AIMHelp: Generating Help for GUI Applications Automatically

Yashasvi Appilla Chakravarthi, Christof Lutteroth, Gerald Weber
Department of Computer Science
The University of Auckland
38 Princes Street
Auckland 1020, New Zealand
yapp001@aucklanduni.ac.nz, {lutteroth, g.weber}@cs.auckland.ac.nz

ABSTRACT

Help systems are a requirement in most modern applications. However, current mainstream help systems can be improved to provide information that is more relevant and accurate. This paper introduces a new approach for help systems – AIMHelp – that can generate help information from a running application. Instead of developers having to supply all help information a-priori, a lot of information can be retrieved by monitoring the state of the GUI and the interaction between the user and different system components. This has the advantage that generated help information is consistent with the actual application, unlike pre-defined help information that can easily get outdated as an application evolves.

Categories and Subject Descriptors

H.5.2 [**Information interfaces and presentation (e.g. HCI): User Interfaces**]: GUI

Keywords

Automated help, context-sensitive help, dynamic help, reflection, explanation strategies

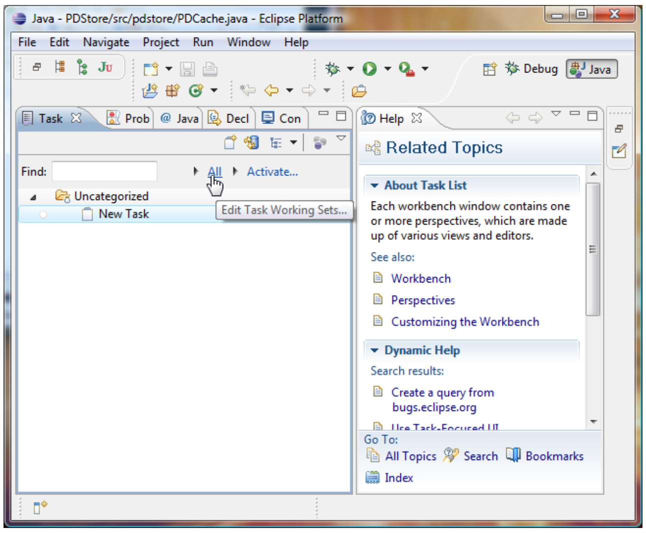
1. INTRODUCTION

Help systems are a requirement for state-of-the-art software applications. However, we assume general agreement that extant help systems can be improved.

In this paper we focus on GUI-based applications. There are several reasonably widespread concepts of dynamic help. We consider dynamic help to cover every help system that is more than a static user manual, e.g. help systems that change with time or context. Applications often use tool-tips to provide verbal descriptions of a specific widget or component in the software. A quick check of major office suites on different major platforms shows that these tool-tips regularly do not provide the user with an access to the help system or the information required within the help system. Context-sensitive help is a feature which typically provides information about a specific feature of the application. It is not offered any more by several major office suites, although this was a novelty in help systems some years ago. In the Eclipse platform, the F1 function key can be used to obtain a context sensitive help for the current focus, see Figure 1. Help indexes are a common part of software applications as well. These indexes are typically offering alphabetical, hierarchical and search-based access.

In all current help systems, creating the help documentation is a manual task. As a consequence, completeness of help systems is not guaranteed, not monitored and not maintained. We present a system that delivers certain guarantees of availability of help. We report on an ongoing project, and we present the proposed final system as well as the current state of the implementation. Methodologically, the contributions are first a theoretical definition of two different notions of guaranteed help. This theoretical notion is positioned on the abstraction level that is typically referred to as analysis or specification in various software methodologies. The second contribution is a design for both notions of guaranteed help. This design is platform independent and we present it in the context of the Auckland User Interface model. Thirdly we report on the current prototypical implementation on the open source operating system Haiku, and further implementation steps.

For purposes of the presentation of our approach, we assume two use cases for the help systems in a software product. The first use case is to find out how to perform a particular task if the users know what they want to do with the system. The second use case is to find out what a component that the user stumbles upon is doing. With our proposed help system, we address both use cases and ensure that the help system provides relevant information to the users. Our theoretical notions of guaranteed help contribute to the understanding of service orientation of software. Service orientation is often discussed in a way that regards the actual users of a system only very indirectly as stakeholders of the service oriented architecture, if they are considered at all [7]. In our concept of guaranteed help, it becomes clear that end-user needs should be considered more prominently in service definitions, and this has direct consequences for the usability of a system.



