Data Exchange

The same data might sometimes be needed at different levels of the system. For instance, some output data (or different representations thereof) might reside inside the graphics system, inside the UIMS and inside the application. This is done for efficiency reasons, i.e. to minimise data exchange and data transformations. (Compare with traditional "cache-ing"). However, modification of data at a lower level must now also lead to modification of the corresponding data at the higher levels. This can be done via properly controlled shared access mechanisms. Alternatively, the higher levels might have to inspect the data at the lower levels and deduct updates from there (via an inverse transformation).

Interaction Primitives

It would be very helpful to the interaction designer if there existed "standard interaction primitives" that were very well suited for building complex user interfaces. The abstractions (LID's) that exist in current graphics systems were mainly designed to enable portability. They are considered inappropriate for building user interfaces. It is unlikely that these interaction primitives can be put on top of a graphics system. However, they might merely extend the GS (and as such share the physical device drivers). An example of such abstract interaction primitive is a "rubberband" function. An appropriate set of these primitives has not yet been defined.

5. REFERENCES

Anson E (1982) The device model of interaction. Proceedings of SIGGRAPH 82. Computer Graphics 3:107-114

Borufka HG, Kuhlmann HW, ten Hagen PJW (1982) Dialogue cells: a method for defining interactions. IEEE CG&A, July:25-33

Herman I, Krammer G, Tolnay-Knefely T, Vincze A (1984) Picture book about user interface management and associated representation kernels. Proceedings of this workshop

Kamran A (1984) Issues pertaining to the design of the abstract interaction handler. Proceedings of this workshop

Matthys J (1984) The input tool model - a personal experience. Proceedings of this workshop

Pfaff G (1983) Construction of operator interfaces based on logical input devices. Acta Informatica 19:151-166

Strubbe HJ (1984) Components of interactive applications. Proceedings of this workshop

ten Hagen PJW (1984) The relation between a UIMS and an application. Separate Subgroup report. Proceedings of this workshop

Van den Bos J, Plasmeijer MJ, Hartel P (1983) Input-Output tools: a language facility for interactive and real-time systems. IEEE Trans. Software Eng. 3:247-259

Report on Dialogue Specification Tools

M. Green

Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada

INTRODUCTION

One of the goals of User Interface Management Systems (UIMS) is the automatic (or semi-automatic) construction of user interfaces. In order to accomplish this a description of the user interface to be implemented must be available. This report addresses three questions related to user interface descriptions. These questions are: what user interface descriptions are required to automatically produce a user interface, how do these descriptions relate to the human factors of user interfaces, and how can the existing user interface description techniques be classified? In order to address these questions an abstract model of a UIMS has been developed. This model does not represent how a UIMS should be structured or implemented, instead it presents the logical components that must appear in a UIMS. Each of these components has a different function, and different description techniques are required for each one. The logical model of a UIMS is shown in fig. 1. This model is similar to ones proposed by Edmonds [Edmonds 1982], Green [Green 1984], and Olsen [Olsen 1984]. Each of the components of this model will now be discussed.

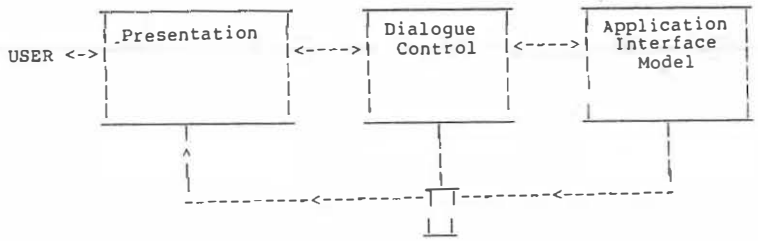

Fig. 1 Logical model of a UIMS

The presentation component is responsible for the external presentation of the user interface. This component generates the images that appear on the display screen, and reads the physical input devices, converting the raw input data into the form required by the other components in the user interface. The user interface employs an abstract representation for the input and output data. This representation consists of a type or name that identifies the kind of

data, and the collection of values that define the data item. This representation is similar to that used to represent tokens in a compiler [Aho and Ullman 1977]. For this reason the term token is used for the chunks of information handled by the UIMS. The main purpose of the presentation component is to translate between the external physical representation of the tokens, and their internal abstract representation. In most cases this translation is a simple one-to-one mapping.

