Visualization and Manipulation of Structured Information

Uwe Brinkschulte, Marios Siormanolakis, Holger Vogelsang
Institute for Microcomputer and Automation, University of Karlsruhe, Haid-und-Neu-Str. 7, 76131 Karlsruhe, Germany
E-mail: {brinks|sior|vogelsang}@ira.uka.de

Abstract
In modern object oriented visual information systems the internal state of the entire system consists of the internal states of many objects in different services, probably distributed over a heterogeneous network of computers. Man machine interaction in such an application is based on the visualization of states on one hand and the modification of states on the other hand. A solution for this problem is the invention of symbols as a graphical representation of structured information, held by objects. The symbols are handled by a man machine service, which is constructed as a server.

1 Overview

The main task of any man machine service (MMS) is to inform a human user of a system's state and to enable modifications of this state. Due to the fact that man can perceive and handle information fastest in a visual way, this is the best channel to inform about complex system states. The optical channel is also a good choice to support human interaction. This is achieved in feeding back the user's actions. A MMS has to provide two services:

- the visualization of structured information
- and the supported modification of structured information

Based upon these services any communication between user and machine can be realized. This paper presents the concept and an implementation of a man machine service, usable as a server in a heterogeneous network of computers.

2 Symbols

A symbol is a graphical representation of a structured data type [2]. The user can define symbols very flexible with a tool called *Symboleditor*. The defined symbols are stored in a configuration database for usage within the application. Symbols can be defined hierarchically, i.e. a symbol can contain other symbols or base symbols. Changing a value of a data type connected to a symbol leads to a different graphical representation. Changing the graphical representation (e.g. the user moves a symbol interactive) leads to a different data type value. The relations between data type values and the resulting images can be defined. This relation is either continuous, where we provide linear or logarithmic functions, or discrete. Common attributes to symbols and base symbols which can be modified are:

- The *position* specifies the location of a symbol or a part of it.
- The *scale* specifies the dimension of a symbol or a part of it.
- The *orientation* allows to rotate a symbol or a part of it.

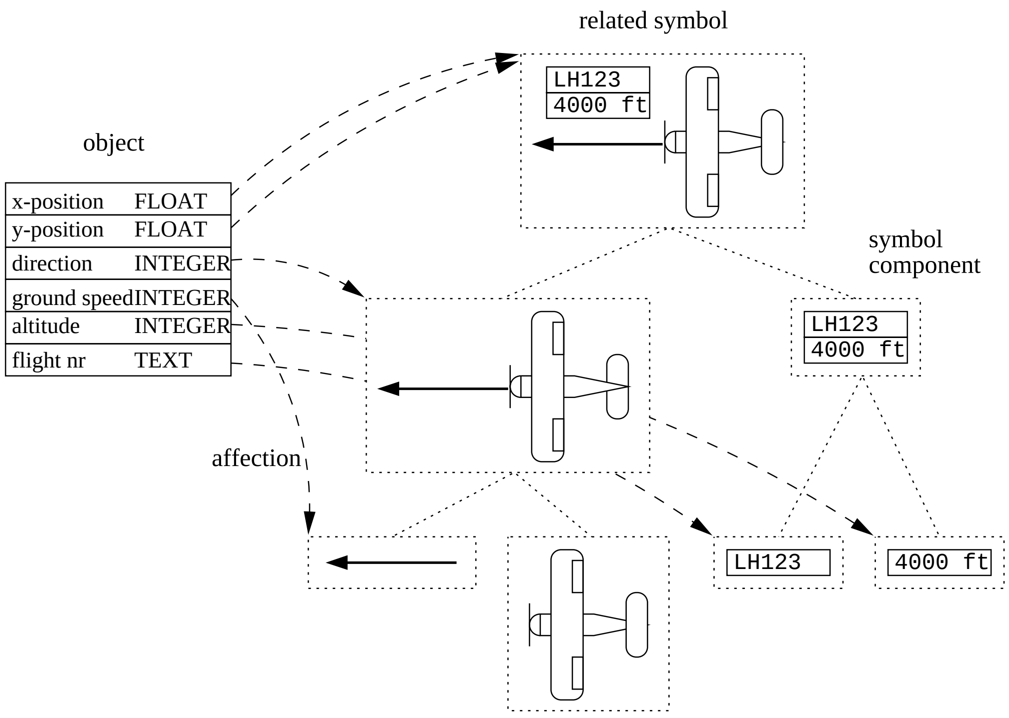

Fig. 1. Example of a complex symbol

- The *visibility* allows to show respectively hide a symbol or a part of it.

