A UIMS Architecture for Focus Processing in a Graphical User Interface

Manuel A. Pérez
Robert J. K. Jacob

Code 5534
Human-Computer Interaction Laboratory
Naval Research Laboratory
Washington, DC 20375-5337
{perez | jacob} @itd.nrl.navy.mil

ABSTRACT

Today's graphical user interfaces remember little from one transaction to the next; each command exists nearly independently. Humans, however, typically draw on previous elements of a dialogue in their communications. We are seeking to add some of the characteristics of human dialogues to graphical interfaces. This paper describes our research into this problem and our initial results in answering three questions: What are the appropriate analogues of conversational focus in a graphical human-computer conversation? Where does this type of processing best fit within a user interface management system paradigm? What mechanisms can be used to realize it?

NATURAL DIALOGUE IN A DIRECT MANIPULATION INTERFACE

In a direct manipulation or graphical interface, each command or brief transaction exists as a nearly independent utterance, unconnected to previous and future ones from the same user. Real human communication rarely consists of such individual, unconnected utterances, but rather each utterance can draw on previous ones for its meaning. It may do so implicitly, embodied in a conversational focus, state, or mode, or explicitly ("Do the same sorting operation you did before, but on these new data").

Our goal is to connect these properties of human dialogue to direct manipulation or graphical interaction styles. While some natural language human-computer interfaces attempt to exploit these characteristics of human dialogue, they have been notably absent from graphical interfaces. Natural dialogue is by no means restricted to natural language. Most research on the processes needed to conduct such dialogues has concentrated on natural language, but some of them can be applied to any human-computer dialogue conducted in any language. A direct manipulation dialogue is conducted in a rich graphical language using powerful and natural input and output modalities. The user's side of the dialogue may consist almost entirely of pointing, gesturing, and pressing buttons, and the computer's, of animated pictorial analogues of real-world objects. A dialogue in such a language could nevertheless exhibit useful dialogue properties, such as following focus.

In natural language, focus represents the attentional space of the participants in a dialogue [14], and it is a property of a dialogue between two (or more) participants. It contains the objects and actions that are most relevant to the conversation [3, 4]. Natural language processing systems use focus to resolve ambiguous utterances. Focus contributes to the connectedness of dialogue by allowing economy of expression (e.g. pronoun references and anaphora). One approach to focus processing in the computational linguistics field [3, 4] uses a semantic net partition to represent those items from a conversation that are in focus. The net is partitioned into spaces, classified as explicit or implicit. Explicit focus spaces contain objects and actions that have been used in a conversation. Implicit focus spaces contain items that are closely related to the items in the explicit focus spaces. For example, while talking about cars, the concept car would be in an explicit focus space, while the parts of the car would be in an implicit focus space. This type of focus partitioning is used to resolve pronoun referents, and to detect topic shifts initiated by references to items outside of the current focus.

The work described in this paper is part of a collaborative project at NRL, see submission by Marsh and Wauchope [13].

Focus and Direct Manipulation Dialogue

The most popular design for today's graphical interfaces is an object-action paradigm. These interfaces require the user to select an object followed by the selection of the action to be performed on the object. Because objects have to be

selected for an action to be performed, this dialogue design is called the current-selection paradigm [2]; all actions are performed on the currently selected objects. The structure of a dialogue in these systems is very simple. At any point in the dialogue, you have two choices: select an object, or perform an action on the selected objects. This dialogue structure has some advantages. First, selection of objects for actions is always done the same way, thus making that part of the interface simple and consistent. Second, current-selection paradigm benefits from the use of recognition of valid commands instead of recollection. Invalid actions (based on the contents of the selection) are disabled; and only those that apply to the current selected objects can be performed. Thus the user can see what actions apply in the current context.

The advantages of a simple dialogue structure come at a price. First of all, current-selection works best for applications with homogenous objects and actions. These applications are often elaborated by permitting current selection to be a set with one or more objects. Management of the selection set can be cumbersome, especially if the application has many different types of objects that do not share the same types of actions.

