

DiamondHelp: A Generic Collaborative Task Guidance System

Charles Rich and Candace L. Sidner

■ DiamondHelp is a generic collaborative task guidance system motivated by the current usability crisis in high-tech home products. It combines an application-independent conversational interface (adapted from online chat programs) with an application-specific direct-manipulation interface. DiamondHelp is implemented in Java and uses Collagen for representing and using task models.

Our diagnosis of the current usability crisis in high-tech home products (see sidebar this page) identifies two fundamental underlying causes: the exhaustion of conventional interaction paradigms and the lack of consistency in user interface design. This article addresses both of these causes by introducing a new framework for building collaborative task guidance systems, called DiamondHelp.

The dominant current paradigm for human-computer interaction is direct manipulation (Shneiderman and Plaisant 2005), which can be applied with great effectiveness and elegance to those aspects of an interface that afford natural and intuitive analogies to physical actions, such as pointing, dragging, sliding, and so on. 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, and so on.

To make matters worse, once one goes beyond the most basic functions (such as open, save, and so on), 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.

The DiamondHelp framework consists of three components: (1) an interaction paradigm based on task-oriented human collaboration, (2) a graphical user interface (GUI) design that combines conversational and direct-manipulation interfaces, and (3) a software architecture of reusable JavaBeans.

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 later on.

Figure 1 illustrates the DiamondHelp user interface design. The top half of the screen is the generic conversational interface, while the bottom half is a direct-manipulation interface for a combination washer-dryer. Although our immediate motivation has been the usability crisis in DVD recorders, programmable thermostats, combination washer-dryers, refrigerator-ovens, and so on, DiamondHelp is not limited to home appliances; it can be applied to any software interface.

DiamondHelp grew out of a longstanding research thread on human-computer collaboration, organized around the Collagen system (see the Collagen sidebar). Our work on Collagen, however, has been primarily concerned with the underlying semantic 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 the Software Architecture Section), the DiamondHelp

The Usability Crisis in High-Tech Home Products

... 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. (*Business Week*, 2004)

Half of all “malfunctioning products” returned to stores by consumers are in full working order, but customers can’t figure out how to operate the device. (Reuters, 2006)



Figure 1. DiamondHelp for Combination Washer-Dryer.

framework can also be used independently of Collagen.

Finally, in relation to this special issue of *AI Magazine*, we would say that mixed initiative is at the heart of DiamondHelp since collaboration and mixed initiative are virtually coextensive concepts: all collaborative systems are mixed initiative, and most interesting mixed-initiative systems are collaborative (see further discussion in the Mixed Initiative section).

Interaction Paradigm

Although 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” (also known as 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.

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. J. C. R. Licklider observed this fundamental distinction more than 40 years ago in a classic article, which is well worth revisiting (see the Licklider sidebar and Lesh et al. 2004).

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, and so on, which are typically not integrated into any clear paradigm. (See Related Work 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 provide a place to include direct manipulation as part of the collaborative paradigm; that is, one can choose to implement the application’s GUI

