Visualization and Manipulation of Database Contents Using Corresponding Symbols

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

This paper deals with the problem of representing complex database contents to human operators and manipulating these contents in an intuitive and user definable way. This approach makes use of the graphical facilities of todays computer systems. Information presentation is achieved by displaying graphical components on a screen, where information manipulation is done via haptical interfaces (e.g. mouse, joystick) with optical feedback. We introduce corresponding symbols as universal and user definable interface between database contents and graphical presentation. The graphical attributes of these symbols (like form or color) correspond with the attributes of database information to be represented. Further we introduce a tool to create and arrange corresponding symbols and compose user interfaces for database visualization and manipulation.

1. INTRODUCTION

A main research area of our institute is the design of automation systems. To aid this design process, an open service architecture for automation systems called ***OSA***+ (**O**pen **S**ystem **A**rchitecture **- Pl**atform with **U**niversal **S**ervices) was created [1].[*)] A ***service*** is a reusable software component, designed to solve a given problem by processing tasks and results.

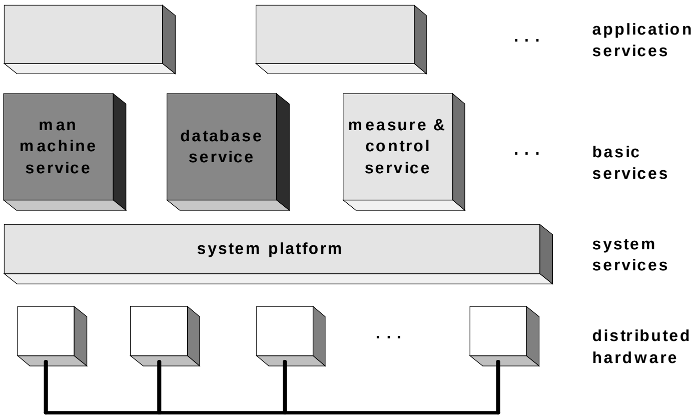


Figure 1: OSA+ service architecture

Based upon a distributed system service (the CORBA [3] -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***.

[*)] This research arises from a cooperation with the IEEE task force on Engineering of Computer Based Systems (ECBS) [2]

The ***man-machine service*** was presented on Visual 96 [4]. 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.

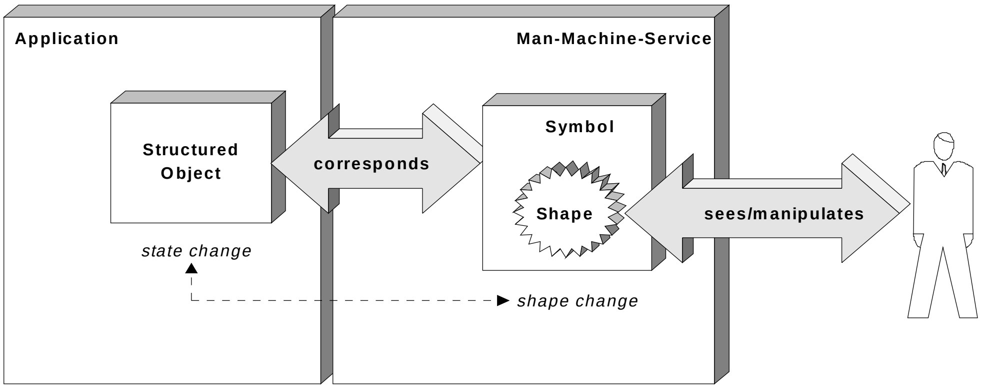


Figure 2: Visualization and manipulation of object states using symbols

Symbols are composed recursively from basic symbols like polygons, ellipses, arcs, text, etc. The attributes of a structured object correspond to the graphical attributes of a symbol (and symbol components) like form, color, fill pattern, visibility, text, size, position, rotation angle and scale. The relationship functions between corresponding attributes can be discrete or continuous, where linear or logarithmic functions are provided.