The external-internal mapping can be viewed as a dictionary, with one entry for each of the external and internal data items. This entry indicates how the token is to be translated. The presentation component has no control over the contents of the dictionary, it cannot change the external-internal mapping. The dictionary can only be changed by the dialogue control component. The dictionary is a generalization of the physical/logical device bindings that appear in some graphics packages. In terms of Foley's language model [Foley and Van Dam 1982] the presentation component deals with the lexical aspects of the user interface.

The dialogue control component defines the structure of the dialogue between the user and the application program. This component receives a linear sequence of input tokens from the presentation component, and a linear sequence of output tokens from the application. Based on these two sequences of tokens the dialogue control component determines the structure of the interaction, and routes the tokens to their appropriate destinations. The dialogue control component can be viewed as the mediator between the user and the applications program. The user, through the presentation component, makes requests and supplies data to the application program. The input tokens representing these requests and data are examined by dialogue control, and routed to the appropriate routines in the applications program. Similarly the application program generates requests for data and answers to the user requests. The dialogue control component must channel these output tokens to the appropriate parts of the presentation component.

Unlike the presentation component, the dialogue control component must maintain a state and have control over it. The actions performed by this component will usually depend upon the context of the dialogue, therefore, any notations for it must be able to handle dialogue states and state changes. This component can also control the state of the presentation component, thus a means of specifying these state changes is also required.

The application interface model is a representation of the application from the viewpoint of the user interface. This component defines the interface between the UIMS and the rest of the application program. The application interface model defines the semantics of the application. This representation includes the data objects that are maintained by the application, and the routines the user interface can use to communicate with the application. This component contains the information required by dialogue control for routing tokens to the appropriate place within the application. This component also contains constraints on the use of the application routines. This allows the user interface to check the semantic validity of the user input before the application routines are invoked. This information can also be used in error recovery and undoing user actions. This component of a user interface has not appeared explicitly in any of the existing UIMSs. But, most of them have some mechanism for performing some of the functions we have assigned to

it. The exact nature of this component will evolve as it is incorporated into UIMSs and used in practice.

HUMAN FACTORS

In this section the relationship between the human factors of user interfaces, and the three user interface components is explored. There are two aspects to this relationship. First, how can human factors experts help us design the three user interface components? By dividing the user interface into three logical components it may be possible to develop better design guidelines, than by considering the user interface as a whole. Second, given descriptions of each component, is it possible to evaluate these descriptions from a human factors point of view?

Starting with the first aspect and the presentation component of the user interface, it appears that name spaces are a key issue. The presentation component is responsible for mapping between the user's symbolism, and the internal representation of the user interface. When the user approaches an application he has a certain set of symbols he uses to name the physical and conceptual objects in the application. In order to have a smooth dialogue the presentation component must deal with the user in terms of these symbols. The design of this set of symbols is where human factors enters the design of the presentation component. We need advice on how the symbols should be designed, how the symbol space should be structured, and the effects of interactions between symbols. Should the symbol space be designed in such a way that all the symbols are obviously disjoint, or is there something to be gained from the interaction between symbols? In order to address these questions advice from experts in human factors and cognitive psychology is needed.

In the dialogue control component the key human factors issue is command and dialogue structure. It is fairly well known that certain command formats and dialogue structures are superior to others, and numerous sets of guidelines have been produced for command language design. It has also been shown that the interaction between commands can have an effect on the usability of a user interface. There is more readily available information on the design of this component than the other two components. But, there are still a large number of human factors issues to be addressed. One of the key issues is the effect of chunking and closure on dialogue design. How can we design a dialogue to take advantage of the user's natural chunking ability? Also, how can we design a dialogue so the closures occur at natural places, and the dialogue does not overload the user's processing abilities?

