Graphical user interfaces for heterogeneous distributed systems

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 computer systems the internal state of the entire system consists of the internal states of many objects, probably distributed over a heterogeneous network of computers. The man machine interaction in such an application is based on the visualization of states on one hand and the modification of states in combination with event generation on the other hand. This paper describes concept and realization of a reusable service for general man machine communication.

Keywords: graphical user interface, data visualization, service, object-oriented service

1 OVERVIEW

This research arises from a cooperation with the IEEE task force on ECBS and has two main goals: First, to find a common standard for the process of creating large computer systems and second to define standards for reusable software components, the ”services”. A service is designed to solve a given problem by processing tasks and results. With a given communication platform it is possible to place the services on different hardware systems independent of the specification. This is realized by distributed objects for heterogeneous systems (Figure 1).

Three basic types of services can be distinguished: The *system services* are needed for communication and process scheduling. The *basic services* are located above the system service. At this point of time, there are a layer for man machine interaction, for measurement and control and a database system for persistent data. The third type of service is application dependent, which means, that these services are build for a special solution, whereas the other ones only have to be configured for a given task.

One of the most important services is the man machine interaction. The task of any man machine service (MMS) is to inform a human user of a system's state and to allow 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 general functions:

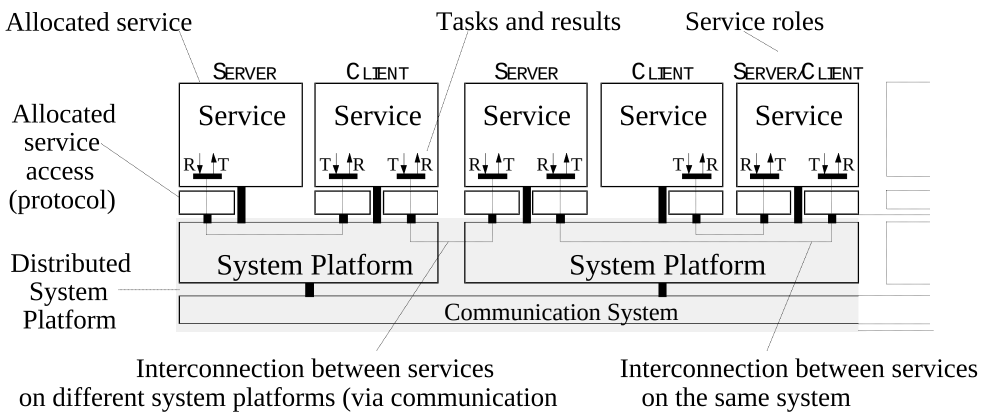
Allocated service
Tasks and results
Service roles
SERVER
CLIENT
SERVER
CLIENT
SERVER/CLIENT
Service
Service
Service
Service
Service
Allocated service access (protocol)
R T
T R
T R
R T
R T
T R
R T
T R
System Platform
System Platform
Distributed System Platform
Communication System
Interconnection between services on different system platforms (via communication
Interconnection between services on the same system

Figure 1: Physical structure of a service-oriented system

- the visualization and
- the modification

of structured information.

Based upon these functions 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

The basic idea of the man machine service is the introduction of symbols as state-pictures of structured objects of an application, e.g. process variables of a control unit. The user can define symbols and their behavior on events very flexible with an interactive tool, the *symbol-editor*. Symbols are composed of base-symbols, such as lines, circles . . . and other user-defined symbols. As a result, symbols may contain a hierarchy of components. These are stored in a configuration database for usage within the application. After the configuration or construction, the symbols are available in the application. To use them, they have to be connected with an object. Changing a value of this object leads to a different graphical representation. Changing the graphical representation (e.g. the user moves a symbol interactive) leads to a different object value. The relations between object values and the resulting images can be defined. This relation is either continuous, where linear or logarithmic functions are provided, or discrete. Figure 2 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.

