A New Man-Machine-Interface at BESSY

R. Müller

Berliner Elektronenspeicherring-Gesellschaft für Synchrotronstrahlung m.b.H. (BESSY), Lentzeallee 100, 1000 Berlin 33, FRG

H.–D. Doll[*] *I. J. Donasch, H. Marxen*[*]*, H. Pause*

Gesellschaft für Oberflächenanalytik und Computertechnologie m.b.H. (SPECS), Voltastraße 5, 1000 Berlin 65, FRG

Abstract

A UIMS (*user interface management system*) has been developed, that is completely based on non-proprietary software. Central part of our UIMS are processes (*mapper*) that act as universal X–clients for each specified X–server. Mapper (graphic server) and applications (graphic clients) exchange requests by an event driven interface. The communication protocol is free from any graphical information. The most powerful mapper client is a *form interpreter*, that can be programmed to act as an equipment access server. Mapper and form interpreter allow to compose control panels and synoptic views of the machine with statements in a simple and comprehensible UIDL (*user interface definition language*).

Introduction

The Berliner Elektronenspeicherring-Gesellschaft für Synchrotronstrahlung m.b.H. (BESSY) operates an 800 MeV storage ring dedicated to the generation of synchrotron light in the VUV and soft X-ray region [1].

Currently a new control system based on a distributed computing environment is developed and gradually installed at BESSY [2]. It replaces the aged control system [3] of the running light source BESSY and has to serve as the kernel for the control system of the planned 3rd generation light source BESSY II. Standards (IEEE 802.3, 802.4) or industry conventions (TCP/IP, X11.4, etc.) are used wherever possible. Large high resolution, bitmap oriented colour graphic screens with mouse and keyboard will become the standard operator console.

For a period of time old and new control system have to be operated in parallel to supply a reliable and unperturbed operation of the running light source. That imposes constraints and structural elements to the new system. Different aspects of the old system have their effect on a new user interface:

- The mental image (metaphor) must contain the existing control structure.
- Appearance characteristic (the look) has to be as similar to the familiar system as possible.
- Interaction sequencing (the feel) must take into account the experience and habits of trained and skilled operators.
- Well established driving and diagnostic programs will be ported to the new system with minimal effort. In a first stage they go into operation running in an ordinary terminal window.

Therefore our development is more in a danger to produce a replica of the existing system on a modern platform than to have any innovative impact on developments in other laboratories.

Nevertheless the basic concept of our new man machine interface system is of general interest:

- Associated with every X11 display station is a central graphics server program we call *mapper*, whose services have to be claimed by any application program. That program hides representational aspects completely from the applications.
- User interactions can assign code fragments in an action language to variables of a special application program we call *form interpreter*. Any access to the equipment can be specified and evaluated within this action language.
- Only non proprietary software has been used. We are not bothered by licence policies and independent of any company.

The Graphics Server

Characteristic for any UIMS (*user interface management system*) is the separation of code that implements the user interface to an application and the code of the application itself. The specification of the user interface is supported at a high level of abstraction.

The most common approach is to 'paint' the user interface representation with an interactive editor. Semantics and specification of the interface to the application is usually added in a special purpose language. Only at this stage modularity of software, separation of code for the graphical tasks and the code of the application itself is given. A user interface builder program binds the elements to the final application program.

[*]Now at Dr.Brunthaler, Industrielle Informationssysteme, Berlin

0-7803-0135-8/91$01.00 ©IEEE

Every application encorporates its own representational part. It has to be recompiled whenever a change becomes necessary.

In our system modularity with respect to representation and application functionality is especially emphasized. The representational aspects are encapsulated within a separate server program we call mapper. From the application programs point of view the running graphic server (mapper) is the frontend process that 'knows' how to present application variables on the associated display and that reports user interactions in an 'understandable' way.

An application programmer, who wants to offer variables to user interactions has to build up and maintain a connection (based on a socket link) to the mapper of the desired interaction display. The corresponding software interface is a complex object we call *application form interface* (Fig. 1). Routines and data structures necessary for the dialog have to by linked to the application program and are provided by a library.

