

Agents and GUIs from Task Models

Jacob Eisenstein, Charles Rich

TR2002-16 March 14, 2002

Abstract

This work unifies two important threads of research in intelligentagents.

ACM International Conference on Intelligent User Interfaces (IUI)

This work may not be copied or reproduced in whole or in part for any commercial purpose. Permission to copy in whole or in part without payment of fee is granted for nonprofit educational and research purposes provided that all such whole or partial copies include the following: a notice that such copying is by permission of Mitsubishi Electric Research Laboratories, Inc.; an acknowledgment of the authors and individual contributions to the work; and all applicable portions of the copyright notice. Copying, reproduction, or republishing for any other purpose shall require a license with payment of fee to Mitsubishi Electric Research Laboratories, Inc. All rights reserved.

Submitted October, 2001; revised and released February 2002.

Agents and GUIs from Task Models

Jacob Eisenstein

USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA, 90292, USA
jacob@isi.edu

Charles Rich

Mitsubishi Electric Research Laboratories
201 Broadway
Cambridge, MA, 02139, USA
rich@merl.com

Abstract

This work unifies two important threads of research in intelligent user interfaces which share the common element of explicit task modeling. On the one hand, longstanding research on task-centered GUI design (sometimes called model-based design) has explored the benefits of explicitly modeling the task to be performed by an interface and using this task model as an integral part of the interface design process. More recently, research on collaborative interface agents has shown how an explicit task model can be used to control the behavior of a software agent that helps a user perform tasks using a GUI. This paper describes a collection of tools we have implemented which generate both a GUI and a collaborative interface agent from the same task model. Our task-centered GUI design tool incorporates a number of novel features which help the designer to integrate the task model into the design process without being unduly distracted. Our implementation of collaborative interface agents is built on top of the COLLAGEN middleware for collaborative interface agents.

Keywords

Model-based interface design, task models, software agents, collaboration.

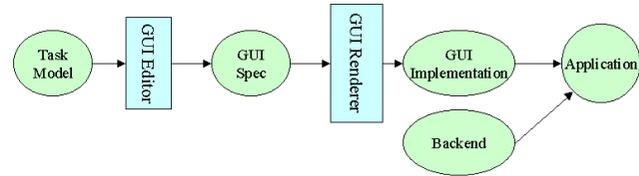
INTRODUCTION

Figure 1 at the right gives an overview of our work. Figure 1(a) shows the typical task-centered graphical user interface (GUI) design process. Since GUIs are usually intended to support some particular set of human tasks, the first job of the designer using this methodology is to formalize these intentions as an explicit task model. Eliciting and formalizing task models is itself an arduous job, which requires supporting tools and methodology [16, 10, 3]. In this work, however, we take the task model as our starting point. Also, notice that even though this approach automates the implementation of the GUI, the application developer still needs to implement the “backend” processing to make a complete application.

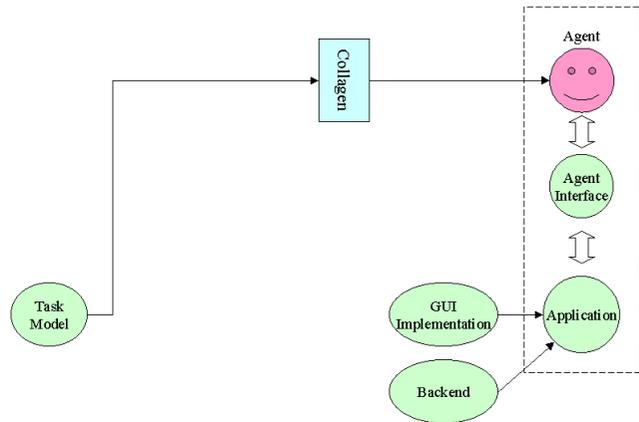
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

IUI'02, January 13-16, 2002, San Francisco, California, USA.

