Illustrating Macros with Existing Documentation

Eugene R. Creswick
School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, OR 97331
creswick@gmail.com

Lawrence Bergman, Tessa Lau, Vittorio Castelli, Daniel Oblinger
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598
{bergmanl,tessalau,castelli,oblio}@us.ibm.com

ABSTRACT

This paper presents an approach that enlivens existing documentation, to efficiently support users performing procedural tasks. We describe an approach that automatically creates, in real-time, correspondences between the actions performed by users and the steps described in online documentation for the procedure being performed. Using these correspondences, our Macro Illustrator system can highlight the relevant portions of the documentation and provide the user with a visual indication of the progress being made. Users are therefore offered in-context documentation, which helps them track their current position in the procedure, and answers questions about the next steps to be taken. This approach works with existing documentation, requiring no additional markup on the part of the documentation author. We present an algorithm for extracting actions from documentation and aligning these extracted document actions with observed user actions through a system of similarity metrics. Empirical evaluation of this algorithm shows that it performs significantly better than a strawman approach.

ACM Classification: I.2.7 [Artificial Intelligence]: Natural Language Processing - Text Analysis. K.6.1 [Management of Computing and Information Systems]: Project and People Management - Training.

General Terms: Algorithms, Documentation, Human Factors

KEYWORDS: computer-based training, information extraction, online documentation, online help systems

1. INTRODUCTION

Computer users are routinely confronted with ancillary tasks, such as configuring network settings, setting up a development environment, and creating e-mail filters, that can affect their productivity. Some rare tasks are performed so infrequently that users must resort to documentation or to external help to be able to accomplish them. Collectively, these tasks impose a burden on time and resources that can have a negative impact on the users' actual responsibilities. There are two approaches that are commonly used to mitigate this problem: documentation and automation.

Detailed documentation, in either electronic or printed form, typically consists of formatted text accompanied by images of the graphical user interface (GUI), and provides a visual representation of the associated procedure. These documents, however, are static sources of information and lack the interactive nature of the interface with which the user is working. A user who relies on documentation is required to make the correspondence between the document and the application's GUI. This task can be challenging even for moderately complex procedures, and is further complicated by the all too common phenomenon of documentation obsolescence, and by the variability of the application's appearance due to differences in user preferences and environment configuration. In a previous study [14] we observed that these discrepancies between the documentation and the interface have a detrimental effect on users' ability to complete documented procedures successfully. Moreover, users tended to miss portions of the instructions, particularly instructions involving conditional branches. In the study, we noticed that users had difficulty completing tasks successfully in general, and that in particular they commonly lost track of their position within the procedure.

The need for good documentation or for external support could be completely obviated if effective task automation were widespread. There are two main approaches to task automation: programming (e.g., scripting) and macro recording. Building a program that guides a user through a procedure, for example, by means of a scripting language, is in general an onerous task. The programmer must be careful to account not only for the procedure structure, but also for the variability of the user interfaces required by the procedure. Macro recording provides a cheaper and quicker alternative to automation, but comes with severe limitations. A macro recorder observes a user performing a task by interacting with a GUI. It records every user action, and produces a "macro", namely, a description of the actions that can be repeated at a later time. Macro recorders do not require programming, and building a macro is only as expensive as performing the corresponding procedure. However, macro recorders are typically not robust with respect to variations in the appearance of the application's GUI, and can capture only trivial procedure structure.

Widespread task automation is not a reality at this point in

time, and therefore the need for good, effective documentation is very much a reality.

The presence of documentation alone, however, does not resolve all of the problems that plague users when performing these ancillary tasks, there is a need for more intelligent help systems. The root of such an intelligent help system is in the ability to convey what is going on to the user, and what to do next. Users manually performing procedures may become lost and need assistance to find their location in the documentation they are using. Static instructions could benefit from a tie between documentation and the actions being performed on the interface. Currently, an enormous amount of online documentation exists for myriad tasks; however, as mentioned above, the user is required to build correspondences between steps in the documentation and the relevant portions of the application interface. Some intelligent help systems, such as Follow-me wizards [2] and HelpTalk [19, 20] remove this burden from the user by generating documentation automatically. These dynamically generated documents are in the form of human-readable scripts, in the case of Follow-me wizards, or in the form of answers to very specific questions about the user interface, in the case of HelpTalk.

This paper addresses the problem of incorporating existing documents for use in intelligent help systems. Our approach provides users with a guide as they perform actions, this guide is based on documentation. To accomplish this, we have designed algorithms that autonomously extract actions from legacy documentation as well as align user actions with the extracted actions. The implementation of these algorithms, Macro Illustrator, is also presented here along with an empirical evaluation of the alignment algorithm applied to a corpus of on-line documentation.

The organization of the rest of the paper is as follows. Section 1.1 describes the Macro Illustrator interface and outlines the benefits of such a system. In Section 2 we discuss the related work, Section 3 describes the algorithm developed and Section 4 presents a comparison of our algorithm with two straw-man approaches. Section 5 contains conclusions and describes areas for future work.

1.1 The User Experience

Macro Illustrator has been implemented as a plug-in for the Eclipse Workbench[1]. This plug-in behaves similarly to the Eclipse help system, Figure 1 shows the Macro Illustrator interface with a simple procedure loaded.