Each workbench window contains one or more perspectives, which are made up of various views and editors.

Figure 1: An example of the context-sensitive help in the Eclipse platform.

In Section 2, requirements for help systems as they are given in the literature are reviewed. Section 3 gives a comprehensive overview of help systems research. Section 4 introduces our novel approach to dynamic help, and Section 5 gives a brief overview of the technologies that it is built on. Section 6 presents a prototype implementation of the AIMHelp system, and Section 7 discusses advantages and challenges of our approach. Section 8 concludes the paper.

2. HELP SYSTEM REQUIREMENTS

Previous research has identified a number of criteria that a help system should meet. Some of the qualities that the help system should provide are consistency, navigability, completeness, relevance, conciseness, coherence, fidelity and reuse [2].

The help system needs to be able to provide information to the users in the simplest manner. Thus, the users will be able to follow the information quickly and understand what it implies. There are a number of explanation strategies that a help system can use to converse with the users [4]. The help system can display the information as animations or videos which provide the users with a visual demonstration of the information. Another explanation strategy is to display the information in a textual format, where each instruction is displayed as a set of rules. A third strategy is to provide a dialog based instruction set, where the user can interact with the dialogs to navigate through the help system. In these strategies, the help system needs to be able to convert a set of rules into human readable language. Examples are troubleshooting wizards, e.g. for network connections.

The help system also needs to provide information that is relevant to the user. Thus, the system needs to keep track of the state of the application, the user's actions and the history of other inputs such as number of errors, frequency of command usage etc. [15, 3]. Based on the collected data, the system needs to be able to analyze what the user wants to do and provide relevant information to the user. Thus, the help systems need to meet these criteria in order to provide efficient help to the users of a software application.

3. STATE OF THE ART HELP SYSTEMS

There has been research on a number of dynamic help system designs which have been implemented as prototypes. Each of these designs has a unique underlying concept. These state-of-the-art designs only address the second use case for a help system as described in 1. This section discusses several of these designs and analyzes the advantages and disadvantages of each approach.

3.1 The Two Module Approach

HelpTalk is a dynamic help generator in UIDE (User Interface Design Environment) [15]. It clearly separates the help access and help generation as two different mechanisms. UIDE separates its application into two separate models - the application and the interface models. The application model can have a number of interface models linked to it. Each of these models consists of their respective actions and objects. These actions and objects are linked to one another through action-object mappings. These models are represented by their respective blackboards during the runtime of the UIDE. It also consists of a User Interface Controller (UIC) which processes the user's interactions, invokes the application action and displays dialogues in order. HelpTalk has access to the UIDE's entire knowledge base and produces its comments using the state that the blackboards are in. Once the help is generated, it displays an animation of how to perform the specific task. Using the knowledge base, HelpTalk generates the textual help by explaining the reasons why the interface is in that particular state. The reasons are phrased based on the models present in the UIDE's knowledge base. Then, it generates an animation that displays the procedures required to complete the textual help displayed.

This approach has a number of advantages. One of the advantages is that the separation of the models i.e. the application and the interface models allows the distinction between the different actions and objects and the mappings between them. Another advantage of this approach is that it enables the help system to display the instructions from the knowledge base in textual or animated format. However, one of the drawbacks to this approach is that all the actions need to be divided between the application model and the interface model. This is both time-consuming as well as redundant in certain applications. Another example of a framework that uses the two module approach is the Crystal framework which is similar to the HelpTalk framework [3].

3.2 The Snippets Approach

CogentHelp is a prototype tool that generates dynamic help for user interfaces built using Java Abstract Windowing Toolkit [2]. It uses a different mechanism for creating help documents using snippets and servers. CogentHelp accepts a set of 'human-written' snippets as an input and attaches these snippets to their respective widgets in the user interface. When the help information for a specific widget is generated, the snippets for the specific widget are linked together. The help system consists of different views including a thumbnail and a tree view. The help topics are delivered through an

HTTP server via a Java Servlet API. When a help request is made, the help server is loaded through a particular URL. This URL encodes the current state of the system, some parameters and attributes as well as the 'snippets' for the corresponding widgets. The generated HTML forms can be used by the program authors to edit snippets using 'frames' which allow the authors to add in missing keywords in the snippets. The help authors generate exemplars which are frameworks that support different text generation methodologies. These exemplars follow object-oriented design and can inherit from one another. They have been implemented in the system to make use of Java's object-oriented design and to help the authors. Once the exemplars are prepared, the help documents are generated by the server.

