ObjView: A Task-Oriented, Graphics-Based Tool for Object Visualization and Arrangement

G. FRIEDRICH, W. HOELLINGER, Ch. STARY, & M. STUMPTNER

Technical University of Vienna
Institut für Angewandte Informatik und Systemanalyse
Abteilung für Verteilte Datenbanken und Expertensysteme
Paniglgasse 16 - 181/2
1040 Vienna
Austria

Abstract

The ObjView tool provides interactive, graphics-based access to objects, relationships and methods, supporting the varied activities of designing components for automatic routing systems. The visualization uses a object-oriented representation of complex technical systems including context specific information (attributes, methods, etc.) which encourages users to manipulate and view parts of the system in a modular fashion. It supports navigation along physical and functional object relationships, as well as direct selection of objects from a graphical representation of the internal system model. A preliminary version of the application has been implemented to demonstrate the functional and organizational features of the user interface.

1. WYSIWYW (What You See is What You Work)

The task of data modelling is twofold: first, a real-world system is formally described. Second, the resulting model must be applicable to tasks within a problem domain. For modeling physical systems, object-oriented data models turned out to be particularly adequate (Madsen et al., 1988). After identifying and representing class structures the primary objective in object manipulation is to determine inter-objective behaviour.

The visibiiity of objects influences the manipulation of system components to a great extent. Over the last years a variety of representation as well as access mechanisms have been developed to provide efficient storage of objects, relationships and methods. Considerably less progress has been made in the area of providing users with convenient manipulation facilities concerning specific tasks of object handling (e.g., Pintado et al., 1988). Most of the advances that have been made in this area concern object-oriented development systems. For example, object-oriented modeling of electronic books led to the visualization of classes and their relationships (Pasquier-Boltuck et al., 1988). However, it is often neglected to provide manipulation and structuring operations comparable to activities in the real world. For example, Pasquier-Boltuck et al. represent a "book" by an ordinary scrollable text window. Thus, the possibility of random access to different sections, that is, opening the book in the middle, is absent, although present in any paper publication without change of representation.

In this paper we describe the ObjView tool as a step towards real-world activity support. In particular, ObjView is a design tool for automatic routing systems. It supports activities concerning the creation and manipulation of electronic components which are

represented by an object-oriented approach. Furthermore, the tool introduces a number of novel concepts which can be incorporated into extant and future knowledge representation access packages.

One of the most important features of any graphics-based object management system is the visual representation of object instances and their relationships. Some criteria for visualization can be articulated:

(1) The simultaneous and coherent display of all types of instance-relevant information is inevitable for an activity-oriented work environment (e.g., routing tasks).

(2) The clustering of information allows a modular representation and permits simultaneous display of several levels of abstraction, e.g., attributes and relations which are necessary for supporting problem solving activities.

(3) The functionality of the tool must include facilities to reorganize specific instantiations and object constellations (e.g., connections between functional units).

(4) If facilities for tailoring the interface to individual users' preferences are provided, it should be possible to reset the display to be visually familiar.

The integration of organizational context (specification of user tasks) and further principles of cognitive ergonomics (combination of textual and graphical representation, using the 'Desktop' metaphor) provide users with a feeling of continuity and stability over the course of lengthy system interactions.

2. Object Representation

The ARTEX knowledge-based system (Retti et al., 1987) has been developed for audio routing systems, i.e., switching systems for high-quality audio signals which are typically used in broadcast studios. The principal task of a routing system is to flexibly connect input channels with output channels. This is done by means of a switch matrix having up to 1000 inputs as well as outputs. This matrix is functionally divided into submatrices and is distributed over several hardware modules which are mounted in racks.

The hardware is composed of standard plug-in modules. There are rather few types of modules, but the modules can be interconnected in many different ways for a specific installation. The current analog routing system has about 20 different types of audio and control components. A typical system comprises 200 to 1000 replaceable units.

ARTEX has to achieve several goals during the life-cycle of the audio routing system. Those goals are the configuration of individual routing systems, quality assurance of newly assembled systems and fault detection during operation of the routing system (Fleischanderl et al., 1989). The class of routing systems that ARTEX is dealing with allows a large number of different configurations. All these variants have to be coped with by the configuration knowledge.