Second of all, when an action requires two parameters of different types, it is not clear how the CSO paradigm should handle this situation. Take for example, the duplicate command in most drawing programs. The duplicate command requires two parameters: the object to be duplicated, and the location where the duplicate will be placed. Current selection does not allow for a simple implementation of the selection of the two parameters. One solution is to split the duplicate command into two separate commands, copy and paste, as it is done on the Macintosh [1].

Another limitation of current-selection dialogues is that the only information maintained from one action to the next is the selection itself. There is no mechanism for actions to use other information that spans more than one exchange [10]. The current-selection provides only one level of focus, and must be explicitly specified by the user every time, even though sometimes there is only one object of interest based on the current context.

The graphical user interface could keep a history of the user's current focus, tracking brief digressions, meta-conversations, major topic shifts, and other changes in focus. Unlike a natural language interface, the graphical interface would use inputs from a combination of graphical or manipulative modes to determine focus. Pointing and dragging of displayed objects, user gestures and gazes as well as the objects of explicit queries or commands all provide input to determine and track focus [14]. Moreover, focus would not be maintained as a single object, but rather a history of the course of the user-computer dialogue. It is necessary to track excursions or digressions in the dialogue so focus could be restored as necessary. In addition, it would be helpful to track focus by categories. This would allow the user to refer to "the ship" even though the current focus is another object. In that case, the recent history of focus would be searched to find a ship of the appropriate category. Finally, focus is not necessarily a concrete object; it might be a class or category of objects ("all blue ships") or a more abstract entity ("the previous command").

As a simple example of the use of such focus information, the user might give a command (verb) without specifying its object, and the interface would supply the object based on the user's current focus. A more sophisticated approach would deduce the object of the command based on the recent history of the user's focus, rather than its single latest manifestation. The nature of the command might constrain the possible objects. For example, "display hull speed" might apply only to ships. If the current focus were not such a ship, the interface would backtrack through recent focus objects to find the last applicable ship and use it as the inferred object of the command. Further, a "retrieve data" command might indicate a shift from a digression back to the main dialogue; hence the appropriate object of this command would be not the current (digression) focus but the previous (main dialogue) focus.

Human dialogue often combines inputs from several modes. Deixis often involves a pointing gesture that does not precisely specify its object; the listener deduces the correct object from the context of the dialogue and, possibly, from integrating information from the hand gesture, the direction of the user's head, tone of his or her voice, and the like [5]. A user could, similarly, give a command and point in a general direction to indicate its object. The interface would disambiguate the pointing gesture based on the recent history of its dialogue with the user and, possibly, by taking into account other information about the user from physical sensors. An imprecise pointing gesture in the general direction of a displayed region of a map could be combined with the knowledge that the user's recent commands within that region referred principally to one of three specific locations (say, river *R*, island *I*, and hill *H*) within the region and the knowledge that the user had previously been looking primarily at islands displayed all over the map. By combining these three imprecise inputs, the interface could narrow the choice down so that (in this example) island *I* is the most likely object of the user's new command.

This example combined inputs in several modes and interaction history to disambiguate an imprecise pointing gesture. The same approach applies in the absence of a pointing gesture. The user might simply ask for "that" without pointing. Recent history and focus plus physical information about the user may still be adequate to disambiguate the referent of "that."