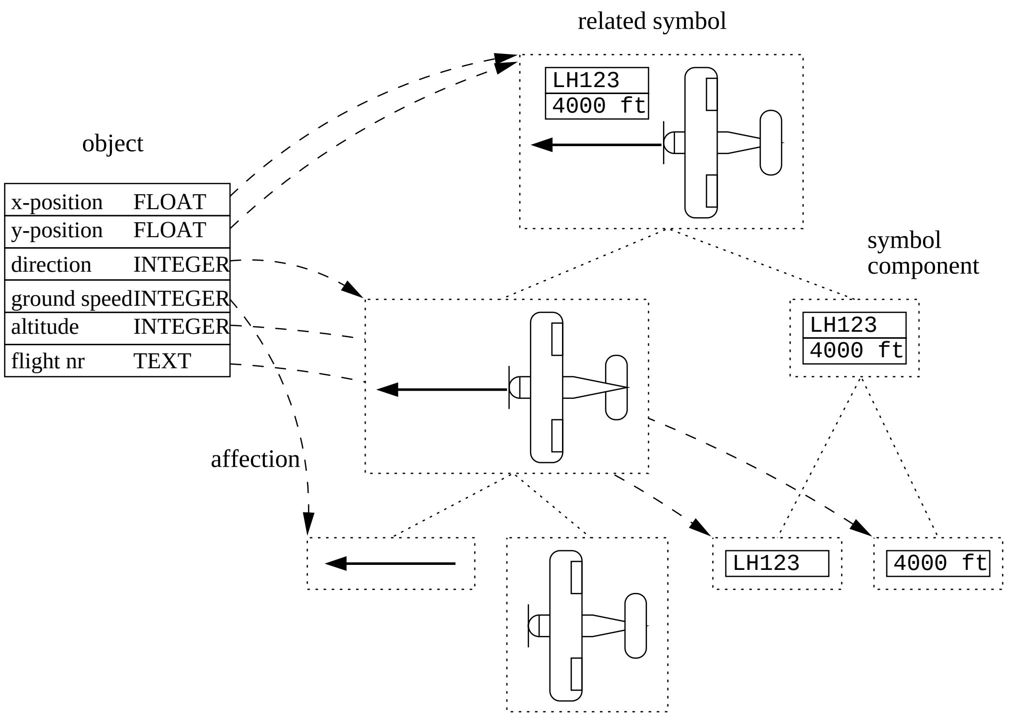

Figure 2: Example of a complex symbol

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. Planes with all their symbols are displayed in windows, using a user defined scaling. It is possible to display multiple planes in one window at the same time, using a stack of planes. On the other hand it is possible to visualize a plane in more than one window at the same time.

Symbols are constructed either by using an interactive *symbol-editor* or by applying the internal API.

3 PRESENTATION OBJECTS

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.

3.1 Predefined presentation objects

There are different types of presentation objects predefined:

- A *picture* is a set of symbols. This is used as an image for a set of application objects. There are no restrictions concerning the object types. The picture is the basis for all other presentation objects.

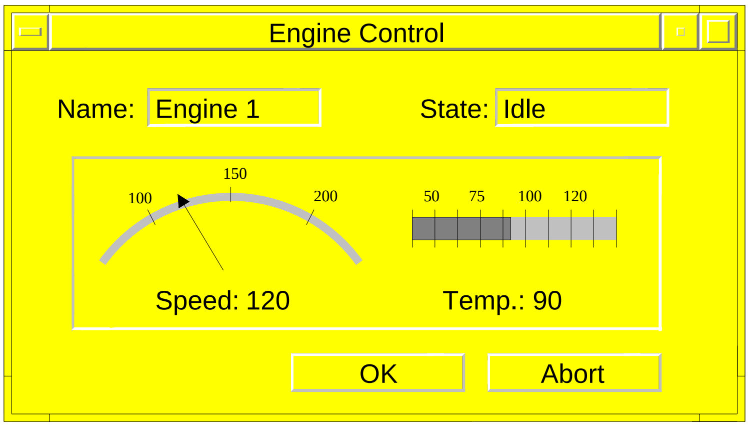

Figure 3: Typical mask

- A *menu* is an image for an object component of an enumeration type, each button shows a selectable value. Any kind of symbol with a boolean state value is usable 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, text fields, . . .).
- A *table* is an image of an array of objects or structures.
- A *text-document* is a picture with a special semantic and behavior.
- A *help-document* is a set of text-documents, connected together.
- A *hierarchical graph* is a set of pictures with a predefined behavior.

Some presentation objects can be build automatically by the service if the type of the corresponding object is known at runtime.

3.2 Example

Figure 3 is a mask-object, defined for the given data type ”EngineControl”. The components ”Speed” and ”Temp” are shown twice: First, using simple text-symbols and second with a graphical symbol. In this example, they are only used for output, although it can be made possible, to change a value by modifying the corresponding symbol, if this is useful in an application.

```
struct EngineControl {
    char        Name[ 20 ],
                State[ 20 ];
    short       Speed,
                Temp;
    };
```

4 EVENTS AND BINDINGS FOR DISTRIBUTED SYSTEMS

Presentations objects themself are useful for the manipulation and visualization of objects. But to allow the user a communication with an application, he must be able to interact with the entire system by creating events. In traditional user interfaces, the application needs an event-loop to recognize such events. The system is build ”around” this loop. This design is not very practicable in service-oriented applications due to the fact, that a service can be composed of many light-weight processes. To overcome this problem, a new mechanism, the *binding*, is introduced. A binding is defined as a connection between an operation and an event on a component of the user interface. The execution of the bounded operation is triggered by the event.

The main properties of this approach are:

- Many internal operations of the man machine service can be bound to events, so that typical user interactions with the system are definable by an interactive GUI tool (see below) without writing any line of code in the application.
- Presentation objects can be bound together to create hierarchical menus, masks and tables.
- User defined operations can be connected to events to create callback functions or methods (in C++). An application is able to catch an event using this technique.
- The interaction with the application is event-driven without the need of an event-loop. The mechanism is not limited to a local computer, it is available system-wide. This contains global call-back functions (or methods in C++) to remote systems.
- The bindings are changeable during runtime with the goal to allow permanent runtime-modifications of the GUI behavior.
- The dynamic behavior model is a portably described for different hardware architectures without the need of recompilation.
- The handling of all unbound events is simplified by applying default-bindings for these kinds of events. These means, that bindings are definable on more than one type of event or on every unbound event. Furthermore, unbound components of the user interface can be connected by bindings to implement a default-handler.

