ADAPTIVE PROMPTING

Thomas Kühme*, Uwe Malinowski*, James D. Foley**

**Siemens Corporate Research and Development, ZFE ST SN 7, Otto-Hahn Ring 6, D-W 8000 München 83, Federal Republic of Germany.*

***Graphics, Visualization, and Usability Center, Georgia Institute of Technology, 801 Atlantic Drive, Atlanta, GA 30332-0280, USA.*

Phone: (+1) 404-894-9148
Fax: (+1) 404-853-0673
Email: kuehme@cc.gatech.edu

Adaptive prompting addresses problems with locating, referencing, and selecting interface items such as elements of directory hierarchies or menu and dialog box entries. By drawing the user's attention to the most appropriate and most likely to be chosen items, adaptive prompting tries to increase users' performance on selections and to provide guidance in explorative environments. Examples for adaptive prompting include an Adaptive Tool Prompter, an Adaptive Action Prompter, and Adaptive Prompting in Dialog Boxes. In order to deal with application contexts and evolving needs and preferences of the user, adaptive prompting employs models of the application and the user. The chosen approach allows for an optional user involvement into the adaptation and for an evaluation of the embedded prompting strategies. Multimodal user interfaces provide further perspectives for adaptive prompting.

Keywords: Adaptive User Interfaces, Intelligent User Interfaces, Multimodal Interfaces, Application Model, User Model

1. Prompting In WIMP Interfaces

The provision of comprehensive prompting is considered to be a strength of WIMP interfaces (windows, icons, menus, pointer) as it is easier to recognize a visible object rather than to recall how to reference it. A major part of users' interaction with systems is selecting items from a set of alternatives, such as tools from a toolbox, files from a directory, actions from menus, or attribute changes from a dialog box. However, only a limited number of selectable items can be presented on the screen at the same time and even a smaller number of items can be perceived by the user at a glance. Thus, there is still some referencing which has to be accomplished by browsing through a structured environment of directory hierarchies, menu hierarchies (menu bar, cascaded menus), or scrollable lists and by searching for relevant items in directories, menus or dialog boxes.

Some forms of a more flexible prompting than that provided by static hierarchies have been introduced in order to reduce the navigation effort. For instance, tools and files (or aliases of them) can be put onto the desktop for a direct access. Menus and dialog boxes are sometimes configurable (or adaptable), i.e. users can determine which items should go into a menu or

dialog box. MS-WORD [8], e.g., provides an adaptive menu which contains the most recently opened files. For directories and lists, users often can choose a filter, i.e. a static pattern, designating items to be included in the presentation.

Prompting may also differ in how items are presented. Often users can adapt the presentation according to their specific needs and preferences. For instance, the presentation style (e.g., “as icons” or “as text”) or the order of items in a list (e.g., “by size” or “by name”) can be chosen in a way that the desired items can be discovered more easily.

Pointer setting strategies for pop-up menus provide a faster access to a certain menu item, often the most recently or most frequently used. With the same intention, a dynamic reorganization of menus has been proposed. Considering user-specific usage patterns is an attempt to make prompting adaptive to particular users.

Context-sensitive prompting includes grey-shading of disabled menu-items. Also, actions may be prompted according to previously selected objects, for instance, in object-specific menus (most typically pop-up menus of editors and browsers) or dynamically changing control panels (e.g., in CAD systems). In addition to increasing users’ performance on item selection, such prompting provides feedback on the actual context and supports users in selecting valid and appropriate items.

2. Goals And Basic Ideas Of Adaptive Prompting

While many of these singular mechanisms have been proven useful in contemporary user interfaces there is no general approach to a more meaningful prompting with respect to users and their tasks. Adaptive prompting, as proposed in this paper, aims to lead the user to those items which are the most important prompts in any given situation.

The interpretation of “most important” depends on whether the emphasis is on increasing users’ performance or on providing guidance. The performance aspect might be covered by items which are most likely to be chosen whereas in the case of guidance the most appropriate items become the important ones. Actually, there is a strong correlation between both aspects depending on users’ experience. For instance, expert users can be expected to most likely choose items which are also very appropriate ones while this does not necessarily hold for novice users.

There are several reasons that a user might need to access only a few out of dozens or even hundreds of available tools, actions, and dialog box entries. Mainly, the user’s actual tasks determine a subset of items which are needed to perform these tasks. In addition, the users’ knowledge and preferences decide about selections. For instance, novice users know only a subset of all the existing actions of an application. Experts, on the other hand, might have developed preferences for certain actions as far as there are redundant actions and different ways to perform a given task.

In a particular situation, the set of actually needed tools or actions may be even smaller. Provided the user is focussing on a certain task or a couple of concurrently performed tasks, the needed items are those to proceed with in order to perform these tasks. In addition, the actual context of the interaction might impose restrictions on the availability of particular items or might rule out items which to choose would not be sensible although they are available.