One of the potential advantages of CogentHelp is the use of 'snippets' or phrases for each widget. This allows the dynamic help system to create help information based on the user's actions and the state of the system. Moreover, this approach also allows the use of different text generation methodologies making the system more flexible. A limitation of this approach is that it does not account for the user's experience level and needs to be used with an efficient text generator.

3.3 Triggering the required action

SmartAide is another help system which takes the automated help generation to the next level [12]. Apart from displaying step-by-step textual instructions to the user, it also executes the action sequence of the task being described and changes the state of the interface. When a user requests help, an AI planning system generates an action sequence which is designed to execute within the user's workspace. It changes the state of the user interface and completes the task that the user is presumed to work on.

This approach has a number of advantages over the other help generation methods. First, the action sequences are executed by changing the state of the system and the help authors do not have to worry about the intermediate states. Also, the action sequences that are executed can be tailored to suit the user's preferences. There are a few drawbacks with this concept. First, all the action sequences are executed even when the user has no intention of performing them. Second, the help system does not provide the user with a learning experience as it executes all the necessary tasks required. Thus, the user does not learn the steps required to perform a specific task.

3.4 Animated Help

In the earlier versions of dynamic help systems, the help instructions were prompted in a textual format. However, most of the modern software applications use graphical interfaces. Thus, textual help does not relay the information needed to navigate through a graphical interface accurately.

The new versions of dynamic help systems introduced a new mechanism where the help instructions were animated. The instructions were listed as a sequence of animated steps, where each animation displayed the process of executing a particular task.

3.4.1 DirectoryTree

Directory Tree is a simple application in UNIX which performs some basic functions with directories and their structures [14, 13]. It consists of a dynamic help system which is based on the animated help mechanism. Preconditions and preferences are set in the DirectoryTree application [6]. These preconditions are attached to a set of animated instructions. When these preconditions and preferences are triggered, the corresponding set of animation instructions is generated. These help scenarios are sequences of actions that need to be performed. The animated help system uses an application context tool to process specific application environments. Once the environment variables are set, the system uses application semantics to understand more about the application mechanism.

This approach has a number of benefits to the user. First, it displays the help scenarios as animated instructions. These instructions are much better at describing graphical interfaces than the textual equivalent of the help system. Second, the help scenarios that are demonstrated are relevant to the user's requirements as a result of the application of the preconditions and preferences in the DirectoryTree program. However, one of the major drawbacks to this approach is the need to store a library of animated videos for each specific scenario. This is not possible for all the help systems implemented in various applications. The second drawback is that different scenarios are created each time the user needs some help. This can lead to confusion among the users.

3.4.2 EdgeWrite

Another application that uses animated help is the EdgeWrite software application [11]. It is a program specifically designed for the disabled to learn how to write characters within a bounded box. The characters are represented by movements to the different corners of a bounded box. The common form of input to the program is the Logitech game pad. EdgeWrite allows the users to be able to write characters by moving a cursor to different corners of the bounded box in the application. Each character is represented by a unique sequence of movements to a corner and hence, helps the user learn how a character is drawn. The dynamic help system in EdgeWrite analyzes the movements of the user and displays the corner that the user needs to follow in order to draw a specific character. It displays an animation that indicates the direction to the corner for a specific character. Thus, the help system helps the user follow the animation and learn the path of the characters quickly.

An evaluation was conducted which compared the results of static help to the animated dynamic help in EdgeWrite [11]. The static help only provided a list of characters at each corresponding corner. 24 students between the age of 20 and 26 participated in the evaluation. The participants were asked to use the static help system first, and then the dynamic help system. The results showed that there was an increase in the number of words entered per minute with the dynamic help. Also, the error rate, effort and entry time were smaller for the dynamic help system as compared to the static help system. These evaluation results showed that the dynamic help system was much more useful to the user and helped them learn the system more efficiently than the static help system.

3.5 The User Experience Level Approach

Most of the dynamic help systems generated a list of instructions based on the history of the user's actions as well as the state of the system. However, they did not take the user's experience level into account. Thus, the same set of instructions was displayed to both novice as well as expert users.

