*see* Figure 4) of the AUI is the only piece of code that poses problems in the final implementation of the plug-in. The AMS interfaces with the Eclipse GUI to dynamically make changes to the menu system. However, highlighting elements by changing their background color proved not to be possible in the current implementation of the Eclipse menu system. Due to the constraints imposed by the Eclipse menu system, we are not able to change the color of a menu element. As a proof-of-concept of our fade and enlighten algorithms, we represent icons to the text of the menu elements in order to illustrate the highlighting. In the future, these problems could be solved if better runtime access to the internal structure of menus is provided by the Eclipse API through extension points. We plan to further refine our implementation and integrate our prototype into the lab environment used by first year programming courses. The integration of our AUI into the lab environment will permit us to perform larger user studies in order to better understand the benefits and shortcomings of our AUI approach for novice users. We also plan to pre-configure possible usage patterns for novice users before the novice user interacts with the AUI. In this case, the AUI can automatically guide users to locate the next elements to click for the very first time they use the AUI.

6 CONCLUSION

In the paper, we propose an AUI architecture. As a proof of concept, we implement the AUI as a plug-in for the Eclipse IDE. The fade and enlighten algorithms are both robust and efficient, and significantly improve the efficiency of a user's interface by shortening the *average time to element* metric and highlighting the next element to click as shown in our case studies. The modular architecture of the plug-in allows for fairly easy migration of the software to other platforms and operating systems, while maintaining the general structure of the code the same. In the future, we plan to work with more volunteers, such as the first year of computer science or computer engineering students as novice users and graduate students with more than 3 years of software development experience as expert users. We want to examine if our adaptive algorithms are plausible and useful for novice users and expert users. We also plan to test the AUI in other Eclipse based products.

REFERENCES

Bowman, B., Debray, S. K., and Peterson, L. L. Reasoning about naming systems. ACM Trans. Program. Lang. Syst., 15, 5 (Nov. 1993), 795-825. "Graphical User Interface." Wikipedia. 19 Apr. 2006. Wikimedia Foundation Inc. 20 Apr. 2006.

"History of the Graphical User Interface." Wikipedia. 18 Apr. 2006. Wikimedia Foundation Inc. 20 Apr. 2006.

Eclipse Platform Technical Overview. IBM Corporation. Object Technology International, Inc., 2003. 1-20.

Nielsen, Jakob. "Usability 101: Introduction to Usability." Jakob Nielsen's Alertbox. 25 Aug. 2003. Nielsen Norman Group. 20 Apr. 2006.

H. Lieberman, “Letizia: An Agent That Assists Web Browsing,” MIT Media Lab, 1995, http://lieber.www.media.mit.edu/people/lieber/Lieberary/Letizia/LetiziaAAAI/Letizia.html.

Vivacqua, H. Lieberman, and N.V. Dyke, “Let’s Browse: A Collaborative Web Browsing Agent,” MIT Media Lab, 1997, http://lieber.www.media.mit.edu/people/lieber/Lieberary/Lets-Browse/LetsBrowse.html.

Liu, Jiming, Chi Kuen Wong, and Ka Keung Hui, “An Adaptive User Interface Based on Personalized Learning”, IEEE Intelligent Systems 1094-7167, 2003.

Eric Horvitz, Jack Breese, David Heckerman, David Hovel, Koos Rommelse, The Lumiere Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users, Microsoft Research, http://research.microsoft.com/~horvitz/lumiere.htm.

Microsoft Office System Preview Site, 2007, http://www.microsoft.com/office/preview/ui/overview.mspx.

Campos José Creissac, Michael D. Harrison, and Karsten Loer, “Verifying User Interface Behaviour with Model Checking”, Universidade Do Minho, Portugal. United Kingdom, University of York.

Margaret-Anne Storey, Jeff Michaud, Marcellus Mindel, Mary Sanseverino, Daniela Damian, Del Myers, Daniel German and Elizabeth Hargreaves. “Improving the Usability of Eclipse for Novice Programmers”, ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, 2003.

Leah Findlater and Joanna McGrenere, “A Comparison of Static, Adaptive, and Adaptable Menus”, Conference on Computer Human Interaction (CHI), 2004.

About the authors

Ying Zou received her B. Eng. degree from Beijing Polytechnic University, her M. Eng. degree from Chinese Academy of Space Technology, and her Ph.D. degree from the University of Waterloo in 2003. Currently, she is an assistant professor in the Department of Electrical & Computer Engineering at Queen's University in Canada. She is a visiting faculty fellow of IBM Centers for Advanced Studies (CAS), IBM Canada. Her research interests include software engineering, software reengineering, software reverse engineering, model driven software development, and business process management. She can be reached at ying.zou(at)queensu.ca

Michael Lerner is a student at Queen's University in Canada. He is currently finishing his undergraduate degree in Computer Engineering. He can be reached at 5ml17(at)queensu.ca

Mr. Leung, Mr. Morisson and **Mr. Wringe** graduated received their B.Sc at the Department of Electrical and Computer Engineering at Queen's University in 2006.