For example, Alice is a software engineer who periodically needs to modify build settings to account for changes in an ongoing project. When this happens she opens a help document for configuring projects in Eclipse. This starts Macro Illustrator. As she performs the first step, by choosing "Preferences" from the "Window" menu, Macro Illustrator highlights the first step in the procedure (shown in Figure 1). After each action is performed, the highlighted portion of the document moves along with Alice's actions, keeping track of her location in the procedure. When Alice reaches Step 4, rather than adding or removing projects, she instead selects

[1]The Eclipse Workbench [10] is an application platform developed by IBM.

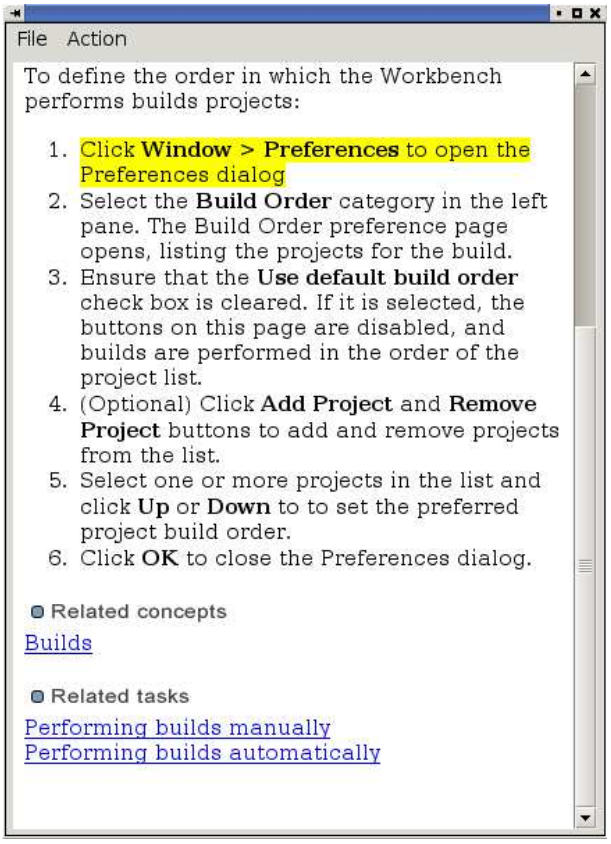

Figure 1: The Macro Illustrator interface, displaying a procedure that a user is performing. The first step in this procedure was performed by the user and subsequently highlighted by the system.

a project and clicks the "Up" button to change the project's build order. Macro Illustrator realizes that she has skipped Step 4, and highlights Step 5, correctly keeping pace with Alice. Alternatively, Alice's build order may have already been correct, in which case she would perform Step 6 by clicking "Ok". Macro Illustrator then skips over both steps 4 and 5, highlighting Step 6.

By watching the user, Macro Illustrator follows in this manner as long as it is activated. The example above addresses a relatively simple procedure, however with more complex tasks we anticipate that the benefits of an automatic guide will increase greatly. Complex conditional actions and subtasks can both cause confusion with users, however either case could be managed with Macro Illustrator.

Macro Illustrator provides the following benefits over current technology:

- Macro Illustrator can help the user keep track of his or her progress though the procedure, of the accomplished subtasks, and of the steps that still need to be taken.
- Macro Illustrator processes documentation autonomously, extracting actions from the documents without requiring any additional markup. This allows legacy documentation to be used without modification.
- Macro Illustrator connects user actions directly to action

descriptions in documentation by providing immediate visual feedback after each user action.

2. RELATED WORK

The intent of Macro Illustrator is to make software procedures easier to perform. This goal is shared by various online help systems that range in complexity from static documentation to complex architectures that monitor a user's actions to predict his or her motives. The distinctions between Macro Illustrator and static documentation should be clear from the introduction and the example. Here we examine more complex computer help systems.

The Apple Guide [1], presents users with a step-by-step procedure. The method of interaction with the Apple Guide is different from that of Macro Illustrator, our system enhances existing documentation, but the Apple Guide is primarily intended to provide automation. Despite the automation of Apple Guide scripts, user interaction can still be guided by "coach-marks" that highlight on-screen widgets during procedure execution. These coach-marks are similar to the highlighting provided by Macro Illustrator; however, Apple Guide scripts must be carefully hand-authored by experts. The coaching provided by the Apple Guide is also independent of legacy documentation.

Eclipse provides a tool similar to the Apple Guide called *Cheat-sheets* [9]. Cheat-sheets engage users in a dialog, guiding them through instructions. These two approaches to automation and documentation differ in that Cheat-sheets can be used in a mixed-initiative manner. Users may either play each step of the cheat-sheet by clicking on a "Play" button, or he or she may perform the step manually. In either case, the cheat-sheet will keep pace with him or her. Like Apple Guide scripts, cheat-sheets must be meticulously created by experts, melding the procedure steps with documentation manually. Macro Illustrator autonomously performs a task similar to that of a cheat-sheet programmer, on existing documentation.

Horvitz, et al. created Lumière [8], an agent that retrieves appropriate documentation when the user exhibits specific behaviors. Lumière utilizes a Bayesian network that was built by observing user behaviors in Microsoft Excel. The agent watches all the user actions, predicting the user's goal at each step. When the user appears to be pursuing certain goals, such as searching the Excel menus, Lumière interrupts the user and offers relevant documentation. This behavior is complementary to Macro Illustrator; once the documentation has been located and presented to the user, Lumière's task is over. Lumière identifies documentation, while Macro Illustrator helps the user once the appropriate documentation has been retrieved.

