

Graphical user interfaces for symbol-oriented database visualization and interaction

Uwe Brinkschulte, Marios Siormanolakis, Holger Vogelsang

Institute for Microcomputers and Automation, University of Karlsruhe
Haid-und-Neu-Str. 7, 76131 Karlsruhe, Germany
E-mail: {brinks|sior|vogelsang}@ira.uka.de

ABSTRACT

In this approach, two basic services designed for the engineering of computer based systems (ECBS) are combined: a symbol-oriented man-machine-service and a high speed database-service. The man-machine service is used to build graphical user interfaces (GUI's) for the database service; these interfaces are stored using the database service. The idea is to create a GUI-builder and a GUI-manager for the database service based upon the man-machine service using the concept of symbols. With user-definable and predefined symbols, database contents can be visualized and manipulated in a very flexible and intuitive way. Using the gui-builder and gui-manager, a user can build and operate its own graphical user interface for a given database according to its needs without writing a single line of code.

Keywords: Database Visualization, Database Interaction, Graphical User Interfaces, Services, Man-Machine-Service, Database-Service, Symbols, GUI-Manager, GUI-Builder, Presentation Objects

1. INTRODUCTION

This research arises from a cooperation with the IEEE task force on *Engineering of Computer Based Systems* (ECBS)¹. One main goal is to define standards for reusable software components, the *services*. A service is designed to solve a given problem by processing tasks and results. Therefore, an open service architecture called *OSA+* (Open System Architecture - Platform with Universal Services) was created².

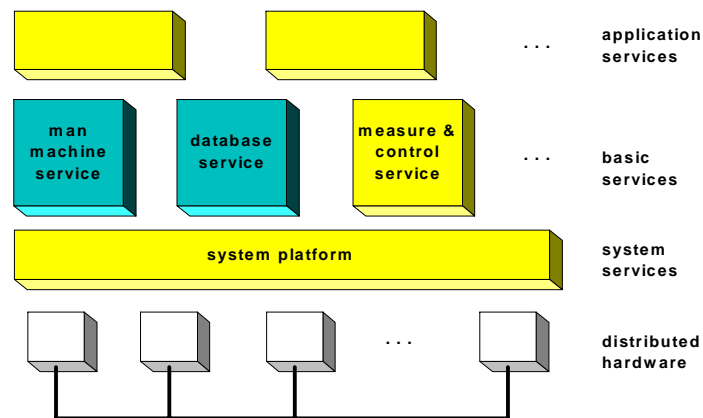
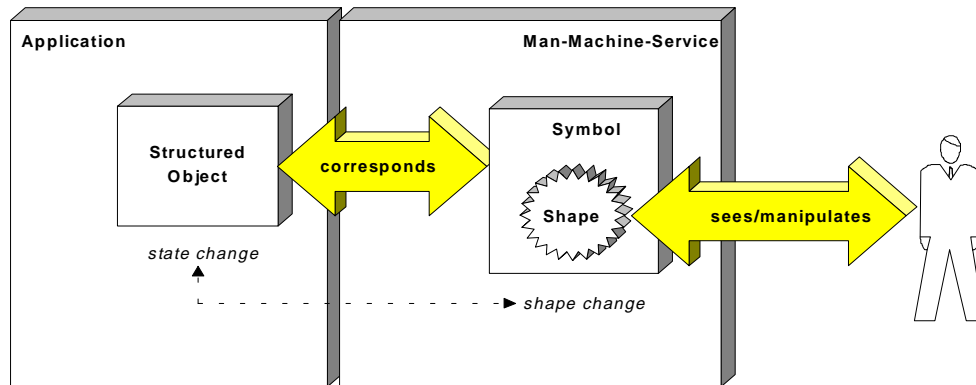


Figure 1: OSA+ service architecture

Based upon a distributed system service (the CORBA³-oriented system platform), several platform- and language-independent basic services have been developed. Two of the most important ones are the *man-machine service* and the *database service*.

The *man-machine service* was presented on EI'96⁴. The main idea is the introduction of *symbols* as graphical representation of object states. A symbol corresponds to a structured object. State changes lead to different visual symbol instances. Interactive symbol manipulation result in altered states.