The generic structure of the routing system is defined inside by means of class descriptions (types of parts) and constraints about how to assemble the system. A single routing system consists of several racks (we will ignore connections between racks in our presentation). A rack contains seven standardized slots (so-called "Double-Euros", or "DE"s), into each of which will fit a motherboard unit ("MB"). A typical rack contains one power-supply motherboard and several audio-motherboards. Each motherboard, in turn, contains an array of slots for plugging in the actual audio components. The type of motherboard determines the corresponding slot types, and therefore the types of basic modules (audio components) that can be plugged in at each slot. The most important kinds of basic modules are input amplifiers, output amplifiers and switching matrixes. The connections between different basic modules on one motherboard are integrated into

the slots. On the other hand, connections between different motherboards use cables and plugs which have to be explicitly plugged in.

As a basis for user manipulation of system configurations and for the representation of system states during the fault detection process, an object oriented model of the system was developed and implemented in C++. The electronic components of the routing system are described by a hierarchy of about 40 classes. In the rest of the paper, when we use the term "model", we refer to the collection of instances of these classes making up the current internal representation, as opposed to the physical system itself.

When objects are presented to the user, this is done in a slightly modified way from their C++ structure. An object is displayed as the combination of four kinds of information about it: its class, its attributes, relations, and tasks. *Attributes* correspond to instance variables (or members, in C++) which contain only primitive data types. Almost all object references within the model are part of two-way connections. As was shown by Rumbaugh, 1987, such constraints are best expressed by explicitly specifying *relations* between objects. Relations are not supported in C++, and are therefore implemented by way of object references; this, however, is not visible to the user. *Tasks* are the operations that a user can perform on objects or relations in the model, and are implemented simply as C++ member functions.

3. Task-Oriented Support

The following section describes the evolution of our system from a simple, command-driven, "one object at a time" manipulation system to a tool that supports navigation in the object space by means of graphics-based interaction utilizing a set of high-level commands, such as consistent multiple instantiations of interconnected entities. Furthermore, the visualization and manipulation techniques for task-specific support within the encompassing ARTEX system are discussed.

3.1 Taking the Problem Domain Into Account

The object-oriented modeling approach, though powerful for representing complex systems, does not provide task-oriented support when handling real-world entities. The reason for this is, that technical systems may consist of a large number of components relevant to the model description. Adding to the problem of sheer weight of numbers is that objects may be participating in a number of relationships, varying from intuitive concepts like a part-subpart hierarchy to more complex interdependencies, such as the position of a switch determining the existence of a link between two other entities. Therefore, to represent these entities and relationships a network consisting of large number of nodes and links is required.

As a result, when using such complex information lattices, the following questions arise: How can relevant information about objects be regained from the network? How can objects be manipulated within a task-oriented environment? There are different approaches known:

First, if the user's level of information (concerning the structure of the object description and the represented knowledge) is sufficient to retrieve and manipulate objects and relationships, a command-oriented manipulation language will be the proper form of interaction. This implies knowledge about

(a) what the relationships between objects look like, i.e. knowledge about the structure of the lattice;

(b) the primitive attributes and methods concerning specific objects, i.e. knowledge about the structure of the objects;

(c) how to formulate sequences of activities for problem solving.

Second, the user is permanently provided with methods and attributes concerning objects. Furthermore, information about relationships between specific objects is offered to the user. Thus, he/she is enabled to process a number of objects along explicitly represented relationships in the object network, starting from a certain entry point.

The third approach best meets the requirements of task-oriented computer support. It presents the structure of the network to the user in terms of the real system, independent of its representation within the application. This kind of approach assumes that the objects in the problem domain have a physical counterpart in the real world. Then, if the graphical representations of objects closely resembles their physical appearance and layout, class membership of an object can be directly inferred from the graphical representation without having to refer to an explicit representation of the class hierarchy. For example, because all of the important electronic components of a routing system are visible from the front, a two-dimensional representation on the screen is sufficient for direct manipulation.

