Developing an Adaptive User Interface in Eclipse

Alex Leung, Scott Morisson, Matt Wringe and Ying Zou
Department of Electrical and Computer Engineering
Queen's University
Kingston, Ontario, Canada
{2akyl, 1mw, 2esm}@qlink.queensu.ca, ying.zou@queensu.ca

ABSTRACT

The usability of user interfaces is often neglected in the design and development of software applications. The Eclipse Integrated Development Environment (IDE) is especially prone to poor usability problems due to the rich functionality introduced by the additions of external plug-ins and Eclipse's board target user base (from novice to expert developers). To improve the usability of Eclipse's user interface, we propose an Adaptive User Interface (AUI) architecture which modifies the existing Eclipse menu system based on statistical user interaction patterns with the user interface. Specifically, the AUI hides infrequently used menu elements, and predicts the next menu elements that a user is likely to click. Moreover, we develop adaptive algorithms that perform a cost-benefit analysis for making modifications to the menus system, and determine the optimal changes to make. A prototype AUI is developed as an Eclipse plug-in. Through an initial case study, we demonstrate the enhancement to the current Eclipse menu system using our Adaptive User Interface.

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 researches have proposed techniques to design and develop adaptive user interfaces. The proposed techniques react to different situations and requirements by they learning the behavior patterns and styles of individual users. Earlier work on adaptive user interfaces [5, 6] focuses on tracking the behavior of users using a Web browser and anticipates items of interest. This prior research analyzes the behavior of users by matching the keywords in Web documents. Microsoft Office Assistant utilizes the result of Lumiere Project [8] that employs Bayesian network to predict the future events and a goal graph to infer a user's need by considering a user's background, actions, and queries. Microsoft Word Office [9] and Windows XP can dynamically hide the menus elements of infrequent use. However, such software is not capable of predicting the possible future steps. [7] presents the techniques that provide personalized, just-in-time assistance to users, and automatically derive user's usage pattern from mouse events and text editing events generated from user interfaces. The result of this research is used to support word editing.

However, with these advancements, the user interface of Integrated Development Environments (IDEs) is often neglected, leaving the UI complex and difficult to use. IDEs often have a wide variety of functions for different development projects and they are designed to cater to a wide range of uses. Therefore, all functionality is made available at all times to a user, increasing UI clutter. IDEs, such as Eclipse (i.e., open source Java development environment) [3], often allow for additional plug-ins to be integrated to increase functionality. These plug-ins have their own UI components (such as, additional menu bars and icons.) which further compound the problem. Usability is an important factor which affects a user's productivity. With a frustrating, complex, and difficult UI, user's efficiency will likely decrease and users may switch to alternative software applications. Therefore, an enhancement to the currently popular WIMPS paradigm must be made to allow for a more efficient and pleasant UI for IDEs.

To improve user experience for current users of IDEs, the Guild project [11] enhances the user interface of Eclipse for the novice developers. However, the user interface presents static functions in the menu. In this paper, we propose an adaptive user interface (AUI) architecture that dynamically adapts the user interface to individual software developer's daily routines, such as coding, compiling, and debugging. Specifically, our adaptive user interface (AUI) is developed as a plug-in for Eclipse, and persistently runs in the background to collect statistics on how the user interacts with the menu system. For example, for the developers who focus on the debugging a code, we will hide the irrelevant menus such as "Files" and "Navigator". Different from earlier work which emphases on the prediction of user's plan, we also develop adaptive algorithms (e.g., Fade algorithm and Enlighten algorithm) that perform a cost-benefit analysis for making modifications to the menu system, and determine the optimal changes to make in the menu system. For example, through the use of the Fade algorithm, the menu elements that are seldom used by the user are hidden. This can reduce the size of the menu elements and allow quick access to menu elements that are most frequently used. To further improve usability, our plug-in also moves menu elements, which the Enlighten algorithm predicts that the user will select next, to the top of their menu. The decisions for hiding and moving menu elements are evaluated by our proposed cost-effective formulas.

