MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

DiamondHelp: A Graphical User Interface Framework for Human-Computer Collaboration

Rich, C.; Sidner, C.; Lesh, N.; Garland, A.; Booth, S.; Chimani, M.

TR2004-114 September 2004

Abstract

DiamondHelp is a reusable Java framework for building graphical user interfaces based on the collaborative paradigm of human-computer interaction. DiamondHelṕs graphical design combines a generic conversational interface, adapted from online chat programs, with an application-specific direct manipulation interface. DiamondHelp provides ä things to sayẗechanism for use without spoken language understanding; it also supports extensions to take advantage of speech technology. DiamondHelṕs software architecture factors all application-specific content into two modular plug-ins, one of which includes Collagen and a task model.

International Workshop on Smart Appliances and Wearable Computing, IEEE International Conference on Distributed Computing Systems Workshops

DiamondHelp: A Graphical User Interface Framework for Human-Computer Collaboration

Charles Rich, Candy Sidner, Neal Lesh, Andrew Garland, Shane Booth, Markus Chimani
Mitsubishi Electric Research Laboratories
201 Broadway
Cambridge, MA, 02139, USA
rich@merl.com

ABSTRACT

DiamondHelp is a reusable Java framework for building graphical user interfaces based on the collaborative paradigm of human-computer interaction. DiamondHelp's graphical design combines a generic conversational interface, adapted from online chat programs, with an application-specific direct manipulation interface. DiamondHelp provides a "things to say" mechanism for use without spoken language understanding; it also supports extensions to take advantage of speech technology. DiamondHelp's software architecture factors all application-specific content into two modular plug-ins, one of which includes Collagen and a task model.

Keyword: Intelligent assistants for complex tasks.

INTRODUCTION

> . . . technology remains far too hard for most folks to use and most people can only utilize a tiny fraction of the power at their disposal. [13]

Our diagnosis of this problem identifies two fundamental underlying causes: the exhaustion of conventional interaction paradigms and the lack of consistency in user interface design. This paper addresses both of these causes by introducing a new framework for building intelligent user interfaces, called DiamondHelp.

The currently dominant human-computer interaction paradigm is direct manipulation [11], which can be applied with great effectiveness and elegance to those aspects of an interface which afford natural and intuitive analogies to physical actions, such as pointing, dragging, sliding, etc. In practice, however, most interfaces of any complexity also include an ad hoc and bewildering collection of other mechanisms, such as tool bars, pop-up windows, menus, command lines, etc.

To make matters worse, once one goes beyond the most basic functions (such as open, save, etc.) there is very little design consistency between interfaces to different products, even from the same manufacturer. The result is that it requires too large an investment of the typical user's time to learn all the intricacies of each new product.

Figure 1: DiamondHelp for combination washer-dryer.

The DiamondHelp framework consists of three components:

- an *interaction paradigm* based on task-oriented human collaboration,
- a graphical *user interface design* which combines conversational and direct manipulation interfaces,
- and a *software architecture* of reusable Java Beans.

The first two of these components are oriented towards the user; the third component addresses the needs of the software developer. Each of these components is described in detail in a section below. Figure 1 shows an example of the DiamondHelp user interface design for a combination washer-dryer.

The immediate motivation for DiamondHelp has been our recent attention to the usability crisis in high-tech home products, such as DVD recorders, combination washer-dryers, refrigerator-ovens, programmable thermostats, etc. However, DiamondHelp is not limited to home appliances; it can be applied to any software interface.

DiamondHelp grows out of a longstanding research thread on human-computer collaboration, organized around the Collagen system [9, 10]. Our work on Collagen, however, has been primarily concerned with the underlying se-

mantic and pragmatic structures for modeling collaboration, whereas DiamondHelp is focused on the appearance (the "look and feel") that a collaborative system presents to the user. Although Collagen plays a key role in our implementations of DiamondHelp applications (see Software Architecture section), the DiamondHelp framework can also be used independently of Collagen.

INTERACTION PARADIGM

Although the concept of an "interaction paradigm" is somewhat abstract and difficult to define formally, it is crucial to the overall development of interactive systems. In particular, the consistent expression and reinforcement of the interaction paradigm in the user interface design (what the user sees) leads to systems that are easier to learn and use. The interaction paradigm also provides organizing principles for the underlying software architecture, which makes such systems easier to build.

The collaborative paradigm for human-computer interaction, illustrated in Figure 2, mimics the relationships that typically hold when two humans collaborate on a task involving a shared artifact, such as two mechanics working on a car engine together or two computer users working on a spreadsheet together. This paradigm differs from the conventional view of interfaces in two key respects.