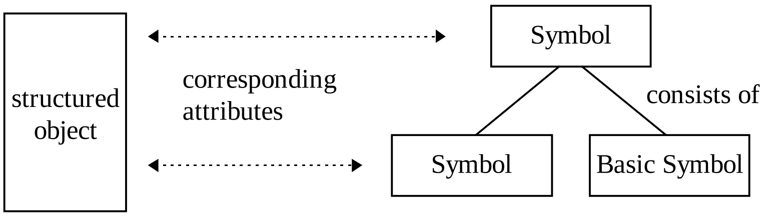
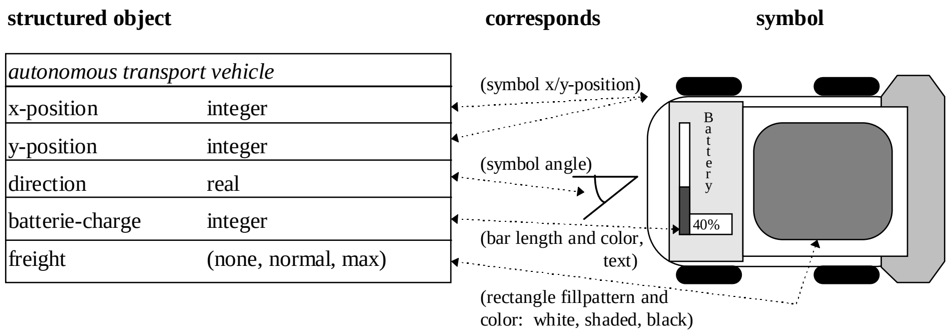


Figure 3: Recursive symbol definition

Figure 4 gives an example. It shows the graphical representation of a vehicle and its states by a corresponding symbol. The symbol consists of several basic symbols. Some symbol and basic symbol attributes correspond to the attributes of the structured object. The symbol position corresponds to the x-position and y-position attribute of the structured object. The symbol angle corresponds to the direction attribute. The battery-charge attribute corresponds to the length and color of a bar (red on low battery charge, else green) and to a text field. The freight attribute corresponds to the fill pattern of a rectangle (white, shaded or black). Manipulating the symbol (e.g. dragging the bar with the mouse) can change the attribute values. Because this makes more or less sense for a given attribute, it can be restricted.

Symbols and their relationships to corresponding objects are created and defined 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.



structured object
corresponds
symbol
(symbol x/y-position)
(symbol angle)
(bar length and color, text)
(rectangle fillpattern and color: white, shaded, black)
Battery
40%

autonomous transport vehicle	
x-position	integer
y-position	integer
direction	real
batterie-charge	integer
freight	(none, normal, max)

Figure 4: Simple example: Visualization and manipulation of an autonomous transport vehicle

The ***database service*** was designed as a high speed process database system [5]. 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 service [6].

In the approach presented here, the man-machine service is used to build symbol-oriented graphical user interfaces for the database service.

2. COMMON DATABASE VISUALIZATION AND MANIPULATION TECHNIQUES

GUI's for database visualization and interaction can be divided in two groups. The first group contains GUI's dealing with specific data items. They are designed to find, show or alter a single or a few sets of data. Several techniques have been developed for that purpose. The most common technique is the use of ***tables, forms*** and ***masks***, which can be arranged by a GUI-builder. This is based on form filling approaches like QBE [7] and is used today in most commercial database systems, e.g. Access [8]. ***Graph based systems*** such as Visual SQL [9] are related, but the user is able to create queries in a graphical way by pointing and clicking on tables and forms. ***Iconic systems*** such as Iconic Query [10] introduce icons to represent data objects (e.g. a table), to show their structure and interrelations and to create queries.

