Universität Karlsruhe
Institut für Mikrorechner und Automation
Prof. Dr.rer.nat. U. Brinkschulte
Dipl. Inform M. Siormanolakis
Dipl. Inform. H. Vogelsang

Haid-und-Neu-Str. 7
76131 Karlsruhe
brinks@ira.uka.de
sior@ira.uka.de
vogelsang@ira.uka.de

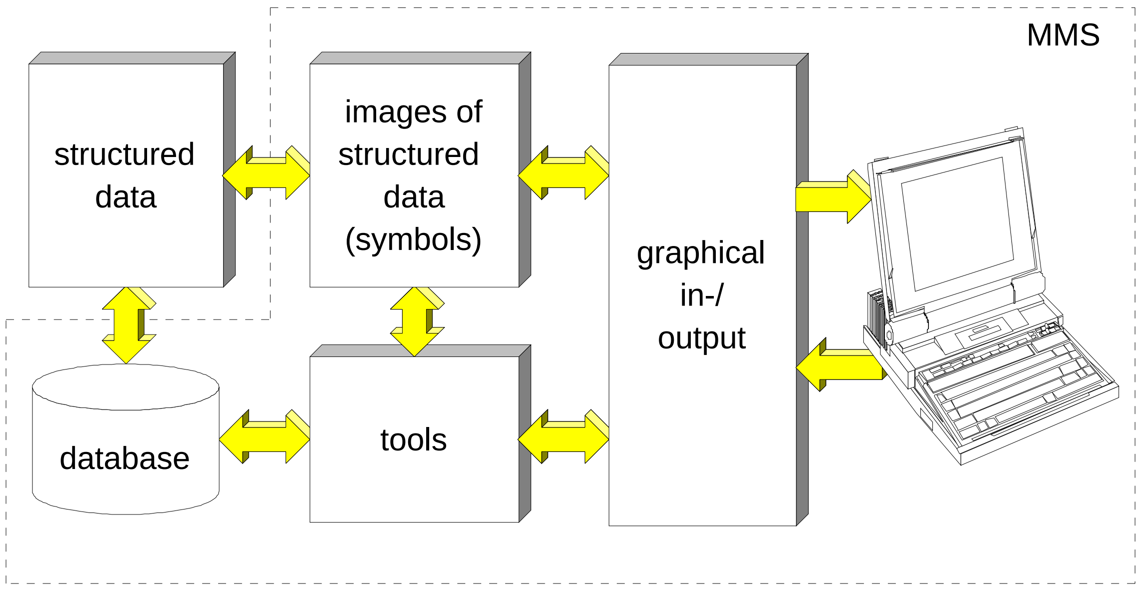
Man-Machine-Service

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
- the supported modification of structured information

Based upon these services any communication between user and machine can be realized. The following picture shows the flow of information from the structured data to the graphical display and reverse.



2 Symbols

A symbol is a graphical representation of a structured data-type. The user can define symbols very flexible with a tool called “Symbol-Editor”. The defined symbols are stored in a configuration database for the usage with the application. Symbols can be defined hierarchicaly — 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 discrete or continuous where we provide linear or logarithmic functions. Possible graphical modifications of symbols and base symbols are:

position: move a symbol or a part of it

scale: scale a symbol or a part of it

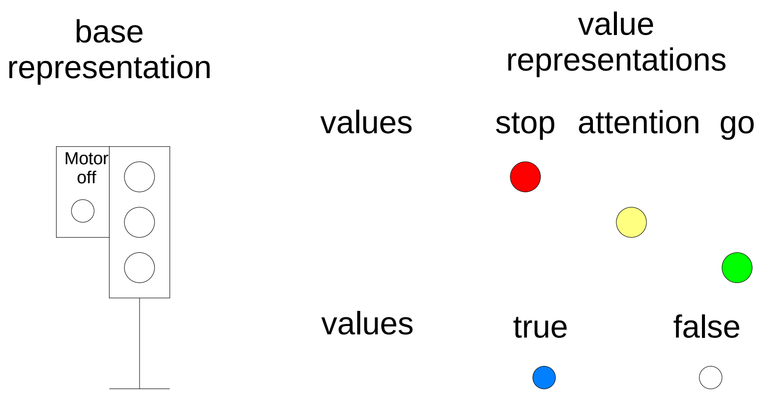
orientation: rotate a symbol or a part of it