Using a representation-independent view, a shift takes place from merely implementation-oriented considerations to an integration of problem domain knowledge, i.e. the presentation of a work space where the tasks that are to be performed by the users actually can be represented by analogy to their real-world counterparts. Some implications for the user interface design have to be taken into account:

(a) Objects need a graphical representation when displayed to the user;

(b) The graphical disposition of objects has to be equivalent to their spatial position in the physical work space;

(c) The tasks to be performed on objects in the model have to represent tasks of the identified work space.

In addition, relationships and attributes may be displayed explicitly.

3.2 Object Visualization and Manipulation

System Creation. Based on the class declarations and additionally defined constraints, a model of the physical system is created by instantiating the relevant components. Starting from an empty rack, motherboard instances are created and plugged into previously empty DE-slots. In this way the motherboard slots are successively filled in. Each such action in the model corresponds exactly to a technician inserting a motherboard into a rack in the physical system. Figure 1 shows one such action in progress (for figures see appendix). The left part of the display shows two racks side by side. In rack 1 all slots already contain a motherboard, each with a number of modules added. The power supply motherboard fills the bottom slot. In rack 2, the top two slots contain audio motherboards, but no modules have been plugged into the motherboards. The other slots are empty.

The mouse cursor was moved over an empty slot in rack 2 (the fourth one from the top). The user selected the task "Insert a MB" and is now prompted to choose the appropriate kind of motherboard. After the choice was made, an object of the appropriate motherboard subclass is generated and inserted in the rack. All subparts of the motherboard are created automatically. This includes, for example, the 19 slots of an audio input motherboard (or AMB_AIM), as in the second slot in the left rack. The units to fill these slots are created individually afterwards.

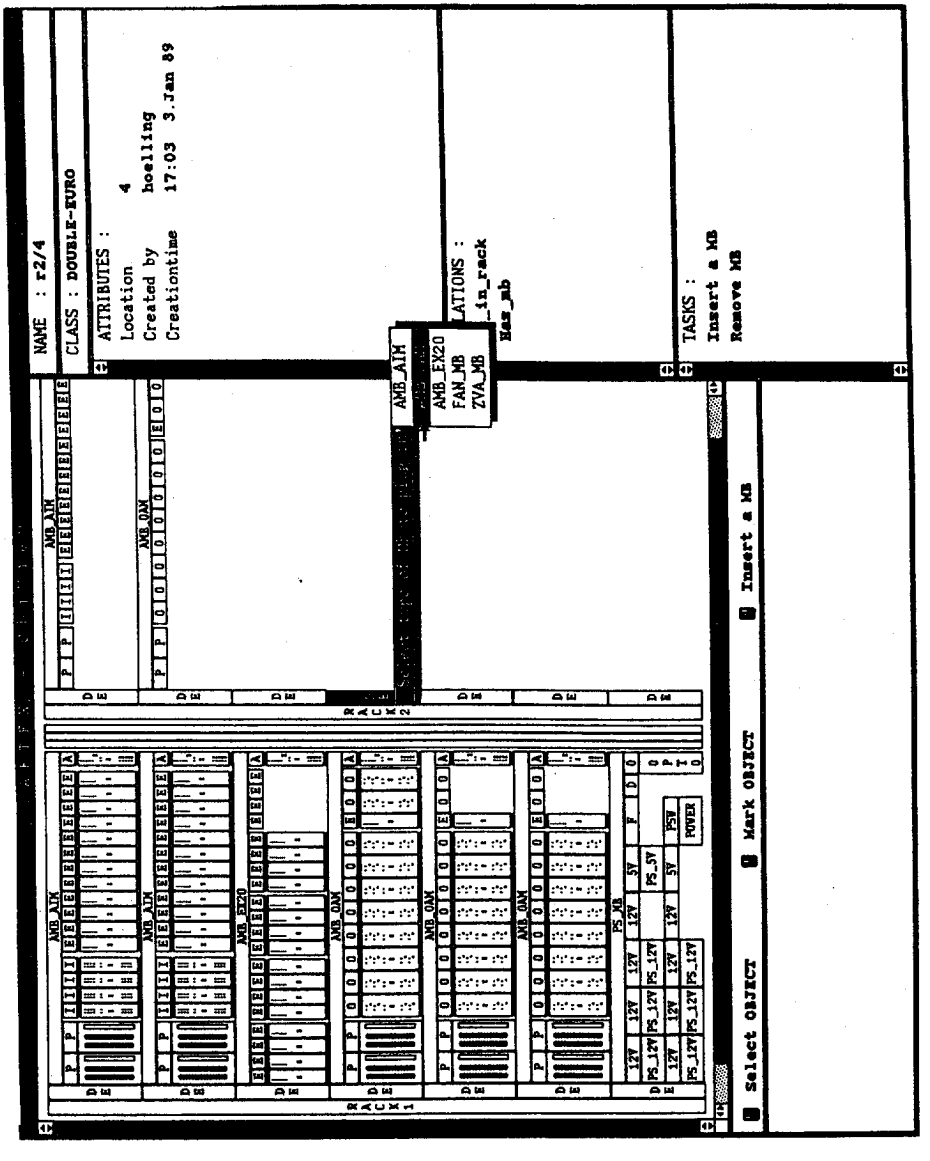
NAME : r2/4
CLASS : DOUBLE-EURO
ATTRIBUTES :
Location 4
Created by hoelling
Creationtime 17:03 3.Jan 89
AMB_AIM
AMB_EX20
FAN_MB
ZVA_MB
LATIONS :
_in_rack
Has_mb
TASKS :
Insert a MB
Remove MB
Select OBJECT
Mark OBJECT
Insert a MB