Simple Example: Visualization and Manipulation of Crossroads

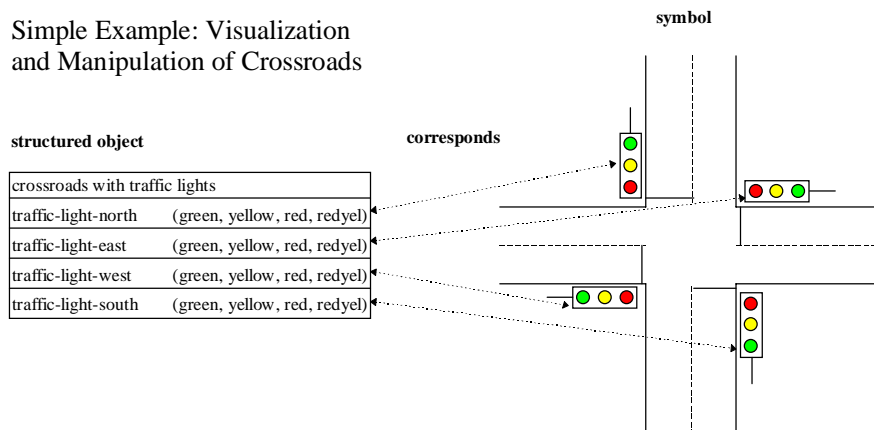


Figure 2: Visualization and manipulation of object states using symbols

Symbols can be created by a *symbol editor*. All created symbols are arranged and positioned in *planes*. Each plane defines a unit of measurement respectively a scale. Planes are displayed in windows using a user defined scaling. A plane can be shown in multiple windows using different scales. Furthermore multiple planes can be shown in a window simultaneously. This allows the creation of easy-to-use and flexible distributed graphical user interfaces (GUI's).

To achieve a complete separation between an application and a GUI, all GUI-elements (windows, planes, symbols, correspondences to structured objects) are stored in a portable database.

The *database service* was designed as a high speed process database system⁵. It provides relational data structures and a simple, set-oriented application program interface. Main properties are high speed data access, soft real-time capabilities, configurable data security and platform independence. For example, the database service is used to store the GUI's created by the man-machine service⁶.

2. CONCEPTION

In the approach presented here, the man-machine service is used to build symbol-oriented graphical user interfaces for the database service. A relational database contains structured objects. They can be visualized by corresponding symbols in a GUI. So the idea is to create a *GUI-builder* and a *GUI-manager* for the database service based upon the man-machine service using the concept of symbols.

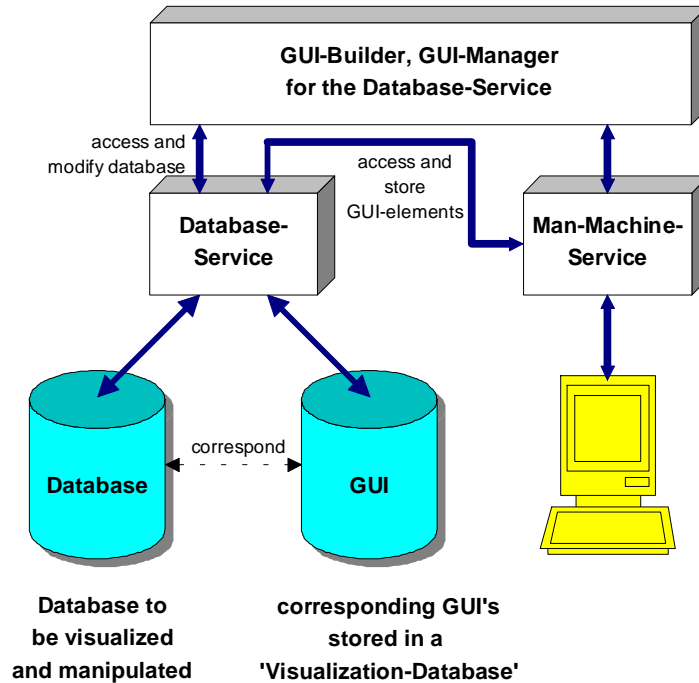


Figure 3: System conception

With user-definable and predefined symbols, database contents can be visualized and manipulated in a very flexible and intuitive way. Using the GUI-builder and GUI-manager, a user can build its own graphical user interface for a given database according to its needs without writing a single line of code