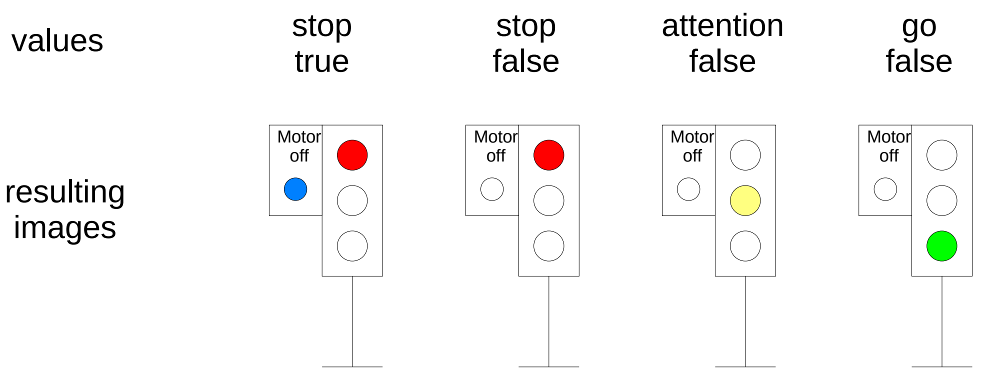
visibility: show/hide a symbol or a part of it

Base symbols have additional attributes like **linecolor**, **linetype**, **fillpattern**, **...** which can be modified. These attributes depend on the type of the base symbol — 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.

2.1 Examples

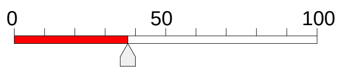
We want to visualize a signal light with a motor-off-sign. The structured data is a record of an enumeration and a boolean type. One for representing the state of the signal light and one for the motor-off-sign. We create a base representation for this symbol which isn’t changed by our values. Additional we define the images which represent our values. The values simply modify the visibility of the value representations. The following picture shows the images and representations we defined and the resulting images we get dependent on the state values.

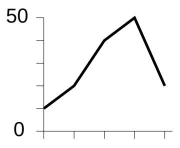




Two more examples for the possibilities of defining symbols for data-visualization follow:

- The value of the user's data-type is represented by a slider and a bargraph which are moved respectively scaled linear.
- A sequence of values ist represented in a histogram.

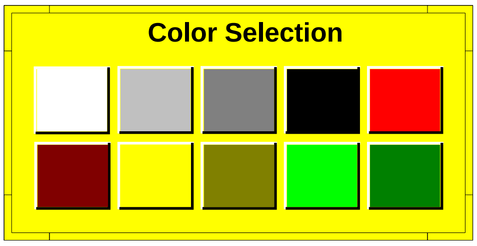
structured data type:	INTEGER
range:	0 ... 100
value:	38
resulting image:	

structured data type:	INTEGER
range:	0 ... 100
value sequence:	10, 20, 40, 50, 20
resulting image:	

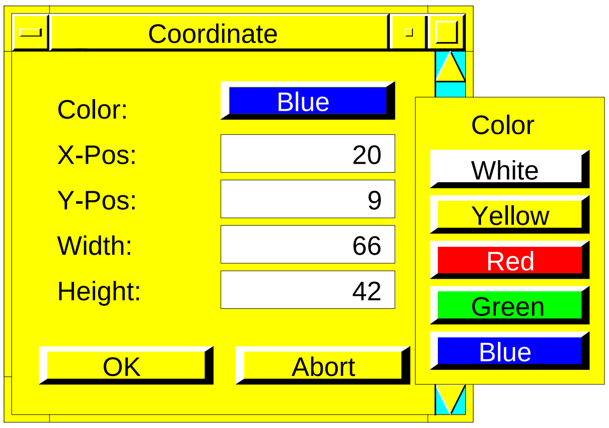
3 Special Symbols

Normal symbols can be used to visualize a big amount of user-defined data-types, but they aren't powerful enough to handle complex structured objects. Therefore a new type of symbol — the presentation-object — is introduced to offer the developer the facility to group symbols together, creating images of complex data-types. 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 for the object-types.
- 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.