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

An interactive GUI editor allows the developer to create user interfaces, consisting of presentation objects and windows together with bindings in a comfortable way. For runtime or application defined interfaces a GUI creation or modification by the service API is possible.

5 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. This is implemented using distributed objects. The new defined standard for distributed objects ”Corba” is not used due to its requirement for a TCP/IP layer. This is not possible in some kind of automation applications.

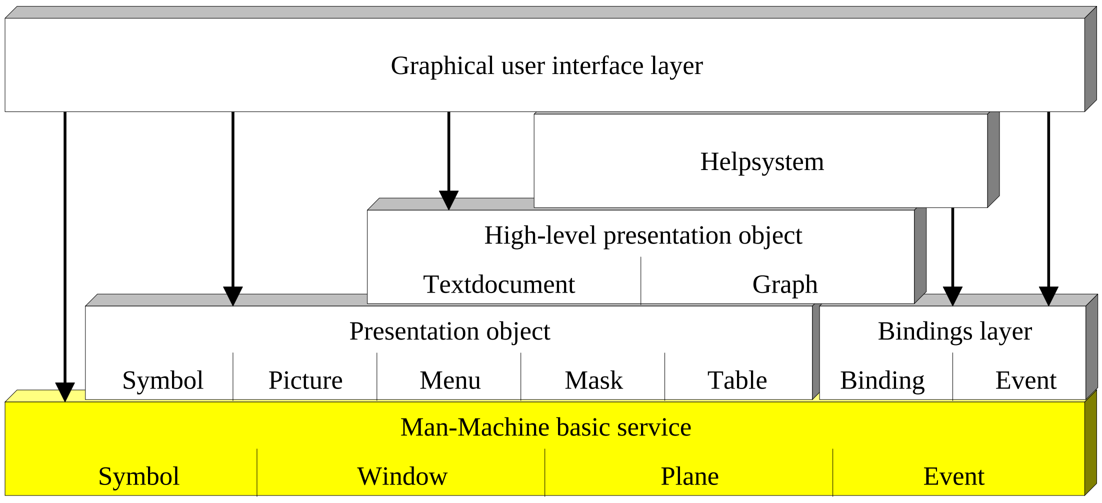

Figure 4: Internal MMS structure

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.[13]

Figure 4 presents the internal structure of the service without the persistent object layer.

6 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,[8] a traffic control system and a project management tool. Free definable symbols have reduced the need for an additional implementation in the application. Bindings are a powerful mechanism to implement a user controlled behavior and reduces the programming overhead.

7 ACKNOWLEDGMENT

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

8 REFERENCES

[1] Len Bass, Joelle Coutaz, ”Developing Software for the User Interface”, *Addison-Wesley Publishing Company*, 1990

[2] Steve Churchill, ”Fresco Reference Manual”, *Fujitsu Ltd.*, 1995

[3] Steve Churchill, ”Programming Fresco — A hands-on tutorial to the Fresco user interface system”, *Fujitsu Ltd.*, 1995

[4] Uwe Brinkschulte, Marios Siormanolakis, Holger Vogelsang, ”Visualization and Manipulation of Structured Information”, *In proceedings of the First International Conference on Visual Information Systems Visual'96*, February 1996, Melbourne, Australia

[5] Ralf Danzer, ”An Object-Oriented Architecture for User Interface Management in Distributed Applications”, *University of Kaiserslautern, Fachbereich Informatik*, 1992

[6] Richard P. Gabriel, ”Persistence in a Programming Environment”, *Dr. Dobb's Journal 12/1992*

[7] Gunnar Johannsen, ”Mensch-Maschine-Systeme”, *Springer*, 1993

[8] C. Lewerentz, T. Lindner, ”Case Study 'Production Cell'. A Comparative Study in Formal Software Development”, *Forschungszentrum Informatik, Karlsruhe, FZI-Publication 1/94*

[9] John K. Ousterhout, ”Tcl and Tk Toolkit”, *Computer Science Division, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720, Draft 8/12/93*, 1993

[10] Kapp, K.-H. and Siormanolakis, M.: ”EGF – Elementary Graphical Functions”, *Institute for Microcomputers and Automation, Internal Report, University of Karlsruhe*, 1994

[11] Julian Smart, ”Reference Manual for wxWindows 1.60: a portable C++ GUI toolkit”, *Artificial Intelligence Applications Institute, University of Edingburgh*, 1994

[12] Al Stevens, ”Persistent Objects in C++”, *Dr. Dobb's Journal, 12/1992*

[13] Holger Vogelsang, Uwe Brinkschulte, ”Persistent Objects In A Relational Database”, *submitted paper to ECOOP'96*, 1996

[14] Holger Vogelsang, ”Archiving System States by Persistent Objects”, *In proceedings of the International IEEE Workshop and Symposium on Engineering of Computer Based Systems ECBS'96*, March 1996, Friedrichshafen, Germany