The problem of selecting an appropriate referent is usefully

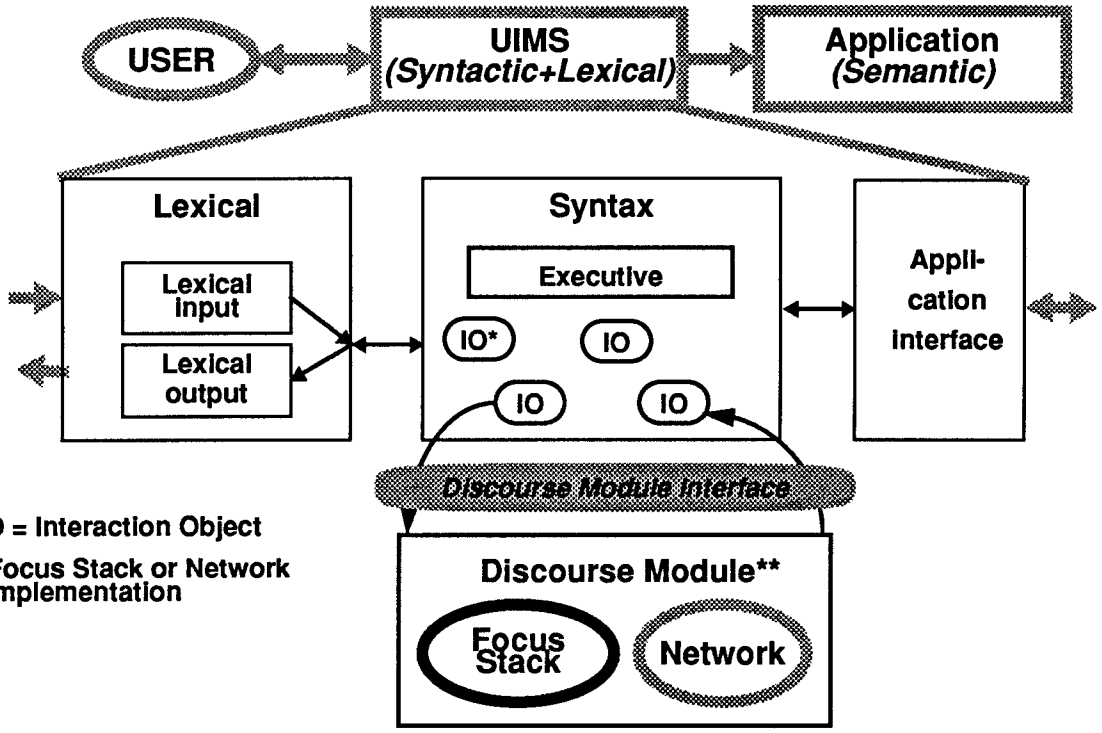

Figure 1. UIMS Software Architecture with Discourse Module

constrained if the user asks for "the aircraft carrier" rather than simply "that." History, focus, and other information may then be combined with the restriction that aircraft carriers are the only pertinent objects to search for the last-referenced aircraft carrier (rather than the last-focused object in general) and thereby determine unambiguously the correct object of the user's command.

A FRAMEWORK FOR HUMAN-COMPUTER DIALOGUE

Human-computer interface design, following Foley and van Dam's methodology [2], is decomposed into three levels: semantic, syntactic, and lexical.

- The semantic level describes the functions performed by the system. This corresponds to a description of the functional requirements of the system, but it does not address how the user will invoke the functions.
- The syntactic level describes the sequences of inputs and outputs necessary to invoke the functions described.
- The lexical level determines how the inputs and outputs are actually formed from primitive hardware operations or lexemes.

Extending the linguistic analogy, we add another level to the interface design:

- The discourse level is concerned with the flow of the interactions over the course of more than one transaction. The semantic, syntactic, and lexical levels are concerned with a single user-computer transaction or brief interaction. The discourse level introduces elements that relate one transaction to another, such as dialogue focus.

Discourse Module in a UIMS Architecture

To incorporate discourse level issues into a user interface management system (UIMS), we extended the typical UIMS software architecture by adding a discourse module. The discourse module's responsibility is to keep track of the focus of the dialogue (i.e. keep track of all objects and actions used in the dialogue). This information is organized in a way that allows the definition of new interaction techniques and new user interface actions.

Figure 1 above shows the components of the UIMS developed at NRL [9, 7], extended with a discourse module. The discourse module works as follows. Tokens from the syntax module that have communicative intent [10] are passed to the discourse module. For example, most of the time moving the mouse pointer around has no communicative intent. In such cases, mouse-move tokens are not passed to the discourse module. But if the mouse pointer is used in a deictic reference not requiring a mouse click (as described in [12]), then mouse-move tokens must be processed by the discourse module. We must decide as part of the design of the interface, which tokens could have communicative intent, so they can be processed by the discourse module.

To use the information stored in the discourse module, we define interaction techniques and user interface actions that request information from the module. Section describes an application used to test the discourse module interface. It also gives examples of how a focus stack implementation is used. Section describes the design for a more complex focus representation (network) and gives some indications of how it will be used, once it is completed.