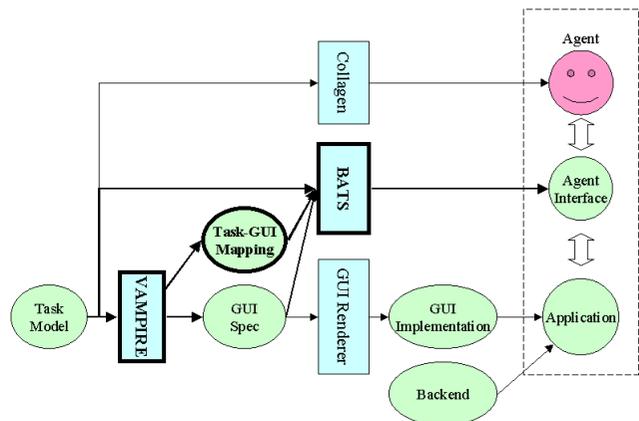
Copyright 2002 ACM 1-58113-459-2/02/0001...\$5.00



(a) Typical task-centered GUI design process.



(b) COLLAGEN-based collaborative interface agent.



(c) Combined architecture (with new elements in bold).

Figure 1: Architecture for generating a GUI and a collaborative interface agent from the same task model.

Figure 1(b) shows the architecture of a COLLAGEN-based collaborative interface agent. COLLAGEN is Java middleware for building collaborative interface agents. COLLAGEN will be discussed briefly later in this paper; please see the references for more detail [13, 12].

Roughly speaking, COLLAGEN takes a task model and gives you an agent which can collaborate with a user in performing the given tasks. For example, in our demonstration system, we have configured COLLAGEN to produce an agent which teaches novice users how to use the application on their first encounter with it (teaching is a kind of collaboration [14]).

Notice that even though COLLAGEN automatically gives you an agent, the application developer still needs to manually implement the “agent interface,” i.e., the program interface through which the agent can interact with the application and observe the user’s interaction with the application. From experience, we have found that implementing this program interface can be a significant barrier to using COLLAGEN. Our combined architecture, described below, eliminates this barrier.

Figure 1(c) shows the combination of (a) and (b), with three new elements. The first new element, called VAMPIRE, is a novel GUI editor we have developed, which generates both a GUI specification and (the second new element) an explicit description of the task-GUI mapping, which can be used by other tools. The third new element, called BATS, automatically produces the agent interface needed for a COLLAGEN agent, given a task model and the task-GUI mapping.

In summary, notice that in Figure 1(c) the only inputs the designer/developer needs to provide are the task model (constructed using tools which are beyond the scope of this work), the graphical and layout design decisions (provided interactively using VAMPIRE), and the backend implementation. Furthermore, since our entire system is implemented using the Java Beans(tm) event architecture, connecting the backend processing with the GUI implementation does not require modifying any automatically generated code.

The following sections of this paper discuss each element of Figure 1(c) in more detail, using a simple File Transfer Protocol (FTP) application we have developed as an example.

We hope that this work will, by its synergy, advance both the practice of task-centered GUI design and the application of collaborative interface agents, because:

- GUI designers will receive a greater return on their investment in task modeling by getting an collaborative interface agent “for free,” and
- the problem of integrating agents with applications [5] is eliminated by automating the creation of the agent interface.

```

top FTP; // toplevel goal

recipe DoFTP achieves FTP {
  // totally ordered steps
  step Login login;
  step Connect connect;
  repeatable step DownloadFile download;
  step Disconnect disconnect;
  constraints {
    login precedes connect;
    connect precedes download;
    download precedes disconnect;
    login.address == download.address; }}

act Login { // non-primitive action
  parameter StringTerm address; }

recipe DoLogin achieves Login {
  // unordered steps
  step EnterAddress address;
  step EnterName name;
  step EnterPassword password;
  constraints {
    address.address == achieves.address; }}

act Download { // non-primitive action
  parameter StringTerm address; }

recipe DoDownload achieves Download {
  // partially ordered steps
  repeatable step SelectFile select;
  optional repeatable step AsciiMode ascii;
  optional repeatable step BinaryMode binary;
  step DownloadFile download;
  constraints {
    select precedes download;
    ascii precedes download;
    binary precedes download;
    download.address == achieves.address; }}

// primitive actions

manipulation EnterAddress {
  parameter StringTerm address; }

manipulation EnterName {
  parameter StringTerm name; }

manipulation EnterPassword {
  parameter StringTerm password; }

manipulation SelectFile {
  parameter StringTerm file; }

manipulation DownloadFile {
  parameter StringTerm address;
  parameter StringTerm file; }

manipulation AsciiMode;

manipulation BinaryMode;

manipulation Connect;

manipulation Disconnect;