The second group contains GUI's to summarize large mount of data for data exploration. A wide range of visualization techniques are used, such as ***maps***, ***coordinate based charts*** (2- or 3-dimensional charts, scattergrams, line charts, ...), ***ratio based charts*** (pie charts, profiles, ...), ***hybrid charts*** (bar charts, histograms, ...) or ***iconic charts***. Most systems offer a combination of these techniques. The connection between the visualization component and the database component can be tightly coupled as in ExBase [11] (visualization component ExVis [12]) or in Sequoia2000 [13] (visualization component Tioga [14]). It can as well be loosely coupled or independent from a specific database component as for example the IBM Data Explorer [15].
A detailed description of these database visualization and manipulation techniques can be found e.g. in [16].

The approach presented here belongs to the first group of GUI's. It is intended to visualize, find and modify specific database items in a very intuitive way using the ***corresponding symbols*** introduced in chapter 1. The visualization component (man-machine service) is tightly coupled to the database component (database service).
Corresponding symbols and the variety of their possible representations exceed iconic representations. Corresponding symbols are a superset over icons. Limiting the attributes of corresponding symbols to visibility and symbol position leads to an iconic presentation. The powerful symbol manipulation functionality allows easy and intuitive change of information respectively database contents.

3. CONCEPTION AND ARCHITECTURE

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.

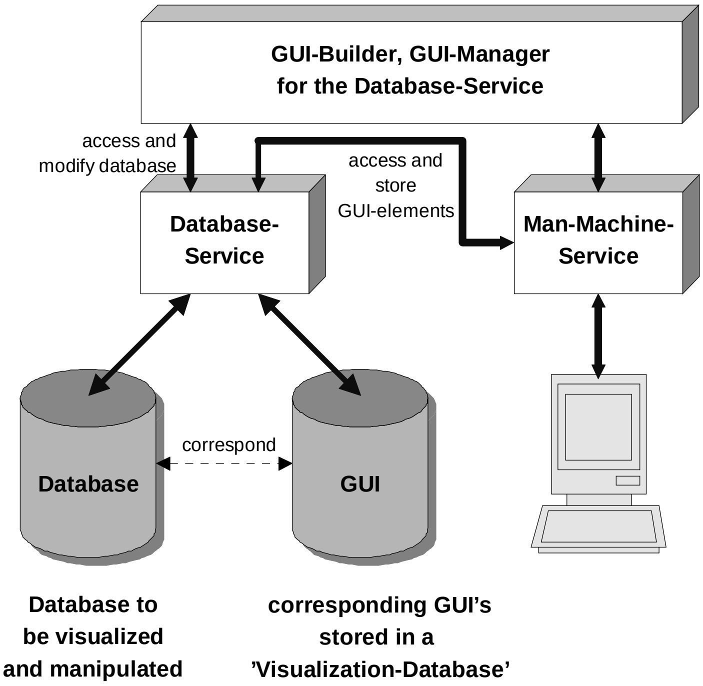


Figure 5: System architecture

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

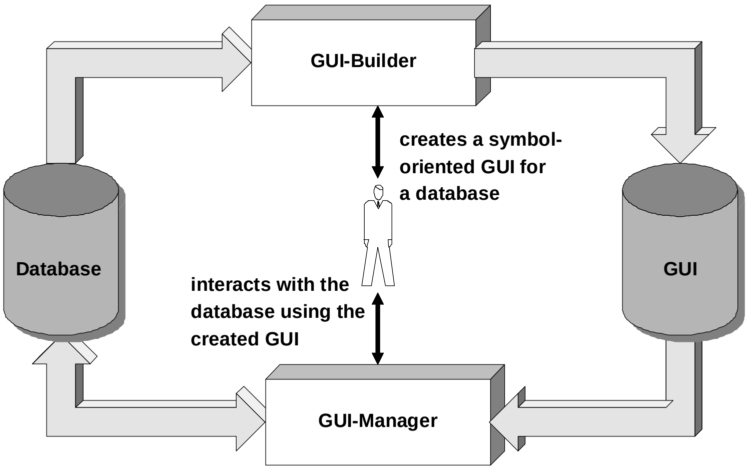


Figure 6: Role of GUI-Builder and GUI-Manager

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. These attributes can correspond to the graphical attributes of symbols (Figure 7).