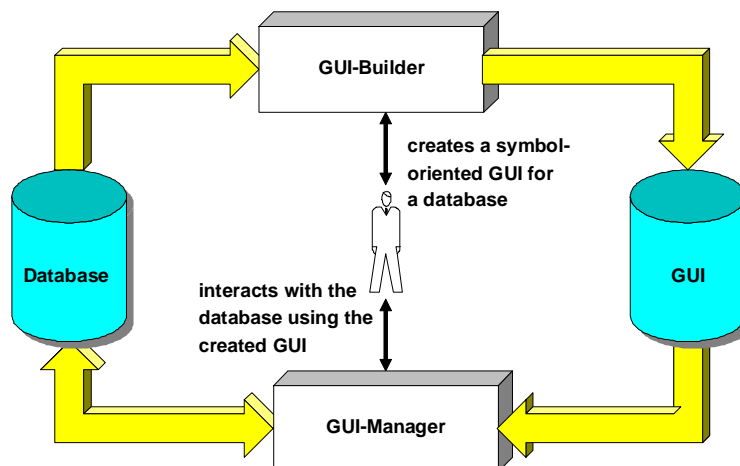


Figure 4: Role of GUI-Building and GUI-Manager

3. ARCHITECTURE

The database service provides relational data structures, that means a database consists of normalized relations. A relation is a set of tuples; each tuple has a given number of attributes. To create a graphical user interface for such a database, the GUI-builder was designed. It consists of two parts, the *symbol-editor* and the *GUI-editor*.

The *symbol-editor* has three main tasks. Firstly, it allows the creation of symbols to visualize and manipulate the attributes of a database relation. For each attribute, a symbol can be either designed by the user or selected from a set of predefined symbols in a symbol library. Predefined symbols are for example sliders, bar graphs, text fields, buttons, etc.

The second task of the symbol-editor is to define the relationship between the attribute value and the resulting image of the symbol. This relationship can be either discrete or continuous, where linear or logarithmic functions are provided. A change in the attribute value leads to a different symbol instance, manipulating the symbol causes a different attribute value.