HelpDesk uses a dialog-based help system which takes the experience level of the user into account [10]. It identifies the user's experience level and provides a set of instructions (goals and sub-goals) based on the experience level. The HelpDesk system architecture consists of a number of modules which are used to analyze the user's input, calculate their experience level and generate the necessary help instructions. When the user provides an input through the mouse or the keyboard, the 'Level Adjusting Agent' module calculates the current experience level of the user and adjusts the user level. The experience level is calculated based on some of the input parameters such as the time taken to perform a specific task, the complexity of the task that was completed and the number of steps taken to reach the goal. Once the experience level of the user is calculated, the 'Utility Updating Agent' and the 'Action Planner' track the user's actions and analyze the intended action by the user. Finally, the 'Content Selection' module divides the action into a number of goals and sub-goals based on the user's calculated experience level. The 'Content Realization' module provides a set of instructions based on the goals and sub-goals that were chosen by the previous modules.

The HelpDesk system is advantageous as it provides a list of instructions based on the user level. Thus, the appropriate set of instructions is presented to the user and makes the application more convenient to use. However, this approach does not display the entire information available to all the users. Thus, some of the information is hidden from the expert users. This is not beneficial to the users as the help system can hide information after placing a specific user in the wrong experience level.

4. AIMHELP - A NOVEL HELP SYSTEM

The previously presented help systems provide a variety of new features. However, they do not guarantee completeness of the help system.

In this section we define two notions of guaranteed help, and we specify features of a new help system that achieves both notions, called AIMHelp (Auckland Interface Manager Help). In the latter sections we report on the design and implementation of AIMHelp on top of the AIM (Auckland Interface Manager). The current implementation project is based on the AIM branch for the Haiku Operating System. Thus, a help system is available for every application without manual provision of the help information.

The notions of guaranteed help address firstly the requirement of completeness, but this requirement is of course a crosscutting requirement that works together with requirements such as navigability. If we accept navigability as a requirement for us, then completeness means that navigability must be provided everywhere.

The first notion of guaranteed help is called *guaranteed help listing.* It requires that every user interface element has to be listed in the help index with its textual name. The second notion of guaranteed help is called *guaranteed help content.* This means that a specified class of effects should be fully documented under those features that trigger these effects.

For guaranteed help listing we use the fact that AIM follows the document-oriented approach. This means, that all user interface elements in AIM are specified in documents. The AIMHelp system traverses all user interface documents and enters all the interface elements into the help index.

For the second notion of guaranteed help content, we have to make assumptions about the way the functionality is provided within the application program. As indicated earlier, this feature will require a very clear system design as envisaged in approaches such as service-oriented architectures. The Haiku operating system, one of the platforms that AIM is provided on, is service-oriented. Therefore we have chosen it for the initial implementation of AIMHelp. In the 'Causes' feature below we will discuss guaranteed help content for an email function as an example. The process in which the service-oriented structure of the Haiku system can be exploited for guaranteed help content will be discussed in the next section.

In general, the degree of guaranteed help content is as much experimental as the vision of service orientation is currently still experimental. However, the guarantee claim of this second notion is not with respect to all things that could possibly be said about a certain feature, but with respect to that what can be obtained by an automatic program.

The AIMHelp system consists of a number of features such as an automated help index, an event log, analysis of causes and analysis of effects. These features will be explained in the following.

4.1 Automated Help Index

The automated help index realizes guaranteed help and provides the user with an index containing all widgets in the user interface, as well as all types of events that can be created in this application. If several widgets have the same name, a disambiguation is performed based on different access paths. For example in a PDF viewer, there might be two widgets called "two-page" one in the print dialogue, actually triggering a two-paged print, and in the View menu, toggling a presentation of two pages at a time.

For all user interface elements, shortcuts from the help system to the user interface element can be provided. The help index gives access to all help information available for a certain keyword. The different types of help information that are generated by AIMHelp are listed as separate features below. The concept of context-sensitive help is firmly built into AIMHelp and is just a function of the Automated Help Index, providing access from each element to the help entry. The information for each keyword can be augmented with manual information as well.

4.2 Event Log
The event log allows the user to browse through the history of past events that have already been generated. It also provides the user with information about the event such as the timestamp, the widget causing the information and the state of the system. It is a user-based feature and allows the user to only access the past events of the same user. This feature also allows the user to replay the event in order to view what has happened (the Triggering the Required Action approach).