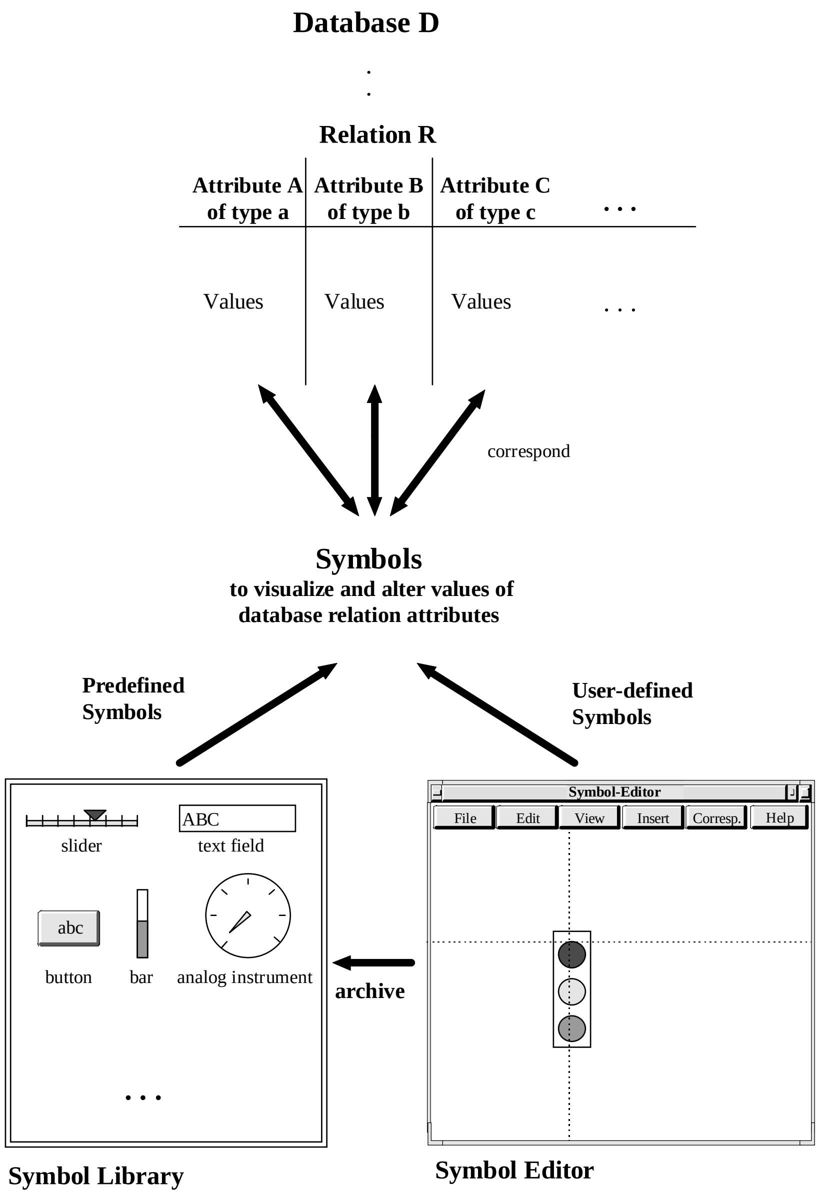


Figure 7: Symbols and database attributes

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. 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 database attributes and the graphical attributes of the symbols. This relationship can be either discrete or continuous, where linear or logarithmic functions are provided. A change in the value of a database attribute leads to a different symbol instance, manipulating the symbol causes a different database attribute value.