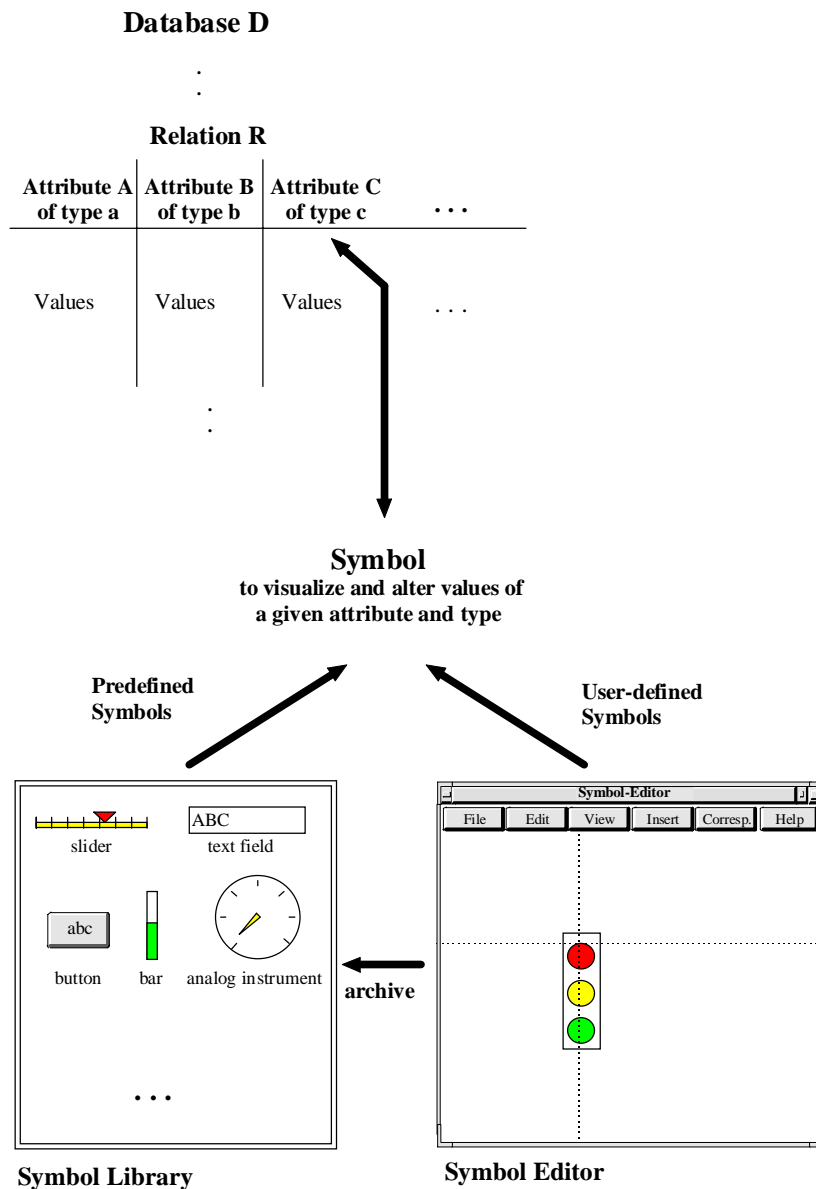


Figure 5: Symbols and attributes

The last task of the symbol-editor is to combine the created symbols to *relation-presentation-objects* (RPO's). Each RPO represents the values of one or more tuples of a database relation. It can be used to show, alter and search for these tuples.

The *GUI-editor* is used to build a graphical user interface consisting of RPO's and other static objects. Static objects are objects, which are not connected to a database relation, for example static text or static pictures. All objects are placed in planes. These planes are shown in windows. Each window can hold multiple planes simultaneously and a plane can be displayed with different scales in multiple windows. This mechanism offers resolution and hardware independent graphical user interfaces. Each generated interface is stored in a *visualization-database*. A database to be visualized can be associated with one or more visualization-databases containing different interfaces for different users.

