The ZOO Metasystem: A Direct-Manipulation Interface to Object-Oriented Knowledge Bases

Dr. Wolf-Fritz Riekert
Siemens AG, K D ST SP 311, AI Programming Systems
Otto-Hahn-Ring 6, D-8000 München 83

Abstract

Transparency and adaptability are important criteria for the usability of computer systems. It is not easy to fulfill these criteria in ill-structured domains by using traditional software concepts. The object-oriented approach, however, allows the construction of metasystems which enable the user to understand and manipulate the underlying conceptual schemata of an application system.

ZOO[1] is a metasystem for the application expert. ZOO visualizes the contents of an object-oriented knowledge base in a graphic, spatial representation as a net of icons. ZOO allows its user to inspect, manipulate, create and delete the components of a knowledge base through direct manipulation of graphic objects.

1. Introduction

There is an increasing demand for computer solutions in ill-structured problem domains such as office automation, planning, design, or diagnosis. In general there is no formal description of the problem domain and the goals of operation are not well-defined. So it is hard or even impossible to find definite general methods for solving problems in these environments.

---

[1]The author was a member of the Research Group INFORM at the University of Stuttgart, when he developed the ZOO system. The Research Group INFORM is supported by the German Federal Ministery of Research and Technology (BMFT) under contract ITW8404A/4, as part of the joint project WISDOM. The development of the ZOO system was supported by the TA Triumph Adler AG, Nürnberg, West Germany.

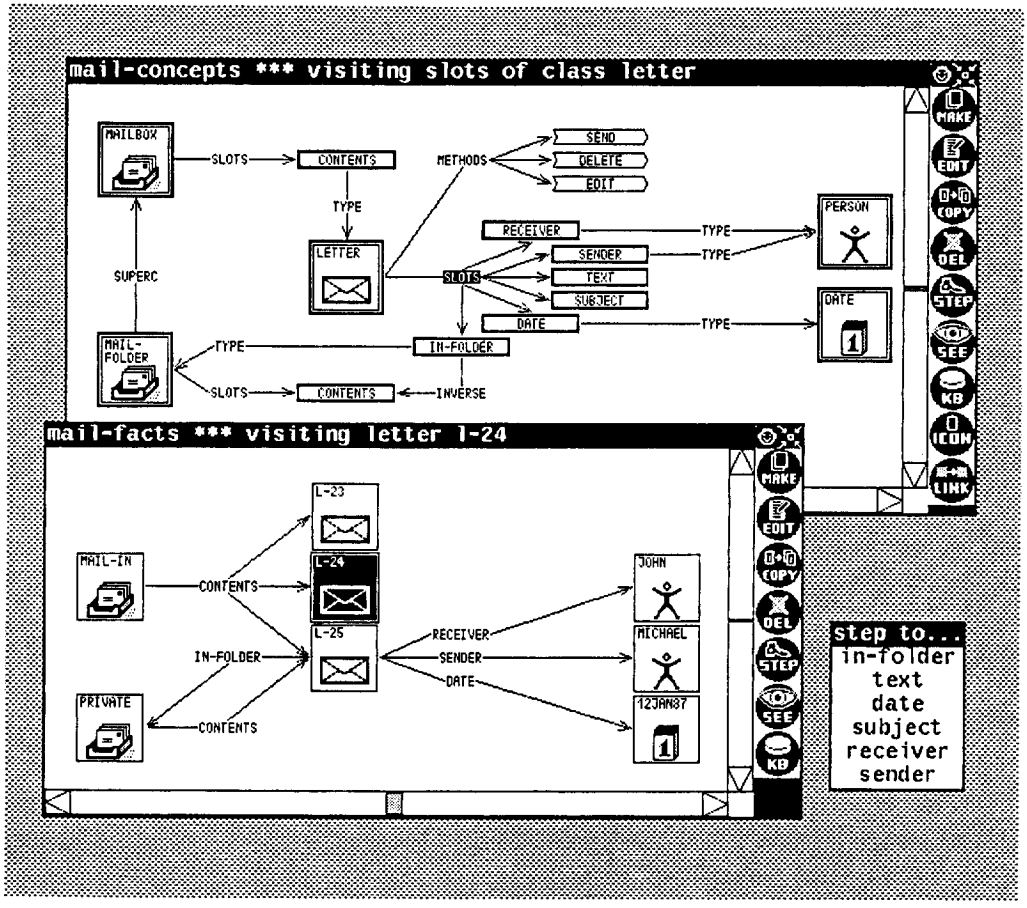

Figure 1: The ZOO metasystem

Therefore the usage of computer software for non-trivial problems in these domains imposes a lot of difficulties on the end user of the system.

