Interner Bericht

An Object-Oriented Architecture for User Interface Management in Distributed Applications

Ralf Denzer

222/92

Fachbereich Informatik

Universität Kaiserslautern · Postfach 3049 · D-6750 Kaiserslautern

An Object-Oriented Architecture for User Interface Management in Distributed Applications

Ralf Denzer

222/92

Universität Kaiserslautern
AG Computergraphik
Postfach 30 49
6750 Kaiserslautern

Mai 1992

Herausgeber: AG Graphische Datenverarbeitung und Computergeometrie
Leiter: Professor Dr. H. Hagen

An Object-Oriented Architecture for User Interface Management in Distributed Applications

Ralf Denzer

University of Kaiserslautern
Postfach 3049
6750 Kaiserslautern, Germany

Abstract

User interfaces for large distributed applications have to handle specific problems: the complexity of the application itself and the integration of online-data into the user interface. A main task of the user interface architecture is to provide powerful tools to design and augment the end-user system easily, hence giving the designer more time to focus on user requirements.

Our experiences developing a user interface system for a process control room showed that a lot of time during the development process is wasted for the integration of online-data residing anywhere but not in the user interface itself. Furtheron external data may be kept by different kinds of programs, e.g. C-programs running a numerical process model or PROLOG-programs running a diagnosis system, both in parallel to the process and in parallel to the user interface.

Facing these specific requirements, we developed a user interface architecture following two main goals: 1. integration of external information into high-level graphical objects and 2. the system should be open for any program running as a separate process using its own problem-oriented language. The architecture is based on two approaches: an asynchronous, distributed and language independent communication model and an object model describing the problem domain and the interface using object-oriented techniques. Other areas like rule-based programming are involved, too.

With this paper, we will present the XAVIA user interface architecture, the (as far as we know) first user interface architecture, which is consequently based on a distributed object model.

Keywords: User Interface Management Systems, Distributed Systems, Object Orientation, Rule Based Programming

1. Introduction

The task of technical plant operation is to manage a complex dynamical system in an optimal way. Most of the difficulties involved in this task are coming from the mass of data and from the causal and temporal relationships within the plant, which are often not exactly known. In critical situations operators have to decide in short time how to react on unforseen events. The decision process is rather difficult because on one hand there is too much "low level" information (up to some hundreds or thousands of alarms within a few minutes), but on the other hand there is a lack of "high-level" information, namely what to do next. This situation has been described by Sachs with the term *cognitive overload.* [3].

A lot of work has been done in the real time community to develop systems supporting operators with diagnostic knowledge, to filter information at runtime or to predict critical situations [3-9]. User interfaces have been improved using high resolution graphics and knowledge based techniques [4,5]. Graphical and pictorial informations are good media to use the human-computer information channel more efficiently, which is extremely important if operators are under time pressure.

In 1989, Foley described some of the goals of future user interface development tools [10], which should provide context-sensitive, knowledge based help. In case of process control systems we think, that not only help but the whole dialogue should depend on the current context. Another aim should be to assist the user interface designers doing their work. As in our case the user interface designer has a lot to deal with the question how to embed remote information into his system, we found it useful to concentrate a part of our work on the integration problem in distributed environments.

There have also been approaches to concurrent and parallel systems and to communication methods in different areas of user interface techniques [11-14], some of them claiming to follow the Seeheim Model [15] and some of them criticizing it. There is still a lot of work to do on the way to highly parallel and direct manipulation user interfaces, such as multi agent models [16,17].

We think that object oriented techniques [18-21] for user interface management systems in combination with knowledge based approaches and distributed processing will play an important role in the future. We have tried an integration of these three fields, mainly focussing on the process control room.

2. Goals

2.1 Adaptive user interface design

To design the user interface we incorporate any kind of useful information: native (internal or remote) data, high-level graphical representations, hypertext, images and knowledge. The knowledge coming from plant designers and operators is specifically used to adapt the graphical representation and the dialogue to a given process context, in order to minimize the time for perception and interaction.

2.2 Integration of remote information

The process control environment is a highly distributed environment. True decision support means integrating many different methods like diagnosis, quality control, models, observers and so on (fig. 1). Because of the methods being of very different kinds of nature, different software systems are used.