There is very little we can say about the human factors of the application interface model. The design of this component should be based on the user's model of the application. In many ways this component reflects the user's view of the semantics of the application. It must provide the operations that the user wants to perform in the application. If a particular operator is missing the user must construct a sequence of operators which has the same effect, and this may not always be possible. This imposes a greater cognitive load on the user than invoking a single command.

The design of both the presentation component and the application interface model require a user's model of the application. In this context a user's model is the model the user has of the application. This model exists in the user's mind and he is usually not aware of it. There are two problems with user's models. The first is, how do we determine the model the user has of the application? Since the user is usually not aware of this model, he cannot produce a written version of it. It may be possible to interview the user, and then construct an approximate model from the results of the interview. But, we currently have no way of knowing if this model is correct. The second problem is, given that it is hard to extract the user's model from the user, is it possible to design a user's model, and then transfer it to the user? If this were possible the first problem disappears, and we are left with the problems of designing good user's models and transferring them to the user. The problems of user's modeling are beyond the scope of this workshop, but these problems must be addressed in order to design good user interfaces.

The second human factors issue, the evaluation of user interface descriptions, is motivated by the precise descriptions of the user interface that are produced when a UIMS is used. The main purpose of these descriptions is to facilitate the automatic or semi-automatic production of a user interface. But, when such descriptions exist there is a great potential to use them for other purposes. Some techniques have already been developed for user interface evaluation. The best known of these are the grammar based techniques of Reisner, and the keystroke model of Card, Moran, and Newell. Both of these techniques require a complete description of the user interface. A UIMS should produce a human factors evaluation of a user interface at the same time it was producing the implementation of it. This would give the user interface designer a feel for how the user interface will perform, and where it could be improved. This could lead to a process of user interface debugging similar to the programming language debugging that is currently done with compiler generators.

NOTATIONS

Over the past few years a number of UIMSs have been designed and implemented. All of these systems have, in some form or another, the three user interface components identified in section 1. The designers of these systems have developed notations for these components, largely without knowledge of what other UIMS designers were doing. One of the major goals of this working group was to analyze, and attempt to classify the notations that have been used in UIMSs. The classification scheme presented here is by no means complete, but it does cover most of the existing UIMSs. It is interesting to note that most of the members of this working group already had a classification scheme in mind before the workshop. The classification scheme that resulted from our discussions does not differ radically from any of these preconceived schemes.

Our classification scheme is based on the three user interface components. Each component has its own set of notations. The notations for dialogue control are further divided into three groups.

Presentation Component

For the presentation component the notations must deal with both input and output. At the present time output notations deal mainly with graphical output. The range of these notations should be extended to cover sound, touch, and movement (as in robotics). Most of the notations for graphical output are based on the routines provided by standard graphics packages. Calls to these routines are combined to form the symbols presented to the user. This form of notation is very similar to a programming language, and in most cases the target language of the UIMS is used as the basis for the notation.

This is not an acceptable form of notation, for the following two reasons. First, a textual language is used to describe something that is graphical in nature. As a result the notation is both hard to use and read. Given a collection of subroutine calls it is not an easy task to determine the image they produce on the display screen. What is really required is a graphical notation for the output symbols. One attempt at this can be found in the MENULAY system [Buxton et.al. 1983]. Second, by using a programming language notation the user interface designer is forced to do the work of the UIMS. The user interface designer should produce a high level description of the symbol, and let the UIMS convert it into a program.

A similar situation exists on the input side of presentation component notations. In most cases the input primitives provided by a graphics package are used as the input symbols generated by the user. Most of the UIMSs do not provide a mechanism for combining the input primitives into more complex symbols at the presentation level. Currently this function is performed by the dialogue control component. The range of input primitive should be extended to cover video, voice, and more complicated body gestures (such as character recognition). As of yet no really useful input notations have appeared.

The last issue to be dealt with in presentation component notations is the external-internal mapping. This mapping is controlled by the dialogue control component, but it resides in the presentation component. There are several issues associated with this mapping. The first is whether the mapping is a simple table lookup, or can some form of simple decision making be involved. These decisions could be based on the values of the token. For example, the presentation of an output token may depend upon the magnitude of its values. A detailed presentation would be given for small values, and an overview for large values. If decision making is included in the mapping what form should it take, and how would it be expressed? The presentation component contains the definitions of internal and external tokens. The dialogue control component must be able to reference these tokens in order to establish the mapping between them. How does the presentation component assign a name to the tokens, and how are these names passed to the dialogue control component? These issues must be addressed in any notation for the presentation component.