Other techniques for assisting users with tasks have also been explored. Forms/3, a spreadsheet programming language [3, 4], focuses on integrating techniques from the area of professional software engineering with end-user programming languages. This approach, termed *End-User Software Engineering*, uses primarily visual techniques to help users validate their spreadsheets. One technique, called *Surprise-Explain-Reward* uses a system of intelligent tool tips to provide in-context help that changes dynamically based on the state of the system. Wilson et al. have demonstrated that the Surprise-Explain-Reward system is sufficient to educate end-users in the software engineering devices built into Forms/3 [21]. This type of education bridges the gap between documentation and the interface at the expense of specificity. The tool-tips provide very general assistance regarding the interface, unlike Macro Illustrator, which offers help with executing specific procedures.

Sukaviriya presents a dynamically generated help on specific interface widgets in Cartoonist [18, 19], and later HelpTalk [20]. Based on a model of the user interface, these systems are meant to answer user questions about specific widgets in the interface. The Whyline [13] answers similar questions about program behavior in the Alice programming environment. In Alice, users are able to ask "Why did..." and "Why didn't..." questions about runtime failures. Ko and Myers have shown that the Whyline improves debugging performance, however no work has been done to extend this approach to standard computer use.

3. APPROACH

Our goal is to enliven documentation without requiring any special markup in the documentation. To this end, Macro Illustrator needs the ability to automatically extract actions from the documentation and then match these document actions with the corresponding actions performed by a user (termed user actions). Because of the dynamic nature of the visual feedback, each user action must be immediately aligned with the corresponding document action and displayed to the user. The process of performing this mapping between document actions and user actions is decomposed into a sequence of subgoals, which are briefly described here and discussed in further detail in the following sections.

First, a *Document Preprocessor* analyzes the documentation to find locations in the documented procedure that are likely to be procedure steps. Second, the interactions between the user and the application are recorded by an *Instrumented Environment* that abstracts them into user actions. The most recently demonstrated user actions and the annotated document are then correlated by an *Alignment Engine*, which creates the desired correspondence. Finally, the *Document Viewer* highlights the section of text identified by the Alignment Engine, visually indicating the action in the procedure that the user just performed. Figure 2 shows the interactions and dependencies between these modules.

3.1 Document Preprocessor

The first functionality of Macro Illustrator is to process every new document, to extract action descriptions from the text. Typically, documentation is created with human readers in mind, and is therefore only semi-structured. In order to support legacy documentation, Macro Illustrator must act fully autonomously, extracting information from these semi-structured documents without any additional input. Because these documents are not strictly structured, and because of the ambiguous nature of the English language, this step initially appears as difficult as the general natural language processing problem, the description of which is beyond the scope of this paper. As a reference for the reader, we men-

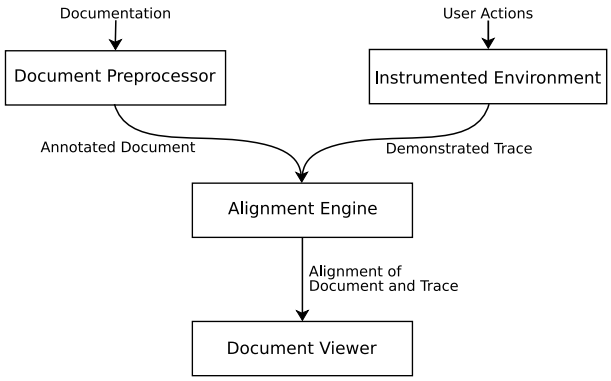

Figure 2: The four-part architecture of Macro Illustrator. Modules are depicted as rectangles, and communication between modules follows the arcs.

tion McCord's system [16, 17], which uses slot grammars to build deep parsers for natural language processing.

A deep parser, such as the slot grammar approach mentioned above, can extract detailed information about the steps in a procedure. However, such a parser is also prone to errors due to required background knowledge or poor English in the documentation. (Translated documentation is a prime example of the later case.) One example where McCord's deep parser fails to meet the needs of our domain is the action description: "Open the print dialog and select preferences." This command is ambiguous, and the term "select" is often parsed as an adjective, but it is actually being used as a verb.

The parsing problem is further complicated because many documented procedures contain meta-information about the procedure, or background information about the task at hand. This additional information may help the user determine when a possible procedure is applicable, or direct actions that are outside the scope of the environment (for example, configuring hardware). This additional content is not directly applicable to the procedure steps, and should not be highlighted for the user. (Such information may still be of value to the user, and is displayed.) The Document Preprocessor filters this meta-information by annotating the portions of the document that appear to be instructional. These annotated portions of the document are then further processed to extract the action(s) described in those sections of the document. The annotated portions are later highlighted by the Document Viewer when the document actions are aligned with demonstrated user actions.

To avoid the issues with deep language parsing (described above), we chose to take advantage of the semi-structured nature of online documentation to extract actions. Through an informal examination of instructional documentation from various sources, including 18 procedures from the Eclipse online help system, randomly selected IBM RedBooks[2], and various web-based documentation sources, we found that nearly all instructions are presented in bulleted or enumerated lists. Based on this, our approach first identifies elements of such lists. Each list item is considered a potential document action. In the case of nested lists, the content of the outer list prior to the first element of the nested list is identified. This filters much of the meta-information from the documentation, leaving only the potential document actions.