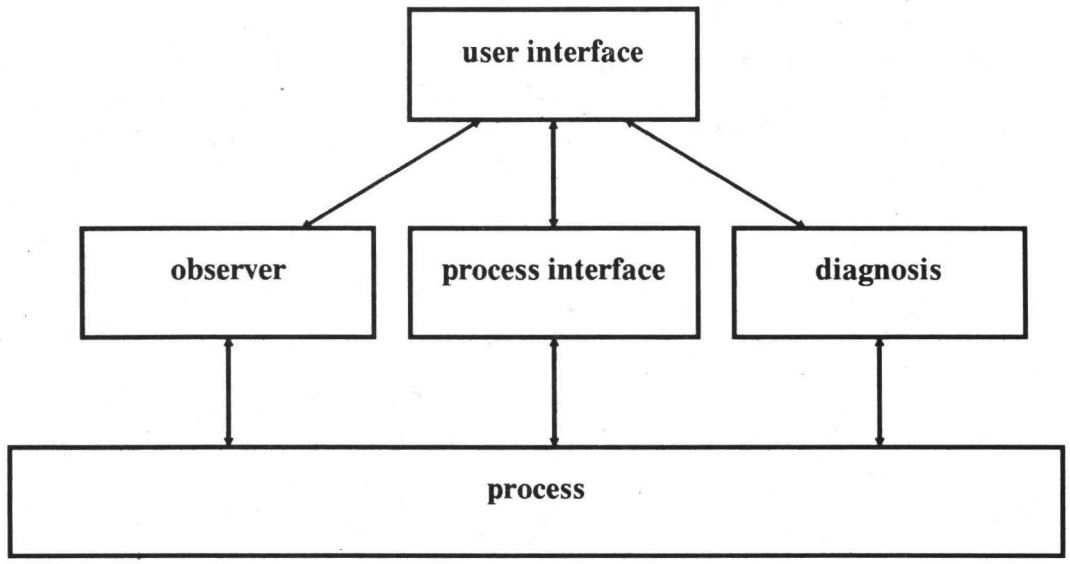

Fig. 1: Distributed environment

Therefore, we concentrated one part of our work on the following question: *how can we integrate informations generated by any process distributed over a network into the user interface, doing this in an easy and very general way, thus keeping our system open?* Thereby we use the word "easy" in terms of "as easy as possible as a principle" and "easy to actually do the integration of remote data".

A specific requirement from process control was, that the whole information processing has to be asynchronous (as we never know when events occur) and, if not realtime, at least prioritized.

3. Architecture

3.1 Main concepts

Our architecture consists of several parts which are independent of each other, namely

- a language- and network- independent communication model,

- an extendable message protocol,
- an object interface providing links between remote objects and
- an object system acting as the user interface.

Each of the parts is built on top of the previous one and uses its features. Nevertheless, a given application does not have to use each of the parts. E.g. it is possible to run a C-program as a measurement process on a remote node without using the fourth layer; the C-program only has to stick to the communication model (which does not have a syntactical view on the messages it transports) and to the message protocol/object interface (which do have syntax).

The *communication model* uses standard communication mechanisms as they are defined by the OSI reference model. The kernel of the communication model is a decentralized and asynchronous message passing system, which provides link and message management. Each process in the distributed environment can communicate with each other. On top of the message passing system we have defined a C-interface which is used to interface to other programming languages (fig. 2).

The *message protocol* and the *object interface* define the way applications share informations. The main concept used is the connection of objects via *request-* and *state-change-messages*. An object willing to use an information residing in a remote object has to send a