Dialogue Control Component

The dialogue control component has the most highly developed notations. Since most of the existing UIMSs have concentrated on this component, it has more notations and there is considerably more experience with their use. The notations that have been used for this component mainly fall into three groups. The major difference between these groups is the model the notations have of the user interface. The transition network notations view the user interface as a collection of states, and the user's actions cause transitions between these states. The grammar group views the dialogue between the user and the computer as a language, and uses grammar based techniques to describe this interaction. The event group views the user interface as a collection of events and event handlers. When the user interacts with the computer, one or more events are generated, which are processed by the event handlers, possibly generating more events. This is not a complete classification scheme for dialogue control notations, since at least one well known system does not fit (the Tiger system of Kasik [Kasik 1982]).

The transition network group is the oldest group of notations for dialogue control. This approach to user interface management dates back to at least 1968 with the work of Newman [Newman 1968]. This style of notation has been used in a number of systems since that time, and a considerable amount of experience has been gained in its use. A pure transition network, consisting solely of states and transitions between these states, is not powerful enough to handle a wide range of user interfaces, and tends to be hard to use. This is due to the fact that most user interface have a large number of states, with a large number of possible transitions between them. As a result, several schemes have been developed for partitioning the network. One approach is the user of subnetworks. A subnetwork is an independent transition network having its own set of states and transitions. A subnetwork can be used to replace any of the states or transitions in another network. Once a subnetwork has been entered it retains control of the dialogue until one of its terminal states is reached. When this happens control returns to the point, in the calling network, where the subnetwork was invoked. Thus, compound dialogues can be built up from smaller dialogues, each represented by a subnetwork. An extension of this approach is recursive transition networks or RTNs, where the subnetworks are capable of invoking themselves recursively. Experience with transition networks indicates that RTNs are necessary to handle the types of user interfaces that arise in practice. An example of a notation based on RTNs is SYNICS [Edmonds 1981].

Experience with transition network based systems indicated that a multi-threaded implementation is desirable. This allows the user to interact with several networks at the same time. This can be used in help processing, where there is a separate network for the help command. The user can invoke the help network without fear of losing his place in the original dialogue.

The grammar based notations use techniques from programming languages for both the description, and implementation of the dialogue control component. These notations are based on using context free grammar to describe the dialogue between the user and the program. As with transition diagrams, pure context free grammars do not form an ideal notation for the dialogue control component, so numerous extensions have been made. Most of these extensions give the dialogue designer

more control over the order in which the user's input is parsed. For example, it may be possible to specify that the ordering of the symbols in a production is arbitrary. This allows the user to enter these symbols in any order. Other extensions deal with error detection, error recovery, and the ability to undo parts of the dialogue. It should be noted that the descriptive power of RTNs and context free grammars is the same. There is some indication that RTNs may be easier to use than context free grammars. Two examples of grammar based notations are SYNGRAPH [Olsen and Dempsey 1983] and DIALOG [Derksen 1983] [Borufka,. et.al. 1982]. Closely related to this approach is the work of van den Bos [van den Bos 1980].

The event model is not as well known, or as highly developed as the other two groups of dialogue control notations. Since the event model is not widely known a brief description of it is presented here. The event notations are based on the concepts of events, and special procedures called event handlers. In may ways events are similar to the input and output tokens discussed in section 1. Each event has a name and a collection of data values. An event is generated each time the user interacts with an input device. These events are processed by one or more of the event handlers associated with the input, or display device involved in the interaction. An event handler is a procedure that performs a set of actions based on the name of the event it receives. These actions include passing output tokens to the presentation component, passing input tokens to the application interface model, performing some calculation, or generating new events. The collection of events processed by an event handler can be viewed as a state. The set of event handlers active (able to receive events) at any one time defines the legal user actions at that point in the dialogue. This set can be changed by disassociating an event handler with a particular device, or associating an new event handler with a device. An event may be sent to more than one event handler. In this case each event handler is responsible for one aspect of the event's processing, such as the different levels of feedback, error checking, and routing to high levels of the user interface. The event model is in many ways similar to object oriented programming as in Smalltalk [Goldberg and Robson 1983].