These potential document actions are then searched for the occurrence of verbs from a list of action verbs, termed *go-verbs*. (The creation of this list is discussed in Section 4.) Each occurrence of a go-verb becomes one action in the annotated document, and that action is associated with the text from the list item containing the go-verb. When there is more than one occurrence of a go-verb in a list item, a document action is extracted for each instance of a go-verb. List items that contain no go-verbs are not annotated, and therefore these items are not considered by the Alignment Engine. Each annotated step contains one go-verb and the corresponding text from the documentation. Included in these annotations is the location and length of the descriptive text for each action.

3.2 Instrumented Environment

To determine what step the user is on, Macro Illustrator compares the two types of actions previously described: *document actions* and *user actions*. Each widget, such as a button, menu, or checkbox, triggers an event when it is activated. These basic widget events correspond to the instructions in online documentation. For example, the instruction: "Click Ok" is commonly found in instructional documents, and it corresponds directly to the event of a user clicking on a button labeled "Ok".

Each recorded action consists of two parts: The type of action performed, which is dependent on the *type* of widget that was activated, and the descriptive text associated with that particular widget. As each action is performed the actions are stored in a list, which we term a *trace*. This trace (or partial trace, if the demonstration is not yet complete) is then given to the Alignment Engine to be aligned with the documentation.

3.3 Alignment Engine

The core challenge of Macro Illustrator is the creation of a correspondence between the user actions, recorded by the environment, and the document actions extracted by the Document Preprocessor. The module devoted to this task is called the Alignment Engine.

This task is similar to many string-alignment tasks—the input to the Alignment Engine consists of two strings (sequences) of actions. Each "character" is either a user action or a document action. There has been a wealth of research [5, 6, 12] in string matching when the characters of each string can be immediately checked for equivalence, however, user actions and document actions are rarely exactly equal. Jagadish, et al. address the topic of similarity-based queries, in which characters are similar, but not identical [11]. Their approach, however, requires specification of a pattern language of bounded complexity.

Recall that the Alignment Engine must provide the user with immediate visual feedback when aligning user actions and

[2]RedBooks are exhaustive documents describing complex procedures such as confi guring database systems and web servers.

document actions. Because of this, each alignment must be based on the extracted document actions and the user actions performed so far—the alignment of user actions and document actions is of no use to the user after his or her task is complete, so this alignment must be performed as soon as the user actions are performed.

We have designed an algorithm that performs this alignment by calculating a similarity score for each possible pairing of demonstrated user actions with document actions from the documentation. The calculated similarity scores are used to determine the document action that corresponds to the last user action. This similarity score is determined by the product of three similarity metrics:

$$\begin{aligned}\text{SimAlign}(U, D) &= \text{temporal\_sim}(U, D) \\ &\quad \times \text{description\_sim}(U, D) \\ &\quad \times \text{action\_type\_sim}(U, D)\end{aligned}$$

Where U and D are a user action and document action, respectively. The three similarity metrics, temporal similarity, description similarity and action type similarity, are described below.

Temporal similarity: Software help procedures are predominantly designed to be read and performed in order. Therefore, intuitively the n^{th} action performed by the user is likely to correspond to an action near the n^{th} action described in the documentation. This intuition is maintained by the Alignment Engine through a temporal similarity metric. The index of the user action is known, and can be used to determine a likely window of actions in the documentation. A Gaussian distribution is used to calculate a similarity between the current user action and all document actions. The mean and standard deviation for this Gaussian distribution are chosen in the following way: To account for optional steps, such as Step 4 in Figure 1, the temporal similarity metric uses a mean equal to the index of the document action immediately after the document action aligned with the previous user action. Specifically, if the user action A_k is aligned with the document action D_l, then the temporal similarity metric for action A_{k+1} will center on the document action D_{l+1}. In all cases, the standard deviation of the distribution is set to be one third the length of the documented procedure because this ensures a positive temporal similarity score for all document actions (a similarity score of 0 would nullify the effects of the other two similarity metrics).

Description similarity: Recall from Section 3.2 that each document action consists of a section of text taken directly from the documentation. This text often describes the widget with which the user should interact, and some of the context for that widget. Additionally, each widget in the interface typically has a textual label, or other descriptive text associated with it (for example, tool-tips). This descriptive text is collected by the instrumented environment when the user demonstrates an action. In this way two lists of descriptive words are collected, one list for the document action (W_d) and one list for the user action (W_u). The description similarity (S_d) of two actions is the normalized word-overlap between these two lists of words:

$$\frac{\mid W_d \cap W_u \mid}{\mid W_d \mid + \mid W_u \mid}$$

Word overlap was used to calculate description similarity because the widget titles (such as the words "Ok" and "Cancel" commonly found on buttons) are usually written verbatim in documentation, but occur relatively uncommonly in the associated explanatory text.

Action type similarity: Instructional documentation for software procedures typically uses a fixed vocabulary when describing actions. We collected a set of these action words through examination of 18 software procedures from the Eclipse on-line help documentation. This set then formed the list of go-verbs (mentioned in Section 3.1 and used to inform the Document Preprocessor). Because the number of widgets is relatively small, and most widgets are acted upon in unique ways, this collection of verbs is also useful in predicting the similarity of actions. While the user's physical action is usually the same (depressing the left mouse button), the name given that action by the documentation is often unique to the type of widget the action is to be performed on. For example, users are instructed to *click* on buttons, and *check* check boxes. We combined the go-verbs with more specific actions collected from the interface to build an ontology of actions, in which the hierarchy represents the the intuitive relationships between action types. This ontology is shown in Figure 3.