First, notice that the diagram in Figure 2 is symmetric between the user and the system. Collaboration in general involves both action and communication by both participants, and in the case of co-present collaboration (which is the focus of DiamondHelp) the actions of each participant are directly observed by the other participant.

One consequence of this symmetry is that the collaborative paradigm spans, in a principled fashion, a very broad range of interaction modes, depending on the relative knowledge and initiative of the system versus the user. For example, "tutoring" (aka intelligent computer-aided instruction) is a kind of collaboration in which the system has most of the knowledge and initiative. At the other end of the range is "intelligent assistance," wherein the user has most of the knowledge and initiative. Furthermore, a collaborative system can easily and incrementally shift within this range depending on the current task context. This kind of "mixed initiative" is the exception rather than the rule in conventional interfaces.

Second, and at a deeper level, the primary role of communication in collaboration is not for the user to tell the system what to do (the traditional "commands"), but rather to establish and negotiate about *goals* and how to achieve them. Licklider observed this fundamental distinction over 40 years ago in a classic article, which is well worth revisiting [3]:

> [Compare] instructions ordinarily addressed to intelligent human beings with instructions ordinarily used with computers. The latter specify precisely the individual steps to take and the sequence in which to take them. The former present or imply something about incentive or motivation, and they supply a criterion by which the human executor of the instructions will know when he has accomplished his task. In short: instructions directed to computers specify courses; instructions directed to human beings specify goals. [6]

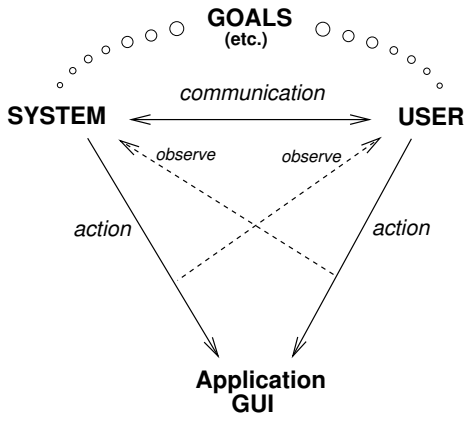


Figure 2: The collaborative paradigm.

The problem with the "command" view of user interfaces is that it demands too much knowledge on the part of the user. Conventional systems attempt to compensate for this problem by adding various kinds of ad hoc help facilities, tool tips, wizards, etc., which are typically not integrated into any clear paradigm. (See the Related Work section for specific discussion of the relation of DiamondHelp to wizards and Microsoft's Clippy.) In contrast, collaboration encompasses all of these capabilities within the paradigm.

Finally, notice that Figure 2 does provides a place to include direct manipulation as part of the collaborative paradigm, i.e., one can choose to implement the application GUI using direct manipulation. This is, in fact, exactly what we have done in the user interface design for DiamondHelp, described in the next section.

USER INTERFACE DESIGN

Our overarching goal for the DiamondHelp user interface design was to signal, as strongly as possible, a break from the conventional interaction style to the new collaborative paradigm. A second major challenge was to accomodate what is inescapably different about each particular application of DiamondHelp (the constructs needed to program a thermostat are very different from those needed to program a washing machine), while preserving as much consistency as possible in the collaborative aspects of the interaction. If a someone knows how to use *one* DiamondHelp application, they should know how to use *any* DiamondHelp application.

In order to address these concerns, we divided the screen into two areas, as shown in the example DiamondHelp interfaces of Figures 1, 3 and 4. Figure 1 has been implemented in Java;

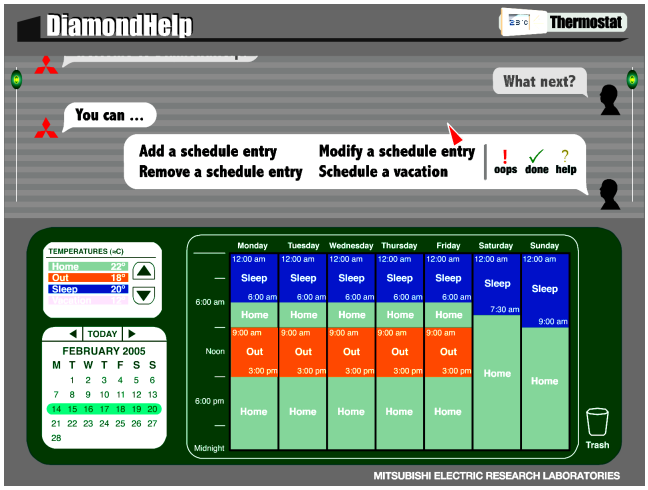


Figure 3: DiamondHelp for programmable thermostat.