Figure 8 shows the most import relationship functions between database attributes and corresponding symbol attributes. Depending on the database attribute type, several functions can be applied to affect different graphical attributes of the symbol.

database attribute type	**relationship function type**	**can affect the following attributes of a symbol (or symbol component)**
enum	*discrete enum [s]*	form, color, pattern, visibility, position, size, angle
subrange, int, real	*discrete range [s]*	form, color, pattern, visibility, position, size, angle
subrange, int, real	*continuous linear*	form, position, size, angle
subrange, int, real	*continuous logarithmic*	form, position, size, angle
char	*alphatext [k]*	text
subrange, int	*inttext [k]*	text
real	*realtext [k,l]*	text

s : number of states
k: number of characters or digits
l: number of digits behind the decimal point

Figure 8: most important relationship functions

Assignments between database attributes and symbols can be made n:m. This means, a symbol can correspond to several database attributes and a database attribute can correspond to several symbols. In the example shown in Figure 9, database attribute A corresponds to symbol 1. Attributes B and C correspond to symbol 2, which means, each database attribute controls a specific graphical attribute of the symbol, e.g. color and angle. Additionally, attribute C corresponds to symbol 3. So this attribute has two graphical representations.

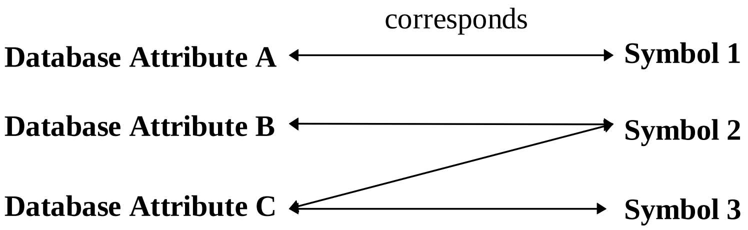


Figure 9: n:m correspondations

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. In that context a database relation must not be a physical relation. It can be a virtual relation as well, created from physical database relations by join, restriction or selection operations.

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.