Based on these assumptions, adaptive prompting tries to detect the items which are most appropriate and most likely to be chosen by the user. Strategies to determine appropriate items

and to predict the possibly next selection are based on evaluating knowledge sources some of which are already available in existing research prototypes of user interface environments. For instance, the interface itself may provide information about the user's current focus (pointer position, current selection). An embedded model of the application can be used to trace the context of application objects, actions, and tasks and to supply the corresponding information. A model of a particular user's knowledge about and preferences for certain items can support the decision which items are likely to be selected by this user.

Once the system determines which items can be assumed to be the most important ones it presents them to the user in a way that the user's attention is drawn to these items and that the user can perceive them at a glance. The number of items presented as being important should follow early experiments which determined the value 4.2 to be the average number of items perceivable at a glance [10].

At least, there are two alternatives to bring certain items to the user's attention without impairing the accessibility of all the other items. First, the system could provide the user with a separately presented preselection of the corresponding items and, second, the system could change the appearance of these items within the regular environment. The tool prompter and the action prompter described below are examples for the preselection method while the adaptive prompting in dialog boxes is an example for the appearance changing approach.

In either case, adaptive prompting is intended to be a complementary aid, not a substitute for existing interfaces. Explorative working in a rich environment should always be possible if desired, but not by all means necessary in situations with a clear focus.

Adaptive prompting strategies can obviously not be expected to never fail. On the contrary, the unpredictability of users' behavior and the self-imposed restriction to the fairly small number of items perceivable at a glance will bring down the rate of successfully guessing the set of candidates for the next selection. Even if an item which the user is going to select is contained in an offered preselection this might not always be the best alternative from which to select the item. For instance, if the user knows exactly where to find an action within a pull-down menu and if, besides, the current pointer position is close to this menu the search for the action in a separately presented preselection obviously requires more effort than the selection from the menu.

However, all this does not argue against the prompter approach as long as users do have the opportunity to select the items from the regular environment. There is already a gain of performance if there are at least some situations in which a selection from the prompter is considerably faster. So, even experts can benefit from the prompter with respect to performance if only sometimes they are provided with an item which they do not know where to find.

Beside performance aspects, adaptive prompting attempts to provide guidance by presenting and thereby suggesting items which are assumed to be appropriate in a given situation. However, the provision of guidance has to be designed very carefully since wrong assumptions about the appropriateness of items can cause fatal problems. Obviously, misleading the user would be even worse than no guidance at all.

3. Examples Of Adaptive Prompting

In the examples given in this section, adaptive prompting is applied to three major elements of direct manipulation interfaces. The Adaptive Tool Prompter facilitates tool selection in a

desktop environment; the Adaptive Action Prompter addresses problems of action selection by means of menus, control panels, and direct manipulation of graphical objects; and Adaptive Prompting in Dialog Boxes aims to make dialog boxes easier and more effective to use. While this section is concerned with what adaptive prompting can specifically be about and to what purposes it might serve the next section explains how adaptive prompting can be technically achieved.

3.1. The Adaptive Tool Prompter

The adaptive tool prompter presents a choice of references to tools (programs) and related files (data objects) which are considered to be the user's current working set. The user can select tools and files from either the prompter or the regular environment (i.e., desktop, file manager) whatever is more convenient in a given situation. The prompter contents is being dynamically adapted to context changes and evolving needs and preferences of the user. The number of contained items is kept small (e.g., 3-5 tools and 4-6 files per tool), as opposed to a desktop which usually keeps constantly filling up with icons, thus becoming difficult to survey. See Figure 2 for an example contents of the tool prompter.