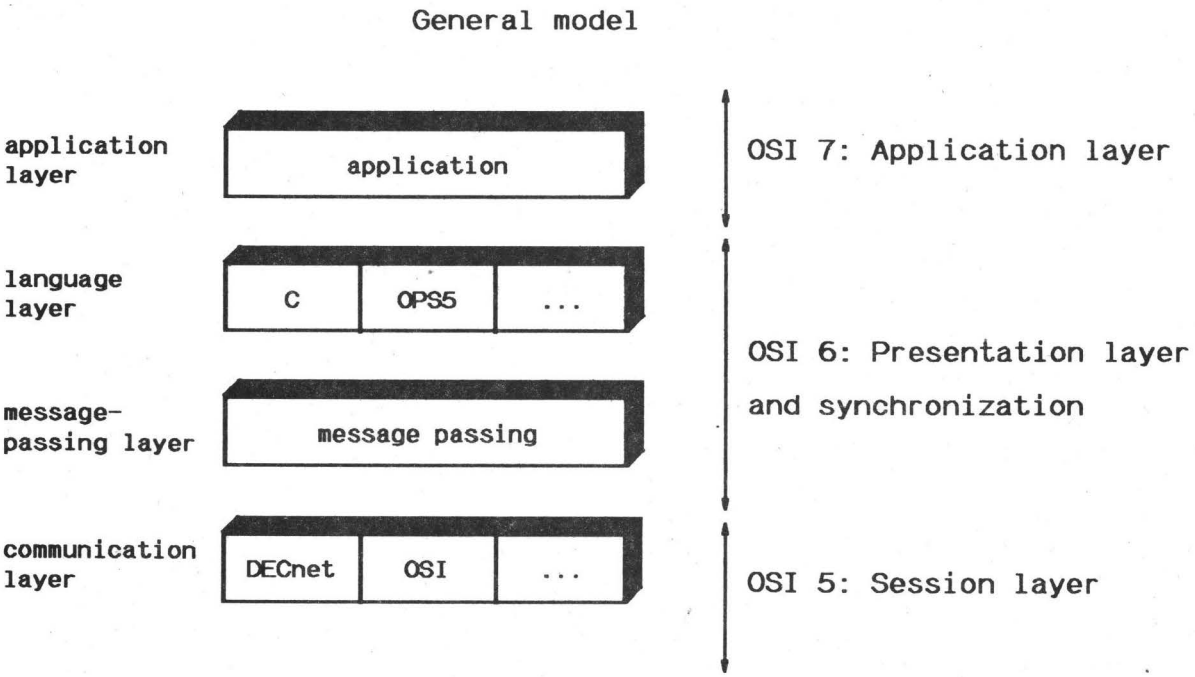

Fig. 2: Layers of the communication model

request to the remote object. As a response, the remote object sends back the actual state and will send the new state in the future whenever the state changes. As you will see later on, it is not the objects responsibility to do such. It is rather the *object system* handling all of the communication and update for all of the objects residing in the application.

The message protocol is extendable. We use *tags* to identify what kind of information is transported (e.g. TAG1 for requests, TAG2 for state changes). Introducing new tags, it is possible to extend the functionality of the communication system. Currently, we are thinking about transporting whole objects including their methods to remote applications, which is possible in a symbolic environment. We could use new tags to implement this without changing the message protocol or the object interface.

The *object system* on top of our architecture (which we use as the user interface) is independent of the layers beyond. An application in the distributed environment could use its own object description, hence making it possible to choose an implementation language which is well suited to the specific problem the application has to solve.

Our main goal was to produce high-level interaction objects which could be used anywhere in the user interface (also as multiple instances), transporting their whole functionality with them. In case of critical situations in a technical plant, it is necessary to reduce the interaction at a minimum level, hence giving the operators the time to focus on their problem and not on searching the interface for the point where they can do what they have to do. From our point of view, this is the main disadvantage of the technology currently used. As mentioned earlier, it was also a goal to make the dialogue adaptable to a given process context.

Another goal was to support the designer of the user interface. There are two main areas where we support the designer:

- integration of remote information into the user interface and
- creating and maintaining object databases used as user interface applications

We will introduce our object description in the sequel.