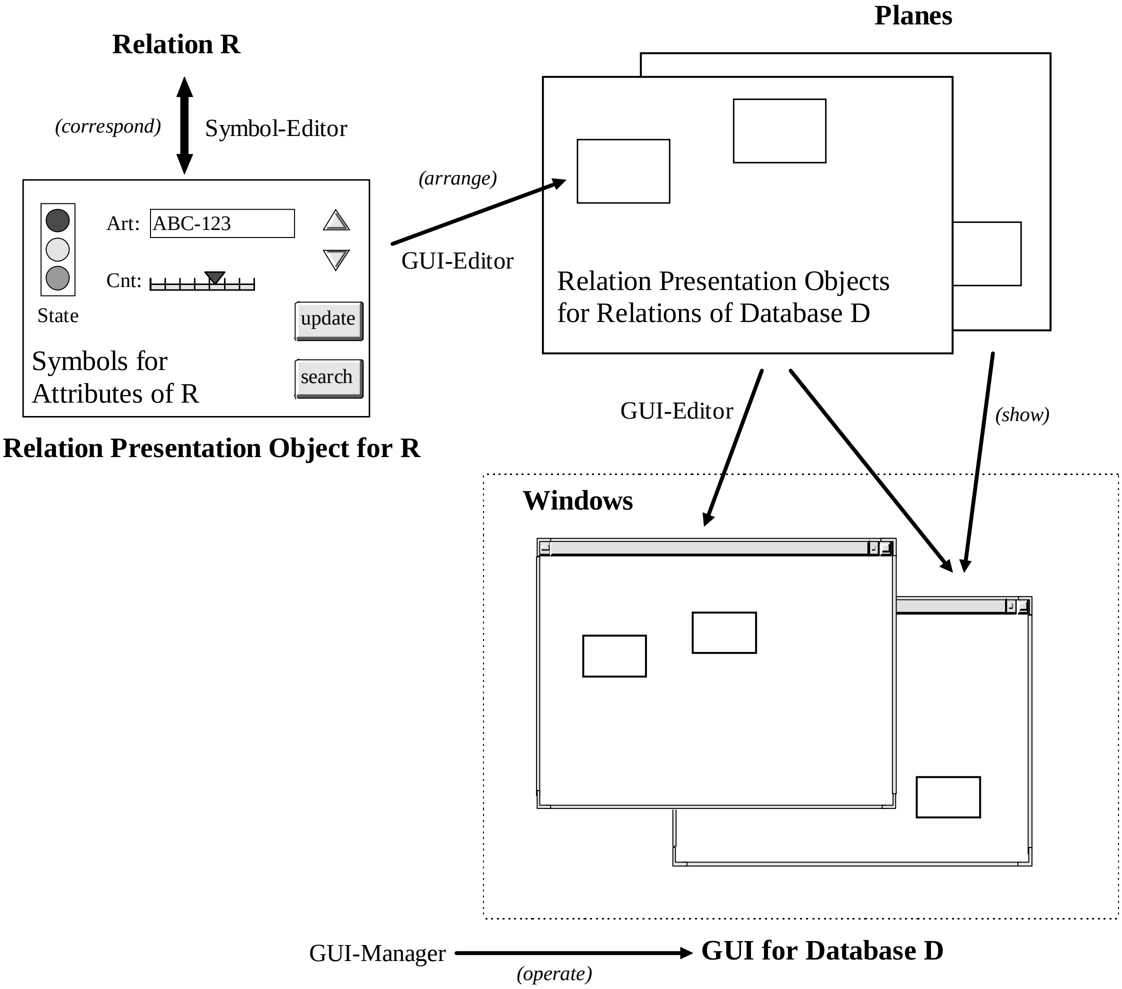


Figure 10: Creating a GUI for a database

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 Production

Computer-Type	Productnumber	Production Stage	Delivery Month	Storage-Count
enumeration type (labtop, tower, tabletop)	*alphanummeric string[20]*	*subrange type (1 .. 4)*	*subrange type (1 .. 12)*	*integer*
tower	EN-11223-A	2	8	125
labtop	LT-33128-Z	4	9	26
. . .	. . .	. . .	. . .	. . .

⋮

Corresponding symbol definitions:

