A REASONED APPROACH TO ERROR HANDLING
Position Paper on Work-in-Progress

Tamara Babaian, Wendy Lucas
Bentley College, Computer Information Systems Department, Waltham, MA, 02452, USA
Email: tbabaian@bentley.edu, wlucas@bentley.edu

Keywords: Planning, reasoning, error handling, collaboration, ERP systems.

Abstract: It is widely acknowledged that Enterprise Resource Planning (ERP) systems are difficult to use. Our own studies have revealed that one of the largest sources of frustration for ERP users is the inadequate support in error situations afforded by these systems. We propose an approach to error handling in which reasoning on the part of the system enables it to behave as a collaborative partner in helping its users understand the causes of errors and, whenever possible, make the necessary corrections. While our focus here is on ERP systems, this approach could be applied to any system for improving its error handling capabilities.

1 INTRODUCTION

Enterprise Resource Planning (ERP) systems present many challenges to their users. Not least among these is the typically inadequate support provided in error situations. Even trained users can be stymied by the lack of information conveyed in error messages concerning the source of an error and possible alternative actions for how it can be handled. Information on system-provided data that was entered by another user is virtually impossible to track down without the specialized knowledge possessed by highly trained power users. System resources that could help users avoid some types of errors, such as the calendar function for entering correctly formatted dates, are available if one knows where to find them but are not offered up by the system in times of need.

We propose an innovative approach to error handling that is inspired by the collabororative view of system-user interaction (Terveen, 1995; Grosz, 1996; Shieber, 1996). This view specifies that the system must act as a partner to its users by supporting them in the increasingly complex environments of modern applications (Grosz, 2005). Note that this stream of research is different from Computer-Supported Cooperative Work (CSCW), which is concerned with computing technology that supports human collaboration.

The novelty of our approach to error handling comes from the application of reasoning capability for enabling the system to act as a collaborative partner in times of need. To that end, the system must be knowledgeable about the resources it has at its disposable and must reason about which ones to offer and when to offer them to its users. The system must, therefore, be able to plan a course of action in response to error conditions and then carry out that plan. We have chosen to focus on ERP systems because of the known paucity of their error handling capabilities, but this approach can be applied to any system for helping users in error situations.

This work is part of a larger project focusing on achieving significant improvements in ERP usability, where usability is defined as “the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use” (ISO 9241-11, 1998). As part of this project, we conducted a field study of ERP system users in a Fortune 500 company and categorized and described the usability issues they encountered (Topi et al., 2005). Our field study revealed that error handling was the most painful and time-consuming aspect of daily operations, resulting in users creating hundreds of pages of informal documentation to support them in their system usage (Topi et al., 2006). The special role of a power user was also created as an additional support mechanism. The majority of the “power” users’ efforts were directed at diagnosing and resolving the most difficult and obscure errors faced by other users who sought their help.

One of the major frustrations facing users comes from the lack of transparency inherent to ERP systems. Data-to-process and process-to-process relationships play a critical role in defining system functions, but they are too complex and too numerous to be known in their entirety by an individual user or user group. The support an ERP system affords its users can be significantly strengthened by improving error diagnosis and recovery techniques based on its knowledge of such processes and relationships. From the collaborative standpoint, the system can provide better support to its users by sharing that knowledge effectively.

The next section of this paper discusses related work. We then describe our proposed approach for improving support to users in error situations. This is followed by our conclusions and directions for future work.

2. RELATED WORK

Several streams of research have examined the mechanisms for supporting a user's operations based on the explicit knowledge embedded within the system itself. Most notably, examples of these mechanisms have come from research on collaborative interfaces and tutoring systems. The fact that these two streams have similarities in their methodologies is not surprising, since the interaction between a tutor and her student is a type of collaborative activity (Davies et al., 2001).

Collaborative interfaces view the process of humans using a software system as a collaborative effort in which the human user and the software system are partners working towards achieving shared goals. These partners have different natural strengths and, therefore, the effectiveness of their collaboration depends to a large extent on the allocation of the subtasks that builds on the respective strengths of the system and the human user (Shieber, 1996).

To achieve successful collaboration, the partners must also maintain awareness of the overall strategy for achieving the goal and understand how the steps taken by each partner contribute to that strategy. To enable this kind of awareness on the part of the system requires equipping it with the knowledge of the structure of the tasks for which it was designed. This structure is commonly referred to as a task model. Such a model can be specified using various means: it can be either implicitely built into the system's processes or explicitly specified in a declarative fashion and made available to the system as a component. Many intelligent systems (e.g., Johnson et al. 2003, Eisenstein and Rich, 2002, Traum et al., 2003) use explicit representation of task models as *plans* (Russell and Norvig, 2002). Generally speaking, a plan is a structure that combines action descriptions, specified via preconditions and effects, in a way that guarantees achievement of a specified goal or a set of goals as a result of executing that plan.