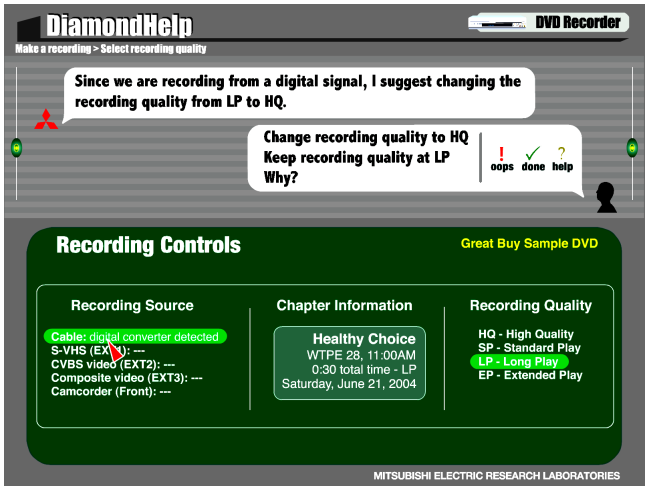


Figure 4: DiamondHelp for DVD recorder.

Figures 3 and 4 are Flash simulations. The top half of each of these screens is the same distinctive DiamondHelp conversational interface. The bottom half of each screen is an application-specific direct manipulation interface. Dividing the screen into two areas is, of course, not new; our contributions are the specific graphical interface design and the reusable software architecture described below.

Conversational Interface

To express and reinforce the human-computer collaboration paradigm, which is based by human-human communication, we adopted the scrolling speech bubble metaphor often used in online chat programs, which support human-human communication. The bubble graphics nicely reflect the collaborative paradigm's symmetry between the system and the user, discussed above. This graphical metaphor also naturally extends to the use of speech synthesis and recognition technology (see Extensions and Variations section below).

The top half of every DiamondHelp interface is thus a conversation ("chat") between DiamondHelp (the system), represented by the Mitsubishi three-diamond logo on the left, and the user, represented by the human profile on the right. All communication between the user and system takes place in these bubbles; there are no extra toolbars, pop-up menus, etc.

The basic operation of the conversational part of the DiamondHelp interface is as follows. Let's start with the opening of the conversation, which is the same for all DiamondHelp applications:

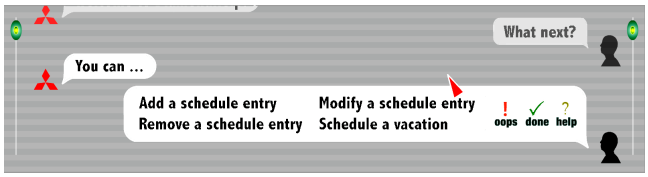
After the system says its welcome, a user bubble appears with several options for the user to choose from. We call this the user's "things to say" bubble. At the opening of the conversation, there are only four choices: "What next?," "oops," "done," and "help." Notice that the last three of these options are specially laid out with icons to the right of the vertical line; this is because these three options are always present (see Predefined Things to Say section below).

At this point, the user is free either to reply to the system's utterance by selecting one of the four things to say *or* to interact with the direct manipulation part of the interface. Unlike the case with traditional "dialogue boxes," the application GUI is never locked. This flexibility is a key aspect of how we have combined conversational and direct manipulation interfaces in DiamondHelp.

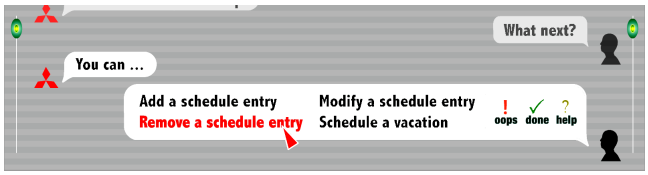
In this scenario, the user decides to request task guidance from the system by selecting "What next?" in the things-to-say bubble. As we have argued elsewhere [10], *every* interactive system should be able to answer a "What (can I do) next?" question.