- Due to the lack of well-defined goals and methods it is hard for a human to accept the solutions found by the computer. The user does not know the underlying conceptual model of the decisions made by a computer. The program documentation does not help, because it usually describes the software structure of the system rather than the concepts of the application domain.
- Since the application domain is not sufficiently formalized the computer software used is destined for incompleteness. The system designer can only follow heuristics which

still have to be verified in the field. The use of the system will reveal new criteria which require a modification of the software by a software specialist. A permanent communication problem between the user and the programmer of the software is the result, taking a lot of time. Therefore most of the users would rather perform the necessary modifications by themselves.

So there are two criteria with increasing importance for the evaluation of complex software systems: the *transparency* and the *adaptability* of the system. These criteria are not fulfilled while using traditional application systems in ill-structured problem domains. The reason for this deficiency is the way knowledge is represented in traditional systems: Since the knowledge is encoded in a collection of procedures and data structures, its only use is the run of the application system. For all other purposes, the code of a program is a poor representation of knowledge: neither does it help much if we need documentation, explanation or help for the usage of the system nor does it support its augmentation or modification by itself.

Object-oriented application systems, however, offer a way out of this problem. The behavior of these systems is driven by a *knowledge base* consisting of objects rather than by a collection of procedures. There is no limit for the interpretation of the knowledge encoded in objects, so it is possible to design system components which access and manipulate the contents of an object-oriented knowledge base. Such system components operate on a higher level of abstraction than the underlying application system, therefore they are called *metasystems*[2]. Having access to the knowledge sources used by the application system, metasystems are able to explain and to control the behavior of an application system.

The ZOO[3] system presented in this paper is a metasystem which serves two purposes: It is both an instrument for inspecting the contents of an object-oriented knowledge base in a two-dimensional graphic representation and a tool for the modification of knowledge bases by direct manipulation of sensitive screen objects. Graphics is a very popular means used for problem specification and for the documentation of technical systems, so it is used in the ZOO system. ZOO intentionally appears to its user as simple as a system for the display and the construction of business graphics, whereas its functionality is much broader. Every graphic object displayed by the ZOO system represents an item of the system-internal knowledge base. Creating a new graphic object on the screen is a request for generating a new component of the knowledge base as well. Supporting the construction of a graphic diagram and a knowledge base at the same time, ZOO connects the processes

---

[2]The first knowledge-based metasystems were developed in Stanford: Typical examples are the systems EMYCIN/Teiresias and Meta-DENDRAL which are metasystems for the known expert systems MYCIN and DENDRAL [3]

[3]The name ZOO stands for a Zoo Of Objects.

of specification, implementation and documentation of knowledge-based systems in parallel. Thus ZOO may also be considered as a rapid-prototyping tool.

ZOO is running on a VAX 11/780 under Berkeley UNIX 4.2 and requires a BBN BitGraph terminal as communication device. ZOO makes use of the window-system WLISP [5] and the object-oriented language ObjTalk [13] which were both developed by members of the research group INFORM. The basic implementation language of all software components is Franz Lisp [7].

2. External Representation of Knowledge

In our software systems, we use ObjTalk [13; 14], an object-oriented language, for the representation of knowledge. ObjTalk is a language which combines the actor concept and the message passing paradigm of Hewitt [10] and the frame ideas of Minsky [12] in a way similar to KL-ONE [2] or LOOPS [1].

ObjTalk allows the representation of object-level knowledge about concrete facts as well as meta-level knowledge[4] about abstract concepts:

- Concrete facts about the real world may be represented by ObjTalk *objects* and their state. ObjTalk provides *slots* for every object in order to store properties and relations to other objects. ObjTalk allows the classification of objects: Every object belongs to a *class* of which it is called an instance.
- Abstract Concepts may be represented by a special kind of ObjTalk objects, called *conceptual objects*. Classes are conceptual objects representing the common properties of their instances. All classes are forming a specialization hierarchy. Classes provide *slot descriptions*, *rules* and *methods* which are conceptual objects as well.