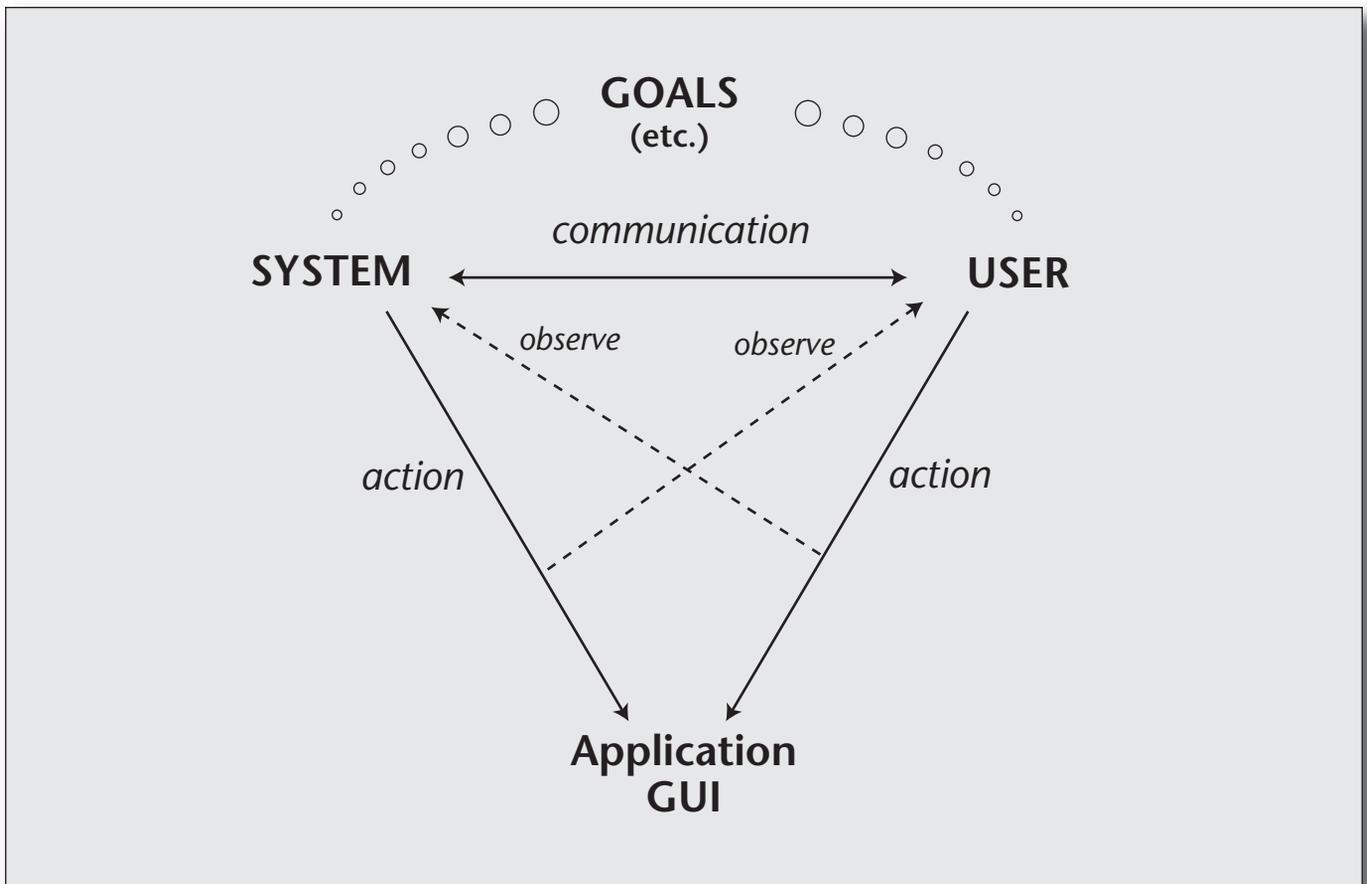


Figure 2. The Collaborative Paradigm.

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 accommodate 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 someone knows how to use one DiamondHelp application, he or she 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. The washer-dryer has been imple-

J. C. R. Licklider in 1960 on Human-Computer Collaboration

[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. (Licklider 1960)

mented in Java; the other two appliances 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 later.

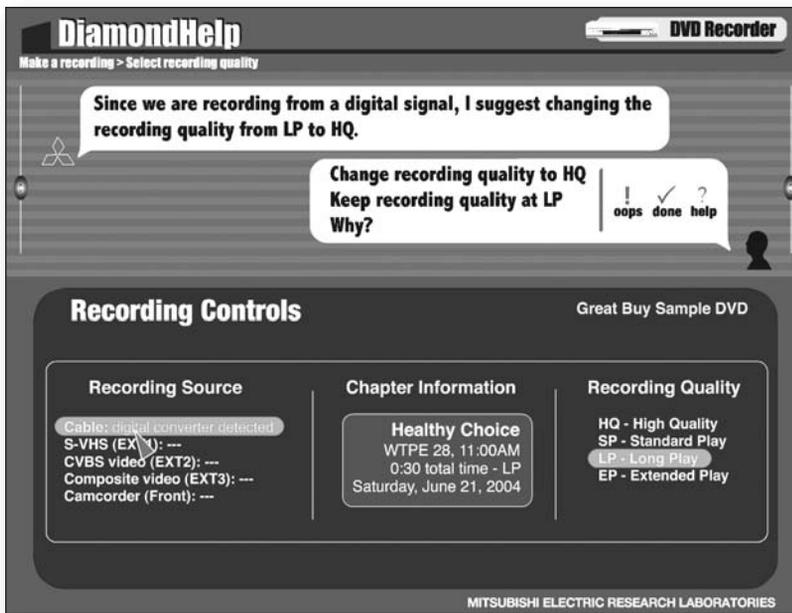


Figure 3. DiamondHelp for DVD Recorder.

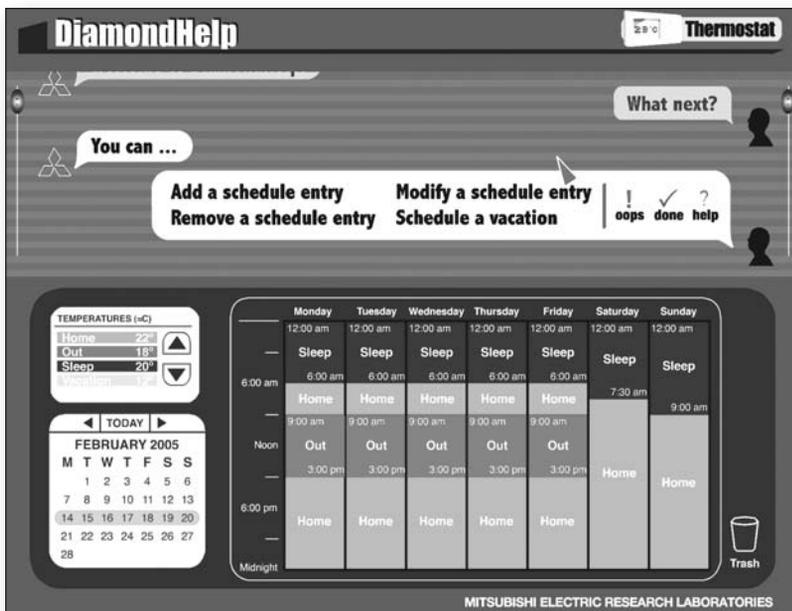


Figure 4. DiamondHelp for Programmable Thermostat.

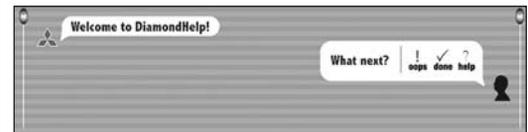
Conversational Interface

To express and reinforce the human-computer collaboration paradigm, which is based on 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 previously. This graphical metaphor also naturally extends to the use of

speech synthesis and recognition technology (see the conclusion).

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, and so on.

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).

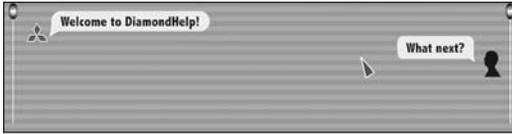
At this point, the user is free either to reply to the system’s utterance by selecting one of the four things to say or to interact with the direct manipulation part of the interface. Unlike the case with traditional “dialog boxes,” the application GUI is never locked. This flexibility is a key aspect 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 (see Rich, Sidner, and Lesh [2001]), every interactive system should be able to answer a “What (can I do) next?” question.