Action type similarity is calculated by indexing two action types into the ontology in Figure 3. One action type is extracted from the documentation (a go-verb), the other action type comes from the user's action on the interface. Indexing into the ontology returns a pointer to the corresponding node for that action type. The length of the shortest path between two nodes in the ontology can be easily computed. This length is calculated for all pairings of the current action and document actions and the resulting scores are normalized. Because similar actions are near each other in the ontology the normalized scores are complemented to obtain a similarity score.

Each similarity metric estimates the probability that the user action matches the given document action. The final similarity score for a particular pairing of user action U_k and document action D_i is the vector of values of $SimAlign(U_k, D_i)$ for all i. This generates a similarity score for each D_i for a given U_k. Next, the user action U_k is aligned with the document action D_m that has the highest similarity score. The system then instructs the Document Viewer to highlight the section of the documentation corresponding to the chosen document action D_m.

3.4 Document Viewer

The user interface to Macro Illustrator is a window termed the *Document Viewer*, that takes the place of the standard Eclipse on-line help browser. Through the interface provided by the Document Viewer, the user is able to load and navigate HTML documentation in a manner similar to that of the

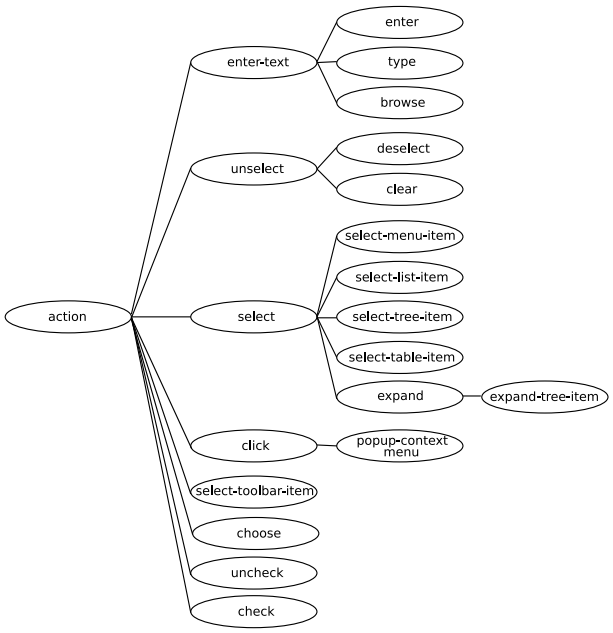

Figure 3: The action type ontology.

Eclipse help system. In addition to this standard functionality, the Document Viewer provides the visual ties between the user's actions and the static images and text of the procedure documentation. It is this visual, interactive guidance—provided in the context of the user's working environment—that sets the Macro Illustrator apart from other on-line help systems such as RoboHelp [15].

Figure 1 shows the Document Viewer after a user has recorded the first step of the selected procedure. The user has selected the "Preferences" menu item from the "Window" menu. As the user performs a procedure, the highlighted region follows along with each step the user takes, highlighting the relevant portions of the documentation and scrolling the document to keep pace with the user.

4. EVALUATION AND RESULTS

To evaluate the usefulness of the Macro Illustrator system, we have initially evaluated the Alignment Engine on a corpus of documented procedures for the Eclipse workbench. This evaluation provides a quantitative measure of the success of action alignment, and gives insights into the areas that need improvement before conducting an in-depth empirical evaluation. The SIMILARITY-BASED algorithm was compared with two other approaches described below.

Many documented procedures are designed to be read from start to finish, and performed in a linear sequence. Therefore, it seems reasonable that software procedures would have few branches, and a very simple algorithm would suffice for the alignment step of Macro Illustrator. One such algorithm simply aligns the first user action with the first document action, the second user action with the second document action, and so on. This algorithm, termed the PURE TEMPORAL alignment algorithm, generates a temporal mapping between the demonstrated procedure and the documentation. In this section we describe a performance comparison between the SIMILARITY-BASED algorithm and the PURE TEMPORAL algorithm on a test corpus. For a baseline comparison, we also tested a RANDOM alignment algorithm.

4.1 Data Collection

We randomly selected 50 documents from the Eclipse online help system. The documents selected met the following criteria:

- Documents must be instructional procedures. Many of the documents in the Eclipse on-line help contain only explanatory text or reference material (such as tables of shortcut keys).
- Our instrumentation must support all actions required to complete the procedure. Drag and drop and editing of file content are the most common features that our instrumentation does not support.

These documents describe simple procedures in Eclipse. Example procedures are: opening files, creating projects, creating CVS repositories and configuring the Eclipse environment. While these tasks may seem simple, they are representative of two key types of procedures: Essential tasks that may prove problematic to novice users, such as file management in Eclipse, and; Complex tasks that are performed rarely, such as configuring a development environment to make use of local version control systems or modifying the visual interface of your environment.

The algorithm in Section 3 includes two tunable parameters: the list of go-verbs and the creation of the action type ontology. Because of this, data collection was broken into two phases: initially a training set of procedures was collected, then, after tuning the algorithm a test set of procedures was collected.