Most of the event based notations have the appearance of programming languages. This is mainly due to the procedural nature of the event handlers. These notations need some way of describing both the events, and how the events are processed. In most notations both of these components are combined into one description, that of the event handler. That is, each event handler defines all the events it can receive, even if these events are used in several event handlers. This results in self contained event handlers, at the price of increasing the redundancy of the descriptions, and possibly introducing inconsistencies. The description of an event handler contains one section for each of the events it can process. The section for an event starts with a description of the event itself, followed by the sequence of actions required to process the event.

Application Interface Model

Since the application interface model has not explicitly appeared in an existing UIMS, there have been no notations developed for it. In most UIMSs the application procedures called by the user interface form an implicit application interface model. While this identifies the application procedure used by the user interface, it tells us

very little about the application, and these calls are usually embedded in other descriptions making them hard to find. The application interface model must contain certain information about the application. To be useful the application interface model must cover at least the following three areas. First, it must contain a description of the application data structures that are of interest to the user and the user interface. The description of these data structure is at an abstract level, and implementation details are usually not important. The UIMS only needs to know the information that is stored in the data structures, and how it can find it. Second, there must be a description of the application procedures available to the user interface. This description must include the name of the procedure, and the operands it expects. This part of the description defines the interface between the user interface and the application. Third, the constraints on the application of the operators must be outlined. These constraints include any pre-conditions for the operators, and any ordering restrictions on them. This allows the user interface to filter out some of the semantically illegal operations before they reach the application. While this is the minimum amount of information that must be included in the application interface model, there are several other things that would be useful. One of these is the effects, or post-conditions of the operators. This allows the user interface to anticipate the effects of commands, provide automatic help, and automatically perform some undo processing. Another useful component of an application interface model is procedures for performing standard tasks in the application. These procedures could be used by the user interface to guide naive users, and provide sophisticated help facilities.

Since notations for the applications interface model have not been developed, a classification scheme for them cannot be presented. Instead, several possible notations are presented, along with several of the issues that arose in our discussions. One possible notation is objects and operators. The objects correspond to the data structures in the application program, and the operators correspond to the application procedure available to the user interface. A notation of this kind, called UML, is described in [Green 1981] and [Green 1984]. In order to handle a wide range of user interfaces this type of notation must treat object descriptions as type definitions, and allow the creation of arbitrary numbers of object instances. In a number of applications a network of objects is required, so notations that only support object hierarchies are not desirable. Another issue related to this type of notation is the parameters to the operators. An operator definition will contain a number of implicit parameters, but should these be the only objects available to the operator? Should there be global objects that represent the context of the interaction? If there are, how are these global objects defined and referenced?

Another possible notation for the application interface model is based on relations (as in relational database [Date 1981]), and first order logic. This idea was prompted by the work done by Garret and Foley in graphical databases [Garret and Foley 1982]. The relations are used to represent the data structures in the application, while statements in first order logic model the effect of the application procedures.

Two issues that must be addressed by any notation for this component are invariants, and sequential relationships between application procedures. Invariants state properties of the application that are always true. There are two reasons why they are a useful part of the

applications interface model. First, they form a concise description of general properties of the application. In most cases these properties could be described by other means, but this will usually entail a large amount of redundant material. Second, the UIMS can use them to detect errors in user input, and as the basis for guiding the user through interaction sequences. One issue related to invariants is whether they should be passive or active. A passive invariant describes some property of the application, while an active invariant takes an active part in the computation. That is, either the application, or the UIMS will perform whatever actions required to maintain the truth of an active invariant.

Some of the procedures in the application cannot be performed in arbitrary sequences. For example, a file cannot be processed until it is open, therefore an open call must always precede any read or writes on the file. This is an example of a sequencing constraint.