As part of the interface animation, whenever the user makes a things-to-say selection (by clicking or touching the desired word or phrase), the not-selected items are erased and the enclosing bubble is shrunk to fit only the selected word or phrase. Furthermore, in preparation for the next step of the conversation, the completed system and user bubbles are "grayed out" and scrolled upward (if necessary) to make room for the system reply bubble and the user's next things-to-say bubble:

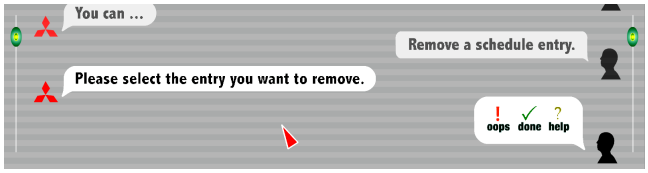
DiamondHelp applications always reply to “What next?” by a “You can…” utterance with the actual options for the possible next task goals or actions presented inside the user’s things-to-say bubble:



In this scenario, from DiamondHelp for a programmable thermostat (see Figure 3), the direct manipulation interface shows the family’s schedule for the week. The entries in this schedule determine the thermostat’s temperature settings throughout the day. The user chooses the goal of removing a schedule entry:



The system replies by asking the user to indicate, by selection in the direct manipulation interface, which schedule entry is to be removed. Notice that the user’s things-to-say bubble below includes only three choices. “What next?” is not included because the system has just told the user what to do next:



Although the system is expecting the user next to click (or touch) in the direct manipulation interface (see Figure 5), the user is still free to use the conversational interface, e.g, to ask for help.

Scrolling History

As in chat windows, a DiamondHelp user can at any time scroll back to view parts of the conversation that have moved off the screen. This is particularly useful for viewing explanatory text, which can be quite long (e.g., suppose the system utterance in Figure 4 was several lines longer).

The oval beads on each side of the upper half of the screen in Figures 1, 3 and 4 are standard scroll bar sliders. We also support scrolling by simply dragging anywhere on the lined background of the upper window.

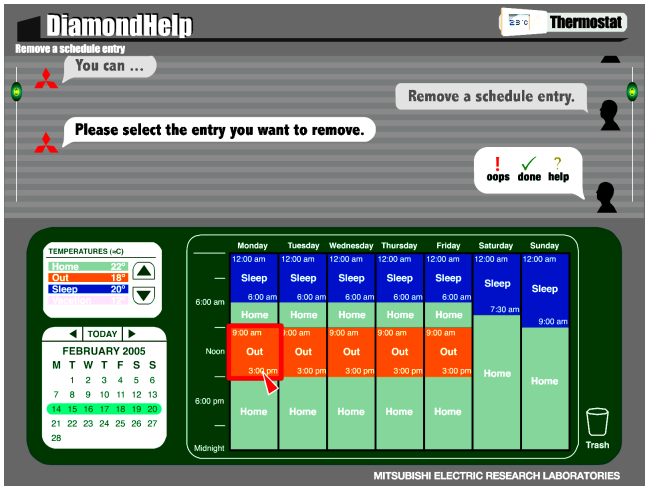


Figure 5: Selection in the direct manipulation area (see small triangular cursor in first column of schedule).

An interesting DiamondHelp extension to explore is allowing the user to “restart” the conversation at an earlier point in the history and move forward again with different choices. This could be a good way to support the backtracking paradigm for problem solving and is also closely related to our earlier work on history-based transformations in Collagen [9].

Things to Say

The user’s things-to-say bubble is partly an engineering compromise to compensate for the lack of natural language understanding (see Extensions and Variations section below). It also, however, partly serves the function of suggesting to the user what she can do at the current point [12, 1].

In the underlying software architecture, each choice in the things-to-say bubble is associated with the semantic representation of an utterance. When the user selects the desired word or phrase, the architecture treats the choice as if the missing natural-language understanding system produced the associated semantics.

From a design perspective, this means that the wording of the text displayed for each things-to-say choice should read naturally as an utterance by the user in the context of the ongoing conversation. For example, in Figure 1, in reply to the system’s question “How can I help you?”, one of the things-to-say choices is “Explain wash agitation,” not “wash agitation.”

At a deeper level, to be true to the spirit of the collaborative paradigm, the *content* of the conversations in DiamondHelp, i.e., both the system and user utterances, should concern not only primitive actions (“Remove a schedule entry”), but also higher-level goals (“Schedule a vacation”) and motivation (“Why?”).

When Collagen is used in the implementation of a DiamondHelp application, all the system utterances and the user’s

things-to-say choices are automatically generated from the task model given for the application (see Collagen Collaboration Plug-in section). Without Collagen, the appropriate collaborative content is provided by the application developer, guided by the principles elucidated here.