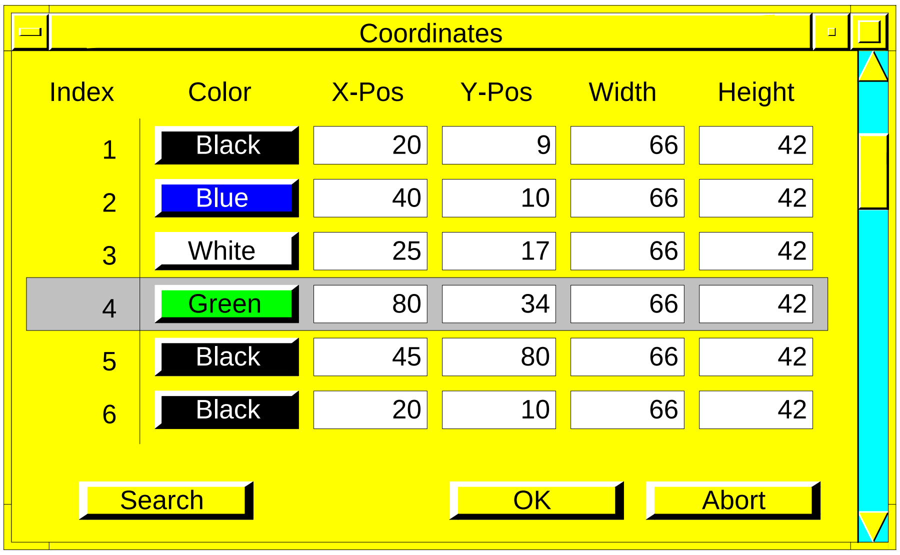


- A **Mask** is an image for an object: Modifyable components of the object can be changed by the manipulation of the corresponding symbols (sliders, buttons, textfields, ...)



Color:	Blue
X-Pos:	20
Y-Pos:	9
Width:	66
Height:	42

- A **Table** is an image of an array of objects.



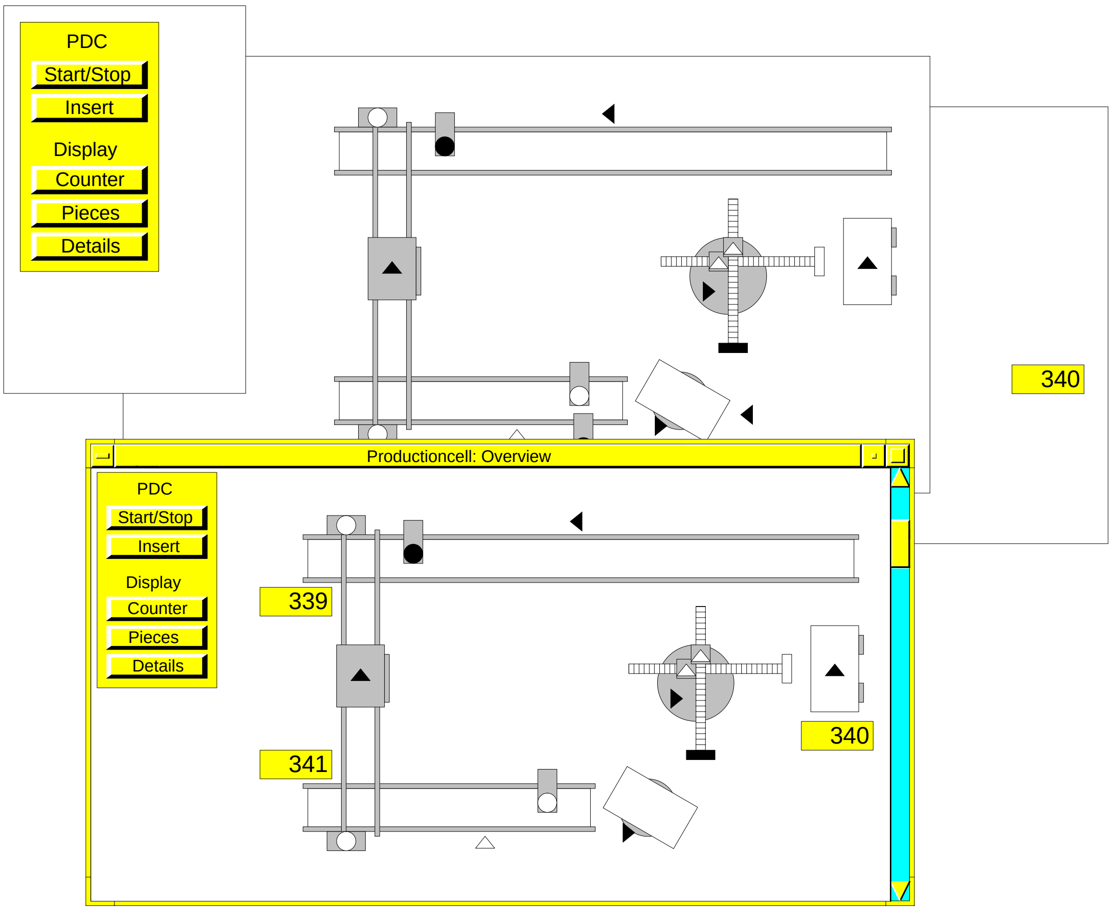
Index	Color	X-Pos	Y-Pos	Width	Height
1	Black	20	9	66	42
2	Blue	40	10	66	42
3	White	25	17	66	42
4	Green	80	34	66	42
5	Black	45	80	66	42
6	Black	20	10	66	42