INTERRELATIONS BETWEEN COMPONENTS

Figure 1 shows the interfaces between the components in our model of a user interface. These interfaces represent the flow of information or control between the components. In order to completely describe the user interface, the nature of these interfaces must be understood.

There are two issues related to the interface between the presentation component and dialogue control component that effect user interface notations. One of these issues is the form of the tokens flowing between these components. The are a number of ways in which this flow can be viewed. In one view, dialogue control treats the presentation component as a collection of logical devices. In this view there are problems related to the definition of device classes, and device characteristics. One device characteristic is the type of the value produced by the device. Most graphics packages have a fixed set of types that can be used for device values. In the case of the presentation component, a fixed type structure may be too restrictive. On the other hand, the full type definition facilities of modern programming languages may be more than is required here, and needlessly complicate the interface between these components. Work needs to be done on the set of types required to support the communications between these two components. A related issue is the handling of picks. In existing graphics packages there is one type for all pickable objects. This type usually refers to the display file segment containing the object selected, and not the object itself. With the presentation component, each pickable object type could have its own device type, and the pick would contain a pointer into the application data structure identifying the object selected. This would relieve the dialogue control component of determining the object selected by the user. Another important device characteristic is its mode (as in the GKS model of graphical input [Rosenthal et.al. 1982]). What are the interpretations of event, sample, and request modes at the level of the presentation component? If the logical device view is not taken, some of these issues disappear. Another view of this interface is to treat the output of the presentation component as a simple stream of tokens, without any information on the devices that produced them (either logical or physical). In this view the problem of device modes does not occur, but the type problems remain.

Another important feature of this interface is the manner in which dialogue control exerts its control over the presentation component. Dialogue control is responsible for the state of the presentation component, it controls the external-internal mapping. How much information does dialogue control need in order to adequately perform this function? It must at least know the names of all the internal and external tokens, otherwise it will not be able to define the mapping. But, does it need more detailed information about these tokens? For example, should the physical devices associated with the external tokens be available to dialogue control? Or, is there a set of general device properties that would serve its needs? Another issue is whether dialogue control or the presentation component is responsible for enabling or initializing devices. The presentation component deals directly with the devices, so there is some argument for having it responsible for device initialization and control. On the other hand, dialogue control is responsible for the state of the presentation component, so it knows when the devices should be enabled. It may be necessary to share this responsibility between these two components.

At the present time there are far more issues associated with this interface than there are answers. More experience with UIMS implementations is required before a more detailed description of this interface can be produced.

The major issue associated with the interface between dialogue control and the application interface model, is the access dialogue control has to the application data structure. In particular, can dialogue control directly access the application data structure without informing the application? An argument can be made that dialogue control must always call an application routine whenever it wants to change the application data structure. This isolates the user interface from the implementation of the application data structure, and ensures that all modifications to the data structure are legal. On the other hand, this approach can be too inefficient for some applications, and places the burden of error recovery and undo processing on the application. This was an issue at the previous workshop is Seattle, and still appears to be unresolved. One possible, but inefficient, solution is to give the UIMS its own copy of the application data structure. The UIMS is free to modify this data structure without informing the application routines. At key points in the dialogue (determined by dialogue control) the UIMS copy of the data structure is used to update the application copy. In this approach error recovery and undo processing can easily be accomplished by restoring the UIMS copy of the data structure from the application's version.

There are two issues associated with the interface between the presentation component and the application interface model. The first issue deals with picking. The result of a picking operation should be some object in the application data structure that is currently displayed on the screen. The presentation component is responsible for the allocation of screen space, and the appearance of everything on the screen. But, it does not know about the application data structure. When a pick occurs the presentation component knows the coordinates of the pick, but it doesn't know how to relate these coordinates to the contents of the application data structure. Some form of cooperation between the two components is required to determine the object the user selected. There needs to be some correlation mechanism between the coordinates in the presentation

component, and the data structure maintained by the application.