An obvious limitation of the things-to-say approach is that there is only room for a relatively small number of choices inside the user bubble—six or eight without some kind of nested scrolling, which we would like to avoid. Furthermore, since we would also like to avoid the visual clutter of drop-down choices within a single user utterance, each thing-to-say is a fixed phrase without variables or parameters.

Given these limitations, our design strategy is to use the direct manipulation interface to enter variable data. For example, in DiamondHelp for a washer-dryer, instead of the system asking "How long is the dry cycle?" and generating a things-to-say bubble containing "The dry cycle is 10 minutes," "... 15 minutes," etc., the system says "Please set the dry cycle time," points to the appropriate area of the direct manipulation interface, and expects the user to enter the time via the appropriate graphical widget. Figure 5 is a similar example of using the direct manipulation interface to provide variable data, in this case to select the thermostat schedule entry to be removed.

Predefined Things to Say

Most of the things to say are specific to an application. However, a key part of the generic DiamondHelp design is the following set of user utterances which should have the same meaning in all applications: "What next?," "Never mind," "Oops," "Done," and "Help." In addition, to save space and enhance appearance, we have adopted a special layout for the last three of these utterances, which are always present.

"What next?" has already been discussed above. "Never mind" is a way to end a question without answering it (see Figure 1). "Oops," "Done," and "Help" each initiate a subdialogue in which the system tries to determine, respectively, how to recover from a mistake, what level of task goal has been completed, or what form of help the user desires.

When Collagen is used in the implementation of DiamondHelp, the semantics of the predefined things-to-say are automatically implemented correctly; otherwise this is the responsibility of the collaboration plug-in implementor (see Software Architecture section).

Task Bar

An additional component of the DiamondHelp conversational interface is the "task bar," which is the single line located immediately above the scrollable history and below the DiamondHelp logo in Figures 1 and 4 (the task bar in Figure 3 happens to be blank at the moment of the screen shot). For example, in Figure 1, the contents of the task bar reads:

Make a new cycle > Adjust wash agitation > Help

The task bar, like the scrollable history and the predefined "What next?" utterance, is a mechanism aimed towards helping users when they lose track of their context, which is a common problem in complex applications. When Collagen is used in the implementation of DiamondHelp, the task bar is automatically updated with the path to the currently active node in the task model tree (see Figure 7). Otherwise, it is the responsibility of the collaboration plug-in implementor to update the task bar with appropriate task context information.

Currently, the task bar is for display only. However, we have considered extending the design to allow users to click (touch) elements of the displayed path and thereby cause the task focus to move. This possible extension is closely related to the scrollable history restart extension discussed above.

Application GUI

The bottom half of each screen in Figures 1, 3 and 4 is an application-specific direct manipulation interface. The details of these GUI's are not important to the main point of this paper. If fact, there is nothing in the DiamondHelp framework that guarantees or relies upon the application GUI being well-designed or even that it follow the direct manipulation paradigm, though this is recommended. (The only way to guarantee this would be for DiamondHelp to automatically generate the application GUI from a task model. We have done some research on this approach [2], but it is still far from practical.)

Furthermore, to achieve our overall goal of consistency across applications, an industrial-grade DiamondHelp would, like conventional UI toolkits, provide standard color palettes, widgets, skins, etc. However, this is beyond the scope of a research prototype.

Two design constraints which *are* relied upon by the DiamondHelp framework are: (1) the user should be able to use the application GUI at any time, and (2) every user action on the application GUI is reported to the system (see Manipulation Plug-in section below).

Finally, it is worth noting that, for the three applications presented here, it is possible to perform every function the application supports entirely using the direct manipulation interface, i.e., totally ignoring the conversational window. While this is a pleasant fact, we also believe that in more complex applications, there may be some overall advantage in relaxing this constraint. This is a design issue we are still exploring.

SOFTWARE ARCHITECTURE

In contrast to most of the discussion of above, which focuses on the the user's view of DiamondHelp, this section addresses the needs of the software developer. The overarching issue in DiamondHelp's software architecture is *reuse*, i.e., factoring the code so that the developer of a new DiamondHelp application only has to write what is idiosyncratic to that application, while reusing as much generic DiamondHelp framework code as possible.