For viewing and manipulating an object, we need an external representation for it. ObjTalk provides such a format, yet it is dedicated to programmers and consists of a Lisp-expression which would define the object when it is loaded or evaluated. This format is not suited for the end user of a system implemented in ObjTalk.

If there is a terminal with a high-resolution screen, there are new ways of displaying the internal objects and their state in a representation closer to the needs of the user. This was shown by the success of the new graphic user-interfaces which came up with Xerox' Star computer [17] and are a standard now in a lot of small computers. Objects and operations originating in an office environment are visualized by graphic symbols, so-called *icons* and may be activated and processed by *direct manipulation* [16; 11]. Icon-based

---

[4]The terms "object-level knowledge" and "meta-level knowledge" have been introduced by R. Davis [4]

direct-manipulation interfaces are very easy to learn and understand. A Star-like interface, however, does not fulfill all the needs of external knowledge representation:

- The Star interface is limited to the office world, while we want to represent objects of arbitrary domains.
- The Star interface supports only hierarchical relations, e.g. between a document and its folder, but a universal system should represent arbitrary relations between objects.
- It must be possible, to represent abstract concepts (such as the concept *folder*), not only concrete objects (such as the folder *XYZ*).

On the other side there are knowledge engineering tools which meet most of these requirements, but suffer from a poor use of graphics, such as the SMALLTALK-Browser [8]. KEE and LOOPS only display specialization and instantiation hierarchies of objects, they do not support other relations between objects. Moreover, their graphic representations cannot be changed through direct manipulation. KEE's SIMKIT system possesses a complete direct-manipulation interface, yet it is only useful for simulation applications. The idea behind the ZOO system described here is to combine the graphic capabilities of an icon-based system with the universality of a knowledge engineering system.

3. The ZOO Graphic Representation Format

Common objects and conceptual objects of the knowledge representation language ObjTalk may be represented in a graphic form by the ZOO system. For this purpose ZOO provides two kinds of graphic primitives: *icons* and *arrows*.

Icons are used for the graphic representation of *objects*. Usually such an icon has a square shape. It is labelled by the name of the object. It contains a graphic symbol which, in general, visualizes the class membership of the object. *Relations* to other objects are represented by labelled arrows pointing to the graphical representation of the corresponding objects while the label of the arrow contains the name of the relation. Figure 2 shows the graphic representation of an object, a German computer system, in relation to other objects. In summary, object-level knowledge appears as a network of icons as nodes and labelled arrows as links.

Meta-level knowledge consisting of a knowledge base of conceptual objects is represented basically the same way as object-level knowledge. *Classes* appear as icons which are surrounded by a fat border (see figure 3). By default, the graphic symbol used for the visualization of the class will be transmitted to its instances and to its inferiors in the specialization hierarchy. The specialization hierarchy of classes is represented internally by a relation with the name "superc", so it may be displayed using labelled arrows as described above. The other conceptual objects, such as slot descriptions, methods and rules are linked to the network of classes by labelled arrows representing the respective relations.