The other issue involves the flow of output data from the application to the presentation component. In theory, all the information from the application, that is to be displayed, must go through dialogue control. In most cases dialogue control is not interested in the actual data, it assigns the data a format or template, and passes it on to the presentation component. Since dialogue control does not need to process this data, it could be directly transferred to the presentation component, saving some processing time. This flow of data is represented by the arc flowing from the application interface model to the presentation component in fig. 1. Dialogue control has control over this flow of information, it assigns the formats to the output data, and establishes the pipe line between the application and the presentation component. Once the pipe line has been established, dialogue control does not take part in the information transfer. This approach is particularly effective when large amounts of data must be transferred from the application to the screen.

SUMMARY

In this report we have presented some of the issues pertaining to the notations used in UIMSs. At the present time there are a small number of implemented UIMSs, so there is some experience to draw upon. This is the first workshop of this nature where there has been a significant number of participants with implementation experience. This experience seemed to raise more issues than it resolved. The conclusion that can be drawn from this is there is still a considerable amount of work to be done in notations for UIMSs.

One of the major problems encountered by this working group was the inability to compare different UIMSs. The systems discussed in this group have been used to produce user interfaces for different application areas, with different interaction styles and requirements. This makes it very difficult to compare the ease of use (for the user interface designer), and the quality of the resulting user interface. This problem could be partially solved by constructing a standard set of user interface problems. Each of the UIMSs could be used to solve these problems, and the results used as a means of comparison. This set of problems could be viewed as a benchmark test for user interface management systems. Along with developing the problems, techniques for measuring ease of use and the quality of the user interface must also be developed.

REFERENCES

[Aho and Ullman 1977] Aho A.V., J.D. Ullman, *Principles of Compiler Design*, Addison-Wesley, Reading Mass., 1977.

[Borufka, et.al. 1982] Borufka, H.G., H.W. Kuhlmann, P.J.W. ten Hagen, "Dialogue Cells: A Method for Defining Interactions", IEEE Computer Graphics and Applications, vol.2 no.5, p.25, 1982.

[Buxton et.al. 1983] Buxton W., M.R. Lamb, D. Sherman, K.C. Smith, "Towards a Comprehensive User Interface Management System", SIG-

GRAPH'83, p.35, 1983.

[Date 1981] Date C.J., *An Introduction to Database Systems*, Addison-Wesley, Reading Mass., 1981.

[Derksen 1983] Derksen J., "Een ontwerp van programmeergereedschap voor dialoogsystemen gebaseerd op dialoogcellen (Tools for dialogue systems based on dialogue cells)", TNO-IBBC rapport nr. B1-83-62/68.0.0004, 1983 (in Dutch).

[Edmonds 1982] Edmonds E.A., "The man-computer interface - a note on concepts and design", Int. J. Man-Machine Studies, vol.16, p.231, 1982.

[Edmonds 1981] Edmonds E.A., "Adaptive Man-Computer Interfaces", in Coombs M.J., J.L. Alty (ed.), *Computing Skills and the User Interface*, Academic Press, London, 1981.

[Foley and Van Dam 1982] Foley J., A. Van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading Mass., 1982.

[Garret and Foley 1982] Garret M.T., J.D. Foley, "Graphics Programming Using a Database System with Dependency Declarations". ACM Transactions on Graphics, vol.1, no.2, p.109, 1982.

[Goldberg and Robson 1983] Goldberg A., D. Robson, *Smalltalk-80: The Language and its Implementation*, Addison-Wesley, Reading Mass., 1983.

[Green 1984] Green M., "Design Notations and User Interface Management Systems", in this volume, 1984.

[Green 1981] Green M., "A Methodology for the Specification of Graphical User Interfaces", SIGGRAPH'81, p.99, 1981.

[Kasik 1982] Kasik D.J., "A user Interface Management System", SIGGRAPH'82, p.99, 1982.

[Newman 1968] Newman W.M., "A System for Interactive Graphical Programming", SJCC 1968, Thompson Books, Washington DC., 1968.

[Olsen 1984] Olsen D., "Presentational, Syntactic and Semantic Components of Interactive Dialogue Specification", in this volume, 1984.

[Olsen and Dempsey 1983] Olsen D., E. Dempsey, "SYNGRAPH: A Graphic User Interface Generator", SIGGRAPH'83, p.43, 1983.