Being able to reason about the structure of the tasks and monitor the progress made as a result of system-user interaction significantly enhances the collaborative strengths of a system, as demonstrated by the intelligent systems cited above. Rich et. al. (2001) describe several systems that provide assistance to the user via a dialog-based interface to applications. Examples include a VCR-configuration assistant and a tutoring system for teaching a user to operate a gas turbine engine. The task models, represented as hierarchical "recipes" for the actions that can be taken, enable these systems to monitor and recognize the steps taken by the user, interpret user actions within the task context of the interaction, reason about the next steps in the process, and suggest alternatives.

Eisenstein and Rich (2002) present a middleware package that uses natural language to provide automated help generation for simple input-form GUIs. The help messages are generated based on the explicit representation of the tasks that are tied to the related graphical components of the interface.

The Writer's Aid system (Babaian et. al., 2002) demonstrates how reasoning about actions and planning can be used to provide capabilities for the robust and flexible autonomous operation of a system in support of a user's goals. Writer's Aid responds to the user's requests for bibliographic information by autonomously creating and executing a plan for searching and delivering bibliographic items such as papers, citation records, and reference keys. What distinguishes Writer's Aid from the above examples is its ability to dynamically construct and execute a plan from any situation, given only partial information about the world and a model of individual actions.

While automated planning has been used to support human-computer collaboration, what distinguishes the proposed approach is the use of automated planning for creating an error-recovery plan. The approach described below will enable not only monitoring of the progress of the task based on a fixed set of task models, but also creating plans dynamically in support of error recovery. These plans will use a combination of system actions and

Action: **DisplayCalendar()**	Action: **EnterTextIntoField (newtext, field)**
Precondition: *none*	Precondition: *Displayed(field)*
Effect: *if UserMadeSelection(d) then DateFormat(d)*	Effect:*If In(text, field) then ¬In(text, field)*
	In(newtext, field)

Figure 1. Schematic action descriptions to be used by a planner.

user actions. Thus, they will rely significantly on the model of such actions available to the system. The next section presents an example of how this will be accomplished.

3. REASONING IN SUPPORT OF ERROR HANDLING

We propose a novel method based on automated planning for overcoming difficulties in error reporting and recovery. Automated planning is a framework for reasoning about goal achievement via actions (Russell and Norvig, 2002). Our approach will use a planning framework and algorithms to enable error handling that goes beyond the reporting of errors: it will help the user resolve an error by suggesting a dynamically constructed course of corrective action.

As an example of a simple error, consider the following situation: a novice user is entering a delivery date in the date field and gets an error message stating that the date value "12/3/2008" cannot be accepted because it is in an incorrect format. Having no familiarity with the particular ERP system's date specification format, she reads the system help on the topic and, after a couple of attempts, manages to correct the error. Upon hearing about her troubles, her colleague points out a calendar feature in the interface that allows selection and input of the date simply by clicking on it in the system-supplied calendar.

Let us consider how a planning framework can be applied for supporting the user in such error situations. This framework specifies actions as having preconditions and effects using a formal logic-based language. The initial conditions and the goal are also specified using the same language. A plan is a sequence of actions that are executable, i.e., that have their preconditions satisfied at the time they are executed, and achieve the specified goal. To apply the planning framework, each error situation will be identified with a condition that, if not satisfied by the user's actions, will trigger an error.

For instance, to have a date entered correctly in an input field requires satisfying the goal **G:** In (text, field) & DateFormat(text) is true, i.e., the text entered in the field is in the correct date format. To enable the system to develop a course of action for correcting the error, we will also embed the action descriptions corresponding to a selected set of system functions. For example, the function for displaying a calendar and enabling a selection from it could have the specification depicted in Figure 1.

The DisplayCalendar action has no preconditions because it can be executed at any time when the system is running. The conditional effect states that if the user makes a selection, denoted here by *d*, the action will produce *d* in the appropriate date format. The second action description in the same figure defines the system's action of automatically placing some new text into a given field. The precondition of this action is that the field must be displayed in a visible location. There are two effects: first, whatever text was occupying the designated field before the action will not be there after the action is completed, and second, the field will contain the new text.

Returning to our example, the system will establish that *In("12/3/2008", field)* is true, but the text "12/3/2008" is not formatted correctly, i.e., the negation *¬DateFormat("12/3/2008")* is true and therefore goal G is not achieved. The system will detect an error and the planner will then create a plan that will establish goal G given the action specifications embedded in the system and the description of the initial state **I:** *In("12/3/2008", field),¬DateFormat("12/3/2008")*, *Displayed(field)*.