The rest of the paper is organized as follows. Section 2 gives an abstract user model that describes a user's interaction with menu systems. Section 3 discusses the design of our adaptive user interface. Section 4 presents our case studies. Section 5 concludes the paper and describes the future work.

2. USER-MENU SYSTEM INTERACTION

Improving the UI usability is more of an art than a science due to the human factors involved. Different users will have different

usability expectations and usage patterns. However, metrics and certain methodology can still be used to quantitatively measure, compare, and improve usability. In general, the 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, the usability can be decomposed into five attributes: (1) *learnability*, which measures the ease of learning the application functionality; (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 application's functionality; (4) *low error rate*, which measures how the application support users in making less errors; and (5) *user's satisfaction*, which measures how the users enjoy the application. In our research, we aim to improve all five usability qualities through the use of an adaptive user interface.

2.1 User Model

To better understand how to increase usability of the menu system, we establish quantitative usability measurements, and construct a simplified user model that simulates how a user interacts with the menu system and maintains the knowledge of the user's usage patterns. Therefore, optimal modifications to the menu system can be made to improve usability. More specifically, we consider that the user interaction with the menu takes place in the following three steps:

1. The user selects the menu that they wish to view using either the mouse or keyboard. Since the name of the menu is at the top of the workspace, this is where their focus is located.
2. When the menu is shown, the user's focus is still located at the top of the menu.
3. The user searches from the top of the menu system downwards in a linear fashion until their target is reached. If using the mouse, the user has to move the mouse downwards, passing over each menu element in a sequential order. The same logic applies if the user uses the keyboard to access the menu system. In this case, the user has to hit the down arrow key to move down the menu system until they reach the menu element of choice. 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 of where they believe the menu element is located. This also applies to the case that the user knows the shortcuts to access menu and menu elements within the menu structure.

2.2 Model Analysis

One of the metrics that can be determined from the user model is the *average time to element* that is the time takes a user to find any random element within the menu. For a menu with its menu elements randomly distributed, the average number of elements considered before finding the element of interest will be the midpoint of the number of elements (the same as a linear search). The *average time to element* is the number of elements that have had to be considered before the element of interest is selected. By minimizing the *average time to element*, we can maximize the efficiency of the menu system. This can be achieved in two ways: *1)* by removing elements that are unlikely to be selected; and *2)* by moving elements with high probability of being selected to the top of the menu. Moreover, the unit of measure for the *average time to element* is the time it takes the user to go from one menu element to another.

Moving elements that have the highest probability of being clicked next to the top of the menu improves learnability and efficiency [10]. Initially, our adaptive user interface (AUI) plug-in does not modify the menu system except by removing grayed out (disabled) menu elements. For example as shown in Figure 1, some menu elements, such as "Save", "Save All" and "Revert" are disabled because the menu system prevents the user from clicking on. In this case, we hide all disabled menu elements. This shortens most menus and allows quicker access to menu elements that the user can select. This establishes a baseline level for learnability that is equal to the original Eclipse menu system. Menu changes only begin to occur after sufficient usage data has been collected. To improve efficiency, we place the most likely menu element to be clicked at the top. 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. The prediction will move the user's desired menu element to the top, and thus avoid memorizing the functionality of the menu system.

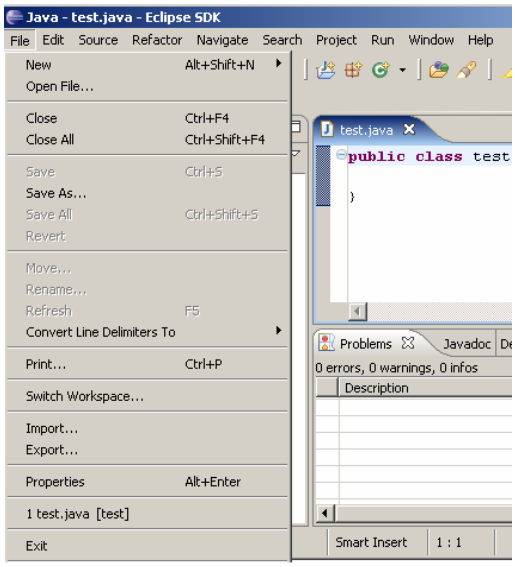

Figure 1: Original Menu System in Eclipse IDE

3. ADAPTIVE USER INTERFACES

To improve the usability of the Eclipse IDE, we develop the adaptive user interface (AUI) as a plug-in for the Eclipse IDE. The plug-in is loaded and executed when Eclipse is started. 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) had a number of menu elements and sub-