Figure 1

Working with a Specific System. After the physical configuration has been chosen, the system state can be changed by setting switches in certain modules. For example, the basic action during operation of the system is establishing a connection between a pair of input and output channels.

Figure 2 shows an aspect of movement through the system. An input amplifier channel (IAC) module has been selected. The user can now choose one of its associated output amplifier channels (OAC) from a menu and thereby make this OAC the new selected object. Note that the chosen relationship (*switched_OACs*) between IA and OA is a purely functional one. The two modules are not physically adjacent, but have been connected by changing the state of a switching matrix (EAM) to create a path between them. There exists a natural way to display functional connections graphically as well. In Figure 3, exactly the path we followed in figure 2 is shown by highlighting its end points and the switching matrix that connects them. The follow-up version of ObjView will allow to visualize all parts of the system (e.g., channels) which leads to higher functionality (e.g., zooming).

While the tasks performed during the creation phase of a system still have to be performed physically (somebody HAS to fasten the screws), the setting of switches is performed on the screen. By sending an appropriate command to the control software of the routing system, the change is immediately effected in the physical system. This can be used for setting a default switching configuration as well as during actual operations.

Default Task. Since certain kinds of tasks are singularly appropriate to some situations, the possibility has been provided to select such a task according to context without actually having to choose an option from the task list in the lower right window. This task is called the *default task.*

The current default task changes dynamically; it depends both on the state of the model and the mouse position over the graphical representation. For example, when moving the mouse over an empty DE-slot in a rack, the only useful action to perform at that spot would be to create a new motherboard, whereas, if a motherboard has already been plugged into the slot, the default task is to remove the motherboard.

To enable easier application, the current default task is displayed below the graphics window (see also section 4).

4. Impacts for Consistent Interface Design

The tool utilizes a high resolution bit-mapped graphics display with a three button mouse to create a highly interactive environment. The user interaction with ObjView takes place primarily through windows and mouse actions. An ergonomic window configuration and mouse representation on the screen are provided but the user may freely resize and position windows to most efficiently utilize screen space. In this section we analyze the Obj-View interface by discussing the consistent integration of organizational aspects concerning the screen as well as the basic elements of visualization and manipulation.

4.1 Screen Organization

Although visual interaction is a rather general concept, cognitive analyses have shown, that by mapping tasks to a visual interface, problem solving can be supported adequately (e.g., Tauber, 1987). By visualizing task-relevant objects, relationships and attributes, a manipulative (versus descriptive) work environment can be designed. The