A plan that could achieve the goal involves two system actions:

1. **DisplayCalendar** – produce a date d such that *DateFormat(d)* if the user makes a selection
2. **EnterTextIntoField (d, field)** – enter *d* into the designated field.

The effect of executing a plan consisting of these two actions and the user entering a valid date would be the satisfaction of goal G. If the date is invalid, in addition to displaying an error message, the system would display a calendar to enable the user to make a selection, thus completing the process of specifying the date correctly.

The above example was deliberately kept simple for the sake of brevity, but it demonstrates the same reasoning mechanism we will use to support errors caused by mismatches between data within different process interfaces. Because the system has knowledge about the context for and linkages between each data element, it will do more than just display an error message; it will identify the task interface that will offer the user a choice of only appropriate values or present choices for possible courses of action to address the error. Even if the user does not have the authority to perform a particular corrective action, she will at least learn about the necessary steps and could ask the appropriate person to perform them.

The benefits of using the planning framework in error recovery are the flexibility and extensibility it affords. Once the planner is embedded within the system, additional functions and/or error diagnostics can be specified in a declarative fashion through textual descriptions such as those presented here. The planning engine will be able to recognize new error conditions and create new solution methods that utilize the newly added functions as appropriate. No modifications to the application code will be needed, as opposed to approaches based on "hardcoding" all responses to error situations, which require modifying significant parts of the error-handling code throughout the application.

4. CONCLUSIONS

We have described a novel approach for error handling based on the application of reasoning capability within a planning framework. This approach enables the system to act as a collaborative partner to its users in helping them navigate their way through error situations. In response to an error, the system will diagnose the cause and dynamically construct and execute a plan for informing the user about the underlying causes of an error and, whenever possible, for guiding her through corrective actions.

The next stage in this research is to implement this approach in an ERP prototype interface for handling a range of realistic error situations. This prototype will then be used for evaluating the effectiveness of our design interventions.

REFERENCES

Babaian, T., Grosz, B. J., & Shieber, S. M., 2002. A writer's collaborative assistant. In *Proceedings of IUI'2002*, 7-14. ACM Press.

Davies, J. R., Gertner, A. S., Lesh, N., Rich, C., Sidner, C. L., & Rickel, J., 2001. Incorporating tutorial strategies into an intelligent assistant. In *Proceedings of IUI''2001*, 53-56, ACM Press.

Eisenstein, J. & Rich, C. 2002. Agents and GUIs from task models. In *Proceedings of IUI'2002*, 47--54, ACM Press.

Grosz, B. J., 1996. AAAI-94 presidential address: Collaborative systems. *AI Magazine, 17*(2), 67-85.

Grosz, B. J., 2005. Beyond Mice and Menus. In *Proceedings of the American Philosophical Society*, 149(4), 529-523.

ISO 9241-11, 1998. Ergonomics requirements for office work with visual display terminals, part 11 - guidance on usability. *International Standards Organization.*

Johnson, W. L., Shaw, E., Marshall, A., & LaBore, C., 2003. Evolution of user interaction: the case of agent Adele. In *Proceedings of the 8th international Conference on intelligent User interfaces,* 93-100. ACM Press.

Rich, C., Sidner, C. L., & Lesh, N., 2001. Collagen: applying collaborative discourse theory to human-computer interaction. *AI Magazine.* 22, 4 (Oct. 2001), 15-25.

Russell, S. & Norvig, P., 2002. *Artificial Intelligence: a Modern Approach*. Prentice Hall.

Shieber, S. M., 1996. A call for collaborative interfaces. *ACM Computing Surveys, 28A*(electronic), 143.

Terveen, L.G., 1995. An overview of human-computer collaboration. *Knowledge-Based Systems Journal, Special Issue on Human-Computer Collaboration, 8*(2-3), 67-81.

Topi, H., Babaian, T., & Lucas, W., 2005. Identifying usability issues with an ERP implementation. In *Proceedings of ICEIS'2005*, pages 128–133.

Topi, H.; Lucas, W.; and Babaian, T., 2006. Using informal notes for sharing corporate technology know-how. *European Journal of Information Systems*, 15(5):486–499.

Traum, D., Rickel, J., Gratch, J., & Marsella, S., 2003. Negotiation over tasks in hybrid human-agent teams for simulation-based training. In *Proceedings of the Second international Joint Conference on Autonomous Agents and Multiagent Systems*, 441-448, ACM, New York.