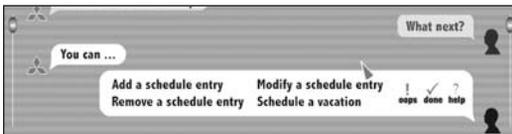


As part of the interface animation, whenever the user makes a things-to-say selection (by clicking or touching the desired word or phrase), the not-selected items are erased and the enclosing bubble is shrunk to fit only the selected word or phrase. Furthermore, in prepa-

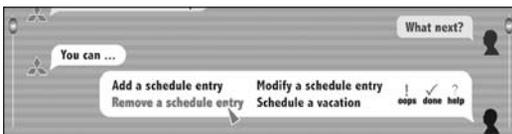
ration for the next step of the conversation, the completed system and user bubbles are “grayed out” and scrolled upward (if necessary) to make room for the system reply bubble and the user’s next things-to-say bubble:



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 4), 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 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, such as 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 (for example, suppose the system utterance in figure 3 was several lines longer).

The oval beads on each side of the upper half

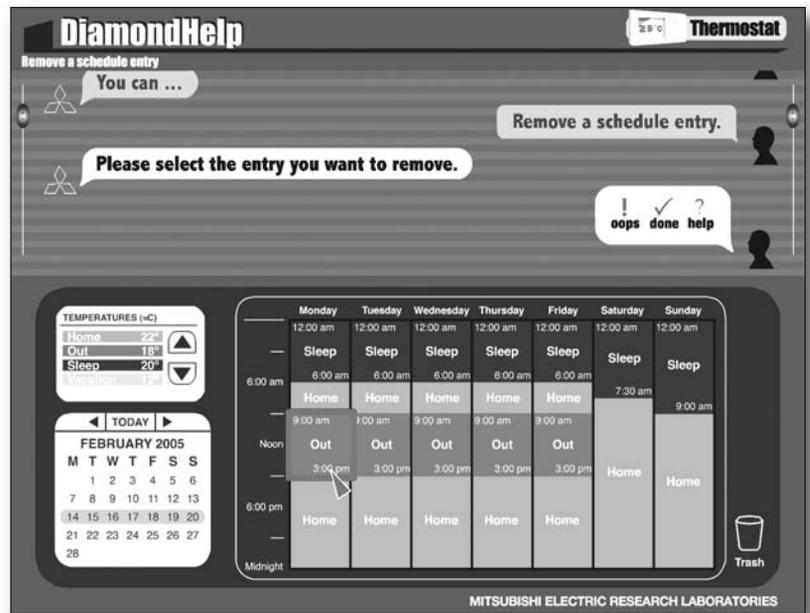


Figure 5. Selection in the Direct-Manipulation Area.

(See the small triangular cursor in the first column of the schedule.)

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.

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; see Rich and Sidner (1998).

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 the conclusion). It also, however, partly serves the function of suggesting to the user what she can do at the current point.

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, that is, 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 system utterances and the user's things-to-say choices are automatically generated from the task model given for the application (see the 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," and so on, 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 through 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.

Use of a textual things-to-say list together with Collagen and speech recognition has been studied by Sidner and Forlines (2002) for a personal video recorder interface and by DeKoven (2004) for a programmable thermostat.

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 appear-

ance, we have adopted a special layout for the last three of these utterances, which are always present.

"What next?" has been discussed previously. "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 the 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 3 (the task bar in figure 4 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 8 and the Collagen sidebar). 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 previously.

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 GUIs are not important to the main point of this article. In 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 [Eisenstein and Rich 2002], 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, and so on. However, this is beyond the scope of a research prototype.

Two design constraints that 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 the Manipulation Plug-in section).

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, that is, 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 previous discussion, 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, that is, factoring the code so that the developer of a new DiamondHelp application has to write only what is idiosyncratic to that application while reusing as much generic DiamondHelp framework code as possible.

Figure 6 shows our solution to this challenge, which we have implemented in JavaBeans 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, and so on, which are beyond the scope of this article.

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 application programming interface (API) with the DiamondHelp framework that 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, for example, "change dry cycle time to 20 min," not "button click at pixel 100,200." Lieberman (1998) further discusses this and other issues related to interfacing between application GUIs 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