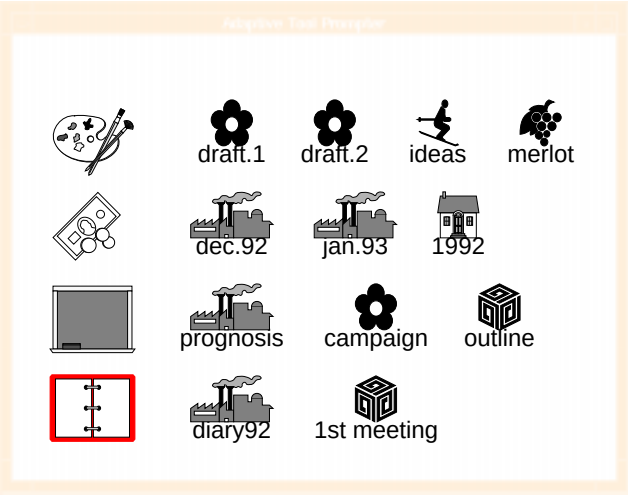


Figure 1. The Adaptive Tool Prompter

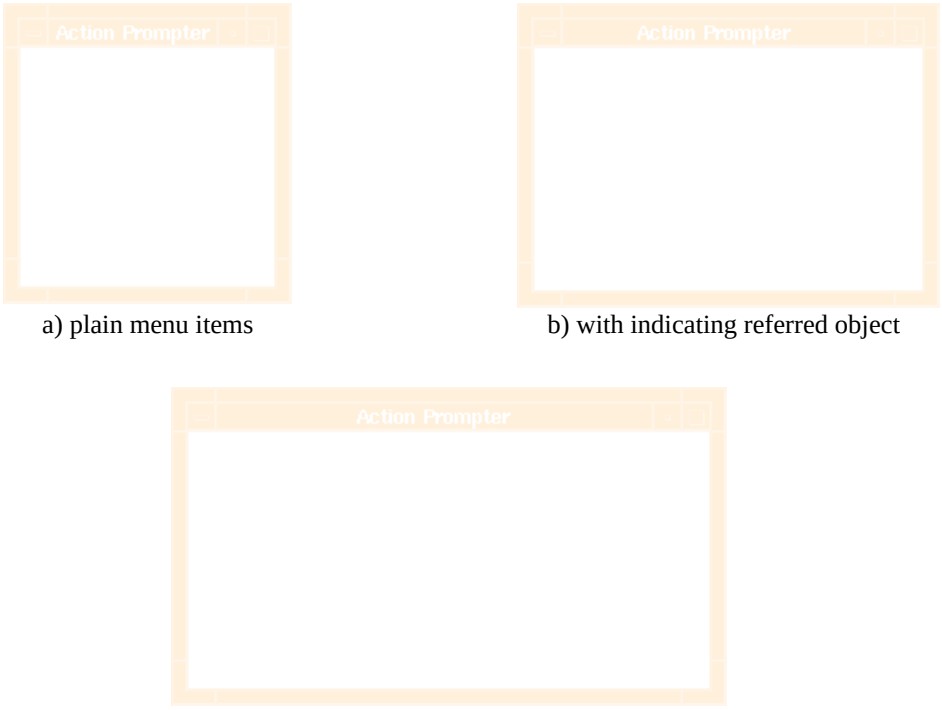
The prompter is able to increase users' performance on tool selection by providing those tools the user knows and wants to use. Guidance is provided by including tools the user might not know and could use well in the current situation. Also, guidance by prompting could be desirable in the case of tools the user do know but is not going to use at the moment although there is a necessity to do so because of external (i.e., not user-initiated) events. For instance, in an office context the user's favorite mail tool would be made available if there were new mail, or, in a production automation context, an emergency management tool would be offered if a power failure occurred in an assembly line.

3.2. The Adaptive Action Prompter

The action prompter is fairly similar to the tool prompter, except that it deals with application actions instead of tools and files. One can think of it as a permanently visible, dynamic menu

(or control panel) into which application actions are mapped by adaptive prompting strategies. The prompter contents is adjusted with every context change caused either by the user interacting with the application or by the application itself. As in the tool prompter, users can select actions from either the action prompter or the regular environment (i.e. from menus, from control panels, by direct manipulation).

A sensible default value for the maximum number of items presented by the action prompter could be 4 or 5. The items can be the same as in the regular menu in which they originally appear or they can be extended by the objects to which the actions refer or by task-oriented explanations with respect to the purpose of an action. Figure 2 shows according instances of the action prompter.



a) plain menu items

b) with indicating referred object

c) with task-oriented explanation

Figure 2. Instances of the Adaptive Action Prompter

Direct manipulation applications provide a huge number of actions selectable for the user. For instance, a typical application would offer: a menu bar with 3-7 submenus with up to 10 or more items each, sometimes even extended by cascaded menus; object-specific pop-up menus, also often cascaded; a control panel or a few single buttons; and direct manipulation operations such as clicking on or dragging objects, often with multiple meanings activated by pressing modifier keys.

As discussed above, menus are considered to be a good idea to provide a comprehensive feedback on which actions are available. However, users have to browse through menu hierarchies in order to locate the desired items and to make their selections. Shortcuts address this problem by allowing for a random access to menu items through pressing certain keys in com-

bination or sequence. But shortcuts, in turn, trade faster accessibility for again more artifacts users have to remember.

With the adaptive action prompter, the most important actions are easily accessible in most cases while otherwise a search through a possibly large menu hierarchy would be necessary. Since the prompter lists the most appropriate actions one beneath the other (as opposed to being distributed over different menus) the user has a good survey of sensible alternatives.

In the case of direct manipulation, a feedback on available actions is usually given by the presentation of the (selected) object and by the pointer shape. In order to find out about accessing actions, users have to select and to traverse an object with the pointer while watching the pointer shape. Sometimes, users even have to put an object into another mode to be able to manipulate it in a certain way. Although users often find it quite intuitive to directly manipulate an application object there is little help on how to discover what the object's behavior is all about with respect to direct manipulation.

Including direct manipulation actions in the prompter is only reasonable for actions the user does not know or does not know how to access. Direct manipulation actions are presented as prompts telling the user that they are available and providing a pictorial information how to select them by manipulating the object in a certain way. If the user decides to select such an action from the prompter a mode is entered which allows for completing the action interactively.