The training set consisted of 18 documents, containing procedures that ranged in size from 3 to 12 steps, with an average length of 6.44 and standard deviation of 2.43.

The algorithm was tuned in the following way: To collect the list of go-verbs, the training procedures were read by one of the authors to collect the action verbs that pertained to actual actions on the interface. This was done manually to avoid the erroneous results that may have resulted from automating this process with a natural language parser. This set was also relatively small, and the actions found are consistent across many applications. The selected verbs were: *click, check, uncheck, select, deselect, clear, enter, choose, type, browse* and *expand*.

The second tunable parameter of the algorithm is the action type ontology discussed in Section 3.3. The structure and content of this ontology was based upon the experience of an expert (one of the authors) performing the procedures as described in the training set of documentation. The content of the ontology consists of the list of go-verbs, with some specialization of certain actions. For example, the go-verb *select* can be applied to various types of objects. Lists, tree objects, and menu items are all "selectable" objects. The information about which type of selection action occurred is recorded by the instrumentation and used when indexing into the ontology. After collecting the content for the ontology,

the structure was determined from the expert's background knowledge about interfaces and action types.

After the algorithm was tuned, a test set was used to evaluate our approach. This test set consisted of the remaining 32 procedures. To preserve validity, no algorithm modifications were made after viewing the test set. These 32 documents ranged in size from 2 to 12 steps, with an average length of 5.25 and standard deviation of 2.32. These procedures were used to test three algorithms, the SIMILARITY-BASED alignment algorithm, the PURE TEMPORAL alignment algorithm, and the RANDOM alignment algorithm.

4.2 Evaluation

The SIMILARITY-BASED alignment algorithm was evaluated by running it on the test set of 32 sample procedures and comparing its performance with that of the PURE TEMPORAL and RANDOM approaches. The RANDOM algorithm aligns each user action U_i to a document action D_r uniformly at random. The PURE TEMPORAL approach, as discussed above, aligns each user action U_i with the document action at the same index, D_i. Given that the Eclipse on-line help procedures are relatively short, and describe simple procedures, it seems reasonable that the PURE TEMPORAL algorithm should perform well. But whenever branching does occur, we expect the SIMILARITY-BASED algorithm to stay on track, keeping pace with the user's actual behavior when document actions are skipped. Whenever the user performs additional steps not described in the document, skips steps, or performs steps out of order, the PURE TEMPORAL algorithm will fail to maintain pace with the user.

To validate the algorithms, sample user input and oracles for each test procedure were created. Each procedure was recorded once, without feedback from Macro Illustrator. After recording a procedure, an oracle was created by manually inspecting the recorded data from the step performed, the text of the document, and the document annotations created by the Document Preprocessor. Each oracle consists of a mapping from user actions to document actions. Each user action is represented by its index in the demonstrated procedure. Similarly, each document action is represented by that action's index in the documented procedure. Due to discrepancies between the documentation and the interface, some user actions may have no corresponding document action. These are represented in the oracle by mapping to a document index -1. For example, the sequence of pairs: $(0,0)(1,0)(2,2)(3,3)(4,-1)$ is an oracle for a five step procedure. The first two demonstrated user actions both correspond to the first document action (this could occur due to obsolete documentation, or parsing errors). The third and fourth user actions map to the third and fourth document actions, and the last user action (U_4) is not represented by any of the steps extracted from the documentation.

As shown above, it is possible that multiple user actions match the same document action, in which case that document action is simply used multiple times. If the user actions are adjacent, the ideal behavior of the Document Viewer is to keep the user guide (the highlighting) stationary.

A drawback to all of the algorithms tested is that they do not allow for actions that are not described by the documentation (such as U_4 above). Therefore, these user actions are always aligned incorrectly.

	RANDOM	PURE TEMPORAL	SIMILARITY-BASED
Correctly aligned:			
Steps/Total	32/168 (19.05%)	90/168 (53.57%)	103/168 (61.31%)
Procedures/Total	0/32 (0.00%)	2/32 (6.25%)	6/32 (18.75%)

Table 1: The number of steps and complete procedures correctly aligned by each of the three algorithms evaluated.

Each algorithm was applied to the original documentation for each procedure along with the demonstrated user traces. The alignment algorithms were supplied with traces one action at a time to simulate the way the trace would be seen by the algorithms if a user were actively using the system. In this manner, every user action was aligned to a document action. The resulting alignments were then compared with the hand-generated oracles.

Each alignment was considered individually—a predicted alignment that would have yielded the desired user experience was considered to be correct. Recall from Section 3.1 that some document actions are represented by the same portion of the document. When this is the case, the correct behavior is to maintain the same highlighting for both steps. Consider two document actions D_a and D_b that are contained in the same sentence in the document. If the user performs the action described by D_a, and it is mistakenly aligned with action D_b, the visualization remains the same as if the action had been correctly aligned. It is also possible that the user's next action is also aligned with the document action D_b. Our evaluation is based on the user experience, allowing for a shifted alignment like the one described above *as long as the user experience is unchanged*. Consider the oracle given above: $(0,0)(1,0)(2,2)(3,3)(4,-1)$. If the document actions D_2 and D_3 are both described by the same sentence (such as "Check the 'print to file' checkbox and click Ok") then both of the following alignments are also valid: $(0,0)(1,0)(2,2)(3,2)(4,-1)$ and $(0,0)(1,0)(2,3)(3,3)(4,-1)$. Neither of these alignments match completely with the oracle, but all three alignments (the oracle and the generated alignments) generate the same user experience. Therefore, all three alignments are considered correct in our evaluation.