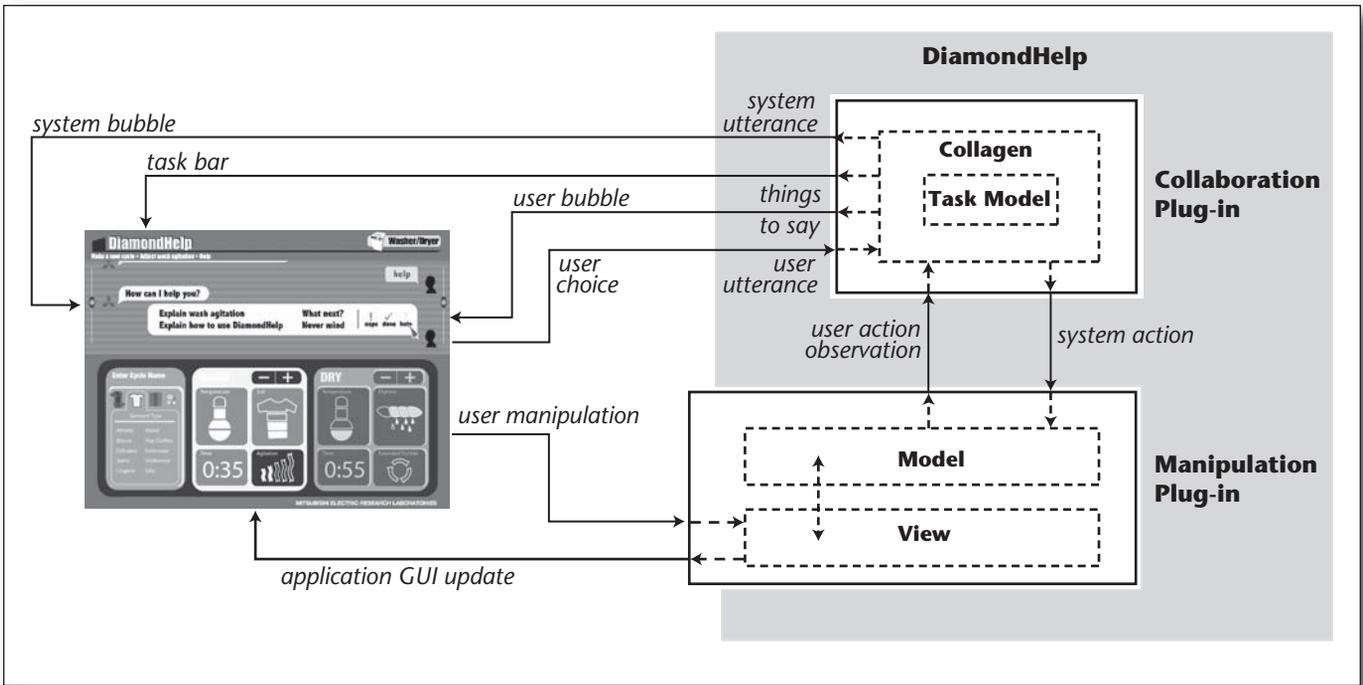


Figure 6. DiamondHelp Software Architecture.

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 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 interface. 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 described in an earlier article (see Collagen sidebar). We will only highlight certain aspects here which relate to DiamondHelp.

The most important reason to use the Colla-

gen collaboration plug-in is that the application developer needs to provide only one thing: *a task model*. All four outputs of the collaboration plug-in described previously are then automatically generated by Collagen as shown in figure 7.

System utterances and actions are produced by Collagen’s usual response generation mechanisms (Rich et al. 2002). 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, and so on). Lastly, predefined utterances are added as appropriate. The further details of this algorithm need to be the topic of a separate article.

Collagen: Applying Collaborative Discourse Theory to Human-Computer Interaction

Collagen (for collaborative agent) is Java middleware for building collaborative interface agents based on Grosz and Sidner's SharedPlan theory of collaborative discourse (Grosz and Sidner 1986, 1990; Grosz and Kraus 1996; Lochbaum 1998). Collagen has been used to build more than a dozen prototypes, including an air travel planning assistant (Rich and Sidner 1998); e-mail assistant (Gruen et al 1999); VCR programming assistant (Sidner and Forlines 2002); power system operation assistant (Rickel et al. 2001); gas turbine engine operation tutor (Davies et al. 2001); flight path planning assistant (Cheikes and Gertner 2001); recycling resource allocation assistant; software design tool assistant; programmable thermostat helper (DeKoven, Keyson, and Freudenthal 2001); mixed-initiative multimodal form filling, and robot hosting system (Sidner et al. 2005).

The two key data structures in Collagen's architecture (see figure 7) are the task model and the discourse state. The two key algorithms are discourse interpretation and generation.

Task Model

A task model is an abstract, hierarchical, partially ordered representation of the actions typically performed to achieve goals in the application domain. Figure 8 shows a fragment of the task model for the programmable thermostat application in figure 4. Although this example does not, task models in general also contain alternative decomposition choices. Collagen currently provides a Java extension language for defining task models, but we will adopt the new CEA standard (see the conclusion) as soon as it is completed.

Discourse State

The discourse state tracks the beliefs and intentions of the participants in the current collaboration and provides a focus of attention mechanism for tracking shifts in the task and conversational context. Collagen's dis-

course state is implemented as a goal decomposition (plan) tree plus a stack of focus spaces (each of which includes a focus goal). The top goal on the focus stack is the "current purpose" of the discourse.

Discourse Interpretation

The basic job of discourse interpretation is to update the discourse state incrementally by explaining how each occurring event (action or utterance by the user or the system) contributes to the current purpose. An event contributes to a purpose if it either (1) directly achieves the purpose, or (2) is a step in the task model for achieving the purpose, or (3) selects one of alternative decomposition choices for achieving the purpose, or (4) identifies which participant should achieve the purpose, or (5) identifies a parameter of the purpose.

If the current event contributes, either directly or indirectly, to the current purpose, then the plan tree is appropriately extended to include the event; the focus stack may also be pushed or popped. Our discourse-interpretation algorithm extends Lochbaum's (1998) original formulation by incorporating plan recognition (Lesh, Rich, and Sidner 1999, 2001) and handling interruptions.

Discourse Generation.

The discourse-generation algorithm in Collagen is essentially the inverse of discourse interpretation. Based on the current discourse state, it produces a prioritized list, called the agenda (see figure 7), of partially or totally specified utterances and actions that would contribute to the current discourse purpose according to the five cases itemized previously.

In general, an agent may use any application-specific logic it wants to decide on its next action or utterance. In most cases, however, an agent can simply choose the first item on the agenda generated by Collagen.

*This sidebar is a summary of Rich, Sidner, and Lesh (2001).

Mixed Initiative

In this section, we review DiamondHelp with respect to the shared themes of this special issue of *AI Magazine* on mixed-initiative assistants.