3.2 Object description

Fig. 3 shows the description of a single object as we store it in our database. An object is described by

- name, priority and object class,
- a set of attributes defining its properties and
- a set of methods defining its behaviour

We store object classes (generalizations of a set of objects) in *object class hierarchies* as it is usually done in object oriented systems. Inheritance of attributes and methods and default

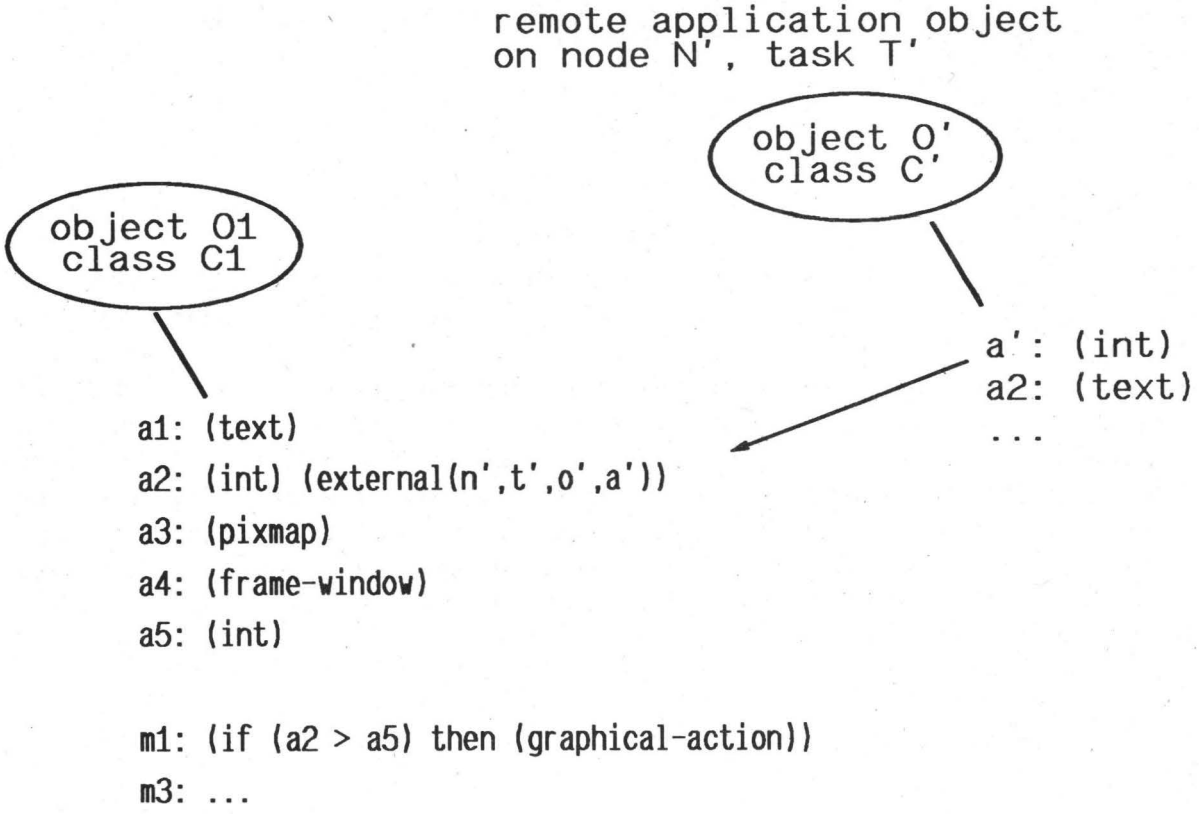

Fig. 3: Object model

mechanisms are also provided. A specific application consists of one or multiple *object hierarchies* and *dialogue hierarchies.*

An object attribute (or a class attribute) can be of *technical nature* or of *graphical nature.* Technical attributes are numbers, strings, state attributes and text attributes. Graphical attributes can be all kinds of graphical representations, as basic graphical primitives, but mainly high-level interaction objects themselves, such as icons, pixmaps, menus etc. An object may have a whole dialogue hierarchie (including frame windows) as an attribute, describing the way the user interacts with the object.

As *methods* to describe the behaviour of an object or object class we use *rules*. Our experiences showed, that rules are well suited to the problem of flexible and adaptable user interfaces. Rules directly implement the state transitions in the technical process as well as in the user interface.

Combining a number of technical attributes, graphical attributes and rules operating on the attributes, it is possible to design *high-level interactive objects* which may be used in any window on the screen. We can integrate help and information windows in the object class or object description to provide any useful information where it is needed, namely in the

interaction with the object itself. As long as the rules operate only on the object itself (which is often the case), it is possible to store the whole interaction in the object class.