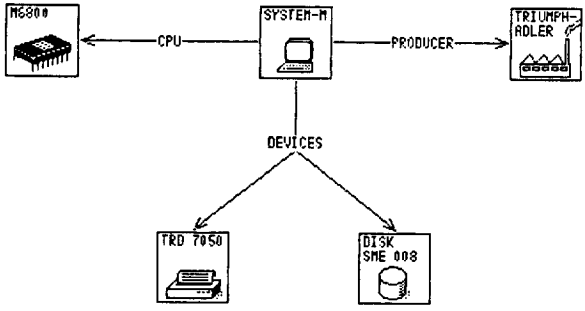

Figure 2: The object *System-M* in relation to other objects

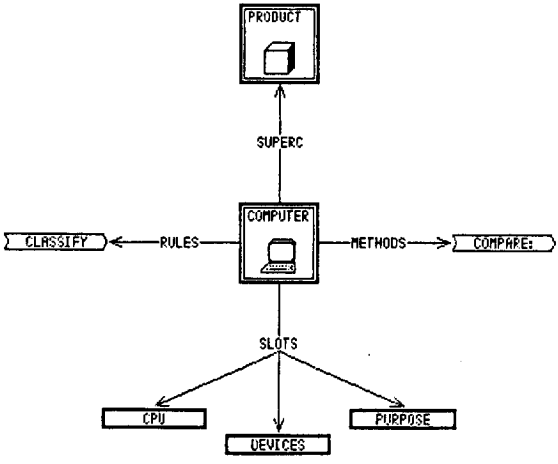

Figure 3: The class *computer* and its surrounding conceptual objects

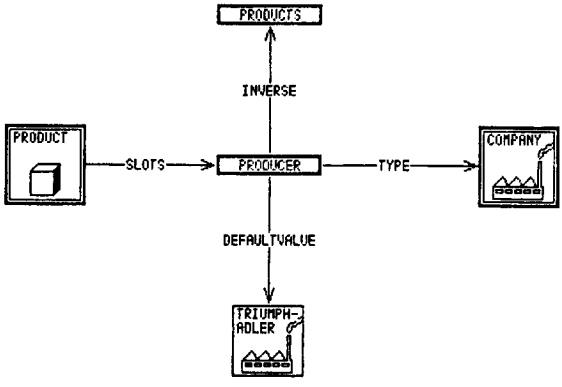

Figure 4: The slot description *producer* from class *product*

Slot descriptions appear as flat rectangles enclosing the name of the slot. Again there may be relation arrows originating from slot descriptions to other icons, displaying the type, the default value and the inverse relation (see figure 4). In a similar way ZOO visualizes *methods* and *rules* within the network of conceptual objects.

4. The ZOO metasystem

The ZOO metasystem may be considered as a user-interface to the whole knowledge represented by ObjTalk objects in a software system. The purpose of ZOO is to present the contents of a knowledge base in the graphic format described above, allowing the user to modify it through direct manipulation. ZOO is operating on the in-core representation of knowledge bases, so it is possible to make changes to a knowledge-based system at run-time. ZOO is able to access conceptual objects representing abstract concepts as well as concrete instances representing facts about the application domain.

The graphic representation of a knowledge base is displayed in a rectangular region on the screen, a so-called *zoo-window* (see figure 1). Each icon represents an object, each labelled arrow represents a relation between objects of the knowledge base. One of these graphic objects (icons or labelled arrows) may be selected by touching it with a pointing device (mouse). All object-specific operations apply to the selected item. The applicable operations are displayed on the border of the zoo-window as function-icons, which can also be activated with the mouse. Objects may be created by copying or instantiating the selected object, relations between objects may be established by drawing arrows from the selected object to other objects. The names of these relations may be selected from a pop-up menu, the contents of which is derived from the class of the selected object. In addition there are functions for storing and retrieving knowledge bases, for viewing objects, for deleting objects, for forgetting relations, and for changing the appearance and position of objects.

There is no central display manager for the graphic objects inside a zoo-window, the display management is done in an object-oriented way instead. Each icon and each labelled arrow is itself represented by an ObjTalk object, a so-called *interaction object* [9]. Interaction objects establish the link between internal objects and their graphic representation on the screen. Therefore interaction objects represent two kinds of knowledge:

- An interaction object possesses graphic knowledge, it describes the position and the appearance of a screen object. This knowledge will be used for the purposes of displaying and activating the graphic objects.
- An interaction object possesses knowledge about the existence of an internal object or about one of its properties. Interaction objects provide methods for the manipulation of their internal counterparts.

Within a zoo-window there are different classes of interaction objects which are used for the graphic presentation of the various kinds of knowledge base objects and their interrelations as described in section 3. The function-icons on the border of a zoo-window do not visualize the objects of the target knowledge base; they represent the functions of the ZOO system instead. Most of these functions apply to the selected graphic object inside the zoo-window. The effect of these functions depends on the class of the selected interaction object. Because of this context-dependent behavior, the number of functions may be kept small, making the user-interface simple and powerful at the same time.

5. Conclusions

The consideration of knowledge aspects leads to a new viewpoint in man-machine communication. The problem of constructing understandable user-interfaces can be reduced to the question of an external knowledge representation adapted to the user's point of view. Our answer to this question is the idea of a two-dimensional representation of knowledge by graphic objects on a high-resolution screen, being subject to direct manipulation with a pointing device.