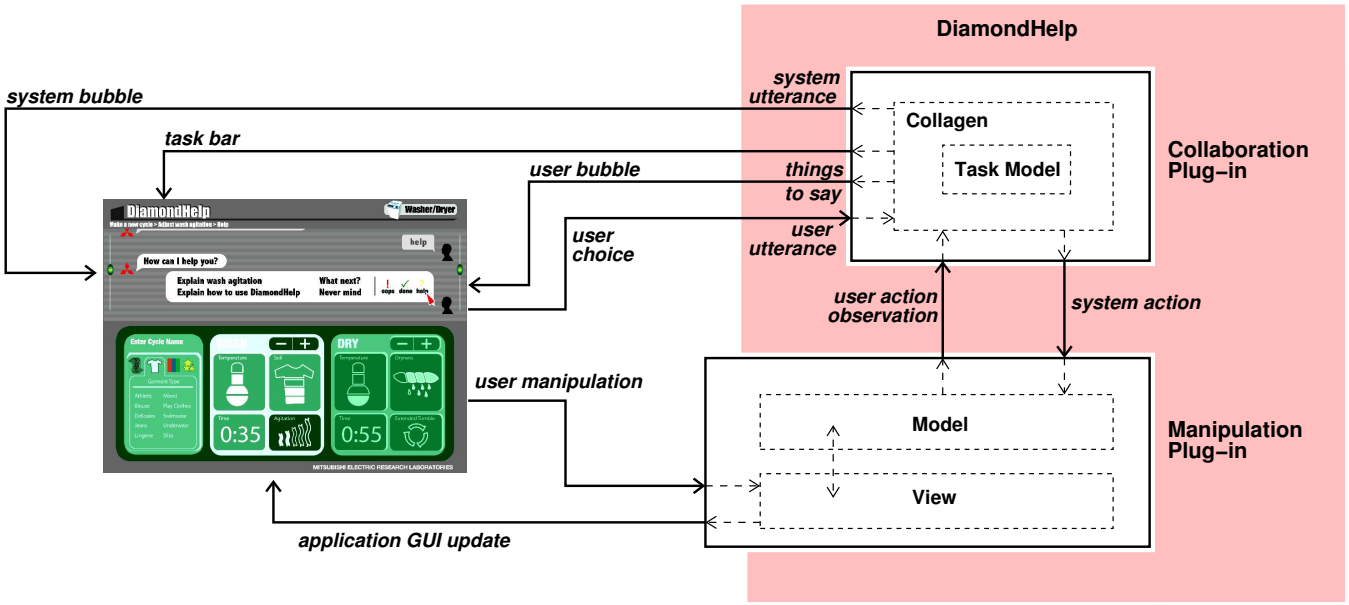


Figure 6: DiamondHelp software architecture.

Figure 6 shows our solution to this challenge, which we have implemented in Java Beans and Swing. (Dotted lines indicate optional, but recommended, components.) All the application-specific code is contained in two "plug-ins," which are closely related to the two halves of the user interface. The rest of the code (shaded region) is generic DiamondHelp framework code.

In addition to the issues discussed in this section, there are a number of other standard functions of plug-and-play architectures (of which DiamondHelp is an instance), such as discovering new devices connected to a network, loading the appropriate plug-ins, etc., which are beyond the scope of this paper.

Manipulation Plug-in

Notice that the manipulation plug-in "sticks out" of the DiamondHelp framework box in Figure 6. This is because it is directly responsible for managing the application-specific portion of the DiamondHelp interface. DiamondHelp simply gives this plug-in a Swing container object corresponding to the bottom half of the screen.

We recommend that the manipulation plug-in provide a direct-manipulation style of interface implemented using the model-view architecture, as shown by the dotted lines in Figure 6. In this architecture, all of the "semantic" state of the interface is stored in the model subcomponent; the view subcomponent handles the application GUI.

Regardless of how the manipulation plug-in is implemented internally, it must provide an API with the DiamondHelp framework which includes two event types: outgoing events (*user action observation*) which report state changes resulting from user GUI actions, and incoming events (*system action*) which specify desired state changes. Furthermore, in order to preserve the symmetry of the collaborative paradigm (see Figure 2), it is the responsibility of the plug-in to update the GUI in response to incoming events, so that the user may observe system actions. As an optional but recommended feature, the manipulation plug-in may also provide the DiamondHelp framework with the graphical location of each incoming state change event, so that DiamondHelp can move a cursor or pointer to that location, to help the user observe system actions.

Note that the content of both incoming and outgoing state change events is specified in semantic terms, e.g., "change dry cycle time to 20 min," not "button click at pixel 100,200." Lieberman [7] further discusses this and other issues related to interfacing between application GUI's and intelligent systems.

Collaboration Plug-in

Basically, the responsibility of the collaboration plug-in is to generate the system's responses (actions and utterances) to the user's actions and utterances. Among other things, this plug-in is therefore responsible for implementing the semantics of the predefined utterances (see next section).