Base symbols have additional attributes like *linecolor, linetype, fillpattern, …* which can be modified. These attributes depend on the type of the base symbol, i.e. a line has two specific points defining it, a polygon has n characteristic points. One data type value can act on many of these modifications simultaneously. Figure 1 shows an example of a complex symbol. The structured information is the position, direction, speed, altitude and the flight number of an airplane. This information plus its types is displayed on the left. The right side shows the hierarchical structure of the airplane symbol. The dashed arrows represent the influence of the information on the symbol. E.g. the direction influences the orientation of the airplane and the speed vector. The length of the speed vector depends on the ground speed and the position of the whole symbol is set according to the airplane position.

3 Planes — Windows

All symbols are arranged and positioned in *planes*. Each plane defines a unit of measurement respectively a scale. One symbol can only be assigned to one plane. A plane allows the grouping of symbols.

The user can define rectangular areas on the screen. Referred to here as *windows*. Windows can superpose each other and the sequence in the window stack can be changed.

Planes with all their symbols can be displayed in windows. A window can hold multiple planes simultaneously and a plane can be displayed with different scales in multiple windows. Each window holds a stack of the assigned planes and their scales. The stack sequence is changeable. Figure 2 shows an example — a roadmap — with two planes displayed in two windows.

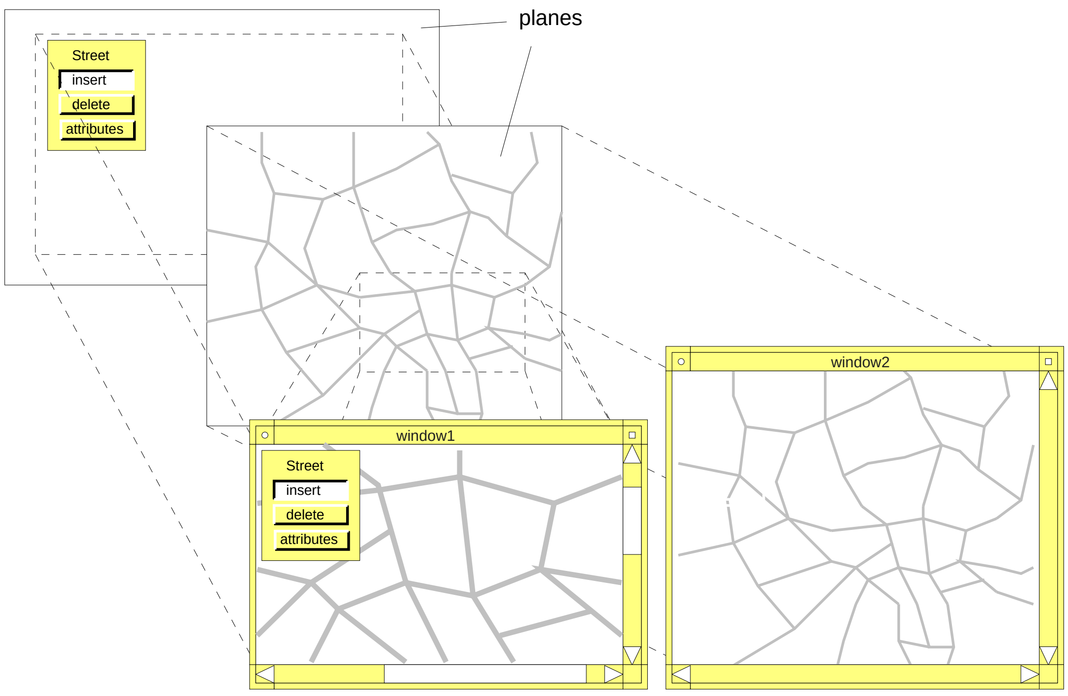

Fig. 2. Planes and windows

4 Presentation objects

Normal symbols are used to visualize a big amount of user defined data types. The *presentation object* is introduced to offer the developer the facility to group symbols together and to create images of complex data type with a special semantic. There are different types of presentation objects predefined:

- A *picture* is a set of symbols as an image of a set of objects of the application. There are no restrictions concerning the object types. The picture is the basis for all other presentation objects.
- A *menu* is an image for a variable of an enumeration type, each button shows a selectable value. Nearly any kind of symbol can be used as a button.
- A *mask* is an image for an object or a structure of an application: Modifiable components of the object can be changed by the manipulation of the corresponding symbols (sliders, buttons, textfields, . . .)
- A *table* is an image of an array of objects or structures.

Presentation objects can be build automatically by the service if the type of the corresponding object is known at runtime. Because every presentation object is derived from the same common class, they share the same (small) set of operations. The presentation objects themself are used to build higher level objects like text editors, hierarchical graphs and help systems. These can be interpreted as pictures with special semantics and a predefined behavior.