```

Figure 2: Complete task model for FTP example (reserved words and comments in italics).

TASK MODELS

Task model representations come in many different variations. In this work, we use the task model representation provided by COLLAGEN, which is a fairly generic hierarchical goal decomposition representation. Although the details of our implementation depend, of course, on the details of this task model representation, we believe our basic paradigm is applicable to any task model representation. To show this, we have included “import” functionality that allows our software to utilize task models written in the ConcurTaskTrees notation [10].

Figure 2 shows the complete task model for a small FTP example, exactly as it is input to COLLAGEN. The syntax of this representation is an extension of Java (implemented by a pre-processor), in which each primitive and non-primitive action type and each goal decomposition rule (*recipe*) is defined as a Java class.

This task model represents the designer’s concept of what needs to be accomplished in the FTP task, abstracted away from the details of how a particular GUI to support the task will be designed. The task model is organized hierarchically. At the top level, the FTP task is broken down into four totally ordered steps: logging in (*login*), connecting to a server (*connect*), downloading one or more files (*download*), and disconnecting (*disconnect*). Each of the non-primitive subtasks is further decomposed by recipes until we reach the primitive actions listed at the bottom of the figure. In this model, there is only a single recipe which achieves each non-primitive action type; in general, there may be more than one.

COLLAGEN’s task model representation also supports a number of other features, including optional and repeatable steps, temporal order and equality constraints, preconditions, and postconditions, some of which are illustrated in Figure 2.

TASK-CENTERED GUI DESIGN

A wide spectrum of approaches have been explored for incorporating task models into the GUI design process. At one end of the spectrum are informal, non-computational approaches [4], which encourage GUI designers to think about the desired task structure when designing a GUI. At the other end of the spectrum are completely automated approaches in which the GUI implementation is automatically generated from a formal task model. Most recent work in this area, however, has been somewhere in the middle.

Figure 1(a) illustrates the architecture of the typical task-centered GUI design process, starting with a formal task model (such as Figure 2). The process begins with a *GUI editor*, which allows the designer to define the look and feel of the GUI in a manner that is somehow guided by the initial task model. The output of this stage is a *GUI specification*, which is an abstract, platform-neutral description of the details of the GUI. This specification is then provided to a *GUI renderer*, which produces (either by code-generation or

at run time) the actual implementation of the GUI in some target environment, such as Windows(tm) or Swing(tm). A key benefit of separating the specification of the GUI from its implementation is that future changes to the GUI can more easily be made to the specification (using the editor), rather than by directly modifying the implementation.

A typical example of this architecture is MOBILE [11], a WYSIWYG GUI editor in which the choice of graphical interactors and the navigational structure of the GUI is guided by the features and structure of the task model. Another example is Teallach [2], a non-WYSIWYG, model-based GUI development environment in which graphical interactors are explicitly linked to elements of the task model. During the design process, Teallach designers can “verify” that the GUI they have designed is consistent and complete with respect to the task model.

In the next section, we describe our own interactive task-centered GUI design tool, called VAMPIRE. Although we have tried to make VAMPIRE as easy to use as possible, the fact remains that building the task model required for such a tool is an inherently difficult process. We believe this is one of the reasons why there has been so little commercial use of this kind of technology. We hope that the new run-time benefits that we will describe later in this paper will make it more attractive for developers to invest the effort to build formal task models.