To begin, let us follow up on our earlier comment that mixed initiative and collaboration are virtually coextensive concepts. Horvitz provides as good a definition as any of mixed initiative:

... an efficient, natural interleaving of contributions by users and automated services aimed at converging on solutions to problems. (Horvitz 1999)

Compare this with the definition of collabora-

tion upon which Collagen (and therefore DiamondHelp) is based:

... a process in which two or more participants coordinate their actions toward achieving shared goals. (Rich, Sidner, and Lesh 2001)

If the two participants in a collaboration are the user and an automated system, then such a collaboration is generally mixed initiative; only in those rare or simple circumstances where one participant provides no contribution to any goal (including clarification, success/failure report, and so on) is mixed initiative lacking. Conversely, the only part of the definition of collaboration that is missing from the definition of mixed initiative is the explicit mention of "shared goals" (and it is perhaps

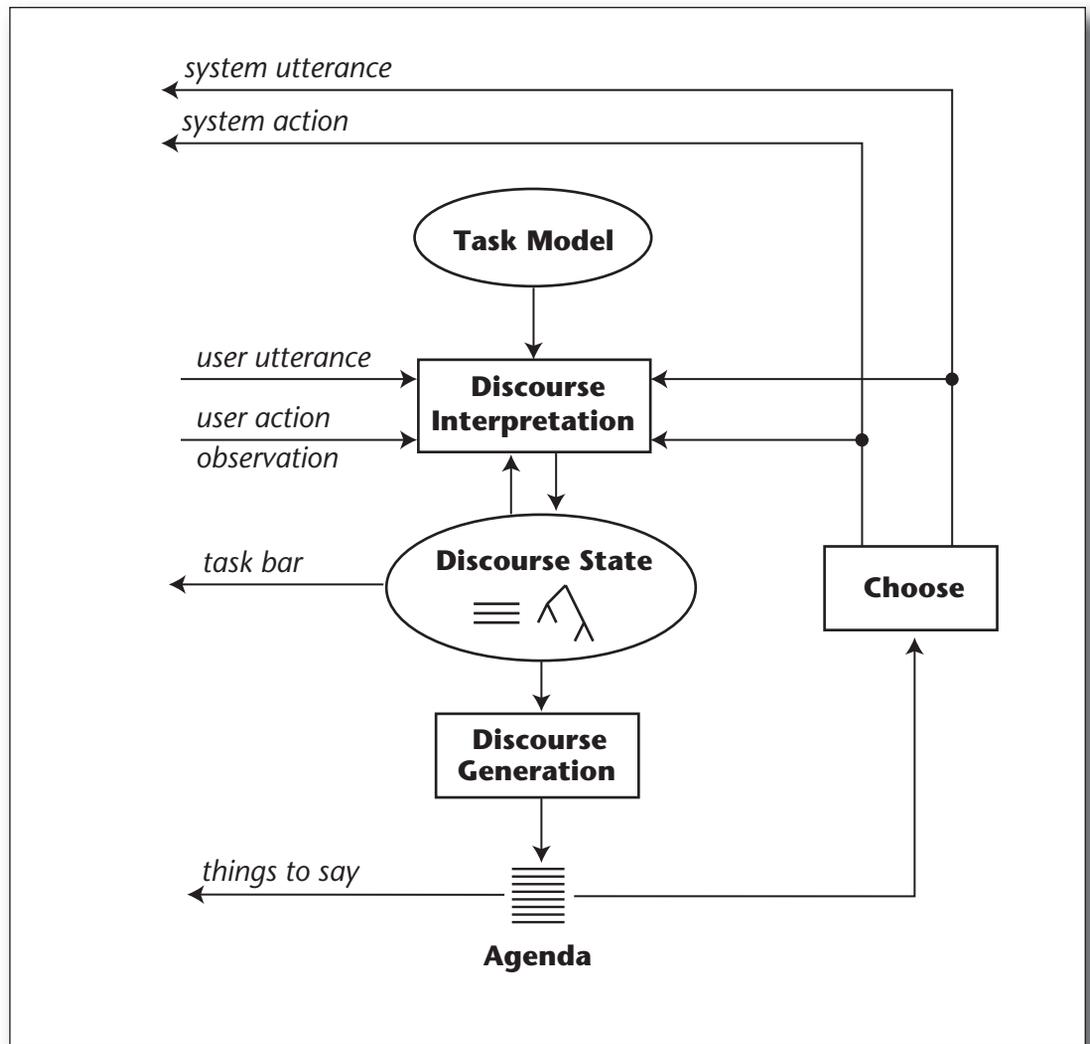


Figure 7. Collagen Architecture.

(Arrows on the left of this figure correspond to arrows with the same labels in figure 6).

implicit in “converging on solutions”). One could imagine some sort of interaction with interleaved contributions without shared goals, but it is hard to believe it would be very efficient or natural.

Division of Labor and Control

In general in a collaboration, decisions about who does what and when are matters for negotiation between the participants. In many concrete settings, however, many of these decisions are conventionalized by the context. Consider, for example, the collaboration between the customer and clerk at a retail counter. Similarly, DiamondHelp has been designed with a certain expected, though adjustable (see the Personalization subsection that follows), division of labor and control.

Basically, DiamondHelp assumes that the

user knows what she wants to do at a high level but needs help carrying out the necessary details. Furthermore, DiamondHelp normally returns control to the user as quickly as possible (by putting “What next?” in the user bubble).

Architecture

The key to DiamondHelp’s power and flexibility as a mixed-initiative system is the high degree of symmetry between the user and the system in all aspects of the architecture, from the user interface, to discourse interpretation, to the task model. Many interactive systems, even mixed-initiative ones, have built-in asymmetries that limit their flexibility.