Presentation-objects can be build automaticly by the service if the type of the corresponding object is known. Because every presentation-object is derived from the same common class, they share the same (small) set of operations.
Based on the *Picture* there are two higher-level presentation-objects, which are usefull in many application:

- Hierarchical graphs as an image of objects with relations and
- Textdocuments, consisting of symbols representing text and other graphical images.

4 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. We call such areas **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 can be changed. The following picture shows an example with three planes displayed in one window.



5 Bindings

It is possible to create *bindings* between symbol-events and operations or between events on presentation-objects and operations. The main ideas behind this 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.

6 Graphical User Interface

The graphical user interface is a group of presentation-objects and windows, which are needed at the same point of time to solve a given task. It is placed in a database to seperate 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.

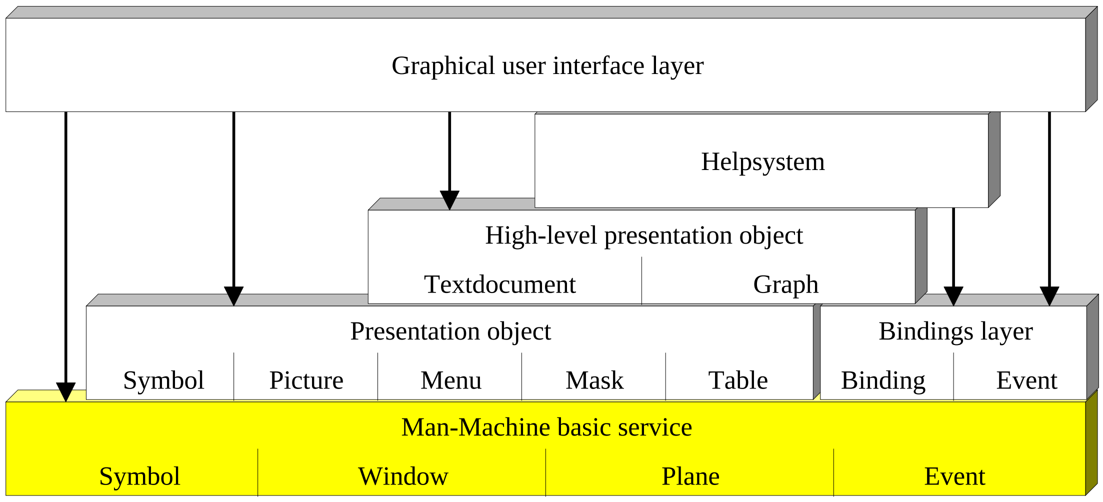
7 Tools

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

- The **symbol-editor** is an interactive tool, which enables the user to construct symbols as an image of predefined data-types. Furthermore: It makes it possible 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 and windows. A library-management simplifys the reuse of prior constructed objects.

8 Summary

The following picture shows the component structure of the entire man-machine-service. The features of the system are appended.



- **Plattform independence**

  Several systems are already supported: X11 (Sparc, Linux-386, Mips), DOS, Windows 3.11

- **Resolution independence by planes**

  Symbols are arranged in planes, using a resolution independend coordinate system.

- **Symbols**

  Complex symbol-types make it easy to handle state display and manipulation. The construction of presentation-objects as groups of symbols is possible.

- **Tools and editor services**

  The use of the symbol-editor and gui-editor enables the user to create complex symbol-types and user-interfaces. These editors are also available as services to be used inside an application.

- **Separation of gui and application**

  Gui's are stored in a database to ensure reusability.

- **Client/Server-Concept**

  Distributed operation in heterogeneous systems: With a given communication plattform it is possible to place the service on different hardware systems independent of the functional specification of the given task.

- **Recording and playing of sounds**