Not only that users have problems to access (i.e., locate, reference) the actions they already know and want to select. Prior to selecting an action, users have to know if the application provides an appropriate action at all (i.e., an action users can describe in terms of its effect). Even more basically, users have to imagine what would be an appropriate action to precede with in order to perform a certain task of the application domain (cf. [6]). While direct manipulation interfaces are appreciated for making all available functions easily accessible at the same time, thus allowing for an explorative working style, they generally do not support the user in dealing with such sequencing problems. Some attempts have been made to overcome this weakness by providing context-sensitive help on how to enable currently disabled actions or animated help on how to perform a sequence of actions [11].

The action prompter addresses sequencing problems by considering known action sequences and relationships between subsequent actions. Sequencing support by adaptive prompting is given implicitly without any interruption of the application dialog.Thus, there are no meta-dialogs which might increase the distance between the user and the field of activities within the application domain.

3.3. Adaptive Prompting in Dialog Boxes

Dialog boxes of complex systems are often unavoidably large and difficult to survey. A considerably large mental effort is necessary for even the simplest change of one single item or for only looking around if all items are set well before clicking OK. Adaptive prompting, as proposed here, means that the most important items of the dialog box are brought to the user's special attention by either highlighting these items or putting some "fog" on top of all the other items. Additional feedback (e.g. by coloring) is provided on how usual (green) or strange (red) a given item value is with respect to the user's history of using the particular dialog box. All items whether highlighted or not are left in their place and are still accessible for the user [7]. Figure 2 shows a sample appearance of an adaptive dialog box.

Figure 3. Adaptive Prompting in a Dialog Box

Not being distracted by the uninteresting parts of the dialog box, the user is able to perceive the most important items at a glance. Most important are those items which are likely to be selected and changed (performance aspect) or which should be observed and controlled (guidance aspect).

4. Concepts and Achievement of Adaptive Prompting

Adaptive prompting aims to be very specific with respect to both the application context and the individual user. Thus, models of the application and the user are necessary ingredients of an adaptive prompter. Numerous prompting strategies use the knowledge supplied by these models to automatically determine items to be prompted while optional user involvement allows for co-operatively controlling the adaptation process.

These issues of how to achieve adaptive prompting are discussed along the two examples of the tool prompter and the action prompter. Adaptive prompting in dialog boxes has been described earlier [7].

4.1. Application Model

Model-based user interface environments aim to capture all the application specific knowledge which is relevant for interaction by an explicitly represented model. The application model supports reasoning about this knowledge at both design-time and run-time of the application. It can be used, for instance, to drive the interaction with the user at run-time and to monitor the dialog.

Tool Prompter

Elements of the underlying application model are: *tools*, *files*, *relationships between tools and files*, *task contexts*, and *projects*.

Relationships between tools and files are expressed by rules such as "tool *t* can be applied to files *a,b,c,...* or to files of type *X*" or "work on file *a* requires tools *t* and *v*". A task context is described by a set of tools which are used together while working on a certain task or group of tasks. A project is defined as a set of files related to a particular project in the application domain. At run time, tools and files monitor their own usage and communicate selections by the user to the user model and to other elements of the application model, i.e., task contexts resp. projects they are elements of. Selections are monitored independent of whether they are made from the prompter or the regular environment. By keeping track of all the selections of their elements, task contexts and projects determine if they are currently active.

Action Prompter

In order to support action prompting, the application for which actions are to be prompted is modeled in terms of: *object classes* and *instances* thereof, *actions* with *parameters*, and *pre-* and *postconditions* of actions [13].

Object classes contain information about which actions are applicable to their instances while action parameters in turn determine to which object types an action can be applied. Pre- and postconditions describe under which circumstances an action is executable and which the expected results are. An action is enabled (i.e., can be selected) if its preconditions are fulfilled while the postconditions of an action determine the context changes which occur when this action has been completed successfully. In addition, the notion of a task context is introduced. Similarly to the tool prompter, a task context is defined as a set of actions which are related to a certain task or group of tasks.

At run time, the application model is responsible for monitoring action selections. The monitoring results are communicated to the user model and are also used inside the application model for context detection. There are three different contexts which are instructive for adaptive prompting: the *object context*, the *action context*, and the *task context*.

The object context considers which application objects are in the user's current focus. Obviously in the focus are, for instance, a selected object, an object that has explicitly received the input focus for keyboard events, or an object which was involved in the last selected action. While these criteria for focus detection are considered on the interface level a more subtle mechanism based on the application model infers further assumptions about the user's focus. This mechanism uses the sequence of interface level detections to create, establish, and switch between object contexts (cf. the notion of the *focus space* in [9]). Objects which are closely connected to an obviously focussed object are included in an object context. In particular, in a hierarchical structure all the objects on the path from the root of the hierarchy to the selected object and all the objects logically contained in the selected object would be considered to be in the user's focus. For instance, a selected chart within a document within an editor would mean that the editor, the document, and the elements of the chart are also in the corresponding object context while another document or another chart are not.