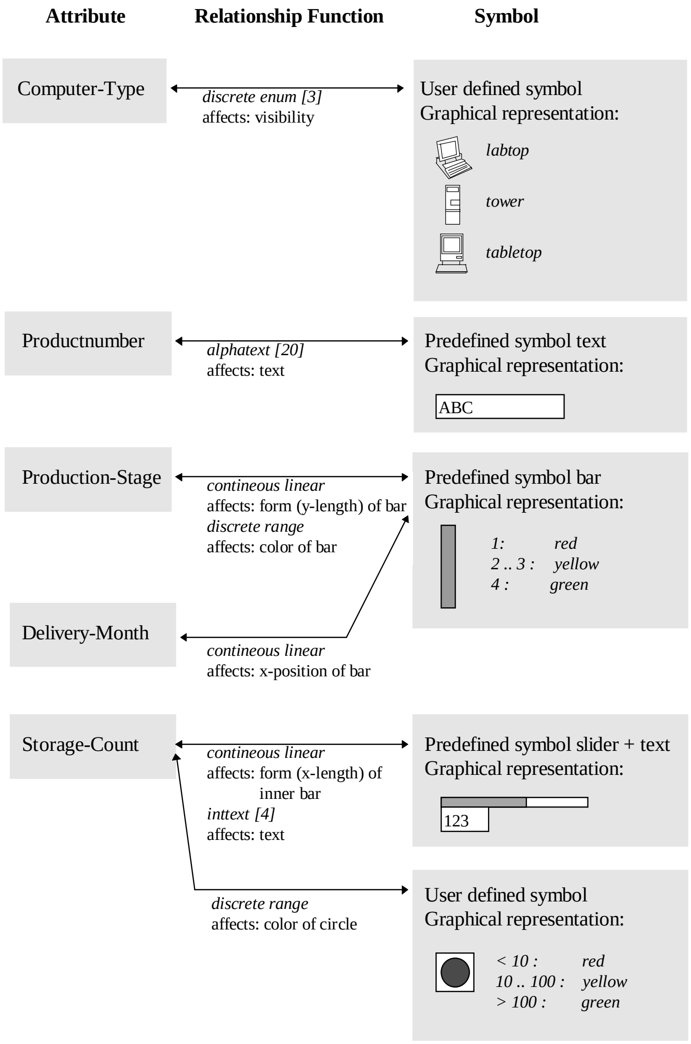
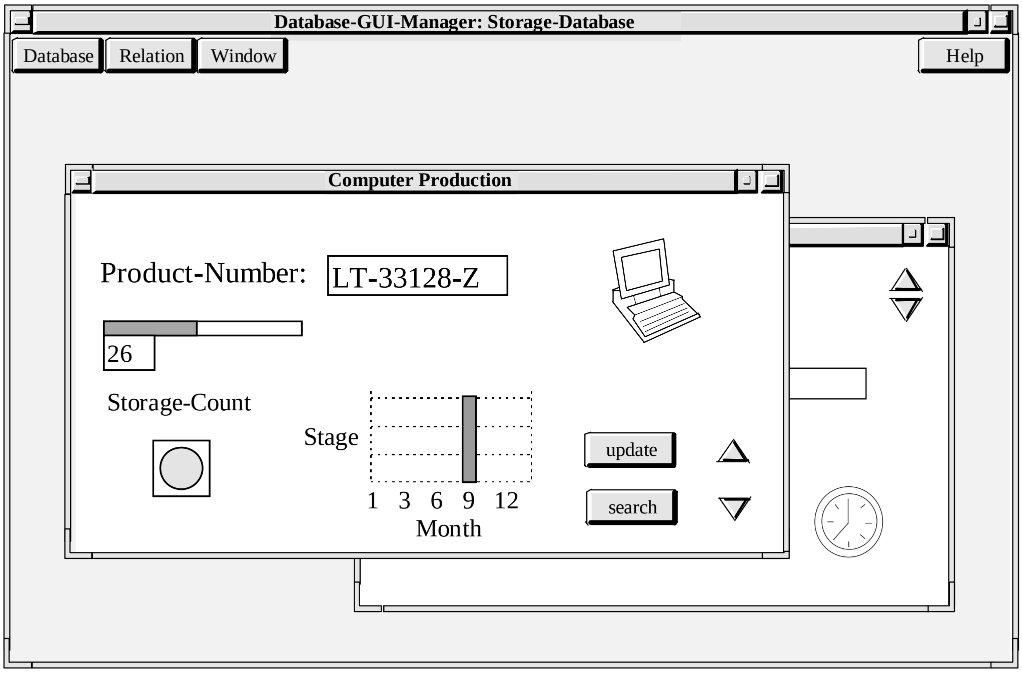
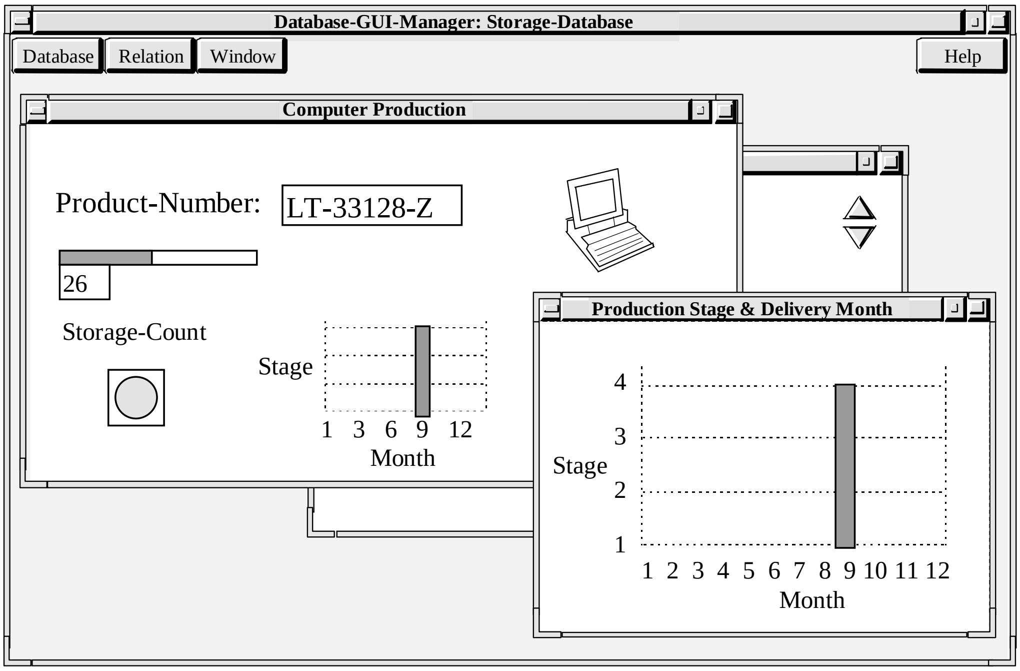