The VAMPIRE Editor

VAMPIRE (for Visual Model-based Pick-and-place Interface Editor) is a task-centered GUI editor we have designed for use with our “agents and GUIs from task models” paradigm. VAMPIRE will be described in detail in a future paper. The key feature of VAMPIRE from the standpoint of this paper is that, in addition to the GUI specification, it generates an explicit *task-GUI mapping* as output (see Figure 1(c) and Figure 4).

Figure 3 is a snapshot of VAMPIRE being used to construct the GUI shown in Figure 5 for the FTP task model (Figure 2). VAMPIRE tries to provide a look and feel that is as close as possible to a traditional WYSIWYG layout tool, while at the same time keeping the designer focused on the task model. The left half of the screen displays the hierarchical task model as a tree. The designer uses the layout area on the right half of the screen to perform typical WYSIWYG design actions, such as selecting interactors from a palette, placing them in windows, and parameterizing them appropriately. However, throughout the process, the designer is also presented with feedback and default suggestions that pertain to the task model.

For example, whenever the designer clicks on an unimplemented element in the task tree, VAMPIRE recommends an interactor by highlighting that interactor in the toolbar palette along the bottom of the right half of the screen. The designer

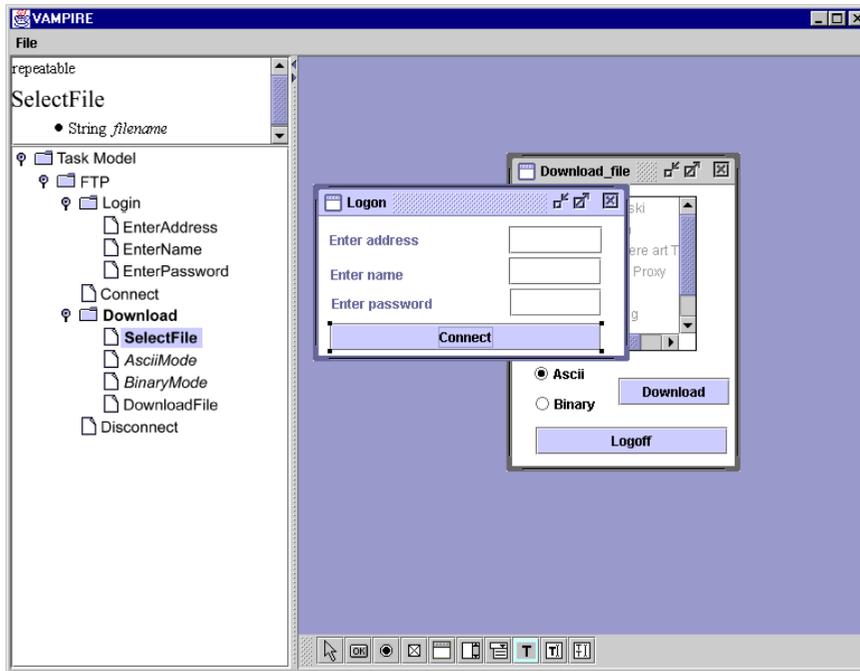


Figure 3: Screen shot of VAMPIRE task-centered GUI design tool being used to construct FTP example.

can now click directly in the layout area to place the recommended interactor. Alternatively, she can ask for a second recommendation by clicking on the task again, or simply go to the toolbar and select whatever interactor she prefers. In any case, the designer must indicate the pertinent task element before placing the interactor on the screen. VAMPIRE thus incrementally builds an explicit mapping between elements of the task model and elements of the GUI. Graphical elements can also be customized in the usual ways (color, font, left vs. right click, etc.) after they are placed.

VAMPIRE uses the task-GUI mapping to provide feedback to the designer throughout the design process. When the designer selects an element of the task tree, the associated GUI elements are automatically selected (highlighted), and vice versa. In addition, the designer can verify at any time that each task is mapped to at least one GUI element.