The collaboration plug-in has two inputs: observations of user actions (received from the manipulation plug-in), and user utterances (resulting from user choices in the things-to-say bubble). It also has four outputs: system actions (sent to the manipulation plug-in), system utterances (which go into system bubbles), things to say (which go into user bubbles), and the task bar contents.

Notice that the collaboration plug-in is responsible for providing only the *content* of the system and user bubbles and

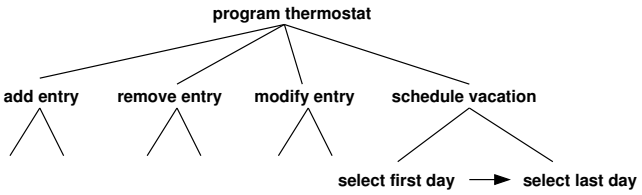


Figure 7: Task model for programmable thermostat.

the task bar. All of the graphical aspects of the conversational window are managed by DiamondHelp framework code. It is also an important feature of the DiamondHelp architecture that the collaboration plug-in developer does not need to be concerned with the graphical details of the application GUI. The collaboration plug-in developer and the manipulation plug-in developer need only to agree on a semantic model of the application state.

For a very simple application, the collaboration plug-in may be implemented by a state machine or other ad hoc mechanisms. However, in general, we expect to use the Collagen version described in the next section.

Collagen Collaboration Plug-in

The Collagen version of the collaboration plug-in includes an instance of Collagen, with a little bit of wrapping code to match the collaboration plug-in API. Collagen has already been extensively described in the literature [9, 10]. We will only highlight certain aspects here which relate to DiamondHelp.

The most important reason to use the Collagen collaboration plug-in is that the application developer only needs to provide one thing: a *task model*. All four outputs of the collaboration plug-in described above are then automatically generated by Collagen.

A task model is an abstract, hierarchical, partially ordered representation of the actions typically performed to achieve goals in the application domain. Figure 7 shows a fragment of the task model for the programmable thermostat application in Figure 3. Collagen currently provides a Java extension language for defining task models. We are also considering an XML version of this language.

System utterances and actions are produced by Collagen's usual response generation mechanisms [8]. The semantics of some of the predefined DiamondHelp user utterances, such as "Oops," "Done," and "Help," " are captured in a generic task model, which is used by Collagen in addition to the application-specific task model provided. Each of these predefined utterances introduces a generic subgoal with an associated subdialogue (using Collagen's discourse stack). Other predefined utterances, such as "What next?", and "Never mind," are built into Collagen.

The third collaborative plug-in output produced by Collagen is the user's things to say. Basically, the things to say are the result of filtering the output of Collagen's existing discourse (agenda) generation algorithm. The first step of filtering is remove everything except expected user utterances. Then, if there are still too many choices, a set of application-independent heuristics are applied based on the type of communicative act (proposal, question, etc.). Lastly, predefined utterances are added as appropriate. The further details of this algorithm need to be the topic of a separate paper.

Collagen's powerful plan recognition capabilities [4, 5] are used to keep the task bar up to date with respect to the given task model.

RELATED WORK

Although an increasing number of intelligent systems have been built recently using the collaborative paradigm (especially tutoring systems), none have focused specifically on the user interface design issues we discuss here, nor do any include a generic software architecture that is comparable with DiamondHelp.

Use of a textual things-to-say list together with Collagen and speech recognition has been studied by Sidner and Forlines [12] for a personal video recorder interface and by Dekoven [1] for a programmable thermostat.

In terms of generic software architectures for connecting arbitrary applications with "help" facilities, a major example is Microsoft's well-known Clippy. Clippy's interaction paradigm might be described as "watching over the user's shoulder and jumping in when you have some advice," and is, in our opinion, the main reason for Clippy's well-known unpopularity among users. In contrast, the collaborative paradigm underlying DiamondHelp emphasizes ongoing communication between the system and user to maintain shared context.

The second major example of generic help architectures are "wizards," such as software installation wizards, etc., which come in many different forms from many different manufacturers. Our feeling about wizards is that they embody the right paradigm, i.e., interactively guiding the user, but in too rigid a form. DiamondHelp subsumes the capabilities of wizards, but also allows users to take more initiative when they want to.

CONCLUSION

The contributions of this work to intelligent user interfaces are twofold. First, we have explicated a novel user interface design, which expresses and reinforces the human-computer collaboration paradigm by combining conversational and direct manipulation interfaces. Second, and more concretely, we have produced the DiamondHelp software, which can be used by others to easily construct such interfaces for new applications.