5 Events and bindings

Presentations objects itself are useful for the manipulation and visualization of objects. But to allow interactively modifiable and dynamically changeable user interfaces, a powerful mechanism for event recognition and execution is created: the *binding*. A binding is an operation, defined on a presentation object. The execution of the operation is triggered by one or more events on this object. The main ideas behind are:

- Several internal operations of the man machine service can be bound to events, so that typical interactions can be created by the GUI-tool (see below) without writing any line of code.
- Presentation objects can be bound together to create hierarchical menus, masks and tables.
- User defined operations can be bound to events to create callback functions. An application is able to catch an event using this technique.

Presentation objects with a predefined behavior on events are implemented using bindings. Objects of a higher level, which are using presentation objects like pictures, are supplying their components with task-specific binding functions to have control over the event responding.

6 Graphical User Interface

The graphical user interface is a group of presentation objects and windows, which are needed at the same time to solve a given task. It is placed in a database to separate the application from the GUI. This allows the reuse of the entire GUI or parts of it in other applications and the on-line modification of the GUI through the application itself.

Any number of GUI's can be used at the same time simultaneously.

7 Interactive tools

The presented approach for a man machine interface requires tools to allow an interactive and comfortable way to create symbols and GUI's.

- The *Symboleditor* is an interactive tool, which enables the user to construct symbols as an images of data types (together with the data types itself). Furthermore: It enables the developer to describe the kind of relation between the image and the data type (proportional display, text display of a value, range of values, ...).
- The *GUI-editor* is used to build graphical user interfaces as a set of presentation objects, bindings and windows. A library management simplifies the reuse of prior constructed objects.

8 Realization

MMS is written in C++ language for manipulating symbols, planes and windows. The platform dependent parts of the MMS are based upon an uniform interface provided by a window service. The functions of this service are grouped in two parts: window manipulations and graphical drawing primitives in windows.

A set of Elementary Graphical Functions EGF [4] implemented on various platforms covers the second part. The first part can either make use of the window functions of the beneath layer (Xlib, Xwm, ...) or use the module *window manager* to provide the same functionality. The service is separated into a client and a server part. The server provides the functionality and the client the access to it. This is implemented using distributed objects.

GUI's are stored in a database, using persistent objects, to ensure reuseability in different projects and to allow the unmodified use on other hardware platforms [5], [6], [1].

The application stores its internal states in a database using persistent objects to ensure the survive of the objects states. The man machine service itself uses the same technique to hold the symbol definition and the user interfaces together with the bindings between the application and the graphical representation.

Figure 3 presents the internal structure of the service without the persistent object layer. Figure 4 shows the MMS basic service and realization for different hard- and software platforms

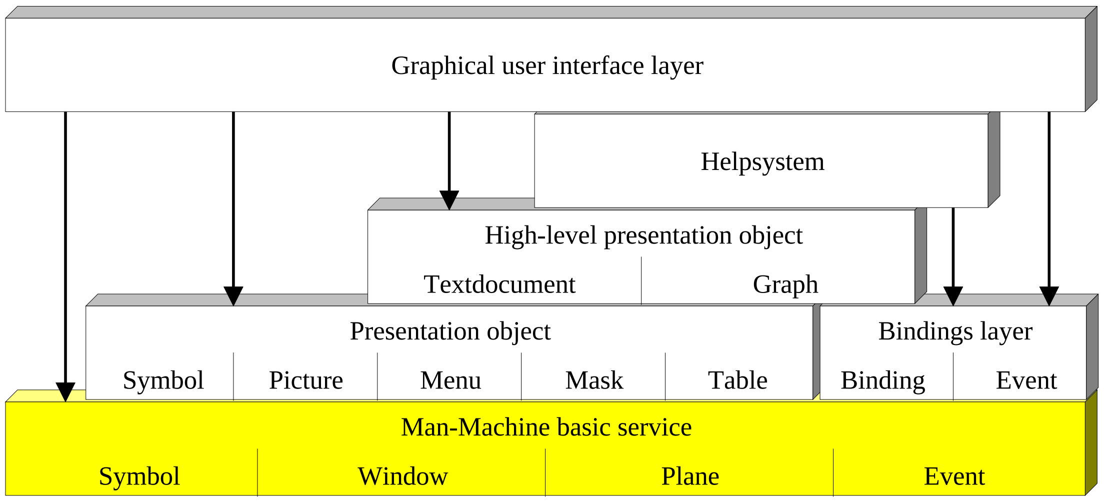

Fig. 3. Internal MMS structure

in detail.