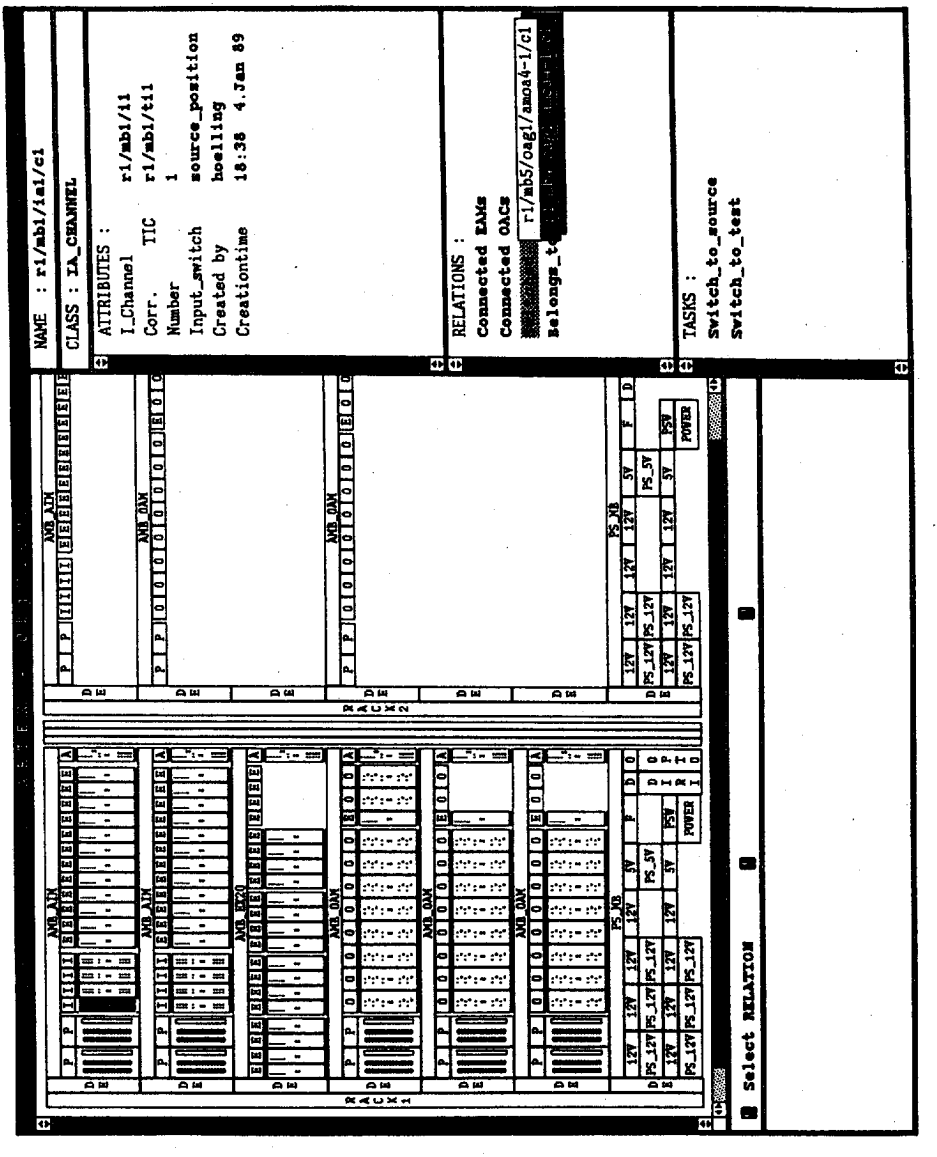
NAME : r1/mb1/ia1/c1
CLASS : IA_CHANNEL
ATTRIBUTES :
I_Channel r1/mb1/i1
Corr. TIC r1/mb1/ti1
Number 1
Input_switch source_position
Created by hoelling
Creationtime 18:38 4.Jan 89
RELATIONS :
Connected EAMs
Connected OACs
r1/mb5/oag1/amoa4-1/c1
TASKS :
Switch_to_source
Switch_to_test
Select RELATION