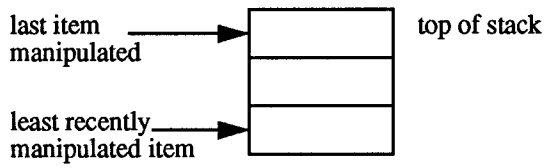

Figure 2. Focus Stack

IMPLEMENTATIONS

We have built two implementations to test some of the ideas presented above. The goal of the first implementation was to test the modularization of the software architecture presented in Figure 1. For this version, we used a simple focus stack as the mechanism to represent focus. The second implementation uses a more complete focus tracking mechanism. This implementation is in the early stages of development; the algorithm and data structures have been tested, but not yet incorporated into the UIMS. The next two sections provide more detail about the two implementations.

Focus Stack Representation

Our first implementation was designed to test the discourse module in the UIMS software architecture described above. The contribution of the first implementation was the clean decomposition of the discourse module and its placement in our existing UIMS software architecture. The implementation of the focus module itself used a simple stack that recorded the objects referenced in user interactions. No representation for user interface actions was provided.

Focus was represented with a stack. The stack contained only application domain objects, with the top of the stack containing the object most recently referenced. When new objects are created, they are added to the top of the stack. Existing objects that are manipulated by the user also move to the top of the stack. Thus, at any point in time, the stack has a list of application objects ordered from top to bottom based on how recently they were manipulated (referenced). Figure 2 shows how the focus stack is organized.

To test this implementation, we designed a simple drawing program (a screen shot is shown in Figure 3) that used focus as an alternative to current-selection interaction. This program can create three types of shapes (using the buttons *Triangle*, *Square*, and *Circle*). Any two circles can be connected with a line using the *Connect* button. Any square can be filled using the *Fill* button. All shapes can be deleted, duplicated, and aligned by their top edges using the *Delete*, *Duplicate*, and *Align Tops* buttons correspondingly. In a current-selection interface, all of these actions would required (at least) one selected shape. In our focus-based interface, the parameters to the actions are taken from the focus stack.

For actions that require one parameter, with no type restriction on the parameter, the item at the top of the stack was used for the parameter to the action. Other actions require a specific type of object as a parameter, for example the *Fill* command in our interface. This action requires a square shape as a parameter. For this action, we use the most recently manipulated square, taken from the focus stack. In Figure 3, the *Fill* button would use the shape square=0x0010.

It is interesting to consider how far down the stack we can move searching for an object. The restriction might be based on time elapsed since the object was last used (a form of forgetting [11]), or we might use task structure to restrict the search (for example see [4]). Currently, our stack has no limit, but we will investigate time-based focus decay in our future work.

The last example involves actions that require more than one parameter, such as the Connect and Align Tops buttons in our interface. For these actions, we pick the highest two objects in the stack that satisfy the command's type restric-