Furtheron, we may add attributes and rules to object instances. This is done to describe specific *relationships between objects*, which may not be generalized in the object class. These relationships describe the relationships between real world objects, e.g. "if tank **B10** is empty, you are not allowed to switch on pump **Pu1** or pump **Pu2**". We use such relationships to manipulate and adapt the dialogue at runtime.

Another key feature of the architecture is, that it is based on a *distributed object model.* Attributes may be *external attributes*. An object **O** may have an attribute **A**, which resides in an attribute **A*** of object **O*** in another application across the network, which is the origin of the information. There is a direkt map from the local triple **[O,A,** value **V]** to the origin of the information [node **N***, task **T***, object **O***, attribute **A***, value **V***]. This is one of the main advantages we see in our architectural approach: there is no programming going on to embed a remote information. We just *define* the attribute as to be external and the object system automatically establishes the link to the remote object as soon as the local object is loaded into the runtime system.

Another advantage of using this map is, that we must not define the origin of an information in the object class. An attribute **state** of pump **Pu1** may reside in the local application whereas the state of pump **Pu2** resides in a remote application. They both may be taken as instances from object class **pumps**, as the map is defined for the attribute **state** after creating the object instance.

An object may have external attributes residing anywhere in the network. They may reside in different applications on different nodes, e.g. a measurement value **[O1, A1, V1]** coming from **[N1*, T1*, O1*, A1*, V1*]** and a diagnosis information **[O1, A2, V2**] coming from a diagnosis system as **[N2*, T2*, O2*, A2*, V2*]**. The map may be defined for every single remote information. Clearly we support the designer in creating the maps in the design system.

4. Implementation

We have implemented the presented architecture under VAX/VMS using DECnet and C for the communication model. We also implemented language layers for C, OPS5 and PROLOG. As we wanted to combine object orientation with rules as methods, we finally decided to use OPS5 as the base of an object system, which we then implemented ourselves. We decided to use OPS5, because OPS5 is event driven from its nature, which applies directly to the user interface and the process model. On the other hand, the OPS5 pattern matcher and the rule selection strategy are well suited for our purposes.

Our experiences using OPS5 were rather positive, because the code is very compact and represents exactly the state transitions of the real world. It is also rather easy to debug. For example, the whole runtime kernel of the distributed communication system (including a prioritizing rule scheduler) , which we tried to implement "water proof", consists of some 20 rules.