The implementation of VAMPIRE is totally XML-based. VAMPIRE's input is an XML version of COLLAGEN's task model notation. The GUI specification produced by VAMPIRE is written in UIML[1]. We have written a UIML renderer that produces a graphical user interface at run-time, and provides a software API to the generated GUI. The task-GUI mapping is implementing using XLink [6], an XML language specifically designed for specifying links between XML models.

COLLABORATIVE INTERFACE AGENTS

The basic intuition underlying collaborative interface agents is that interaction between a human and a computer can be greatly improved if the computer has a model of tasks that the

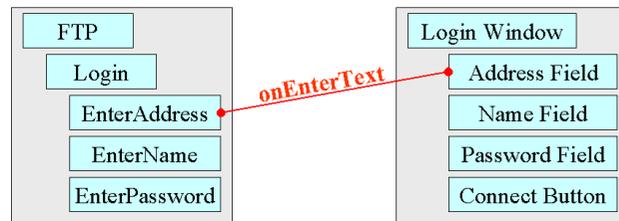


Figure 4: Example of a link in the task-GUI mapping produced by VAMPIRE.

human is trying to perform. A *collaborative interface agent* is a software agent that collaborates with the human user of a (typically complex) computer interface.

The concept of collaboration covers a wide range of kinds of interaction, depending on the relative knowledge and initiative of the user and agent. At one extreme (low user knowledge and initiative), a "first-encounter agent" might use the task model to walk the user through a step-by-step demonstration of how to use an new interface. (We describe such an agent for the FTP task below.) At the other extreme, a collaborative assistant for "power users" might use the same task model to automatically finish tasks that the user has started.

COLLAGEN Middleware

COLLAGEN(for COLLABorative AGENT) [13, 12] is Java middleware developed at MERL to make it easier to implement collaborative interface agents. The architecture of systems built with COLLAGEN is shown in Figure 1(b). Among other things, COLLAGEN provides a generic implementation of discourse interpretation, plan recognition, and plan generation algorithms, all of which take a given task model as data.

However, the developer using COLLAGEN still has to hand-code the *agent interface*, which allows the agent to observe the user's interactions with the application and to interact with the application itself. (In internal COLLAGEN documentation, this component is referred to as the "application adapter.") Basically, the code in the agent interface is a bridge between the primitive action types of the task model and the specific elements that achieve them in the given GUI implementation. This code can be quite onerous to write, particularly if the author of this code is not the same as the author of the GUI implementation. Combining COLLAGEN with task-centered GUI design, however, provides the opportunity for automating the writing of this code, as described in the next section.

The BATS Code Generator

BATS (for Bridging Agents, Tasks, and Software) is a batch program (see Figure 1(c)) that takes three XML documents as input: a task model, a task-GUI mapping, and a GUI specification. BATS' output is a Java implementation of the agent interface needed for a COLLAGEN agent. This automatically generated code connects with COLLAGEN's agent code and the GUI implementation using events.

For example, for the FTP application, the task-GUI mapping includes a link between the `EnterAddress` primitive action type in the task model and the text input field labeled "Server address" in the GUI specification. When the user finishes entering text in this field, the agent interface code generates an event, received by the agent, which includes an instance of `EnterAddress` with the entered text as its parameter. Conversely, when the agent wants to perform `EnterAddress` (e.g., as part of a tutorial demonstration), it sends an instance of `EnterAddress` to the agent interface, which translates it into the appropriate GUI events to cause the associated parameter text to be entered into the "Server address" field.

Another function of the agent interface code is to support the agent pointing at the GUI, as illustrated in Figure 5. From the standpoint of the agent, all that is required to produce the pointing behavior shown in Figure 5 is an abstract call of the form "move the hand to where the `EnterName` action takes place." The agent interface code takes care of layout details.

Agent says "Would you like to learn how to DoFTP?"

User says "Yes."