The action context is given implicitly by pre- and postconditions of actions. It includes several facets some of which can be made explicit by monitoring the values of pre- and postconditions. Particularly interesting for prompting is the information when and why (i.e., by executing which action) the precondition of an action became true. A further evaluation can determine, for instance, those actions which were enabled by the previously executed action. These actions are sensible candidates to be prompted because the user might have executed

the previous action intentionally to enable one of them. On the other hand, some actions might require to be prompted for only after being enabled for a while. For instance, a “save” action is disabled after being executed itself and is enabled after any other action has been selected. The action context information supports a prompting strategy that prompts for a “save” if a certain number of actions (e.g. 100) have been selected after the “save” was enabled.

The task context considers the set of those actions which are usually used when performing a certain task or group of tasks. A task context can be called active if a certain percentage of all actions selected are within this context. On the one hand, this approach is less expressive than task modeling mechanisms dealing with sequences of actions. However, the notion of the task context seems to be just appropriate for action prompting in direct manipulation interfaces. The opportunity to switch between tasks and to choose any possible order of actions provided by direct manipulation dialogs is intrinsically taken into account by the task context approach. In addition, presenting a couple of reasonable alternatives to choose from does not necessarily demand for predicting the very next step in the one and only task the user could be assumed to deal with. Dealing with sequencing in terms of the action context described above might be sufficient in many cases.

Introducing elements of task modeling in the prompting mechanisms does make sense for enhancing the abilities to provide specific guidance on a certain task. The prompter could also be considered as just a convenient instrument to provide the user with tutorial information that is represented or has been produced elsewhere. Further work will be spent on this issue.

4.2. User Model

The user model serves for transforming usage patterns into information about a particular user’s preferences and knowledge. The monitoring facilities of the application model provide the user model with the relevant usage patterns.

Tool Prompter

The user model contains assumptions and performs reasoning about: the user’s knowledge about the existence of tools (count how often a tool was used); the user’s preferences for certain tools (comparative analysis of usage counts and last recently used flags); the user’s overall experience (inferred from how many and how well tools are known and which are preferred); and the user’s preference for performance or guidance prompting (inferred from user’s overall experience).

Action Prompter

As in the case of the tool prompter, not only preferences and knowledge with respect to elements of the interaction such as actions and task contexts are established. Again, also more general assumptions about the user are inferred, such as the user’s overall experience or the preference for either performance or guidance prompting.

The stored usage patterns are not simple counters. They include additional knowledge about contextual correlations. Thus, the user model can supply information about, for instance, the three most preferred actions on objects of type *X* or the most preferred action among actions *A*, *B*, and *C*.

4.3. Prompting Strategies

Prompting strategies consider and prioritize all the different pieces of information contained in the employed models.

Tool Prompter

Candidates for being prompted are those tools and files which are elements of the currently active task contexts or projects. In general, these are still too many, so that a further preselection has to be made based on the user's knowledge and preferences of tools and on usage profiles of particular files within a project. If the user prefers guidance prompting also unknown tools are considered, as opposed to performance prompting where the preferred tools always precede. The maximum number of presented items is adjusted, for instance, according to the user's overall experience (high experience / large number). In addition, event rules describe the reaction on external events, for instance, "on *newMail* prompt *MailTool*".

Action Prompter

The application model of the action prompter provides very comprehensive context information. As opposed to the tool prompter which only deals with simple relationships between tools and task contexts or files and projects, the strategies employed here have to consider three different contexts action selection is related to. The object context determines the objects the user possibly wants to select an action for; the action context states which could be the next in a sequence of actions; and the task context provides informations about actions dedicated to the current task or group of tasks. Considering also the user's preferences provided by the user model, there are many cues to argue which actions to prompt for.

To find appropriate strategies to combine all the available information is a major task. Since the user's style of interaction determines what "appropriate" means hardwiring a particular strategy would not help. For instance, the action context information is hardly relevant for users who are used to switch extensively between tasks. For them, the task context information becomes important. In order to ensure that the prompting strategies match the particular user's needs they have to be adaptive in a broader sense. The strategies not only have to process information which depends on the context and the user; the process itself has to be adaptive.

The information determining the contents of the prompter is accumulated by interpreting each piece of information as a vote for or against certain actions. This procedure is adaptive with respect to the weight of each vote and the order in which the votes are collected. Since it is possible to exclude actions under certain conditions the order is significant.

The weight and the order are predefined by the system designer and can be adjusted by the user. In principal, the prompter could try to improve the setting of weights and order by evaluating the success of its prompting. Such an evaluation can be achieved by observing whether or not the prompter contents matches the actual selection. Mechanisms for self-improving prompting strategies are also an issue for further research.

In addition, rules can be defined to modify the context detection results of the application model and to handle any exceptions to the prompting mechanisms, for instance: "ifSelected *anObject* activate *aTaskContext*" or "ifExecuted *anAction* prompt *anotherAction*". By this means, also certain actions can be excluded from prompting: "Never prompt *anAction*" (par-

ticularly sensible for actions with well-known shortcuts). Similar to the tool prompter, also rules to react on external events are possible.