Figure 2

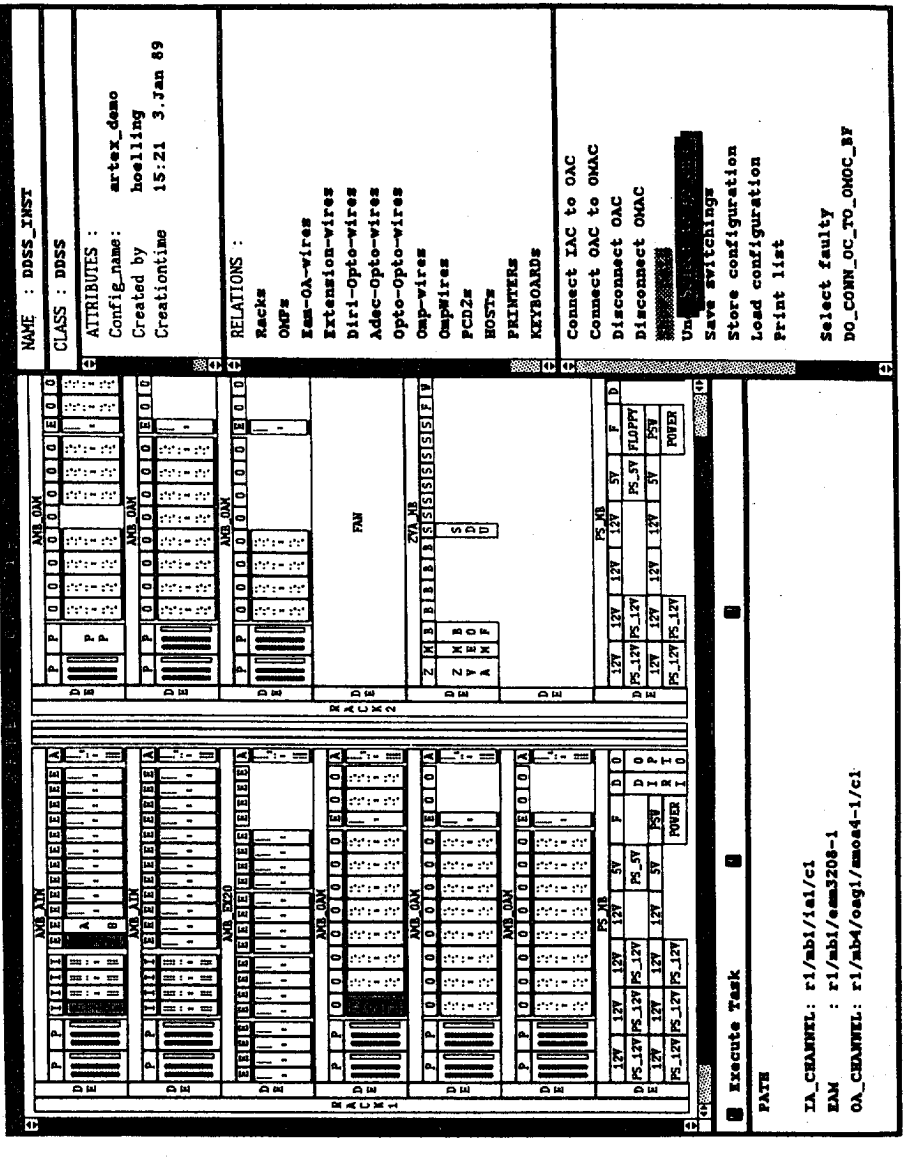
NAME : DDSS_INST
CLASS : DDSS
ATTRIBUTES :
Config_name: artax_demo
Created by hoelling
Creationtime 15:21 3.Jan 89
RELATIONS :
Racks
OMPs
Ram-OA-wires
Extension-wires
Dir1-Opto-wires
Adec-Opto-wires
Opto-Opto-wires
Omp-wires
OmpWires
PCD2s
HOSTs
PRINTERs
KEYBOARDs
Connect IAC to OAC
Connect OAC to OMAC
Disconnect OAC
Disconnect OMAC
Save switchings
Store configuration
Load configuration
Print list
Select faulty
DO_CONN_OC_TO_OMOC_BF
Execute Task
PATH
IA_CHANNEL: r1/mb1/ia1/c1
EAM : r1/mb1/eam3206-1
OA_CHANNEL: r1/mb4/oag1/emoa4-1/c1

Figure 3

work environment provided by ObjView is supported by an adequate screen organization. The following topics of discussion led to the proposed kind of screen organization:

(a) A familiar user's conceptual model must be based on the "Desktop"-metaphor and WYSISWYW.

(b) Consistent interaction do not exclude different screen areas and user tailorability.