In this context, an integrated object-oriented software architecture which includes both the user-interface and the system kernel turns out to be very advantageous. An object-oriented user-interface allows the use of interaction objects which appear to the user as mouse-sensitive graphic objects on the screen of a computer terminal. But within the system they are active data objects communicating with the internal objects of the knowledge base. If the system kernel is driven by an object-oriented knowledge base every interaction object finds a unique internal counterpart and this guarantees a close connection between the internal and the external representation of knowledge.

The object as a fundamental concept of programming and representing knowledge makes it possible to construct metasystems for visualizing and modifying the facts and the concepts of an application system. Together with advanced techniques of man-machine communication, the object-oriented knowledge-based approach makes it easy to construct application systems which are more transparent than traditional programs and can be adapted by the user in many parts. Therefore this approach contributes a lot to the development of user-centered software environments.

References

[1] D.G. Bobrow, M. Stefik: *"The LOOPS Manual"*. Technical Report KB-VLSI-81-83, Knowledge Systems Area, Xerox Palo Alto Research Center (PARC), 1981.

[2] Brachmann. R. et al.: *"KL-ONE Reference Manual"*. BBN-Report 3848, BBN, July, 1978.

[3] B.G. Buchanan, E.H. Shortliffe: *The Addison-Wesley Series in Artificial Intelligence: "Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project"*. Addison-Wesley, Reading, Ma., 1984.

[4] R. Davis, D.B. Lenat: *Advanced Computer Science Series: "Knowledge Based Systems in Artificial Intelligence"*. McGraw-Hill, New York, 1982.

[5] F. Fabian Jr., A. C. Lemke: *"Wlisp Manual"*. Technical Report CU-CS-302-85, University of Colorado, Boulder, February, 1985. Translated by V. Patten and C. Morel.

[6] G. Fischer, M. Schneider: *"Knowledge-based Communication Processes in Software Engineering"*. In *Proceedings of the 7th International Conference on Software Engineering*, pp 358-368. Orlando, Florida, March, 1984.

[7] J.K. Foderaro, K.L. Sklower: *"The FranzLisp Manual"*. Technical Report, University of California, Berkeley, 1982.

[8] A. Goldberg: *"SMALLTALK-80, The Interactive Programming Environment"*. Addison-Wesley, Reading, Ma., 1984.

[9] M. Herczeg: *"INFORM-Manual: Icons"*. Institut für Informatik, Universität Stuttgart, 1985.

[10] C. Hewitt: *"Viewing Control Structures as Patterns of Passing Messages"*. *Artificial Intelligence Journal* 8, pp 323-364, 1977.

[11] E.L. Hutchins, J.D. Hollan, D.A. Norman: *"Direct Manipulation Interfaces"*. In D.A. Norman, S. Draper (editors), *User Centered System Design: New Perspectives on Human-Computer Interaction*. Lawrence Erlbaum Associates Ltd., 1986.

[12] M. Minsky: *"A Framework for Representing Knowledge"*. In P.H. Winston (editor), *The Psychology of Computer Vision*, pp 211-277. McGraw Hill, New York, 1975.

[13] C. Rathke: *"ObjTalk Primer"*. Translated Version by A.C. Lemke, V.M. Patten, C.P. Morel, Dept. of Computer Science, University of Colorado, Boulder - Technical Report CU-CS-290-85 -, INFORM, Institut für Informatik, Universität Stuttgart, 1985.

[14] C. Rathke: *"ObjTalk. Repräsentation von Wissen in einer objektorientierten Sprache"*. Dissertation, Fakultät Mathematik und Informatik der Universität Stuttgart, Oktober, 1986.

[15] W.-F. Riekert: *"Der graphische Wissenseditor ZOO – ein Metasystem zur Visualisierung und Manipulation von Wissensbasen"*. WISDOM-Forschungsbericht FB-INF-86-9, Institut für Informatik, Universität Stuttgart, 1986.

[16] B. Shneiderman: *"Direct Manipulation: A Step Beyond Programming Languages"*. *IEEE Computer* 16(8), pp 57-69, August, 1983.

[17] D.C. Smith, Ch. Irby, R. Kimball, B. Verplank: *"Designing the Star User Interface"*. *BYTE*, April, 1982.