4.3 Results

Table 1 displays the number of steps and procedures accurately aligned by each of the three algorithms. Recall that a step is correctly aligned if the alignment causes the same visualization as the alignment stated in the oracle. A procedure is correctly aligned if and only if every step in the procedure is correctly aligned.

The RANDOM alignment algorithm failed to correctly align any procedures, as expected. The performance of RANDOM should come as no surprise. Since the average procedure length was just over five steps, and RANDOM uses a uni-

form distribution for alignment, roughly 20% accuracy is expected. The PURE TEMPORAL algorithm performed much better, classifying 53.57% of the steps correctly. However, only two of the 32 procedures were classified correctly. This indicates that the procedures are actually not predominantly linear procedures. Rather, nearly every procedure contained at least one optional step. Recall that whenever the user performs additional steps not described in the document, skips steps, or performs steps out of order, the PURE TEMPORAL algorithm will fail to maintain pace with the user. The SIMILARITY-BASED algorithm is more robust to these perturbations, and this is reflected in its performance.

The SIMILARITY-BASED algorithm outperformed both the RANDOM and PURE TEMPORAL algorithms both in correctly classified steps, and correctly classified procedures. The performance of the SIMILARITY-BASED algorithm is significantly better than both the RANDOM algorithm ($p = 2.8031 \times 10^{-6}$) and the PURE TEMPORAL algorithm ($p = 0.024776$)[3].

The SIMILARITY-BASED algorithm, with 61.31% accuracy made a number of errors, many of which may be preventable. We have identified the following four causes of error:

Unlabeled widgets: Some widgets in the Eclipse workbench do not have any descriptive text associated directly with the widget. Text fields are a prime example of this. Semantic labels are present in the interface; however, there is no syntactic tie between the text on the label and the widget that generates the user event that is recorded. Gaeremynck, et al., addressed a similar problem with MORE [7], a system for extracting the semantic relationships from HTML forms. However, designing a general-purpose algorithm to find the relationships between labels and widgets is still an open question.

Similar adjacent actions: The presence of adjacent document actions that are very similar, such as repeated instructions to "Click Next" frequently cause difficulty. This is most problematic when one or more of the actions are optional. It is not always possible for the similarity metrics to differentiate between actions that are similar. Incorporating contextual information, such as the full state of the interface, may mitigate this problem. Documented actions often include some contextual information, such as window titles, to help users disambiguate on-screen widgets. The same information could be used by a similarity metric.

Aggregate actions: Some steps in a procedure are concisely described, yet tediously executed. For example, the instruction: "Select the project(s) you wish to delete." is extracted as one document action, however, the user may need to select a large number of projects, generating a new user action with each selection. If the number of extra actions is large enough, the temporal similarity metric will dominate the similarity calculation resulting in an incorrect alignment.

[3]A single-tailed Wilcoxon test was used to test for signifi cance.

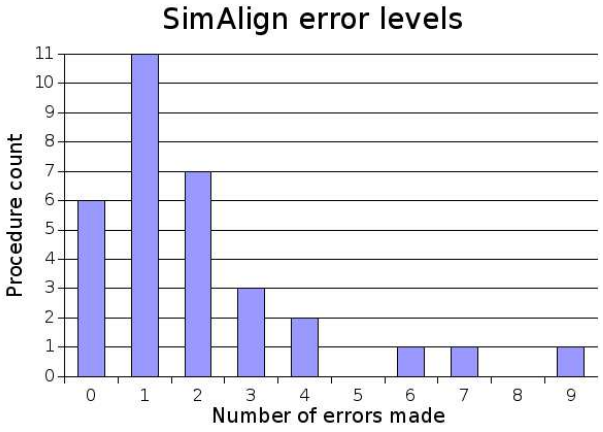

Figure 4: The number of procedures classified with a given number of errors.

Parsing difficulties: Many documents are written in a well-controlled manner, with consistent use of verbs and objects throughout the text. Some instructions, however, do not fit this pattern. The instructions: "Finish entering your search options, for example, to scope the search to specified working sets." translates to checking and unchecking a number of checkboxes, but there is no indication in the text of the type of action needed. Because of this, the Document Preprocessor did not extract an action for this step in the documentation.

Figure 4 shows the accuracy of the SIMILARITY-BASED algorithm in detail. The 32 procedures aligned have been divided into groups based on the number of errors made during the alignment. Six procedures were aligned correctly (0 errors), eleven procedures were aligned with one error, and so on. 24 of the 32 procedures (75%) were aligned with two or fewer errors, indicating that the number of correctly aligned procedures could be increased greatly by resolving some of the issues above. For example, incorporating a larger portion of the screen state when comparing words with the documentation may provide the description similarity metric with enough information to differentiate similar widgets. This would be of greatest benefit when aligning similar adjacent actions, and when dealing with text fields (a problematic widget because of the absence of labels).

5. CONCLUSIONS

This paper has presented a system for assisting users in performing software procedures, and in particular, an approach for integrating documentation with the procedure. This approach keeps track of a user's position in the documentation as he or she performs each task and communicates this information to the user through visual indicators in the document. The described approach does not require additional markup, and can illustrate procedures with existing documentation as they are performed. This novel use of legacy documentation provides users with visual assistance when performing complex procedures.