menus. After sufficient user usage pattern data has been collected, the plug-in will hide menu elements that the user infrequently uses. For instance, when a user continuously creates new projects by selecting the "New" menu element, the AUI plug-in dynamically only exposes the "New" menu element and hides the rest of the menu elements, as illustrated in Figure 2. 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 to the menu back to the original form when no menu elements are hidden.

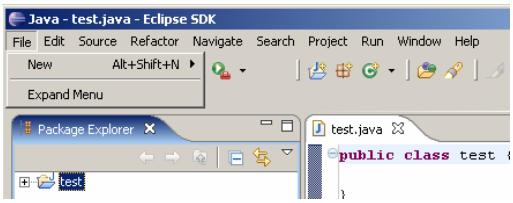

Figure 2: Adapted Menu System in Eclipse IDE

3.1 Architecture for Adaptive User Interfaces

The overall architecture for the AUI plug-in is depicted in Figure 3. Essentially, the user interaction happens among three major components: the Adaptive Menu System (AMS); the AUI Wizard; and the AUI preferences.

- The AMS is the main form of interaction the user has with the menu system.
- The AUI Wizard provides guidance and notification to the user when the user interface is started for the very first time after changes are made to the existing user interface.
- The AUI Preferences are preferences set by the user about how the AUI should act. These preferences will be accessed the same way as all other preferences are accessed in the Eclipse IDE.

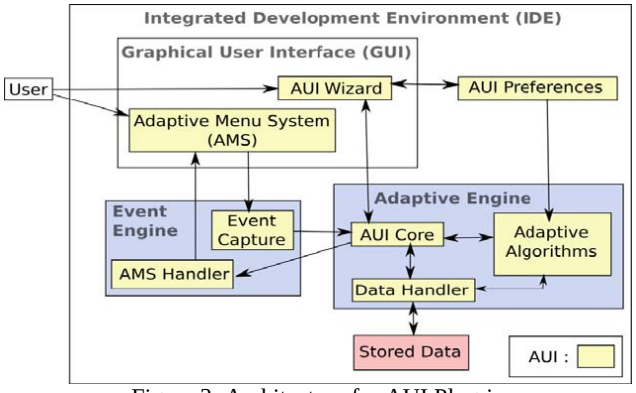

Figure 3: Architecture for AUI Plug-in

When the user interacts with the menu system, events are generated for every menu button clicked. The events are captured by the Event Engine. The Event Engine has two features: One is to notify the adaptive engine that an event has occurred; the other is to update the menu system if the adaptive engine determines to change. More specifically, the Event Capture component in the Event Engine captures all relevant data (e.g., which button was clicked, and the time the event occurred) and passes it to the