Agent says "The first step of DoFTP is to achieve Login with an address."

Agent says "First, perform EnterAddress with an address."

Agent points to location of EnterAddress.

User performs EnterAddress with 'www.merl.com'.

Agent says "Right."

Agent says "Next, now you should perform EnterName with a name."

Agent points to location of EnterName.

User performs Connect.

Agent says "Whoops, it's too soon to do that."

User says "What next?"

Agent says "As I said, you should perform EnterName with a name."

User performs EnterName with 'jacob'.

Agent says "Right."

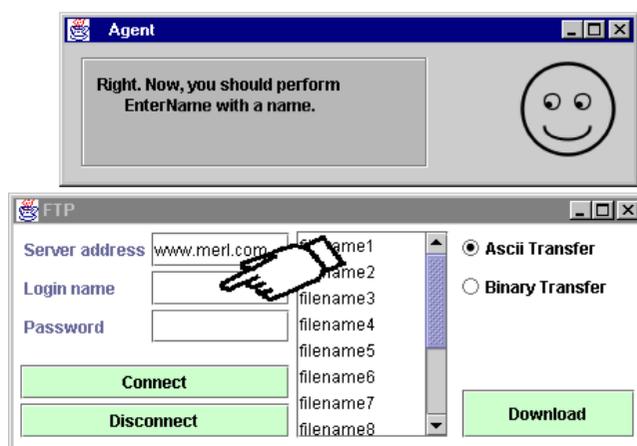


Figure 5: Dialogue and screen shot for automatically generated pedagogical agent.

Agent says "Would you like to learn how to FTP?"

User says "Yes."

Agent says "The first step of FTP is to provide login information."

Agent says "First, type in the server address."

Agent points to where to type in the server address.

User types in 'www.merl.com' as the server address.

Agent says "Right."

Agent says "Next, type in your login name."

Agent points to where to type in your login name.

User presses the connect button.

Agent says "Whoops, it's too soon to do that."

User says "What next?"

Agent says "As I said, type in your login name."

User types in 'jacob' as the login name.

Agent says "Right."

Figure 6: Dialogue of Figure 5 with language generation templates added to task model.

PACO: A Pedagogical Agent

COLLAGEN provides several different versions of its generic interface agent, which vary in terms of the kind of collaboration they support. For our FTP demonstration, we have chosen a pedagogical version of the COLLAGEN agent, called PACO (for Pedagogical Agent in COLLAGEN) [14]. Figure 5 shows part of a typical interaction with this agent. We want to emphasize that the interaction shown in this figure is obtained with no programming or developer input other than the task model in Figure 2, interaction with VAMPIRE to lay out the GUI, and a small amount of “backend” code.

In general, PACO guides the user through a collection of pre-defined example scenarios. Each scenario involves achieving the (or one of the) top level goals of the task model under some set of initial conditions. PACO keeps track of which parts of a task model the user already knows how to do, and offers step-by-step instructions on how to perform the parts of the task that the user does not yet know. PACO also offers confirmation and encouragement when the user does something correctly and corrections when a mistake is made.

Since our use of PACO in this demonstration is targeted at a user’s first encounter with the FTP interface, we simply use PACO’s default scenario, which is to achieve the first top level goal in the default startup state of the application. Note, however, that we are assuming that the user is familiar with the basic domain concepts underlying the FTP application, such as the concept of a server, downloading files, etc. Teaching domain concepts is a more ambitious task; PACO is focused only on teaching *procedural* knowledge.

The dialogue in Figure 5 also illustrates some simple natural-language capabilities of PACO (and COLLAGEN generally), which make use of off-the-shelf speech recognition and text-to-speech software (IBM ViaVoice). PACO’s speech recognition is trivial, since the user can only say a few simple things, like “Yes,” “Ok,” and “What next?”.