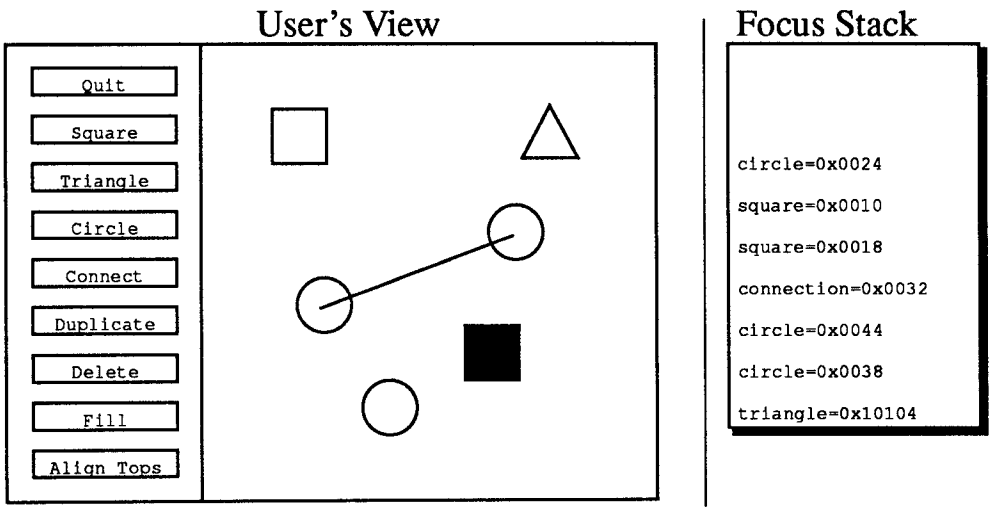

Figure 3. Prototype Application Using Focus Stack

tions. The *Connect* action requires two circles. The result of the action is to draw a line between the two circles, as shown in Figure 3. The difference between the *Connect* and *Align Tops*, aside from their semantic differences, resides in the result in the focus stack. The *Align Tops* uses the top two items from the stack, and leaves the stack unaltered. The *Connect* uses the top-most two circles, creates a connection object (shown in the figure as connection=0x0032), and then places this object (the connection) at the top of the stack. The two circles do not change position in the focus stack, since they were not manipulated directly by the *Connect* command. (This is a design decision for this particular interface to show the usage of the stack.)

This implementation provided us with a quick test of the use of focus in a graphical user interface and for the software architecture. The interface worked as an extended current-selection interface, with selection being implicit and multi-leveled, based on object types and categories. The focus-based interface subsumes the current-selection paradigm. Current-selection still can be achieved by clicking on the object (or objects) of interest before performing the action. We do not claim that this particular drawing program and the examples presented above are better than a current-selection based interface, but rather that we have a software architecture that allows us to design interaction techniques using focus.

Network Representation

The goal of our second implementation was to develop a general design that will manage focus tracking for situations more complex than the simple focus stack described above. Previous work on focus [3] used a semantic net and partitioning of the net to represent focus. We begin with a network representation, similar to the semantic net approach. We divided the focus tracking process in two parts: accumulation of focus-relevant events (also called conversation-relevant in [11]), and determination of focused item(s) based on a given criterion of interest.

The network representation contains nodes and links. Nodes represent objects in the domain and categories (semantic and conceptual). Each node holds an accumulated value indicating the "number" of mentions for the object it represents. Links connect objects to categories forming a directed graph (no cycles are allowed). Each link has a value representing the strength of association between the two nodes it connects.

As items are manipulated in the interface, "points" get accumulated for the objects and categories involved in the interaction. Later, to find a focused item, the user interface software must specify a criterion of interest. This is a representation of critical categories for the focus computation. Depending on a given criterion, different items will be in focus. For example, if a user interface action requires a parameter of type triangle and of color red (from the example in Figure 3, then the criterion of interest for the action would be (triangle = 1.0, red = 1.0, all others = 0.0). This criterion of interest is propagated through the network using a backward propagation constraint-based algorithm similar to the one described in [6]. The result is an accumulation of "focus" or "interest" on the objects in the network based on the given criterion. From the accumulated value, we can find a focused object (that is the object with the highest accumulated value). The criterion of interest can include multiple categories, with each category having a value between 0 and 1. Note that it is possible to have more than one object with the same accumulated value for a given criterion; this would be the result of an ambiguous reference.

We chose a network representation because it will support a far more sophisticated notion of the graphical analogue of focus. Our network representation:

- combines application (semantic) representation with a representation of user's attention.
- is easy to integrate information from several input devices into one representation (very important for multi-modal interaction techniques).
- is flexible and domain independent.
- is computationally easy to implement. Many existing algorithms for constraint propagation and graph traversal apply to our problem.
- will allow us to combine several imprecise indicators of attention (eye-gazes, interaction techniques, natural language mentions) into one integrated representation. The result is a flexible, multi-level representation of focus, driven by (possibly) different dialogue-specified criteria of interest. It avoids having discrete classifications (explicit/implicit focus) as in previous approaches. All items that are manipulated (or mentioned) get "points" each time they are used. The accumulated points combine with a specific criteria of interest and propagate through the network, to represent focus in a very flexible manner.

In the near future, we will finish the network implementation described in section . It will allow us to combine multiple sources of possibly imprecise information, for example, focus from previous commands, eye movements, other measurements, domain-determined importance or relevance. Such a network would support time-integrated selection (selects the item you have spent 75% of the last few minutes looking at or 75% if the last few commands referring to -- even though they were not consecutive).