Table 1: Description of Components of AUI Plug-in

Component	Description
Adaptive Menu System (AMS)	A conceptual idea encapsulating the existing Eclipse GUI and the menu modifications made by the AUI plug-in.
AUI Wizard	Informs the user at the start of Eclipse that the AUI plug-in is running and will be collecting user data.
AMS Handler	Given the desired menu modifications (hiding or highlighting menu elements), this object performs the actual modification to the Eclipse menu system.
Event Capture	Every time the user interacts with the Eclipse menu system, an event is generated. This object captures these events and passes it to the AUI Core for processing.
AUI Core	The heart of the plug-in which orchestrates all the functionalities of the plug-in. It passes captured events to the Data Handler, runs the appropriate algorithms when necessary, and issues menu modifications to the AMS Handler.
Adaptive Algorithms	Includes the Fade algorithm (which is responsible for hiding menu elements) and Enlighten algorithm (which is responsible for predicting and highlighting menu elements). It may call upon the Data Handler to obtain user usage pattern data as input to the algorithms.
Data Handler	Captured events are transformed into tree structures (a menu tree and a sequence tree) for analysis. The Fade algorithm requires data from the menu tree to hide items and the Enlighten algorithm requires data from the sequence tree to predict menu elements. At the termination of Eclipse, all tree data in active memory are stored persistently on disk.
Stored Data	All tree data are stored persistently on disk at the termination of Eclipse. The data is formatted in XML and stored as plain text on disk. When Eclipse starts, this XML data is parsed back into active memory.
AUI Preferences	Using the existing Eclipse preferences system, this allows the user to set global options for the AUI plug-in. The user may turn the plug-in on or off using the AUI preferences.

Event Handler. The Event Handler takes all incoming event data and calls the Data Handler to store it into persistent memory in an XML format. The Event Handler will also determine if any of the Adaptive Algorithms needs to be called. Using the usage pattern data collected from the user and the AUI preferences set by the user, the Adaptive Algorithms determine if any changes to the interface are required. If changes to the interface are required, the Adaptive Algorithms notify the Event Handler which initiates the

AUI Wizard to notify and confirm all changes to be made with the user. If the user accepts any of the changes, the AMS Handler is called to implement the changes. Any changes not accepted by the user will be stored for later reference. Moreover, the Adaptive Algorithms fulfill the usability criteria. The challenging aspects of designing the Adaptive Algorithms are to determine how the algorithms can learn and change the UI based on the user's interaction with the AUI. The detailed description of each component in the architecture is specified in Table 1.

3.2 Usage Prediction

To predict which menu element the user is most likely to select next in the AUI, it is necessary to develop a method that can efficiently track usage patterns in the menu system and applies the collected data for effectively predicting future events. For example, if the user statistically first selects "Copy" menu element under the "Edit" menu, and clicks "Paste" menu element under the "Edit" menu in succession frequently. The AUI plug-in can anticipate that the user is likely to select "Paste" menu element under the "Edit" menu after the user selects the "Copy" menu element under the "Edit" menu. To reduce the access time, the predicted menu element (i.e., the "Paste" menu element) is copied to the top of the menu (i.e., the "Edit" menu) to make it the first element that the user can see. Each menu selection generates an event. We capture these successive events from the menu system in a sequence, so called as an event sequence. We consider that every event sequence corresponds to a single usage pattern, such as copy and paste pattern.

One of the challenges is to design a method that transparently determines boundaries of several event sequences that the user interacts with the user interface over a period of time. It is impractical to require the user to manually mark the starting point and end point of their usage pattern. It is impractical for two reasons: First, it requires additional effort from the user to remember the beginning and end of event sequences which would undoubtedly be annoying and would be hard to remember doing. Secondly, the user might not always be aware of their own usage patterns and not recognize sequence of events as a pattern that occurs often in their usage. In our research, we consider that the event sequences can be separated by intensity of uses. For example, we sample the menus events when the user intensively interacts with the user interface. We measure the intensity by calculating the speed of event generation. When the intensity of use drops, we initiate a new event sequence.

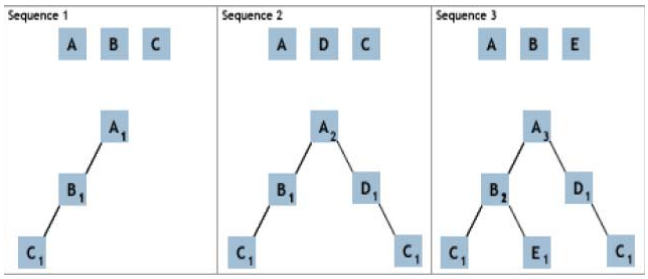

Figure 4: Sequence Tree Building

To capture these event sequences, a tree structure is used where each node represents a menu element selection and each node has a number of children that represent menu elements that have been selected after the current nodes menu element. Each node also has a counter that represents the number of times a path through the node has been taken. Figure 4 shows an example tree being generated from the given sequences.