The screen in ObjView is divided into some areas which allow only to describe (by text or graphics) and some areas enabling the description as well as the manipulation of objects, relationships and activities. The main area is a graphics-based window containing objects of the work space. It resembles the creation of racks, i.e. the environment to perform routing tasks ("Desktop"). ObjView models the real world accurately enough. Its similarity with the routing environment preserves a familiar way of working and the existing concepts and knowledge of users.

Everything which is task-relevant is visible on the screen. The display becomes the work environment by supporting seeing and pointing for user-system interaction. By making the user's activities on objects visible, "WYSIWYG" is extended to "WYSIWYW". The organization of the screen supports the priorities of elements within the discourse of routing. Attributes, relations and tasks are visualized by text-based windows along the right side of the manipulation objects. Messages according to the execution of tasks are displayed at the bottom of the screen. Finally, the interpretation concerning the mouse buttons is displayed in a single line below the graphics-based window (icons+text).

Consistency of representation and interaction asserts that mechanisms are used in the same way whereever they occur. Although the idea of consistent interfaces is well defined, there are few systems where its implementation has been successful (e.g., Smith et al., 1982). In ObjView, the representation of objects, relationships and tasks is consistent in graphics- as well as text-based window types. User requests are kept consistent by processing two steps: First, the user identifies an object of interest by positioning the cursor over that object and pressing a mouse button. Second, ObjView offers menu options according to the identified object of interest. The user selects a desired option with the mouse and the request is processed when the mouse button has been left.

The interpretation of the mouse buttons remains constant (left button: select and execute; intermediate button: mark, when an activity involves more than one object; right button: execute default task). Even, if the screen organization has been individualized to a single graphics-based window with menu-bars at the top of the screen, the mouse buttons have the same interpretation and user requests are processed like described above.

As our practical experiences show, such minimal configuration principles concerning the screen organization have to be taken into consideration to be equivalent to the model developed in the mind of an individual performing the tasks described in section 3.

4.2 Interface Elements

We used the following techniques to design the ObjView interface: graphics, menus, icons, windows, mouse, semantic context refreshing, and individualization.

Although graphical interaction obviously supports dialogue transparency (Kaster et al., 1985) the benefits of graphics are limited when these are not employed to directly assist in solving certain tasks (Ben Basat et al., 1986). Thus, effective problem solving support depends on the representation of task-specific information.

Task-specific information is provided by different types of windows:

(a) a graphics-based window for model representation, text-based windows for describing objects and tasks;

(b) active versus passive windows.

Active windows contain executable activities or elements of manipulation which can be executed and changed by mouse clicks whereas passive ones provide users with static properties of objects and messages according to the execution of tasks.

Besides the known advances in the field of window-based interfaces (e.g., Card et al., 1985) their use for representing the display of the meaning of clicking mouse button is quite new. Windows are also helpful for semantic context refreshing, i.e. if the object of interaction is changed, the display of default tasks, relationships, etc. is changed too.

To enable individual interaction, the size of the windows can be changed. It is also possible to extend the graphics-based window onto the whole screen. In this case, menus for displaying attributes, relationships and the execution of tasks are presented as bars at the top of the screen. This environment facilitates complex task handling (e.g., using more than two racks). The representation of the mouse buttons and messages concerning executed tasks remain at the bottom of the screen. This extension of the orginial work environment can be reset at any time to an overall work-space display.

5. Concluding Remarks

Although theoretical as well as empirical studies in pictorial knowledge representation and problem solving are rare it has turned out that a lot of cognitive activities presuppose pictorial representation: the selection and generation of hypotheses, check for consistency, coreference and coherence, analogical reasoning, etc. Based on the theoretical foundation of pictures as exemplified standard models Janlert, 1988 found out, that pictorial representation is a less direct access to information than words (in contrast to the common belief in AI) because there is no intrinsic relation between a picture and the depicted object. Although a picture (as a part of a pictorial symbol system) represents just a verbal description and implies mapping between representation schemes pictorial knowledge representation facilitates the direct access to work units within a problem area. Furthermore, there is firm evidence that both the "deep", long-term memory representation and the "surface", short-term memory representation are oriented towards objects although computer graphics do not provide complex functions like flexible object conversion, etc. (Weber et al., 1986).