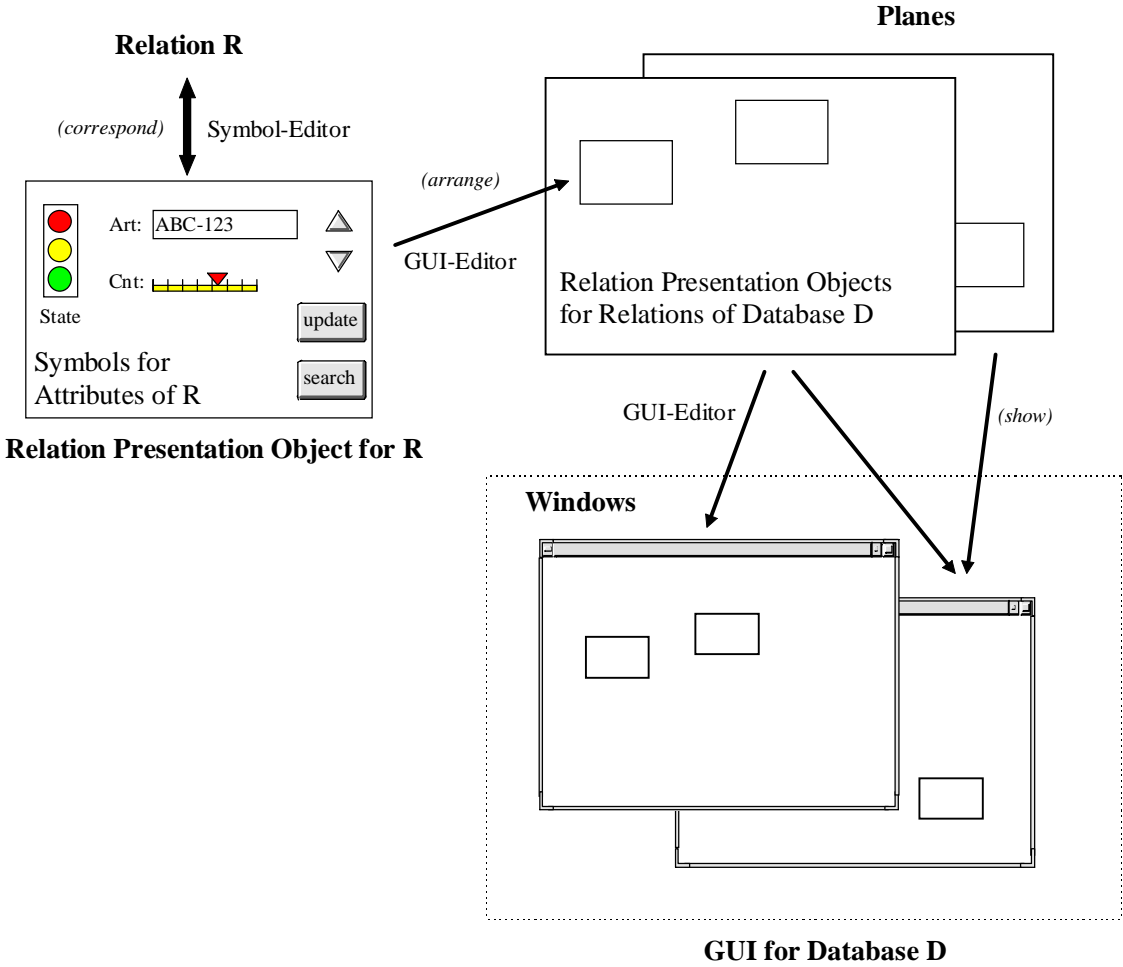


Figure 6: Creating a GUI for a database

The *GUI-manager* finally operates on an interface created by the gui-builder. It accesses the database to be visualized and one of the associated visualization-databases. User interactions for database navigation and modification are handled according to the interface definitions.

4. EXAMPLE

The following simple example shows the construction of a RPO for one relation of a computer manufacturer's storage database:

Extract of database structure:













⋮

Relation Computer Stock

Computer-Type	Productnumber	State	Storage-Count
enumeration type (laptop, tower, tabletop)	alphanumeric string[20]	enumeration type (ordered, ready, under-construction, damaged)	integer
tower laptop ...	EN-11223-A LT-33128-Z ...	ordered under-construction ...	125 26 ...

⋮

Corresponding symbol definitions:

Attribute	Symbol	Graphical symbol representation
Computer-Type	user-defined symbol discrete enum[3]	<i>laptop</i>  <i>tower</i>  <i>tabletop</i> 
Productnumber	predefined symbol alphatext[20]	
State	user-defined symbol discrete enum[4]	<i>ordered</i>  <i>ready</i>  <i>under-const.</i>  <i>damaged</i> 
Storage-Count	predefined symbols slider + inttext[5] + user-defined symbol contineous range[3]	 <i>< 10</i>  (<i>red</i>) <i>10 .. 100</i>  (<i>yellow</i>) <i>> 100</i>  (<i>green</i>)

Note: As demonstrated for the attribute 'Storage-Count', it is possible to assign more than one symbol to an attribute

Figure 7 shows a GUI containing the created RPO (together with a RPO for another relation). In this example each RPO is arranged in a separate plane; each plane is shown in a separate window*. The main window is generated by the GUI-manager.