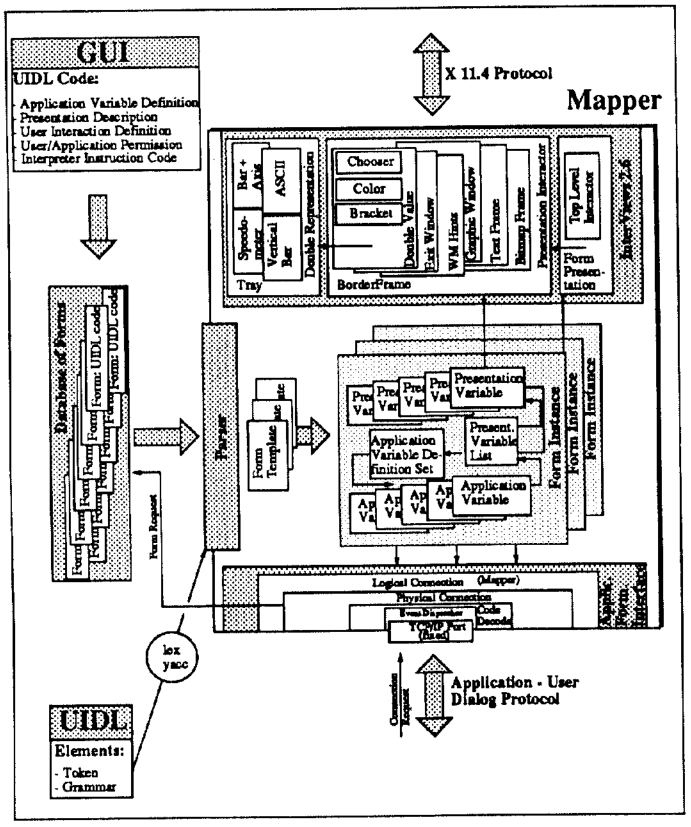


Figure 1: Details of the Graphic Server

Whenever an application that has no connection to a display requests access, the associated mapper reads specified configurational instructions, that are stored in a static (for the time of the applications life) database of files we call *forms* (Fig. 1). The forms specify assignments of graphical representations, user interactions and permissions to modify application variables. The syntax of the forms has been defined by *lex* and *yacc* source files.

A logical connection is build up between objects the mapper creates from the form template (the form instances) both within mapper and application (see Fig. 2 for a sample application program). The application specific data structures of the form instances and the eventdriven communication protocol form the interface between application and user interactions (mediated by the mapper process).

As long as the connection exists the mapper acts as a supervisor. Any event from application and user is reported to the mapper process. The mapper takes care of a consistent state of representational and applicational objects and variables. In the usual operation mode the mapper keeps the form templates in memory even after a connection has been cleared. After a phase of initialisation this type of 'down load' of the GUI improves response times drastically.

Presently the representational objects get their views and interaction feasibilities by means of the InterViews toolkit [4]. Slight modifications (pointer grabbing, polling) had to be introduced to the standard distribution to adopt the version 2.6 package to our needs.

Our first application program was a slider, that can be configured to control any analog device. Then no further individual application program has been written.

Generally application variables are used to notify to the application, that a specific action should be performed. On a next level of abstraction the code of the required action is no more part of the application, but it is assigned to the variable that should initiate the action. Evaluation and execution of the code is then done by an interpreter. The result of this abstraction is a programmable application we call *form interpreter*.

The Form Interpreter

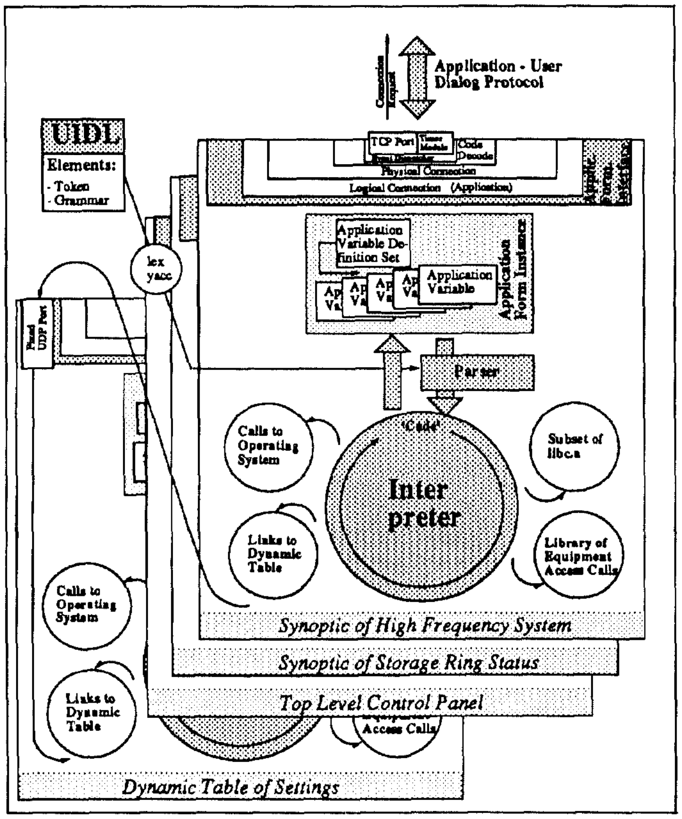
Initially the form interpreter application was intended to supply a flexible tool, that allows to compose a variety of control panels and synoptic views of the machine by means of the static forms. Representation of the form is still provided by the mapper, the description of the semantic aspects is interpreted by the form interpreter. The available sets of application variables reflect the usable functionality.