9 Conclusion

The resulting environment allows pleasant and comfortable development of user interfaces for distributed systems. This has been verified in different applications e.g. a railway control system, the control of a production cell [3], a traffic control system and a project management tool. Free definable symbols have reduced the need for an additional software coded functionality. Bindings are a powerful mechanism to implement a user controlled behavior and reduces the programming overhead.

The major features of this man machine service are:

- Portability
- Flexibility
- Complex graphical objects
- Tools and editor services
- Separation of GUI database and application
- Resolution independence
- Client/Server-Concept
- Recording and playing of sounds

10 Acknowledgment

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

References

[1] Richard P. Gabriel, *Persistence in a Programming Environment*, Dr. Dobb's Journal, 12/1992
[2] Gunnar Johannsen, *Mensch-Maschine-Systeme*, Springer, 1993
[3] C. Lewerentz, T. Lindner, *Case Study "Production Cell". A Comparative Study in Formal Software Development*, Forschungszentrum Informatik, Karlsruhe, FZI-Publication 1/94

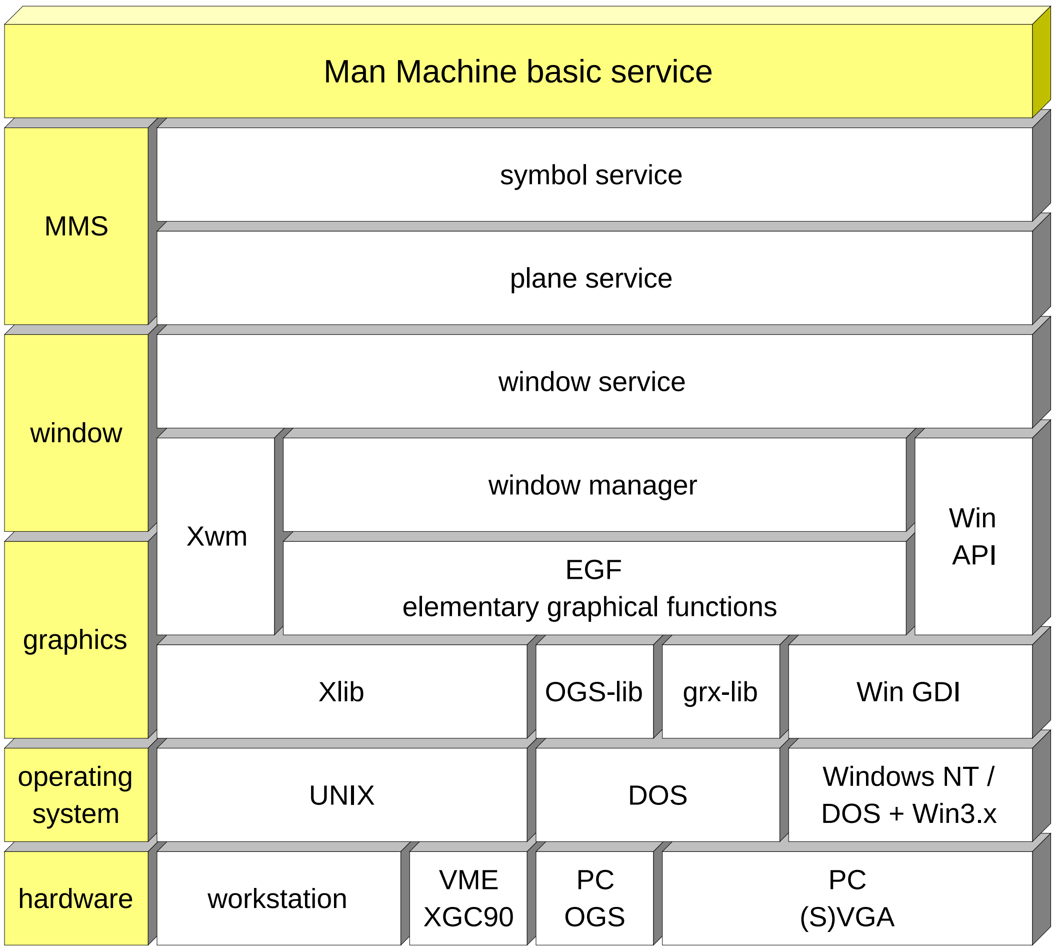

Fig. 4. MMS basic service

[4] Kapp, K.-H. and Siormanolakis, M.: *EGF – Elementary Graphical Functions*, Institute for Microcomputers and Automation, Internal Report, University of Karlsruhe (1994)
[5] Al Stevens, *Persistent Objects in C++*, Dr. Dobb’s Journal, 12/1992
[6] Holger Vogelsang, Uwe Brinkschulte, *Persistent Objects In A Relational Database*, submitted paper to ECOOP’96, 1995