Extensions and Variations

A number of small extensions to DiamondHelp have already been discussed in the body of this paper. Another general set of extensions have to do with adding speech and natural

language understanding technology.

Adding text-to-speech generation to the system bubble is a very easy extension, which we have already done (its an optional feature of the software). Furthermore, we have found that using a system with synthesized speech is surprisingly more pleasant than without, even when user input is still by touch or click.

Adding speech recognition can be done in two ways. The first, more limited way, is to use speech recognition to choose one of the displayed things to say. We have already done this (another optional feature) and have found that the speech recognition is in general quite reliable, because of the small number of very different utterances being recognized. Furthermore, because of the way that the things-to-say mechanism is implemented, this extension requires no changes in the collaboration or manipulation plug-ins for an application.

A second, and much more ambitious approach, is to dispense with the things-to-say bubble and try to recognize anything the user says, which, of course, requires broad underlying speech and natural language understanding capabilities. If one has broad natural language understanding, another variation is to support unrestricted typed input from the user instead of speech.

Finally, it would be trivial (as least from the user interface point of view) to revert the conversational interface to its original function as a chat between two humans, the user and a remote person. This could be useful, for example, for remote instruction or troubleshooting. The system could even automatically shift between normal and "live chat" mode.

Future Work

In addition to building DiamondHelp interfaces for several additional applications, we also plan to conduct first informal and then formal user studies. The purpose of the informal user studies will be simply to get feedback and suggestions on our user interface design. For the formal studies, we would like to compare a DiamondHelp interface with a conventional interface for the same application, and measure the usual parameters, such as task completion time, solution quality, and user satisfaction. One approach for the comparison condition is to use the same direct manipulation interface as in the DiamondHelp condition, but with a written user manual instead of DiamondHelp.

REFERENCES

1. E. DeKoven. *Help Me Help You: Designing Support for Person-Product Collaboration*. PhD thesis, Delft Inst. of Technology, 2004.
2. J. Eisenstein and C. Rich. Agents and GUIs from task models. In *Proc. Int. Conf. on Intelligent User Interfaces*, pages 47–54, San Francisco, CA, Jan. 2002.
3. N. Lesh, J. Marks, C. Rich, and C. Sidner. "Man-computer symbiosis" revisited: Achieving natural communication and collaboration with computers. *IEICE Transactions Inf. & Syst.*, E87-D(6):1290–1298, June 2004.
4. N. Lesh, C. Rich, and C. Sidner. Using plan recognition in human-computer collaboration. In *Proc. 7th Int. Conf. on User Modelling*, pages 23–32, Banff, Canada, June 1999.
5. N. Lesh, C. Rich, and C. Sidner. Collaborating with focused and unfocused users under imperfect communication. In *Proc. 9th Int. Conf. on User Modelling*, pages 64–73, Sonthofen, Germany, July 2001. Outstanding Paper Award.
6. J. C. R. Licklider. Man-computer symbiosis. *IRE Trans. Human Factors in Electronics*, HFE-1:4–11, Mar. 1960.
7. H. Lieberman. Integrating user interface agents with conventional applications. In *Proc. Int. Conf. on Intelligent User Interfaces*, pages 39–46, San Francisco, CA, Jan. 1998.
8. C. Rich, N. Lesh, J. Rickel, and G. A. A plug-in architecture for generating collaborative agent responses. In *Proc. 1st Int. J. Conf. on Autonomous Agents and Multiagent Systems*, Bologna, Italy, July 2002.
9. C. Rich and C. Sidner. Collagen: A collaboration manager for software interface agents. *User Modeling and User-Adapted Interaction*, 8(3/4):315–350, 1998. Reprinted in S. Haller, S. McRoy and A. Kobsa, editors, *Computational Models of Mixed-Initiative Interaction*, Kluwer Academic, Norwell, MA, 1999, pp. 149–184.
10. C. Rich, C. Sidner, and N. Lesh. Collagen: Applying collaborative discourse theory to human-computer interaction. *AI Magazine*, 22(4):15–25, 2001. Special Issue on Intelligent User Interfaces.
11. B. Shneiderman and C. Plaisant. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley, Reading, MA, 2005.
12. C. L. Sidner and C. Forlines. Subset languages for conversing with collaborative interface agents. In *Int. Conf. on Spoken Language Processing*, Sept. 2002.
13. S. H. Wildstrom. Technology & you: Lessons from a dizzying decade in tech. *BusinessWeek*, June 14 2004.