4.4. User Involvement

While in the case of the desktop nearly all the placement of tools and files (i.e. icons) is done explicitly by the user the tool prompter employs built-in mechanisms to determine its contents. However, the prompter approach is not aimed to be at the other extreme end without any user involvement at all. Instead, the user is widely supported in inspecting and controlling the adaptive mechanisms if desired. This approach has been called Computer-Aided Adaptation [3,4]. Although the subsequent examples were chosen from the tool prompter the action prompter offers similar opportunities for the user to get involved.

There is a broad range of optional user involvement including: explicitly setting the maximum number of items presented in the prompter; explicitly adding or removing items to or from the prompter; explicitly determining the currently active task context(s) and project(s); defining and modifying task contexts (task contexts may also be designer-predefined or system-inferred); defining and modifying projects (projects may also be system-inferred); explicitly setting preferences in the user model (overwriting defaults or assumptions inferred by dialog monitoring); and changing the prompting strategies by adding, modifying, or removing rules such as "promptPermanently *MyFavoriteEditor*", "if (*GraphTool* isPrompted) prompt *Calculator*", or "on *diskFull* prompt *Compressor*" (event rule).

It is crucial that the user can obtain all these insights and adjustments with a minimal amount of knowledge and effort. Therefore, the user interface for any kind of user involvement consists only of simple menus and dialog boxes. There are no files to edit and no comprehensive languages to learn. For examples see figures 4-7.