When an event sequence begins, a scan for a tree with the corresponding events occurs, if a tree is located then the node counter is incremented and a pointer to the current tree is kept, otherwise a new tree is created with the root node being the event that occurred. In Figure 4 when the first event sequence occurs, no tree with root event A exists so a new tree is created with root event A. When event sequence 2 occurs a tree with root event A has already been created so the counter to the node is incremented. One advantage of this structure is that it can be used to predict at the same time as it is being built. Since a pointer to the current position in the tree is kept as a sequence of events occurs, the adaptive algorithms can check a list of children nodes and their occurrence counts to predict what the next event will be. When the next event occurs, the event can be inserted into the tree as the tree is being traversed.

Further study into methods of determining when event sequences begin and end would be beneficial. It may be possible to make the time intervals dynamic in that the system would dynamically determine the amount of time between events that occur in sequence often, and use this interval to determine the beginning and end of a sequence with a desired confidence level.

3.3 Adaptive Algorithms

To dynamically personalize the menu system based on the usage patterns, we propose two adaptive algorithms: the Fade Algorithm which handles temporarily removing menu elements from the menu system and the Enlighten Algorithm which handles move the menu elements to the top of the menu. Both adaptive algorithms gather the statistical data stored from the event sequences, determine which menu elements should be changed, and notify the AMS Handler (as shown in Figure 3) to make the appropriate changes to the menu system. Moreover, the adaptive algorithms decide when to make a change to a menu, and which changes to make. In particular, 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 users' behaviors. The underlying variable for both adaptive algorithms is the probability of a menu element being selected, and its variance relative to the rest of the menu elements. We obtain this probability from the stored event sequences in the tree structure.

To determine if a menu element needs to be justified, we need to analyze the interdependencies among elements in the menu and the impact of cost-benefit on the entire menu systems. We utilize the user model as discussed in Section 2.2 to keep track of the changes made in the menu as a whole. The goal of the adaptive algorithms is to determine the number of elements to remove or highlight that will minimize the *average time to element.* The challenge is to change the menus in a way that still keep the usability level the same, or increase it.

3.3.1 Fade Algorithm

As aforementioned, the Fade algorithm determines which elements to be removed from the menu. The basic functionality of the Fade algorithm follows the following steps:

1. Obtain the stored statistical data from the event tree structure
2. Determine the *average time to element* for varying numbers of menu elements removed
3. Select the optimal number of elements to remove, and
4. Determine if there is a significant speedup after removing the menu elements or showing additional menu elements

To calculate the *average time to element* for a menu element, we propose the Fade Equation, as defined in Equation 1. The *average time to element* is determined based on the elements being randomly ordered within the menu, as such the *average time to element* will be the midpoint of the number of elements shown. The ordering is not considered for the *average time to element*, as the order cannot be changed by removing elements. Moreover, the elements to be removed are considered in the order of decreasing probability of being selected.

$$T_{menu}(N) = P_v \cdot T_v + P_h \cdot (T_s + T_e) \quad \text{…….. } \textit{Equation 1}$$

$T_{menu}(N)$: The *average time to element* for a menu that has N elements visible
N: The number of visible elements
P_v: The probability of selecting visible elements
T_v: The *average time to element* for the visible elements
P_h: The probability of selecting hidden elements
T_s: The time it takes to search the visible elements and determine the element of interest is hidden
T_e: The *average time to element* for the expanded menu

We also consider the case that a user fails to find the element that is removed from the menu. In this case, the user has to click the "Expand" menu element as illustrated in Figure 2. The time it takes the user to search a menu and find an element of interest is not displayed in the menu is the number of menu elements listed in the current menu plus 1 for having to select the "Expend" menu element. If M is the number of elements in the original menu, the Fade equation for considering expanding the menu is defined in Equation 2.

$$T_{menu}(N) = P_v \cdot (\frac{N+1}{2}) + P_e(N+1+\frac{M+1}{2}) \text{…. } \textit{Equation 2}$$

To minimize the *average time to element*, we vary the number of visible elements in the menu in all possible combinations. We calculate the optimal number of elements that need to be removed. In this case, we remove the elements with the lowest probability of being selected. Furthermore, we determine the speedup between removing these menu elements and leaving the menu unchanged. The speedup must significantly offset the delay caused by the changes made to the menu system, since frequent changes to the menu system decrease usability and performance. The elements are removed only if the speedup is deemed the maximum in the varying number of visible elements.