4.3 Causes
The 'Causes' feature displays the information related to the cause of a specific event. This feature realizes guaranteed help content, and it is well defined for every distinct service in the system. As an example we consider a case where the user wants to send a document by email. Users expect office applications to be able to send documents directly as emails. In AIMHelp, every system service such as email is listed in the help system. If the user goes to the AIMHelp index of the document editor, the user will find an entry on email. If the user goes to this entry and refers to the 'Causes' feature there, the user will find a list of all user interface elements in the document editor that trigger an email to be sent. This will be a toolbar-button, a menu entry in the File menu, and a hot-key. This help information is generated completely automatically in AIMHelp. The user can choose to either invoke these features outside of the help dialogue, or to use the direct link in the help function.

4.4 Effects
The 'Effects' feature displays the information related to the effect of a specific selected widget does. In AIMHelp, it is guaranteed that the help entry for the email function states that the email service will be called.

5. TECHNOLOGIES BEHIND AIMHELP
AIMHelp is based on a number of technologies such as reflection, message-based interaction and the Auckland Interface Manager. These technologies are described in the following.

5.1 Reflection
Reflection allows a program to observe and/or modify itself during run-time by using meta-computations and metadata [5]. Reflection as a concept has spread from the programming field to the area of user interfaces [9]. On the level of the user interface, reflection can be used to provide insight into an application or customize it. Based on the type of runtime mechanism, there are two common types of reflection: structural and behavioral reflection.

In structural reflection, programs can access their own structure during runtime, e.g. the structure of their code and their data. On the level of the user interface, this can enable users to adapt an applications data structures and functions to their own needs. Behavioral reflection enables access to behavior of a program and its runtime environment during runtime. A program can read information about its current state and modify its own behavior. In the user interface, this can be used to gain insight into the program behavior or adjust the way the system interacts with the user. Both types of reflection provide useful mechanisms for adapting programs during runtime, and providing a dynamic user interface.

Generally, reflection operations fall into the following two categories: introspection and intercession. Introspection allows a program to read information about a specific agent or itself based on the type of reflection that takes place. For example, structural introspection allows the user interface to present application data along with their data structure, while behavioral introspection allows the user interface to acquire information on how the system interacts with the user. Intercession allows a program to modify a specific aspect of it, thereby allowing the user interface access to either the structure or the behavior of the program depending on the type of reflection. Especially introspection can be a powerful tool for dynamic help systems as it enables a system to retrieve help information directly from a program.

5.2 Haiku
Haiku [8] is an open-source operating system with a strong focus on end-user desktop computing. Due to its modular, message-based design it lends itself particularly well to reflection on the level of the user interface. Many of Haiku's functions are implemented as servers that communicate with application through messages. For example, the input server manages device drivers for input devices and passes input events to programs that need to process them. The application server provides windowing system functionality: it manages application windows on the screen and passes input events from the input server down to the right application. Applications, in turn, send drawing commands to the application server, which updates the screen content of the application windows accordingly. These drawing commands are typically sent by the widget toolkit used by the application to draw the widgets of a GUI. Other servers offer functionality for network access, printing, storage and media playback and processing.

Messages are not only used for the communication between an application and the various servers, but also for the communication within the application itself. Widgets do not directly invoke application callbacks when they are activated, but send command messages to the application. Each window has a message loop that processes the command messages send by its widgets and invokes the application logic. This makes it possible to "script" any GUI, i.e. to generate command messages automatically to control and application. It also makes it possible to analyze the events happening within an application during runtime, without changing it.

Messages can be intercepted and analyzed. They are clearly structured, and their structure can be introspected. It is easily possible to capture all the messages generated by a GUI. Through the pervasive use of messages in Haiku, Haiku can be said to be a highly service-oriented system. Consequently, Haiku provides an ideal environment for AIMHelp to reflect on program behavior during runtime and generate pertinent help instructions for a given context.

5.3 AIM - Auckland Interface Manager
The AIM is a platform independent library for GUI specification and layout. It manages the elements of a GUI using

cross-platform high-level specification languages. AIM provides functions for loading, saving, transforming and editing GUIs, but it is up to the GUI developer in how far this functionality is used. Through the layout management functionality, AIM has access to the widgets of an application. This means that it can observe and change the state of an application during runtime, i.e. perform structural introspection on the data represented in the GUI. This makes AIM an ideal place for a dynamic help system. AIMHelp is implemented as part of AIM, which means that it can leverage structural (through the widgets) as well as behavioral (through the messages) reflection to generate help information. As a part of AIM, AIMHelp can provide automated dynamic help to many applications.