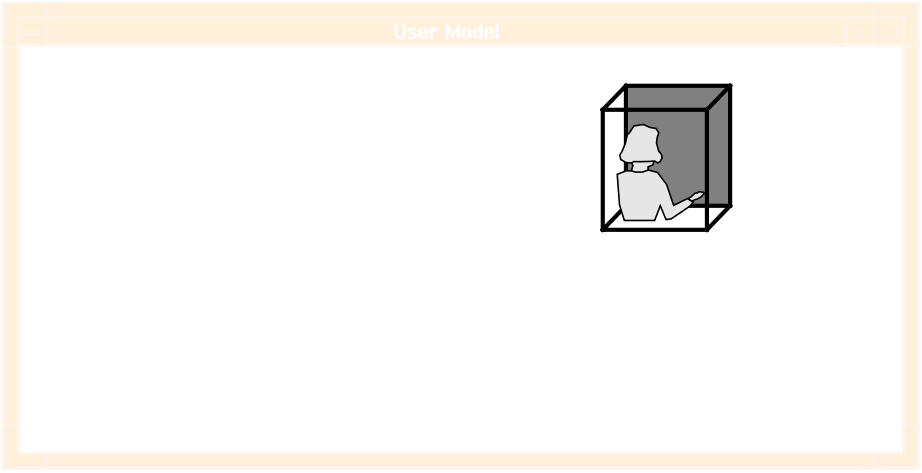


Figure 4. Interface for User Model Adjustments

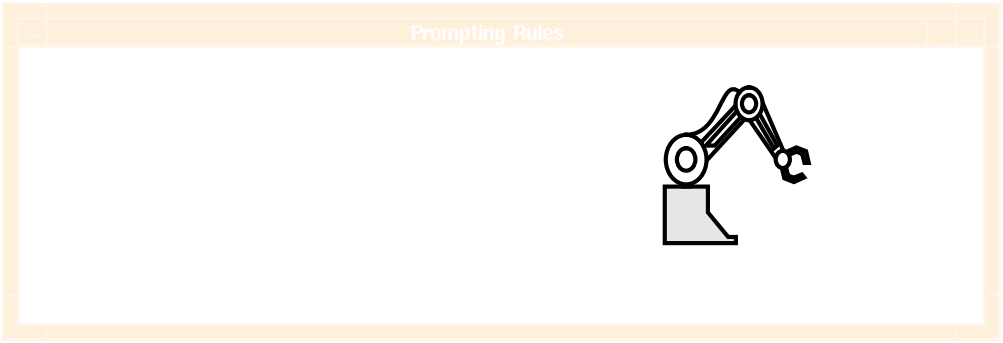


Figure 5. Interface for Prompting Rule Management

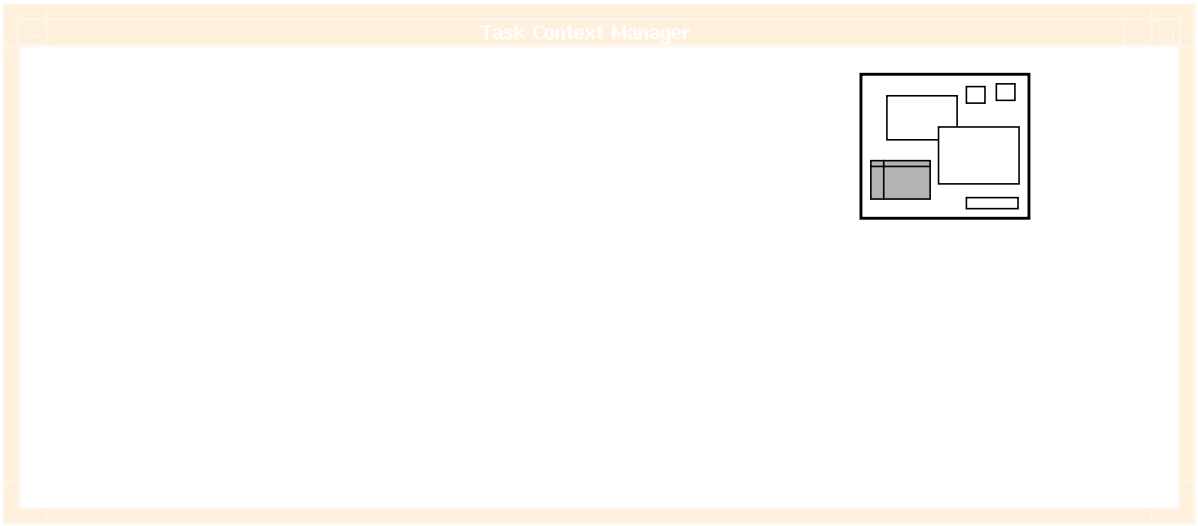


Figure 6. Interface for Task Context Management

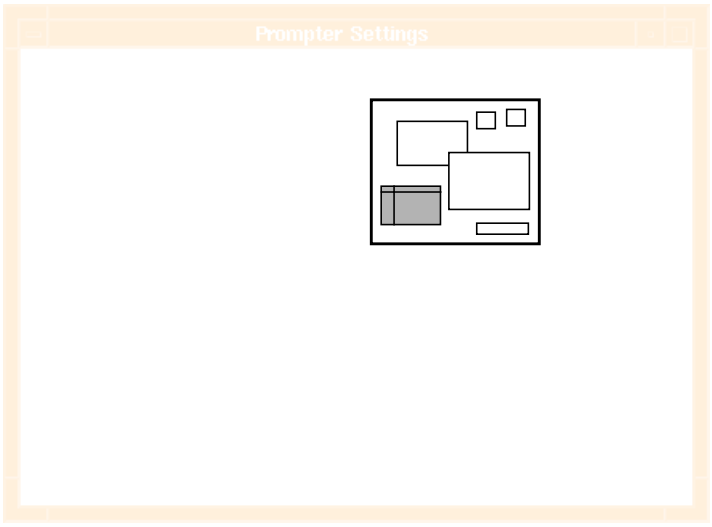


Figure 7. Interface for Prompter Setting Adjustments

5. Adaptive Prompting In Multimodal Interfaces

Multimodal interfaces aim to relieve users from the burden of dealing with system functions and telling the system "how to do" something. Instead, users will be enabled to communicate their intentions and desires (i.e., "what to be done") to the system in a more natural way by means of gestures and spoken or written natural language. On the long run, users will no longer have to select abstract "tools" or "actions". However, adaptive prompting is even more important and applicable in multimodal interfaces.

There are several observations which support this hypothesis. Most of them can be considered again under the two aspects of performance and guidance. A third aspect refers to how adaptive prompting can benefit from multimodality because of the provision of additional focus information and larger opportunities for adapting the presentation of prompting information. However, the latter aspect is beyond the scope of this paper.

5.1. Performance Aspects

Allowing for more natural, intuitive user input requires a much larger effort for interpreting and evaluating this input within the user interface. Since there are still application programs, functions with parameters, and objects with attributes the user interface has to accomplish a mapping from a fuzzy input to definite entities an application deals with. As a matter of fact, the user interface now has the burden of selecting tools, actions, etc. according to the user's interaction with the system.

Internal Prompting

Adaptive prompting mechanisms can greatly support this internal selection process by providing a list of the items which are most likely to be selected. Especially in gesture and speech recognition mechanisms, this information could be used to focus on the most probable interpretations resulting in a potentially higher recognition rate.

Confirmative Prompting

As long as recognition rates are still poor, particularly too poor for certain safety-critical application areas, prompting the user should be considered also or even just with multimodal interfaces. For instance, an adaptive action prompter menu provides the user with a cue what the user interface considers to be most likely meant by the next gesture or speech input. On the one hand, the user would know that inputs which are not covered by the menu are more likely to be misunderstood by the interface than those in the menu. On the other hand, the user can use any prompted command (and perhaps gesture symbol) for producing a speech (or gesture) input which will be understood almost for certain. Optionally, the user can choose the desired action from the prompter.

Referencing

While it is hard to establish and to use contexts and pronominal references in natural language dialogs the additional use of direct manipulation techniques (such as pointing) can overcome these problems in multimodal dialogs [1]. Referencing can become even more simple with adaptive prompting, for instance, in a dialog box (there is little evidence that dialog boxes or something analogous will no longer be needed in multimodal dialogs). Adaptive prompting makes the complex context of a large dialog box shrink to a simple, easy to survey context.

If the user's focus actually is within this restricted context the remaining referencing effort in the speech input can be kept very little. For instance, the speech input "Set the third number in the first column to seven!" would be reduced to "Set number to seven!" with adaptive prompting if one number item and one text item were highlighted. This would be even more reduced to a simple "Seven!" if only the number item was highlighted. This kind of automatic referencing by adaptive prompting can obviously go far beyond the performance of purely natural language referencing and may sometimes even surpass pointing. It might also be very helpful in environments where pointing devices are not available.