[Rosenthal et.al. 1982] Rosenthal D.S.H., J.C. Michener, G.Pfaff, R. Kessener, M. Sabin, "The Detailed Semantics of Graphics Input Devices", SIGGRAPH'82, p.33, 1982.

[van den Bos 1980] van den Bos, J., "High-level graphics input tools and their semantics", in Guedj, R.A., et.al. (eds.), *Methodology of Interaction*, North Holland, p.159, 1980.

Working Group Members:
Jan Derksen, Ernest Edmonds, Mark Green, Dan Olsen, Robert Spence

Report on the Interface of the UIMS to the Application

G. Enderle

SEL AG, Helmuth-Hirth-Strasse 42, 7000 Stuttgart 40, Federal Republic of Germany

SCOPE AND GOAL

The goal of the workshop was to find, evaluate, and describe models, principles, methods, and tasks for the design of User Interface Management Systems. This working group concentrated on aspects of this overall goal that are related to the interface between the application and the UIMS.

One application was given specific attention: Computer Aided Design (CAD). CAD is an application that, on one hand, relies on Computer Graphics as one basic component of the whole system. On the other hand, CAD is also an application area where a favourable user interface is of special importance. The application is the most relevant part of the interactive system. If we look at the application-UIMS interface from the viewpoint of the application, we can identify certain requirements which a UIMS must fulfil to serve the application in an optimal way. A problem of specific interest at the application-UIMS interface is the division of responsibility for graphical output between the application and the UIMS.

PROBLEM AREAS

Three major problem areas related to the application interface of a UIMS were addressed:

- The configuration of a UIMS with respect to a specific application,
- The responsibility for creating and managing graphical output,
- The services and functions available at the communication interface between the application and the UIMS.

In order to gain a common base for the discussion of these questions, an overall structure of the interactive system had to be agreed on. In all problem areas, special attention was paid to CAD aspects.

STRUCTURE OF USER INTERFACE MANAGEMENT SYSTEMS

The application-UIMS interface can be situated at different places within the overall system depending on the design of the structure of

User Interface Management Systems

Proceedings of the Workshop on User Interface Management Systems held in Seeheim, FRG, November 1-3, 1983

Edited by Günther E. Pfaff

With 69 Figures

Springer-Verlag
Berlin Heidelberg New York Tokyo

Eurographic Seminars
Edited by G. Enderle and D. Duce
for EUROGRAPHICS -
The European Association for Computer Graphics
P.O. Box 16
CH-1288 Aire-la-Ville

Editor:
Günther E. Pfaff
GTS/GRAL
Alsfelderstraße 7
D-6100 Darmstadt

ISBN 3-540-13803-X Springer-Verlag Berlin Heidelberg New York Tokyo
ISBN 0-387-13803-X Springer-Verlag New York Heidelberg Berlin Tokyo

Library of Congress Cataloging in Publication Data.
Workshop on User Interface Management Systems (1983 : Seeheim-Jugenheim, Germany) User interface management systems. (EurographicSeminars) Bibliography: p.
1. Interactive computer systems-Congresses. 2. Computer graphics-Congresses.
I. Pfaff, G. (Günther), 1951-. II. Title. III. Series. QA76.9.I58W67 1983 001.64'404 85-2831
ISBN 0-387-13803-X (U.S.)

Printing: Beltz Offsetdruck, Hemsbach/Bergstr. Bookbinding: J. Schäffer OHG, Grünstadt
2145/3140-543210

Editors Introduction

The book contains the proceedings and reports of the "Workshop on User Interface Management Systems", held in Seeheim, Federal Republic of Germany, November 1-3, 1983. The workshop brought together experts in using and developing techniques for managing the dialogue between users and interactive graphics systems. The purpose of the workshop was to produce an agreed report contrasting existing approaches, and outlining directions for future work. Four different areas were defined and addressed at the workshop, namely

a) role, model, structure and construction of a UIMS
b) dialogue specification tools
c) interface of the UIMS to the application
d) user's conceptual model

All participants prepared papers each in one of those problem areas. The papers have been rewritten in the light of the issues discussed during the workshop. Also a subgroup report was produced for each problem area summarizing