3.3.2 Enlighten Algorithm

The functionality of the Enlighten algorithm is to determine which elements should be highlighted. Due the constraints of Eclipse menu system, we are not able to change the color of a menu element. For the sake of proof-of-concept of the Enlighten algorithm, we move the elements to the top of the menu. The basic functionality of the Enlighten algorithm is to gather the stored statistical data in the event tree structure, determine the optimal number of elements to highlight, and then determine if there is a significant speedup to warrant highlighting elements. The Enlighten Equation, as listed in Equation 3, is used to determine the *average time to element* with varying number of elements being highlighted. The elements under consideration for highlighting are ordered by elements with the highest probability of being selected for the current sequence pattern.

$$T_{menu}(N) = \begin{cases} P_h + P_n \cdot (1 + T_{menu}(N-1)) & N \neq 0 \\ T_{remainder} & N = 0 \end{cases} \textit{Equation 3}$$

$T_{menu}(N)$: The *average time to element* when N elements are highlighted
N: The number of elements to be highlighted
P_h: The probability of the selecting the highlighted element N
P_n: The probability of not selecting the highlighted element N
$T_{remainder}$: The *average time to element* for the non highlighted elements

The probability of selecting the highlighted element is the case that when the user correctly have the element at the top of the menu, this will be the first element seen, and hence will only take 1 time unit to find it.

The probability of not highlighting the proper element is plus by 1 (since the user has to look past the first element, and it is not selected) plus the cost of highlighting another element (if another element is to be highlighted) or the average time to the rest of the elements (if no additional elements are considered for highlighting.

It is worth to mention that the *average time to element* for non highlighted elements is computed by considering that the elements are randomly ordered within the menu. As such, the *average time to element* is based on the number of elements being displayed, but not the position and probability of the elements of being selected. Although using the position and probability will give a more accurate *average time to element*, it will also mean that the Enlighten algorithm has to maintain the orders of all elements in the menu. This is not as efficient as a linear search. By considering the non highlighted elements as being randomly ordered, we can offset the computation overhead that maintains the order of all elements.

4. CASE STUDY

As a proof-of-concept, we developed a prototype plug-in that successfully expands/collapses menu elements and hides menu elements according to a user's usage patterns. Figure 1 and 2

demonstrates the differences between the original Eclipse menu and the adaptive "File" menu. In the following subsections, we examine the effectiveness of our proposed adaptive algorithms. Moreover, we evaluate the current menu system for Eclipse based on our experience in developing the prototype used in our case study.

4.1 Test Cases for Adaptive Algorithms

The adaptive algorithms are used to increase the efficiency of the menu system, including the Fade algorithm and the Enlighten algorithm. These algorithms are independent so that the AUI plug-in can choose to use one or both. We design several test cases to validate the adaptive algorithms.

4.1.1 Test Cases for Fade Algorithm

Sample test cases of the Fade algorithm are listed in Figure 5. This figure graphs the *average time to element* to the number of menu elements visible. As can be seen in this figure, the number of elements visible for lowest *average time to element* is 5 elements. With 5 elements shown the *average time to element* is 4.44 seconds, versus all elements shown where the *average time to element* is 8.00 seconds. By showing only 5 elements, a speedup of 44% can be achieved. As can be shown, a significant speedup can be achieved using the Fade Algorithm.