6. AIMHELP IMPLEMENTATION

The AIMHelp system is used with the Auckland Interface Manager and is implemented on the Haiku operating system. Figure 2 gives an overview of the AIMHelp system architecture. In Haiku, when the user interacts with the widgets in an interface, input messages are sent from the application server to the application. The widgets in the application receive and process the input messages, and send their own messages in response to the underlying application. These application messages trigger the application logic. In the Haiku OS, arbitrary information can be passed through the messages. However, typical message based systems at least provide information such as the message target, the widget sending the message, the type of the message, a timestamp and more. Changes in the application state, in turn, usually trigger changes in the GUI. This means that the application sends drawing commands back to the application server.

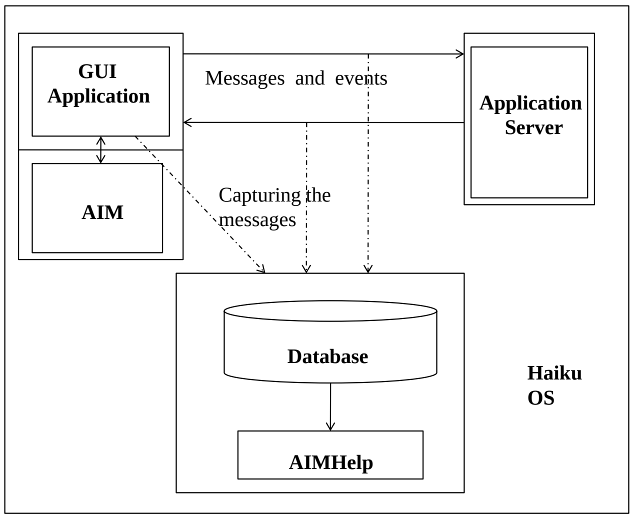
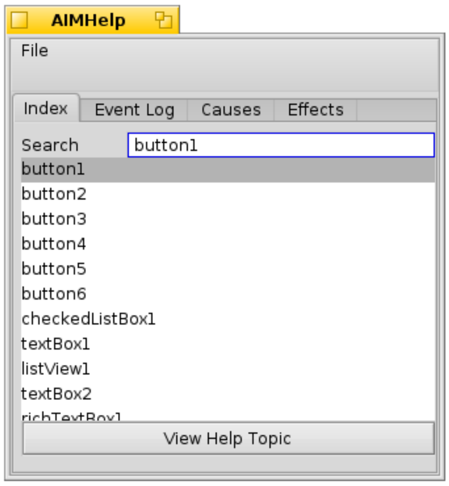


Figure 2: The system architecture of AIMHelp.

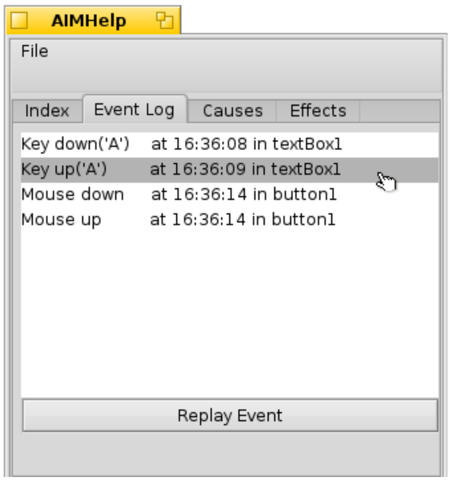
AIMHelp can capture all these messages. This is done by attaching message filters to a window or view. As part of AIM, AIMHelp has access to the widgets of an application and can attach message filters to them. Once a message is caught, the information in the messages is stored in a central database. This includes the sender of the message, the target, the current state of the system and the previous state of the system, as retrieved from the widgets that are accessible through AIM. These messages are stored throughout the operational lifecycle of the AIMHelp system across multiple sessions, so a historical message warehouse is built over time that can be used to analyze the message flow caused by an application. The storage of messages is easily done in Haiku as messages are inherently serializable.

The AIMHelp system is in the development stage and we still need to explore further the possibilities of message analysis in the Haiku OS. Furthermore, we are exploring how the message data can be usefully presented to the end-user.



```
button1
button2
button3
button4
button5
button6
checkedListBox1
textBox1
listView1
textBox2
richTextBox1
```

Figure 3: The automated help index tab of AIMHelp.