For language generation, COLLAGEN provides a simple method of annotating task models with language generation templates. The task model in Figure 2 includes no such annotation, in which case the agent uses default templates to generate utterances of the form shown in Figure 5, such as “First, perform EnterAddress with an address.” However, we can easily associate a generation template (*gloss*) with EnterAddress as follows:

```
manipulation EnterAddress {
  parameter StringTerm address;
  gloss{
    "type in {'#address' as} the server address"
  }}

```

Given this definition, the agent would say “First, type in the server address” as shown in Figure 6. In practice, we would add similar templates to all the elements in a task model.

RELATED WORK

Lieberman [5] gives an excellent, comprehensive discussion of the problems of integrating interface agents with applications. A novel approach, along very different lines from our own, is taken by Zettlemoyer et al. [17]. Rather than using the software implementation of the GUI, they apply machine vision techniques directly to the visual appearance of the GUI. They are able to detect the position and type of various typical interactors, which they then can control by generating the appropriate graphical events, such as mouse clicks. The task structure and its relation to the GUI are not explicitly modeled.

Several researchers have developed techniques for automatically generating help facilities from some form of task/interface model. For example, Sukaviriya and Foley [15] developed a GUI design system which generated animated tutorials to show a user how to accomplish various tasks using the designed GUI. While their research does not support the same flexibility of interaction as our collaborative agent, it was an early demonstration of the possibilities of exploiting a task model to provide run-time guidance for the user.

Moriyon et. al [7] built a system which allows users to ask questions about a GUI, such as “What commands are available?” or “What will happen if I click here?”. Their system does not, however, include a task model, and consequently cannot offer the kind of step-by-step guidance that we provide.

Pangoli and Paterno [9] take a task-centered approach, allowing users to ask questions about how to perform a complex task, and answering these questions in natural language, like our agent. However, since their help system is not integrated with the application, it doesn’t know whether the user has followed its advice properly. It depends on the user to describe the current system state.

Palanque et. al [8] have developed an agent which answers questions about why a GUI widget is disabled, and generates a sequence of steps which will enable the widget. They do not discuss how to integrate this agent with an application.

FUTURE WORK

An effective collaborative agent requires a range of knowledge beyond what is currently represented in our task models. For example, it should know which tasks are most important to the user, so that it can help with those first. It should also know which tasks are most difficult, so that it can be more proactive in assisting with them. Extending COLLAGEN’s task model in these directions, and making it easy for users to provide this kind of information, are important directions for our future research. This kind of knowledge could also improve COLLAGEN’s plan recognition behavior.

We also need to extend the expressive power of our user models. Currently, PACO maintains only a very simple “overlay” model: For each task and subtask, PACO stores a few flags

indicated whether the task has been demonstrated to the user, whether the user has performed it correctly, and whether the user has performed it correctly without prompting. Improved user modeling could include information about a user's personality and learning style, as well as summary characterizations of their history with the system.

Improved user modeling could affect both the GUI design and the behavior of a collaborative agent. For example, a designer might emphasize transparency over efficiency when designing an interface for novice users. A collaborative agent for novices might be more active (take more initiative) than one for experienced users.

Another research direction is to provide a collaborative, task-centered interface *without* a personified agent. For example, we could imagine adding buttons to a GUI labeled "Where am I?", "What can I do next?", "How do I do x ?", etc. (see [12]). Such buttons might become a standard feature of future GUI's, just as the undo button is a standard feature of current state-of-the-art GUI's.

Finally, at the most fundamental level, this research is about the relationship between user interface design and intelligent agents. We would like to explore how the design of conventional graphical user interfaces is influenced by the presence of a collaborative agent. Should the GUI be designed differently, when there is a collaborative agent to help the user?

We would also like to explore how best to incorporate a collaborative agent into a coherent overall interface design. Should there be a visual representation of the agent at all? Should the agent have its own window? How should the user communicate with the agent and vice versa? For example, we are currently experimenting with multi-modal approaches, in which user-agent communication via speech is mixed with conventional GUI manipulation.