The symmetry of the chat-based user interface has already been discussed previously. In figure 7, notice that the discourse-interpretation process is applied to both user and system

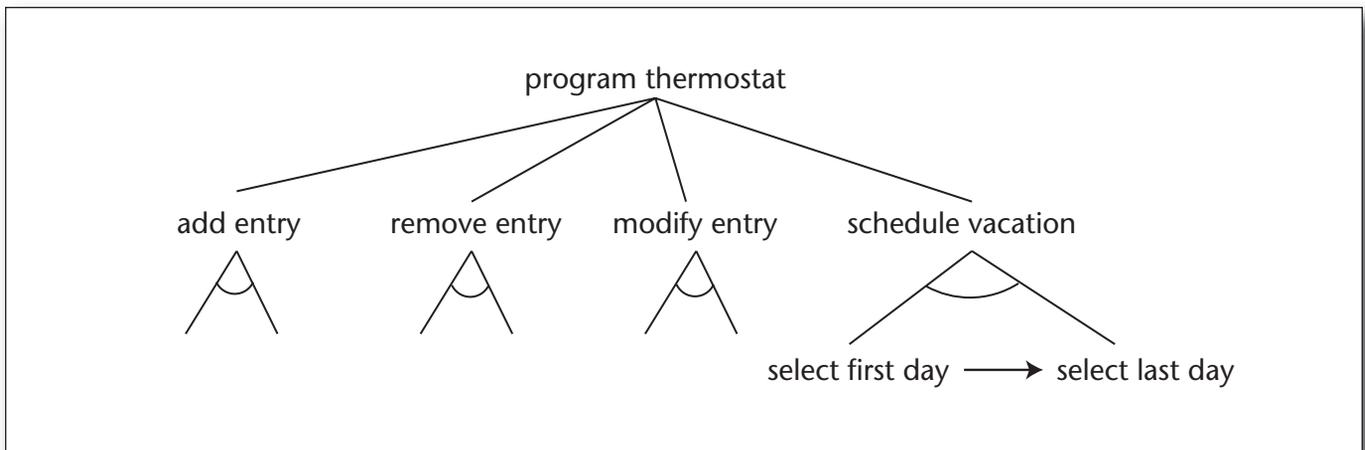


Figure 8. Task Model for Programmable Thermostat.

utterances and actions. This is a reflection of the fact that the collaborative discourse state represents the mutual beliefs of the participants (see Sidner [1994] for more details).

Finally, and most fundamentally, the task model that drives a DiamondHelp interaction generally specifies only *what* needs to be done, not *who* should do it. (The only exception is when one participant is fundamentally incapable of performing some action.) For example, consider the two steps to achieve the schedule vacation goal in figure 8. This task model accounts for an interaction in which the user selects both the first day and the last day through the direct-manipulation area, and also an interaction in which the system selects both days (perhaps based on other knowledge), or interactions in which one day is selected by the user and one by the system. Which interaction actually occurs depends on the actions and knowledge of the user and the system in the particular situation.

Communication and Shared Awareness

As discussed earlier, DiamondHelp is based on a model of co-present collaboration involving a shared artifact/application (see figure 2). Our user interface design attempts to give the feeling of natural communication without actually requiring natural language or speech processing. For shared awareness of the application state, DiamondHelp relies totally on the graphical user interface provided through the manipulation plug-in. DiamondHelp itself provides two mechanisms to support shared awareness of the task/conversation state: the task bar and the scrollable chat window. Collagen also provides a much more com-

plete representation of task/conversation state, called the segmented interaction history (see Rich, Sidner, and Lesh [2001]), which is not currently incorporated into the design of DiamondHelp.

Personalization

In addition to the significant personalization and adaptation inherent in all collaborative interactions, there are at least two specific personalization mechanisms in DiamondHelp worth mentioning here. Both of these mechanisms take advantage of facilities in Collagen, such as the student model, developed for intelligent tutoring (Rickel et al. 2002), which is not surprising, since personalization is a key aspect of intelligent tutoring.

First, as discussed previously, DiamondHelp normally returns control to the user as quickly as possible. However, based on simple observations of the user's behavior, such as timing and errors, DiamondHelp can switch into a mode where it takes control and guides the user through an entire task or subtask, without the user having to ask "What next?" at each step.

A second personalization has to do with whether the system asks the user to perform certain manipulations on the application GUI or simply performs them itself. For example, in the washer-dryer application, the system can either say "Please click here to pop up the temperature dial" or simply pop up the temperature dial window at the appropriate time itself. The former case has the benefit that the user learns where to click in the future; the latter case has the benefit of being faster. DiamondHelp can switch between these modes on a per action basis, depending on whether the user has already performed the action once or twice herself.