FUTURE WORK

For the past year, we have been researching how to incorporate human dialogue properties into the dialogue of a graphical user interface. We have described a UIMS software architecture that allows for focus processing in a graphical

user interface. We have shown a prototype of interaction techniques based on focus and discussed how these techniques subsume CSO interaction techniques. Our research so far has concentrated on the following three questions:

(1) Where does discourse level processing best fit within a user interface management system paradigm?

(2) What are the appropriate analogues of conversational focus in a graphical human-computer conversation?

(3) What mechanisms can be used to realize it?

We have answered (1). Our first implementation allowed us to test the extended software architecture successfully. We can partially claim (2) as well, since our implementation will create behavior that is not exactly like existing human-human linguistic dialogue, but is the graphical analogue to it. Further testing is required. For (3), we have already implemented a focus stack, and we are currently implementing the network representation discussed in this paper.

ACKNOWLEDGMENTS

We want to thank Linda Sibert for her collaboration on this research, our colleagues in the Dialogue Research Program at NRL for helpful debates and discussions on these issues, John Sibert for his cooperation and interest on this research, and Mark Elsom-Cook helpful discussions concerning networks and the spreading activation approach. This work was sponsored by the Office of Naval Research.

REFERENCES

1. Buxton, B. (1991). The "Natural" Language of Interaction: A Perspective on Nonverbal Dialogues. In B. Laurel (Eds.), The Art of Human-Computer Interface Design Reading, Massachusetts: Addison-Wesley Publishing Company, Inc.

2. Foley, J. D., Dam, A. v., Feiner, S. K., & Hughes, J. F. (1990). Computer Graphics: Principles and Practice. Reading, Massachusetts: Addison-Wesley Publishing Company, Inc.

3. Grosz, B. J. (1978). Focus Spaces: A Representation of the Focus of Attention of a Dialogue. In D. E. Walker (Eds.), Understanding Spoken Language (pp. 269-285). New York: North-Holland.

4. Grosz, B. J. (1986). The Representation and Use of Focus in a System for Understanding Dialogues. In B. J. Grosz, K. S. Jones, & B. L. Webber (Eds.), Readings in Natural Language Processing (pp. 353-362). Los Altos, California: Morgan Kaufmann Publishers, Inc.

5. Hill, W. C., & Hollan, J. D. (1991). Deixis and the Future of Visualization Excellence. In Proceedings of the IEEE Visualization Conference. San Diego, CA

6. Hudson, S. E., & King, R. (1988). Semantic Feedback in the Higgens UIMS. IEEE Transactions on Software Engineering, 14(8), 1188-1206.

7. Jacob, R. J. K. (1983). Executable Specifications for a Human-Computer Interface. In A. Janda (Ed.), CHI'83 Conference Proceedings: Human Factors in Computing Systems, (pp. 28-34). Boston: ACM Press.

8. Jacob, R. J. K. (1983). Using Formal Specifications in the Design of a Human-Computer Interface. Communications of the ACM, 26(4), 259-264.

9. Jacob, R. J. K. (1986). A Specification Language for Direct-Manipulation User Interfaces. ACM Transactions on Graphics, 5(4), 283-317.

10. Jacob, R. J. K. (1994). Natural Dialogue in Modes Other Than Natural Language. In R.-J. Beun (Eds.), Natural Dialogue and Interactive Student Modelling Amsterdam: Springer-Verlag, in press.

11. Luperfoy, S. (1992). The Representation of Multimodal User Interface Dialogues Using Discourse Pegs. In Proceedings of the Association for Computational Linguistics.

12. Mac Aogáin, E., & Reilly, R. (1990). Discourse theory and interface design: The case of pointing with the mouse. International Journal of Man-Machine Studies, 32(May), 591-602.

13. Marsh, E., & Wauchope, K. Human-Machine Dialogue for Multi-Modal Decision Support Systems. In this proceedings.

14. Pérez, M. A., & Sibert, J. L. (1993). Focus on Graphical User Interfaces. In B. Hefley (Ed.), Proceedings of the International Workshop on Intelligent User Interfaces, (pp. 255-257). Orlando, Florida: ACM Press.