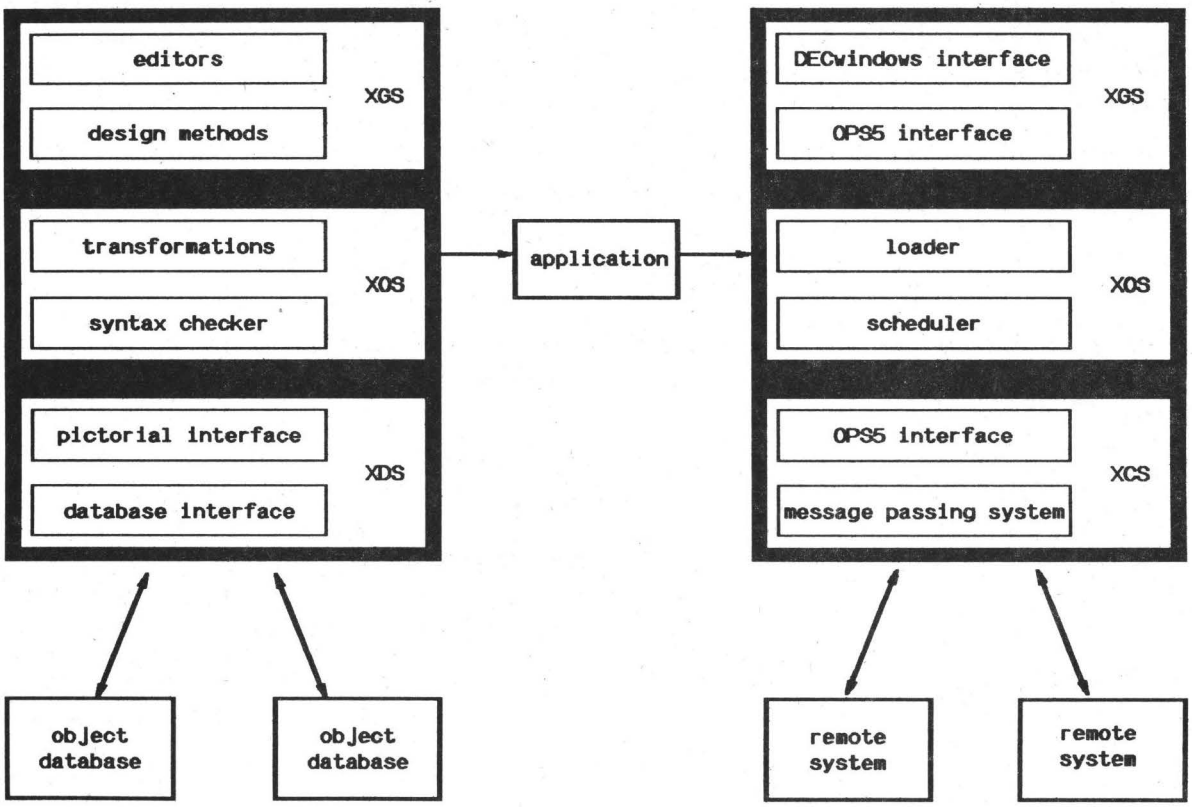

Fig. 4: Implementation

We have split the object system into two parts: a *design system* to create and maintain applications and a *runtime system* to load and execute applications (fig.4). The applications we create using the design system may be graphical and non-graphical applications. E.g. we use the same design environment to design symbolical process simulations running in parallel to the user interface.

The runtime system includes a graphics system based on Xtoolkit/Motif to display and control the dialogue.

5. Applications

We have built two very complex user interfaces in a distibuted environment using our current implementation.

The first user interface was one for the garbage burning part of the garbage burning plant TAMARA. We have reported about this in [1,2,22]. The experiences using rules to create the adaptive behaviour of the user interface were very positive.

The second implementation was for the air quality measurement network of the Austrian federal state of Kärnten [23]. The network has been built by the Austrian research centre Seibersdorf and the work has been done in the context of our cooperations in the field of environmental computer science [26,27]. Most of the current work is concentrated on this field, as presented in recent publications [24,25].

6. Conclusions

The main advantages of object orientation is, that the design environment grows in functionality with every new object class. Using OPS5-rules as methods benefits in a very compact code, because within OPS5 you can use set operations in the condition elements. Thus it is possible to apply one single rule to a large number of objects or even to a whole application. As an example, it is possible to manage the whole alarm display of all objects implementing four rules. Beyond that, the code represents the transitions occuring in the model, which means, that we have a direct map of the way we would describe the reality onto its software representation.

The benefit of the communication model is that it is very easy in principle and that it can integrate programs written in other programming languages. One of the difficulties implementing the model was its asynchronous nature. Because of that, we must provide synchronisation mechanisms for each programming language or tool we want to embed, which is not always easy.

References

[1] Dittrich G., Kerpe R., TAMARA - Versuchsanlage zur schadstoffarmen Müllverbrennung, GVC-Kongress, Baden-Baden, December 1989

[2] Denzer R., User Interface Management in Distributed Systems, in: Denzer R., Hagen H., Kutsche K.H., Visualisierung von Umweltdaten, GI-workshop, Rostock, November 1990, Informatik-Fachbericht 274, pg. 83-97, Springer

[3] Sachs P., ESCORT - an Expert System for Complex Operations in Real Time, Alvey workshop on Deep Knowledge, IEE, London, 1985

[4] Alty J.L., Mullin J., Dialogue Specification in the GRADIENT Dialogue System,in: Sutcliffe A., Macaulay L., People and Computers V, Proc. of the 5. Conf. of the BCS HCI Specialist Group, Nottingham, Cambridge University Press, 1989

[5] Elzer P. et. al., Expertensysteme und hochauflösende Graphik zur Unterstützung des Bedienpersonals in der Prozeßleittechnik, Prozeßrechensysteme 1988, Stuttgart, Informatik-Fachbericht 167, Springer, 1988