CONCLUSION

At a concrete level, we have described two new software technology components that bridge the gap between task-centered GUI design and collaborative interface agents. The first component, VAMPIRE, is a GUI design tool that captures the relationship between a task model and a GUI as a natural by-product of using the tool. The second component, BATS, is a code generator that relieves agent developers of the burden of writing the program interface between an application and an agent.

Beyond these concrete contributions, we also hope that this work will provide an impetus to task-centered approaches to software in general.

Acknowledgements

Thanks to Andrew Garland, Neal Lesh, Candace Sidner, and Steve Wolfman for invaluable feedback and discussions on this research.

REFERENCES

1. M. Abrams, C. Phanouriou, A. Batongbatal, S. Williams, and J. Shuster. UIML: An appliance-independent XML user interface language. *Computer Networks*, 31:1695–1708, 1999.
2. P. Barclay, T. Griffiths, J. McKirdy, N. Paton, R. Cooper, and J. Kennedy. The teallach tool: Using models for flexible user interface design. In *CADUI99: Computer-Aided Design of the User Interface*. Kluwer, 1999.
3. A. Garland, K. Ryall, and C. Rich. Learning hierarchical task models by defining and refining examples. In *First Int. Conf. on Knowledge Capture*, Victoria, B.C., Canada, Oct. 2001.
4. C. Lewis and J. Rieman. Task-centered user interface design. <http://hcibib.org/tcuid/>, 1994.
5. H. Lieberman. Integrating user interface agents with conventional applications. In *Intelligent User Interfaces*, pages 39–46. ACM Press, January 1998.
6. E. Maler and S. DeRose. XML linking language (XLink). Technical report, World Wide Web Consortium, March 1998.
7. R. Moriyon, P. Szekely, and R. Neches. Automatic generation of help from interface design models. In *Proceedings of 1994 Conference on Human Factors in Computing Systems (CHI)*, pages 225–231, 1994.
8. P. Palanque, R. Bastide, and L. Dourte. Contextual help for free with formal dialog design. In *Fifth International Conference on Human-Computer Interaction*. Elsevier Science Publisher, August 1993.
9. S. Pangoli and F. Paterno. Automatic generation of task-oriented help. In *ACM Symposium on User Interface Software and Technology*, pages 181–187, 1995.
10. F. Paterno, C. Mancini, and S. Meniconi. Concurtasktrees: A diagrammatic notation for specifying task models. In S. Howard, J. Hammond, and G. Lindgaard, editors, *Human-Computer Interaction INTERACT*, pages 362–369. Chapman and Hall, 1997.
11. A. R. Puerta, E. Cheng, T. Ou, and J. Min. MOBILE: User-centered interface building. In *CHI: Human Factors in Computing Systems*, pages 426–433. ACM Press, May 1999.
12. C. Rich, C. Sidner, and N. Lesh. Collagen: Applying collaborative discourse theory to human-computer interaction. *AI Magazine*, 22(4), 2001. Special Issue on Intelligent User Interfaces. To appear.
13. C. Rich and C. L. Sidner. COLLAGEN: A collaboration manager for software interface agents. *User Modeling and User-Adapted Interaction*, 8(3-4):315–350, 1998.

14. J. Rickel, N. Lesh, C. Rich, C. L. Sidner, and A. Gertner. Using a model of collaborative dialogue to teach procedural tasks. In *10th International Conference on Artificial Intelligence in Education*, May 2001.
15. P. Sukaviriya and J. D. Foley. Coupling A UI framework with automatic generation of context-sensitive animated help. In *UIST:Symposium on User Interface Software and Technology*, pages 152–166, 1990.
16. R. Tam, D. Mulsby, and A. Puerta. U-TEL: A tool for eliciting user task models from domain experts. In *Intelligent User Interfaces*, pages 77–80. ACM Press, January 1998.
17. L. Zettlemoyer, R. S. Amant, and M. Dulberg. Application control through the user interface. In *International Conference on Intelligent User Interfaces*. ACM Press, January 1999.