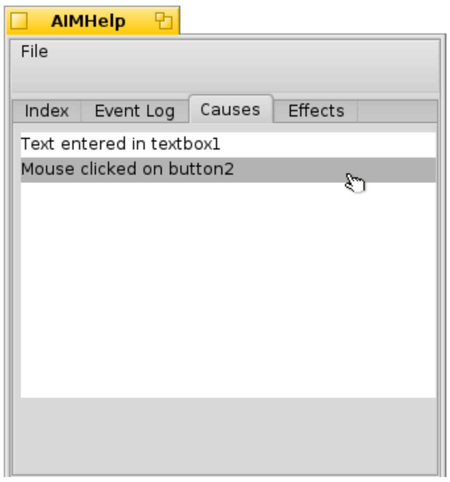


```
Key down('A')   at 16:36:08 in textBox1
Key up('A')     at 16:36:09 in textBox1
Mouse down      at 16:36:14 in button1
Mouse up        at 16:36:14 in button1
```

Figure 4: The event log tab of AIMHelp.

Figures 3, 4 and 5 show the AIMHelp window, which is structured in four tabs that reflect its main features. Figure 3 shows the automated help index, which contains the names of all the widgets in the test application. These names are extracted automatically.

Figure 4 shows AIMHelp's event log. It contains a list of recent messages (in this case keyboard and mouse events) that were caused by the user. The user can select events and replay them by clicking the button below the list of events. Replaying of events is possible because each event



Text entered in textbox1
Mouse clicked on button2

Figure 5: The cause analysis tab of AIMHelp.

entry contains all the necessary information, such as the message target.

Figure 5 shows the tab displaying the results of a simple cause analysis. Once a widget has been selected from the index, this tab lists all messages that, over the recorded history of the system, are frequently observed directly before the selected widget is changed. Since the system distinguishes only a limited number of event types, it is possible to generate a simple textual description for each event type, e.g. "Mouse clicked on widget X".

7. DISCUSSION

In AIMHelp, the notion of guaranteed help listing can already be fully achieved just by virtue of using the underlying technology AIM. The full implementation of guaranteed help listing alone offers already several highly desirable help features. The integration into AIM means that every application with a user interface based on AIM has a help index that fulfills one reasonable completeness criterion, has context sensitive help everywhere, and supports direct shortcuts from the help system to the system functions.

The notion of guaranteed help content is more challenging, because the notion of help content is an informal one. We analyze three major subproblems. First, how to obtain a reasonably complete lists of functionality, secondly how to make obtained information human readable, thirdly how to single out functionalities that can be transformed into end-user-readable information. For the first subproblem we use a central database of system usages, and since this database is considered to include the usage of the system during the testing phase, we assume complete coverage of the system. Hence our solution for that problem is at least as good as the coverage approach chosen for the testing phase. If this goes not through as a solution of the subproblem, then it is at least a reduction of the subproblem to a different known problem in testing which is usually regarded as a rather solvable problem. For the second problem we currently focus on creating just information lists, by doing this at places where the user would not expect more than such a list anyway. One example is the complete index, another example is the list of user interface elements that can trigger a certain action. The user might want to expect other information besides that list, but under the heading "list of widgets that allow the sending of email" the user only expects a list. In later versions we consider the use of structured English generators. For the third subproblem there are feasible solutions in our implementation project based on the AIM branch for the Haiku Operating System we can create a green list of services that are defined by the system and which have a human readable explanation.

To provide users with a larger collection of messages from which help information can be generated, a central online database can be used. Such a database can collect the history of all the messages and events that have been triggered by many users. There are privacy issues when collecting user session data in such a database, which can be addressed with statistical database techniques [1]. Nevertheless, not all users will want to provide such data, hence participation in such a database must be voluntary. This is work in progress.

In our presentation we use three different concept pairs, each of them with a slightly different focus. Firstly, we consider two use cases of help, namely the how-to-do vs. the what-does use case. Then we have defined two notions of guaranteed help, the guaranteed help listing vs. the guaranteed help content notion. Finally we employ two different design and implementation strategies, namely analysis of document-oriented GUI specification vs. data mining of service-oriented message interaction. These two strategies are not mutually exclusive, but can and will be combined in the final systems.