[6] Tzafestas G., Ed., Knowledge-Based System Diagnosis, Supervision and Control, Plenum Press, New York, 1989

[7] Khanna R., Moore, R.L., Expert Systems Involving Dynamic Data for Decisions, Int. Expert Systems Conference, Oxford, 1986

[8] IEE Colloquium on Expert Systems in Process Control, London, 1988

[9] IEE Colloquium on The USe of Expert Systems in Control Engineering", London, IEE Digest No. 1987/27, 1987

[10] Foley J.D., Next Generation User Interface Development Tools, in: Hansmann, Hopgood, Strasser Eds., Eurographics '89, September 1989, Hamburg, North-Holland

[11] Roudaud B. et. al., SCENARIOO: A new generation UIMS, in: Diaper, Gilmore, Cockton, Shackel Eds., INTERACT '90, Cambridge, August 1990, North-Holland

[12] Hill R.D., Herrmann M., The Structure of Tube - A Tool for Implementing Advanced User Interfaces, in: Hansmann, Hopgood, Strasser Eds., Eurographics '89, September 1989, Hamburg, North-Holland

[13] Hübner W. et. al., Designing a System to Provide Graphical User Interfaces: The THESEUS Approach, in: Eurographics '87, North-Holland

[14] ten Hagen P.J.W, Schouten H.J., Parallel graphical output from Dialogue Cells, in: Eurographics '87, North-Holland

[15] Pfaff G.E. Ed., User Interface Management Systems, Springer Verlag, 1985

[16] Coutaz J., UIMS: Promises, Failures and Trends, in: Sutcliffe A., Macaulay L., People and Computers V, Proc. of the 5. Conf. of the BCS HCI Specialist Group, Nottingham, Cambridge University Press, 1989

[17] Edmonds E., An Experiment In Interactive Architectures, in: Diaper, Gilmore, Cockton, Shackel Eds., INTERACT '90, Cambridge, August 1990, North-Holland

[18] Koivunen, M.-R., WSE: An Environment for Exploring Window Strategies, in: Hansmann, Hopgood, Strasser Eds., Eurographics '89, September 1989, Hamburg, North-Holland

[19] Breen D.E., Kühn V., Message-Based Object Oriented Interaction Modeling, in: Hansmann, Hopgood, Strasser Eds., Eurographics '89, September 1989, Hamburg, North-Holland

[20] Hübner W., Gomes M.R., Two Object-Oriented Models to Design Graphical User Interfaces, in: Hansmann, Hopgood, Strasser Eds., Eurographics '89, September 1989, Hamburg, North-Holland

[21] Burgstaller J. et. al., On The Software Structure Of User Interface Management Systems, in: Hansmann, Hopgood, Strasser Eds., Eurographics '89, September 1989, Hamburg, North-Holland

[22] Denzer R., Hagen H., Kira G., Koob F., Using Process Knowledge for Adaptive User Interfaces, in: Rzevski G., Adey R. A. (eds.), Applications of Artificial Intelligence in Engineering VI (AIENG VI), Oxford, 1991, Proceedings, pg. 583-596, Computational Mechanics Publication, Elsevier Applied Science, 1991

[23] Denzer R., Schimak G., Visualization of an Air Quality Measurement Network, 6. Symposium on Computer Science for Environmental Protection, Munich, December 1991, in press, Springer-Verlag, 1991

[24] Denzer R., Object-Oriented Modeling of Iconic Dialogues for Environmental Software Systems, 2. Workshop on Visualization of Environmental Data, Dagstuhl, Germany, November 1991, in press, Springer-Verlag, 1991

[25] Denzer R., Kira G., Koob F., Interactive Visualization of Multiple Environmental Measurement Networks, 2. Workshop on Visualization of Environmental Data, Dagstuhl, Germany, November 1991, in press, Springer-Verlag, 1991

[26] Denzer R., Visualization Problems in Environmental Protection, Dagstuhl-Seminary on Scientific Visualization, August 1991, in press

[27] Denzer R., Güttler R., Grützner R., Visualization of Environmental Data - Topics and results of the 1991 Dagstuhl Workshop, 6. Symposium on Computer Science for Environmental Protection, Munich, December 1991, in press, Springer-Verlag, 1991