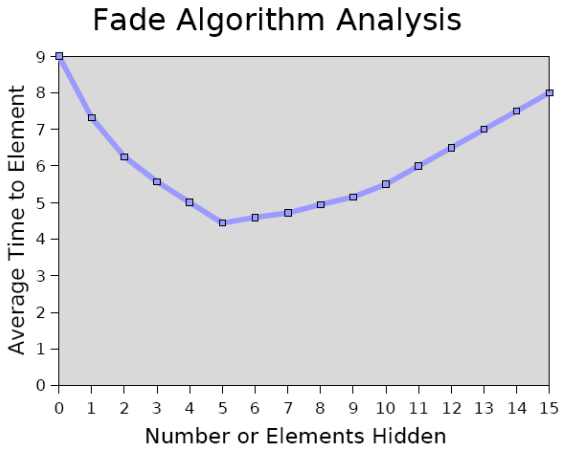

Figure 5: Test Cases for the Fade Algorithm

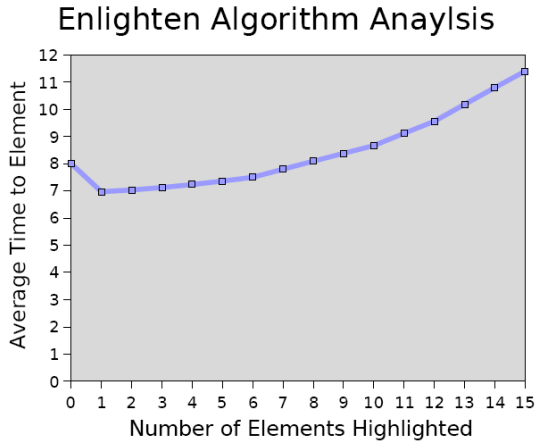

Figure 6: Test Cases for the Enlighten Algorithm

4.1.2 Test Cases for Enlighten Algorithm

Sample test cases of the Enlighten Algorithm are illustrated in Figure 6. This figure graphs the *average time to element* to the number of menu elements highlighted. As can be seen in this figure, the number of elements to highlight for the lowest *average time to element* is only 1 element. With 1 element highlighted the *average time to element* is 6.97 seconds, versus with no element highlighted the *average time to elemen*t is 8.00 seconds. By highlighting 1 element, a speedup of 13% can be achieved. As can be shown, a significant speedup can be achieved using the Enlighten Algorithm.

4.2 Evaluation of the Menu System in Eclipse

The Adaptive Menu System (AMS) portion (*see* Figure 3) 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, not all of the menu elements can be moved to top of the existing GUI system as desired. We observe that we are limited in Eclipse in changing an index or swapping the order of elements in a menu. Since the internal structure that controls the order of menu elements is not accessible at run time. To work around this limitation, we create a new element, render the original element off screen (invisible) and attach a listener to the new element. When a selection event is triggered on the original element, we dispose of the new element, and render the original menu element, making it visible again. Moreover, highlighting elements by changing their background color proved not to be possible in the current Eclipse menu system. In the future, these problems could be solved if better access to the menus and their internal structure is provided by the Eclipse API at runtime through extension points.

5. CONCLUSION

In the paper, we propose an adaptive user interface (AUI) architecture. As a proof of concept, we implement the adaptive user interface as a plug-in for the Eclipse IDE. Our AUI plug-in has shown to improve usability by shortening *the average to element* for a user to find a menu element in our test cases. In the future, we plan to work with 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 aim to examine if the adaptive algorithms are plausible and useful for novice and expert users.

6. ACKNOWLEDGMENTS

This work is based on the results of an undergraduate 499 course project (the 4th year student graduation project) in the Department of Electrical and Computer Engineering at Queen's University. We would like to thank Dr. Michael Greenspan, the course instructor, for his valuable feedback to the work. We also appreciate Mr. Qi Zhang's help to this work.

7. REFERENCES

[1] 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.

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

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

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

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

[6] A. 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/Lets-Browse.html.

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

[8] 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.

[9] 2007 Microsoft Office System Preview Site, http://www.microsoft.com/office/preview/ui/overview.mspx.

[10] 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.

[11] 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.