- Forms are handled (e.g. freeze, unmap, position).
- Subforms are interpreted and handled, trees of forms are generated and managed.
- Actions specified in an action language are started, the context of the actions is specified.
- Periodic actions are scheduled, time instances are reported.
- Background processes are started and terminated.
- Forms are submitted to different displays.
- Gateways to networks on the field level are selected.

The elements of the interpreted action language are very similar to those of 'C'. Certain restrictions in the declaration of variables have to be respected. Some statements like *switch* or *break* are not allowed, the use of labels needs some care. The syntax of the action language is very familiar to 'C' programmers. Built-in functions are some often used routines from the C library and utilities to change the operational mode or support the form handling of the form interpreter itself.

Communication with the equipment is enabled by the built-in library of equipment access calls. At BESSY this library is the standard interface between application programs and equipment. With the equipment access calls equipment can be checked and manipulated. Thus the composition of synoptic views and control panels becomes feasible.

In a client-server model the form interpreter is the equipment access server to abstract clients, namely the synoptic views and control panels specified in the database of forms.



Application - User Dialog Protocol
Connection Request
UIDL
Elements:
- Token
- Grammar
TCP Port
Timer Module
Code Decode
Physical Connection
Logical Connection (Application)
Applic. Form Interface
Application Variable Definition Set
Application Variable
Application Form Instance
lex yacc
Fixed UDP Port
Parser
'Code'
Inter preter
Calls to Operating System
Subset of libc.a
Links to Dynamic Table
Library of Equipment Access Calls
Synoptic of High Frequency System
Synoptic of Storage Ring Status
Top Level Control Panel
Access Calls
Dynamic Table of Settings

Figure 2: Copies of the Form Interpreter at Work

Another set of internal functions supports the interprocess communication needed for the dynamic configuration of equipment lists. These lists sets up the entries of a table managed by a specific form. That form displays (on demand or periodically) the settings of pieces of equipment, that have been entered to the list. The user can store and retrieve his favourite (once dynamically generated) equipment list as well as switch quickly between different lists.

In principle the slider control application could be realized simply by a form that programs the form interpreter to execute the desired actions. It will depend on the programs complexity and the programmers skills, whether it is advantageous to write a new application in a high level language to become a separate mapper client or in the interpreted action language within a new form.

The Graphical User Interface

A graphic editor that could be used to compose or modify synoptic views or panels is not provided for by our GUI system. The definition of our GUI is given in a textual form. Our own UIDL is especially appropriate to associate user interactions with actions that include equipment access. In our forms complex GUI objects are described in a comprehensible way. The tool, that is suited best to extend and maintain the GUI and that reflects the power and flexibility of our system is a plain text editor.

Based on two application programs, the form interpreter and the slider control, a GUI has been composed with about 5300 lines of statements in our UIDL, that provides all synoptic views and interaction tools to the operators, that are today available with the raster scan monitors, tracker balls, angle encoders and terminal screens of the old control system.

Summary

The concept of a graphics server and a form interpreter allows to shape the man machine interface in a descriptive way. Appropriate to the level of abstraction in our UIMS a specific UIDL had to be introduced like in other systems. This is a disadvantage because the usual programmer is not familiar with this language. But in contrast to commercial systems we are in a position to modify and extend language and functionality easily, since everything is present in source code.

The use of the object oriented toolkit InterViews for the realisation of graphical instances made programmers life easy. But to keep in step with current user interface developments it will become necessary to base the representations of our GUI on Xt and the OSF/Motif widget set. In principle this is no problem, but it means much work to compose the required graphical objects already offered by InterViews (e.g. *tray*, nonlinear deformation) within the usual Motif environment of Xlib calls, Xt elements and the OSF/Motif widget set.

References

[1] S. Bernstorff et. al., Physica Scripta, 36, 15 (1987)

[2] G. v. Egan-Krieger, R. Müller, Proceedings of the 2nd European Particle Accelerator Conference, Nice, pp. 872–874, 875–877 (1990)

[3] G. v. Egan-Krieger, W.–D. Klotz and R. Maier, IEEE Transactions on Nuclear Science, NS-30, 2273 (1983)

[4] M. A. Linton, J. M. Vlissides and P. R. Calder, *Composing User Interfaces with InterViews*, IEEE Computer, 8 (1989)