Similar to scheme-manager in the area of data modeling (e.g., King, 1984) we made a step further to activities in handling instantiations of classes within an object-oriented representation scheme. After visualizing the class hierarchy and inter-object relationships (e.g., Ishii et al., 1986) the management of activities according to a certain work environment are supported.

The visualization and manipulation tool ObjView is being developed on a Sun 3/50, and is built upon C++. The focus of our implementation is on demonstrating the features of an task-oriented user interface; efficiency is not an emphasis of the implementation.

In our view, ObjView demonstrate a number of novel concepts in the area of interfaces to knowledge bases. It can serve as the skeleton for the development and demonstration of a wide variety of more advanced concepts for such kind of systems. Interesting research questions are the extension of the display to the representation of more than one physical view of the model (e.g., back view, zooming an amplifier) as well

as logical views (e.g., power supply view). Due to the introduction of new visual features end user tasks will become more complex.

Acknowledgements

We would like to thank Johannes Retti and Herwig Schreiner for their help in developing the preliminary version of ObjView in conjunction with the ARTEX-System (financially supported by Siemens Austria).

Bibliography

Ben Basat, I.; Dexter, A.S.; Todd, P.: An Experimental Program Investigating Color-Enhanced and Graphical Information Representation: An Integration of the Findings, in: Communications of the ACM, Vol. 29, No. 11, pp. 1094-1105, November 1986.

Card, S.K.; Pavel, M.; Farrell, J.E.: Window-Based Computer Dialogues, in: Human-Computer Interaction-INTERACT '84, ed.: Shackel, B., IFIP, pp. 239-243, Elsevier (North Holland), 1985.

Fleischanderl, G.; Friedrich, G.; Retti, J.: ARTEX - Configuration-Driven Diagnosis for Routing Systems, in: Proc. 5. Österr. Artificial Intelligence-Tagung, Innsbruck, 1989 (to appear).

Ishii, H.; Kubota, K.: Office Procedure Knowledge Base for Organizational Office Work Support, in: Proceedings of the Working Conference 'Office Information Systems-The Design Process', IFIP WG 8.4, Linz, pp. 40-57, August 1988.

Janlert, L.E.: Pictorial Knowledge Representation, in: Proceedings ECAI '88, München, pp. 149-151, August 1988.

Kaster, J.; Widdel, H.: Graphical Support for Dialogue Transparency, in: Human-Computer Interaction-INTERACT '84, ed.: Shackel, B., IFIP, pp. 329-333, Elsevier (North Holland), 1985.

King, R.: Sembase: A semantic DBMS, in: Proceedings of the First Intl. Workshop on Expert Database Systems, pp. 151-171, October 1984.

Madsen, O.L; Moller-Pedersen, B.: What Object-Oriented Proggramming May Be - And What It Does Not Have To Be, in: Proceedings ECOOP '88, Oslo, pp. 1-20, August 1988.

Pasquier-Boltuck, J.; Grossmann, E.; Collaud, G.: Prototyping an Interactive Electronic Book System Using an Object-Oriented Approach, in: Proceedings ECOOP '88, Oslo, pp. 177-190,August 1988.

Pintado X.; Tsichritzis, D.: An Affinity Browser, in: Active Object Environments, Technical Report, Centre Universitaire d'Informatique, Universite de Geneve, 1988, pp.51-60.

Retti, J.; Friedrich, G.; Nejdl, W.: ARTEX - Wissensbasierte Fehlererkennung und Fehlerbehebung in einem Vermittlungssystem (Knowledge-based Error Recognition and Correction in a Routing System), in: Proc. 2. Intl. GI-Kongreß, München 1987, pp.217 - 223.

Rumbaugh, J.: Relations as Semantic Constructs in an Object-Oriented Language, in: Proceedings OOPSLA '87, ACM, pp. 466-481, October 1987.

Smith, D.C.; Harslem, E.: Designing the Star User Interface, in: Byte, pp. 242-282, April 1982.

Tauber, M.J.: On Visual Interfaces and their Conceptual Analysis, in: Visual Aids in Programming, eds: Gorny, P.; Tauber, M.J., Springer, 1987.

Weber, R.J.; Kosslyn, St.M.: Computer Graphics and Mental Imagery, in: Visual Languages, ed.: Chang, K., Plenum Publishing, 1986.