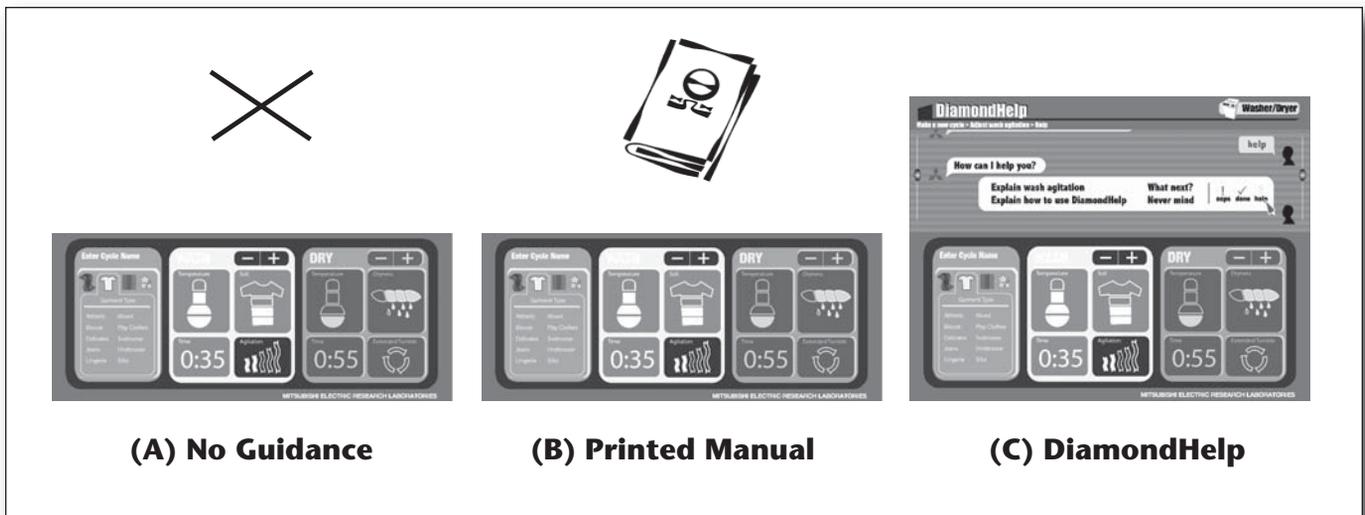


Figure 9. Three Conditions in Planned User Study.

Evaluation

Figure 9 shows the three conditions in a user study we are currently planning to evaluate DiamondHelp using the washer-dryer example. The printed manual in condition B will contain the same information (literally the same text) that is communicated dynamically by DiamondHelp in condition C. Users in each condition will be assigned a set of tasks that will require them to use the advanced programmability features of the washer-dryer. We plan to obtain both objective measures, such as time and quality of task completion, and subjective evaluations of experience.

Related Work

The most closely related work to DiamondHelp, in terms of both application and approach, is the Roadie system (Lieberman and Espinosa 2006), which is also directed towards consumer electronics and shares the task-model approach. Roadie differs from DiamondHelp primarily in not using a conversational or collaborative model for its interface. Roadie also incorporates natural language understanding and commonsense knowledge components for guessing the user's desires,

Also closely related is the Writer's Aid system (Babaian, Grosz, and Shieber 2002), which applies the same collaborative theory upon which DiamondHelp is based to a single specific task, rather than developing a general tool.

Of the articles in this special issue of *AI Magazine*, the most closely related is Ferguson and Allen (2007). The collaborative model and implementation architecture used in that work are very similar to those described here. The

scope of their work, however, is broader than ours, since they also build natural language-understanding components. Furthermore, their architecture includes a beliefs, desires, and intentions (BDI) component that is present only in vestigial form in DiamondHelp/Collagen.

BDI architecture is at the heart of the mixed-initiative personal assistant of Myers et al. (2007). They describe the interaction paradigm used in their work as “delegative,” and contrast it with the collaborative paradigm. We would view delegation as being included within the range of collaboration; see Grosz and Kraus (1996).

Another category of related work is generic software architectures for connecting arbitrary applications with help facilities. The best-known example of this category is Microsoft's 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.

Also in this category are so-called wizards, such as software installation wizards, and so on, which come in many different forms from many different manufacturers. Our feeling about wizards is that wizards embody the right paradigm, that is, 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 are twofold. First, we have explicated a novel user interface design that 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.

A number of small extensions to DiamondHelp have already been discussed in the body of this article. 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 (it's 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.

Alternatively, it would be trivial (at 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.

Finally, returning to the original motivation of this work, namely the usability crisis in high-tech home products, the Consumer Electronics Association (www.ce.org) has recently created a new Task-Based User Interface working group, which will develop a standard (CEA-2018) for representing task models. This stan-

dard has the potential of helping systems like DiamondHelp to have a significant impact on the usability problem.

Acknowledgements

The work described here took place at Mitsubishi Electric Research Laboratories between approximately January 2004 and June 2006. The basic DiamondHelp concept was developed in collaboration with Neal Lesh and Andrew Garland. Shane Booth provided graphic design input. Markus Chimani implemented key parts of the graphical interface in Java.

References

- Babaian, T.; Grosz, B.; and Shieber, M. 2002. A Writer's Collaborative Aid. In *Proceedings of the International Conference on Intelligent User Interfaces*, 7–14. New York: Association for Computing Machinery.
- Cheikes, B., and Gertner, A. 2001. Teaching to Plan and Planning to Teach in an Embedded Training System. Paper presented at the Tenth International Conference on Artificial Intelligence in Education, 19–23 May, San Antonio, Texas.
- Davies, J.; Lesh, N.; Rich, C.; Sidner, C.; Gertner, A.; and Rickel, J. 2001. Incorporating Tutorial Strategies into an Intelligent Assistant. In *Proceedings of the International Conference on Intelligent User Interfaces*, 53–56. New York: Association for Computing Machinery.
- DeKoven, E. 2004. Help Me Help You: Designing Support for Person-Product Collaboration. Ph.D. Dissertation, Delft Institute of Technology, Delft, the Netherlands.
- DeKoven, E.; Keyson, D.; and Freudenthal, A. 2001. Designing Collaboration in Consumer Products. In *Proceedings of the ACM Conference on Computer Human Interaction, Extended Abstracts*, 195–196. New York: Association for Computing Machinery.
- Eisenstein, J., and Rich, C. 2002. Agents and GUI's from Task Models. In *Proceedings of the International Conference on Intelligent User Interfaces*, 47–54. New York: Association for Computing Machinery.
- Ferguson, G., and Allen, J. 2007. Mixed-Initiative Systems for Collaborative Problem Solving. *AI Magazine* 28(2).
- Grosz, B. J., and Kraus, S. 1996. Collaborative Plans for Complex Group Action. *Artificial Intelligence* 86(2): 269–357.
- Grosz, B. J., and Sidner, C. L. 1986. Attention, Intentions, and the Structure of Discourse. *Computational Linguistics* 12(3): 175–204.
- Grosz, B. J., and Sidner, C. L. 1990. Plans for Discourse. In *Intentions and Communication*, ed. P. Cohen, J. Morgan, and M. Pollack, 417–444. Cambridge, MA: MIT Press.
- Gruen, D.; Sidner, C.; Boettner, C.; and Rich, C. 1999. A Collaborative Assistant for E-Mail. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems, Extended Abstracts*, 196–197. New York: Association of Computing Machinery.
- Horvitz, E. 1999. Uncertainty, Action and Interac-

tion: In Pursuit of Mixed-Initiative Computing. *IEEE Intelligent Systems* 14(5): 17–20.