5.2. Guidance Aspects

Although multimodal interfaces facilitate more intuitive dialogs than ever possible in conventional direct manipulation interfaces they will not completely relieve the interface from providing guidance. This assumption is based on the every day experience that new capabilities are fully exploited as soon as they appear. Interfaces and underlying applications will become more powerful but hardly simpler to use in their entirety.

Intuitive Guidance

As described above, adaptive prompting provides guidance by particularly offering the most appropriate items to be proceeded with (objects, actions, etc.). Since prompting is a regular part of the user's interaction with the system this kind of guidance fits intuitively into the productive application dialog.

Guidance for Pen-Based Systems

Guidance seems also to be necessary in pen-based systems. For instance, it cannot be assumed that users remember all the possible gestures an interface designer might have thought of as "intuitive". An optional guidance on gestures which are understood by the system would be more than helpful. However, there is rather little screen space on most pen-based systems where to provide this guidance. Adaptive prompting would be a solution since it presents only those items which are most important in a given situation. Hence, space consumption could be limited to what is just necessary.

6. Ongoing Work

Prototypes of the tool prompter and the action prompter have been developed using and extending SX/Tools [5] for the physical user interface portion and UIDE [12, 13] for application and user modeling. Mechanisms for adaptive prompting in dialog boxes are going to be included. Implementation details will be described in a separate paper.

An evaluation of prompting strategies can be achieved by comparing the proposed prompts with the actual user input. With some restrictions, such an evaluation can be carried out offline by using session protocols. Thus, different prompting strategies can be tested very efficiently. First results of these evaluations will be presented at the conference. Further evaluation includes extensive user-testing with respect to questions such as: Do users accept (i.e. use) prompting? Does its usage improve performance *and* quality of dialogs? How does the optional user involvement affect adaptive prompting?

There is virtually no application area adaptive prompting would not support. However, areas which could benefit from adaptive prompting in particular include medical information sys-

tems and traffic management systems. These systems have in common several demanding problems in regard to the user interface which are addressed by adaptive prompting. Among these problems are: a community of users who are mostly no computer experts and range widely in their domain expertise (adaptivity); an application domain which usually involves stress situations (performance prompting); and a need for extensively using all the emerging presentation media and interaction modalities which make interfaces more complex and not always simpler to handle (guidance prompting).

Acknowledgements

We thank Hartmut Dieterich, Matthias Schneider-Hufschmidt, and Piyawadee “Noi” Sukaviriya for many helpful conversations about related topics. Thanks also to Al Badre for his comments on a draft of this article.

References

1. P. R. Cohen: *The Role of Natural Language in a Multimodal Interface*. Proc. UIST’92, Monterey, CA, Nov 15-18, 1992.
2. W. D. Gray, W. E. Hefley, D. Murray (Eds.): *Proceedings of the 1993 ACM International Workshop on Intelligent User Interfaces*. Orlando, FL. ACM Press, New York, 1993.
3. T. Kühme: *A User-Centered Approach to Adaptive User Interfaces.* In [2], pp. 243-245, 1993.
4. T. Kühme, H. Dieterich, U. Malinowski, M. Schneider-Hufschmidt: *Approaches to Adaptivity in User Interface Technology: Survey and Taxonomy.* Proc. of the IFIP TC2/WG2.7 Working Conf. on Engineering for HCI, Ellivuori, Finland, Aug 10-14, 1992.
5. T. Kühme, M. Schneider-Hufschmidt: *SX/Tools - An Open Design Environment for Adaptable Multimedia User Interfaces*. Proc. Eurographics ‘92, Computer Graphics Forum, Vol. 11, No. 3, pp C-93–C-105, 1992.
6. C. Lewis, P. G. Polson: Cognitive Walkthroughs: *A Method for Theory-Based Evaluation of User Interfaces.* CHI’92 Tutorial Notes, Monterey, CA, May 4, 1992.
7. U. Malinowski: *Adjusting Forms to Users’ Behavior.* In [2], pp. 247-249, 1993.
8. Microsoft Word 5.0, Microsoft Corporation, 1991.
9. M. A. Peréz, J. L. Sibert: *Focus in Graphical User Interfaces.* In [2], pp. 255-257, 1993.
10. G. Sperling: *The Information Available in a Brief Visual Representation.* Psych. Monogr., Vol. 74, No. 11, 1960.
11. P. Sukaviriya, J. J. de Graaff: *Automatic Generation of Context-sensitive “Show&Tell” Help.* Technical Report GIT-GVU-92-19, Georgia Institute of Technology, July 1992.
12. P. Sukaviriya, J. Foley: *Supporting Adaptive Interfaces in a Knowledge-based User Interface Environment.* In [2], pp. 107-113, 1993.
13. P. Sukaviriya, J. Foley, T. Griffith: A Second Generation User Interface Design Environment: The Model and The Runtime Architecture. Proc. INTERCHI’93, Amsterdam, The Netherlands, April 24-29, 1993.