The three concepts are each well motivated and foster the understanding of the desired features and the feasibility of their implementation. That they are not mutually identical, but overlapping, is at first glance a possible reason for critique, but we consider it as a mere fact of our subject of research, and hence discussing all three concept pairs helps to understand a very real aspect of complexity of the underlying problem. The two use cases do not match with the two guarantee notions because both use cases profit to some extent from both guarantee notions. Equally both use cases can profit from both implementation strategies. Finally the guarantee notions and the implementation strategies have an overlap; for example, without the data mining component, the guaranteed help listing would only cover the user interface elements and the access to the functionality from the help entry, but many functions that the application can actually invoke would be missing. There can be functions of the system which are not directly represented by widgets, and they are entered into the help index by the data mining component.

8. CONCLUSION

In order to deal with the increasing complexity of the software systems, dynamic help systems have been defined to display help information to the user based on their actions and the state of the system. However, most of the dynamic help systems use a pre-defined list of data in order to generate the information for the users. AIMHelp is a dynamic help system that is different from other dynamic help systems in that it focuses on automatic content generation and aims at guaranteeing the availability of help content. It is integrated into Auckland Interface Model (AIM), and the current implementation project is based on the AIM branch for the Haiku Operating System.

Messages are generated when the user interacts with the widgets in the Haiku OS. The AIMHelp catches these messages and stores the data in a central database. When the user requests help information, it analyzes the messages from the database and displays information about causes and effects of messages. In conclusion, AIMHelp is different from the other dynamic help systems as it can generate help information automatically.

9. REFERENCES

[1] N. R. Adam and J. C. Worthmann. Security-control methods for statistical databases: a comparative study. *ACM Comput. Surv.*, 21(4):515–556, 1989.

[2] D. Caldwell and M. White. Cogenthelp: a tool for authoring dynamically generated help for java guis. In *SIGDOC'97: Proceedings of the 15th annual international conference on Computer documentation*, pages 17–22, New York, USA, 1997. ACM.

[3] D. Chau, A. Ko, B. Myers, and D. Weitzman. Answering why and why not questions in user interfaces. In *CHI'06: Proceedings of the SIGCHI conference of human factors in computing systems.* ACM, 2006.

[4] R. Cullingford, M. Rueger, M. Selfridge, and M. Bienkowski. Automated explanations as a component of a computer-aided design system. In *IEEE'82: IEEE Transactions on Systems, Man and Cybernetics*, volume 12, pages 168–181, 1982.

[5] F. Demers and J. Malenfant. Reflection in logic, functional and object-oriented programming: a short comparative study. In *Proceedings of the IJCAI'95 Workshop on Reflection and Metalevel Architectures and their Applications in AI*, 1995.

[6] J. Foley and D. Gieskens. Controlling user interface objects through pre- and post conditions. In *SIGCHI'92: Proceedings of the SIGCHI conference on Human factors in computing systems*, Monterey, California, 1992. ACM.

[7] N. Gold, C. Knight, A. Mohan, and M. Munro. Understanding service-oriented software. *IEEE software*, 21(2):71–77, 2004.

[8] Haiku Inc. The Haiku Operating System, 2008. http://www.haiku-os.org/.

[9] C. Lutteroth and G. Weber. Reflection as a principle for better usability. In *ASWEC 2007: Proceedings of the 18th Australian Software Engineering Conference.* IEEE Press, 2007.

[10] P. Maloor and J. Chai. Dynamic user level and utility measurement for adaptive dialog in a help-desk system. In *SIGDIAL'00: Proceedings of the 1st SIGDial Workshop on Discourse and Dialogue*, pages 94–101, 2000.

[11] B. Martin and P. Isokoski. Edgewrite with integrated corner sequence help. In *SIGCHI'08: Proceeding of the 26th annual SIGCHI confenrence on Human factors in computing system*, pages 583–592, 1990.

[12] A. Ramachandran and R. Young. Providing intelligent help across applications in dynamic user and environment contexts. In *IUI'05: Proceedings of the 10th international conference on Intelligent User Interfaces.* ACM, 2005.

[13] P. Sukavirirya and J. Foley. Coupling a ui framework with automatic generation of context-sensitive animated help. In *SIGGRAPH'90: Proceedings of the 3rd annual ACM SIGGRAPH symposium on User Interface Software and Technology*, pages 152–166, 1990.

[14] P. Sukaviriya. Dynamic contruction of animated help from application context. In *SIGGRAPH'88: Proceedings of the 1st annual ACM SIGGRAPH symposium on User Interface software*, pages 190–202, 1988.

[15] P. Sukaviriya, J. Muthukumarasamy, A. Spaans, and H. 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. ACM, 1994.