There are a number of directions for future work. We plan to evaluate the Macro Illustrator system with user studies

to compare our approach with standard training techniques, and to determine users' tolerance for error. The findings from these studies will drive the refinement of Macro Illustrator. Our evaluation of Macro Illustrator also brought to light four elements of documentation that proved to be difficult for our approach to handle. These difficulties present areas for improvement. Incorporation of the context of each user action into the similarity calculations may improve performance with unlabeled widgets and similar adjacent actions. The difficulties with aggregate actions are due to the temporal similarity metric; weighting the contribution of each similarity metric would reduce this issue. Finally, additional experience with natural language processing techniques is required to ease the problems encountered when extracting actions from documentation.

6. ACKNOWLEDGMENTS

This work has been funded in part by the EUSES Consortium via the National Science Foundation (ITR-0325273). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

1. Apple Computer Inc. *Apple Guide Complete: Designing and Developing Onscreen Assistance*. 1996.

2. Lawrence Bergman, Vittorio Castelli, Tessa Lau, and Daniel Oblinger. DocWizard: A system for authoring follow-me documentation wizards. In *UIST '05: Proceedings of the 18th annual ACM SIGGRAPH symposium on User interface software and technology*, Seattle, WA, September 2005. [in submission].

3. Margaret Burnett, J. William Atwood, Rebecca Djang, Herkimer Gottfried, James Reichwein, and Sherry Yang. Forms/3: A first-order visual language to explore the boundaries of the spreadsheet paradigm. *Journal of Functional Programming*, 11(2):155–206, March 2001.

4. Margaret Burnett, Andrei Sheretov, and Gregg Rothermel. Scaling up a 'What You See Is What You Test' methodology to spreadsheet grids. In *Proceedings of the IEEE Symposium on Visual Languages*, pages 30–37, Tokyo, Japan, September 1999.

5. David Eppstein, Zvi Galil, Raffaele Giancarlo, and Giuseppe Italiano. Efficient algorithms for sequence analysis. In *SEQS: Sequences '91*, 1991.

6. Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. In *Proceedings 1994 ACM SIGMOD Conference, Mineapolis, MN*, pages 419–429, 1994.

7. Yves Gaeremynck, Lawrence D. Bergman, and Tessa Lau. MORE for less: model recovery from visual interfaces for multi-device application design. In *Proceedings of the 8th International Conference on Intelligent User Interfaces*, pages 69–76. ACM Press, 2003.

8. Eric Horvitz, Jack Breese, David Heckerman, David Hovel, and Koos Rommelse. The Lumière project: Bayesian user modeling for inferring the goals and needs of software users. In *In Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 256–265, Madison, WI, July 1998.

9. IBM. *The Eclipse Platform Plug-in Developer Guide*. http://help.eclipse.org/help30/index.jsp, Last accessed: November 30, 2004.

10. IBM. The Eclipse Workbench. http://www.eclipse.org, Last accessed: December 6, 2004.

11. H. V. Jagadish, Alberto O. Mendelzon, and Tova Milo. Similarity-based queries. In *Proceedings of the 14th Symposium on Principles of Database Systems*, pages 36–45, 1995.

12. Jong Y. Kim and John Shawe-Taylor. Fast string matching using an n-gram algorithm. *Software — Practice and Experience*, 24(1):79–88, 1994.

13. Andrew J. Ko and Brad A. Myers. Designing the whyline: a debugging interface for asking questions about program behavior. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 151–158, New York, NY, USA, 2004. ACM Press.

14. Tessa Lau, Lawrence Bergman, Vittorio Castelli, and Daniel Oblinger. Sheepdog: learning procedures for technical support. In *Proceedings of the 9th International Conference on Intelligent User Interface*, pages 109–116. ACM Press, 2004.

15. Macromedia. Robohelp. http://www.macromedia.com/software/robohelp/ last accessed: April 1, 2005.

16. Michael C. McCord. Slot grammars. *Computational Linguistics*, 6(1):31–43, 1980.

17. Michael C. McCord. Slot grammar: A system for simpler construction of practical natural language grammars. In *Proceedings of the International Symposium on Natural Language and Logic*, pages 118–145. Springer-Verlag, 1990.

18. Piyawadee Sukaviriya. Dynamic construction of animated help from application context. In *UIST '88: Proceedings of the 1st annual ACM SIGGRAPH symposium on User Interface Software*, pages 190–202. ACM Press, 1988.

19. Piyawadee Sukaviriya and James D. Foley. Coupling a ui framework with automatic generation of context-sensitive animated help. In *UIST '90: Proceedings of the 3rd annual ACM SIGGRAPH symposium on User interface software and technology*, pages 152–166. ACM Press, 1990.

20. Piyawadee N. Sukaviriya, Jeyakumar Muthukumarasamy, Anton Spaans, and Hans J. J. de Graaff. Automatic generation of textual, audio, and animated help in uide: the user interface design. In *AVI '94: Proceedings of the workshop on Advanced visual interfaces*, pages 44–52, New York, NY, USA, 1994. ACM Press.

21. Aaron Wilson, Margaret Burnett, Laura Beckwith, Orion Granatir, Ledah Casburn, Curtis Cook, Michael Durham, and Gregg Rothermel. Harnessing curiosity to increase correctness in end-user programming. In *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 305–312, Fort Lauderdale, FL, April 2003.