Figure 11a shows a GUI containing the created RPO (together with a RPO for another relation). In this example each RPO is arranged in a separate plane; each plane is shown in a separate window. The main window is generated by the GUI-manager. In Figure 11b, a part of the plane containing the created RPO is shown in an additional window using a larger scale. Furthermore, static text was placed in separate planes to allow different grid text for both windows. In general, any assignment between RPO's, static objects, planes and windows can be made.


Product-Number: LT-33128-Z

Storage-Count: 26

Figure 11a


Product-Number: LT-33128-Z

Storage-Count: 26

Figure 11b: sample GUI's

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 platform-independent visualization-database (separate from the database to be visualized) makes it possible to create many different interfaces for the same database. The created interfaces can run on different hard- and software platforms. It also allows the reuse of parts of an interface (for example symbols or relation-presentation-objects) for other databases.

6. REFERENCES

1 G. Schweizer, M. Voss
Systems Engineering and Infrastructure for Open Component Based Systems.
Eurocast 95, Computer Aided Systems Theory, Innsbruck, Austria, 1995

2 G. Schweizer
Foundations for the ECBS Process
ECBS’96, International IEEE Symposium and Workshop on Engineering of
Computer Based Systems, Friedrichshafen, Germany, 1996

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
Visualization and Manipulation of Structured Objects
Visual 96, Melbourne, Australia, 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

7 M. Zloof
Query by Example
IBM Systems Journal 16, 1977

8 Access 7.0 User Manual
Microsoft, 1996

9 J.H. Trimble, D. Chappel
A Visual Introduction to SQL
Wiley, New York, 1990

10 Iconic Query Reference Manual
IntelligenceWare, Los Angeles, 1992

11 J. P. Lee
Data Exploration Interaction and the Ex Base System
Database Issues for Data Visualization, IEEE Visualization Workshop 93, San Jose 1993
Springer, New York, Lecture Notes in Computer Science 871

12 G. G. Grinstein, R. M. Pickett, M. S. Williams
Exvis, An Exploratary Visualization Environement
Eurographics 92, Cambridge, England, 1992

13 M. Stonebraker, J. Frew
The Sequoia 2000 Architecture and Implementation Strategy
Sequoia 2000 Technical Report 93/23, University of California Berkley, 1993

14 M. Stonebraker, J. Chen, N. Nathan, C. Pax ton, A. Su, J. Wu
Tioga: A Database-Oriented Visualization Tool
Visualization 93, San Jose, USA, 1993

15 IBM Visualization Data Explorer User’s Guide
IBM, New York, 1993

16 K. Parsaye, M. Chignell
Intelligent Database Tools & Applications
Wiley, New York, 1993