Lesh, N.; Marks, J.; Rich, C.; and Sidner, C. 2004. “Man-Computer Symbiosis” Revisited: Achieving Natural Communication and Collaboration with Computers. *IEICE Transactions Information and Systems* E87–D(6): 1290–1298.

Lesh, N.; Rich, C.; and Sidner, C. 1999. Using Plan Recognition in Human-Computer Collaboration. In *Proceedings of the Seventh International Conference on User Modeling*, 23–32. New York: Springer-Verlag.

Lesh, N.; Rich, C.; and Sidner, C. 2001. Collaborating with Focused and Unfocused Users under Imperfect Communication. In *Proceedings of the Ninth International Conference on User Modeling*, 64–73. New York: Springer-Verlag.

Licklider, J. C. R. 1960. Man-Computer Symbiosis. *IRE Transactions on Human Factors in Electronics* HFE-1: 4–11.

Lieberman, H. 1998. Integrating User Interface Agents with Conventional Applications. In *Proceedings of the International Conference on Intelligent User Interfaces*, 39–46. New York: Association for Computing Machinery.

Lieberman, H., and Espinosa, J. 2006. A Goal-Oriented Interface to Consumer Electronics using Planning and Commonsense Reasoning. In *Proceedings of the International Conference on Intelligent User Interfaces*, 226–233. New York: Association for Computing Machinery.

Lochbaum, K. E. 1998. A Collaborative Planning Model of Intentional Structure. *Computational Linguistics* 24(4): 525–572.

Myers, K.; Berry, P.; Blyth, J.; Conley, K.; Gervasio, M.; McGuinness, D.; Morley, D.; Pfeffer, A.; Pollack, M.; and Tambe, M. 2007. An Intelligent Personal Assistant for Task and Time Management. *AI Magazine* 28(2).

Rich, C., and Sidner, C. 1998. COLLAGEN: A Collaboration Manager for Software Interface Agents. *User Modeling and User-Adapted Interaction* 8(3–4): 315–350.

Rich, C.; Lesh, N.; Rickel, J.; and Garland, A. 2002. A Plug-In Architecture for Generating Collaborative Agent Responses. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*. New York: Association for Computing Machinery.

Rich, C.; Sidner, C.; and Lesh, N. 2001. Collagen: Applying Collaborative Discourse Theory to Human-Computer Interaction. *AI Magazine* 22(4): 15–25.

Rickel, J.; Lesh, N.; Rich, C.; Sidner, C.; and Gertner, A. 2001. Using a Model of Collaborative Dialogue to Teach Procedural Tasks. Paper presented at the Workshop on Tutorial Dialogue Systems, Tenth International Conference on Artificial Intelligence in Education, 19–23 May, San Antonio, Texas.

Rickel, J.; Lesh, N.; Rich, C.; Sidner, C.; and Gertner, A. 2002. Collaborative Discourse Theory as a Foundation for Tutorial Dialogue. In *Proceedings of the 6th International Conference on Intelligent Tutoring Systems*, 542–551. Berlin: Springer.

Shneiderman, B., and Plaisant, C. 2005. *Designing the*

User Interface: Strategies for Effective Human-Computer Interaction. Reading, MA: Addison-Wesley.

Sidner, C. L. 1994. An Artificial Discourse Language for Collaborative Negotiation. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 814–819. Menlo Park, CA: AAAI Press.

Sidner, C. L.; Lee, C.; Kidd, C.; Lesh, N.; and Rich, C. 2005. Explorations in Engagement for Humans and Robots. *Artificial Intelligence* 166 (1–2): 104–164.

Sidner, C. L., and Forlines, C. 2002. Subset Languages for Conversing with Collaborative Interface Agents. Paper presented at the International Conference on Spoken Language Processing, Denver, CO., 16–20 September.



Charles Rich is a professor of computer science at Worcester Polytechnic Institute. He was previously a distinguished research scientist at Mitsubishi Electric Research Laboratories. He earned his Ph.D. at the MIT Artificial Intelligence Laboratory, where he was a founder and director of the

Programmer’s Apprentice project. Rich is a fellow and past councilor of AAAI, as well as having served as chair of the 1992 International Conference on Principles of Knowledge Representation and Reasoning, cochair of the 1998 National Conference on Artificial Intelligence, and program cochair of the 2004 International Conference on Intelligent User Interfaces. His e-mail address is rich@cs.wpi.edu.



Candace L. Sidner is a division scientist at BAE Systems Advanced Information Technologies. She was previously a senior research scientist at Mitsubishi Electric Research Laboratories. She earned her Ph.D. at the MIT Artificial Intelligence Laboratory. Sidner is a fellow and past councilor of AAAI,

as well as having served as president of the Association for Computational Linguistics, chair of the 2001 and program cochair of the 2006 International Conference on Intelligent User Interfaces, and cochair of the 2004 SIGdial Workshop on Discourse and Dialogue, and general chair of the 2007 NAACL Human Language Technology conference. Her e-mail address is candy.sidner@baesystems.com.