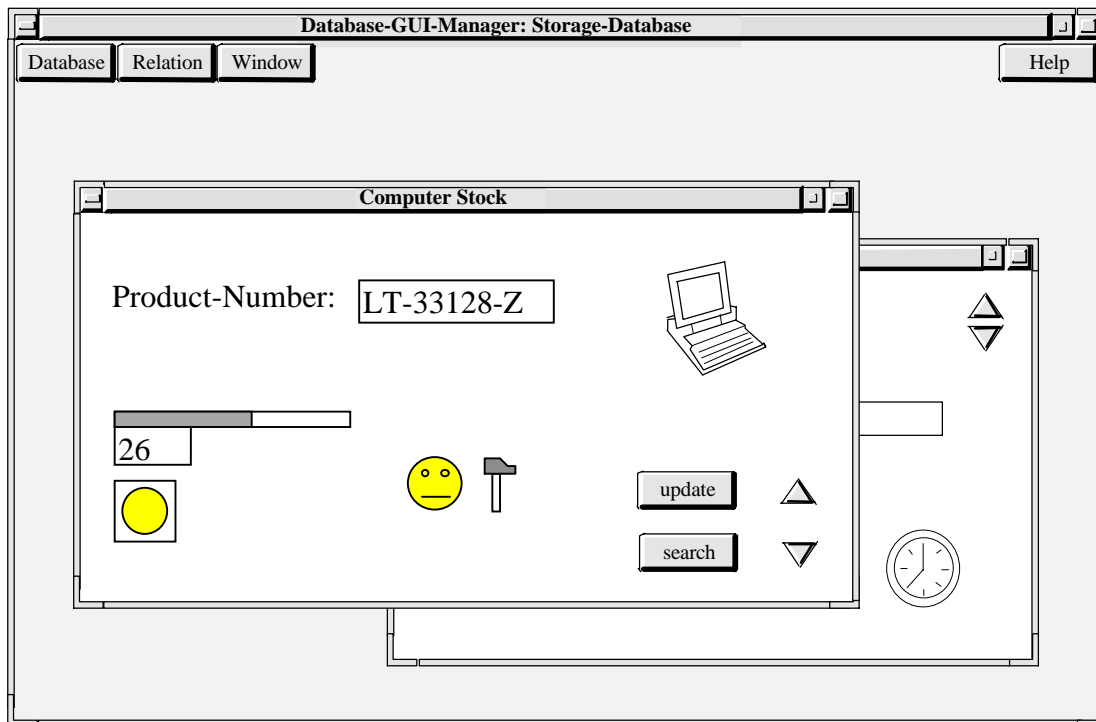


Figure 7: Example GUI

5. CONCLUSIONS

The presented approach combines two existing basic services in two ways. On the one hand, the database service is used to store graphical user interfaces created by the man-machine service. On the other hand, the man-machine service is used to visualize databases of the database service. This has several advantages. The symbol-oriented structure of the man-machine service allows the design of easy-to-use and intuitive interfaces for database visualization and interaction. No code needs to be written by the user. Storing an interface in a visualization-database (separate from the database to be visualized) makes it possible to create many different interfaces for the same database. It also allows the reuse of parts of an interface (for example symbols or relation-presentation-objects) for other databases.

6. ACKNOWLEDGEMENT

This paper is based on research done at the Institute for Microcomputers and Automation - Prof. Schweizer and Prof. Brinkschulte

* This is not necessarily so. Any assignment between RPO's, planes and windows can be made as shown in figure 6.

7. REFERENCES

- 1 G. Schweizer
Foundations for the ECBS Process
ECBS'96, International IEEE Symposium and Workshop on Engineering of
Computer Based Systems, Friedrichshafen, Germany, 1996
- 2 G. Schweizer, M. Voss
Systems Engineering and Infrastructure for Open Component Based Systems.
Eurocast 95, Computer Aided Systems Theory, Innsbruck, Austria, 1995
- 3 OMG
The Common Object Request Broker: Architecture and Specification - Revision 2.0
Object Management Group (OMG), Technical Paper 95-07-20, 1995
- 4 U. Brinkschulte, M. Siormanolakis, H. Vogelsang
Graphical User Interfaces for Heterogeneous Distributed Systems
EI'96, International Symposium on Electronic Imaging,
Visual Data Exploration and Analysis III, San Jose, USA, 1996
- 5 U. Brinkschulte
MERLIN - Ein Prozeßdatenhaltungssystem für Echtzeitanwendungen.
Echtzeit 93, Karlsruhe, Germany, 1993
- 6 H. Vogelsang, U. Brinkschulte, M. Siormanolakis
Archiving System States by Persistent Objects
ECBS'96, International IEEE Symposium and Workshop on Engineering of
Computer Based Systems, Friedrichshafen, Germany, 1996