Graphics Server and Action Language Interpreter greatly Simplify the Composition of a Graphical User Interface.

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

Abstract

A GUI (*graphical user interface*) has been developed with a UIMS (*user interface management system*) that has been build up using exclusively non-proprietary software. A *graphics server* encapsulates completely representational aspects, mediates between user interactions and application variables and takes care of a consistent state of graphical and applicational objects. The most flexible and powerful client of the graphics server has a built-in interpreter section. On user interactions the graphics server passes code fragments in a C type language to this application program where the requested operations are executed. Graphical representations, semantics of user interactions and interpreter instructions are defined in a database written in a simple and comprehensible UIDL (*user interface definition language*).

1 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 gradually installed at BESSY [2, 3]. It replaces the aged control system [4] 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.

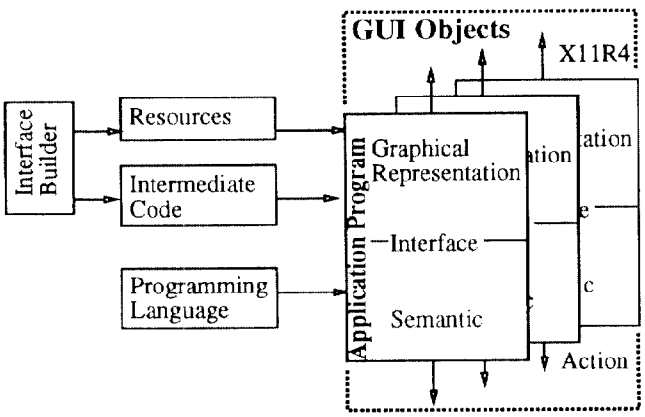

Figure 1: Products of an interactive Interface Builder

The new operator consoles are large high resolution, bitmap oriented color graphic screens with mouse and keyboard, the GUI (*graphical user interface*) is based upon the X-window system [5]. Within this graphic system the view of an elementary GUI object is typically a window that is most frequently composed of a tree of subwindows. It has certain representational elements, that can be further specified by resource descriptions, and a definite user interaction semantic.

The complicated task of building GUI's with the X-window system is simplified by a couple of toolkit libraries. Further a reasonable number of commercial and public domain GUI builders are available. They are mostly based upon one of the toolkits and follow the corresponding style guide. The complex UIMS's (*user interface management systems*) isolate application code from the implementation of graphical representation by an intermediate dialog layer [7]. These support environments facilitate coding work for the end user without requiring extensive programming skill.

Interactive GUI builders usually start with the painting of windows. The semantics is then attached to the graphical entities using the interface typical to the builder. These GUI builders produce intermediate code (e.g. in C). After compilation the results are application programs for the GUI objects or trees of GUI objects (Fig. 1). Maintenance, developments and tuning have to deal with the intermediate code.

Non-interactive UIMS are partly programmable in a special purpose language (UIDL), that has to be compiled, partly interpreted by a run time system.

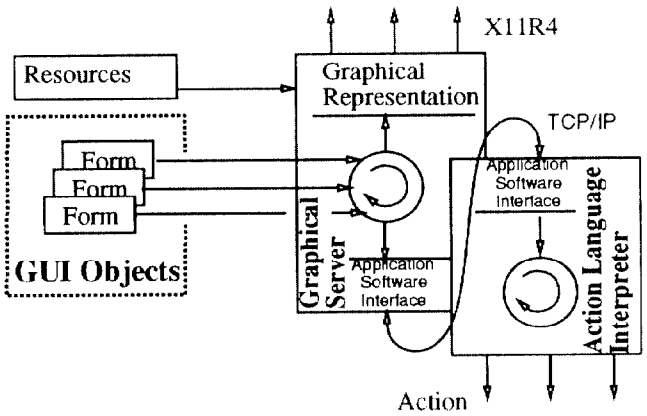

Figure 2: The BESSY non-interactive UIMS

The architecture of the non-interactive UIMS predominantly used at BESSY is characterised by an interpreting run time system. Two application programs provide the modularity with respect to representation and application functionality (Fig. 2). They are completely build up from non-proprietary software and present in source code.

The representational aspects are encapsulated within a sepa-

rate graphics server program we call *mapper*. They are realised with the InterViews toolkit [6] but completely invisible to any application program that addresses the graphics server via the application software interface. Semantic aspects are interpreted and the appropriate actions are performed by a flexible and powerful application program we call *action language interpreter*. The different instances of GUI objects are defined in plain ascii description files we call *forms*.

2 GRAPHIC SERVER

Presently the presentational objects are derived from classes of the InterViews toolkit (Fig. 3).

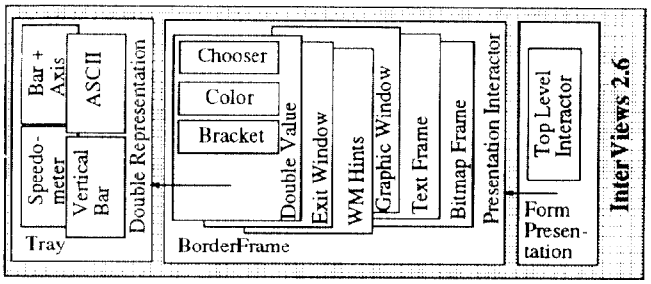

Figure 3: Elements of the Presentation Layer

Central object of the graphic server are the data structures of application variables and the interaction protocol of the event driven dialog layer.

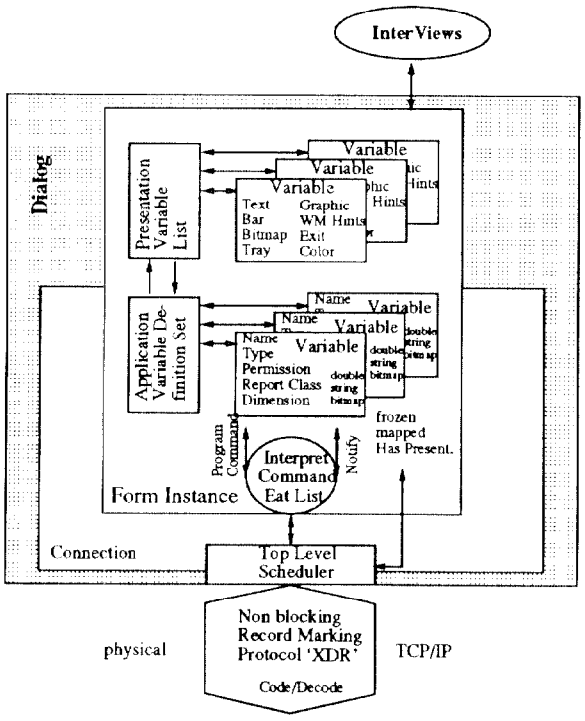

Figure 4: Dialog Layer: Graphic Server Side

On the request to put a GUI object into action the mapper parses the associated form and builds up the *form instance* [3]. These linked list data structures separate representational aspects from the semantic aspects of the variables (Fig. 4). A library provides the application with the methods to build up the corresponding dialog data structure, the *application form instance* (Fig. 5). The major part of the representational work is then done by the communication between application and mapper instances of the forms. These objects are logically connected by a physical connection based on an XDR type protocol on a TCP/IP socket link.

Application programs have no means to change the form objects by themselves. Every desired effect is coded into a series of requests sent to the mapper. If they are legal and executable, the mapper synchronizes the change of the application variables in both form instances by update reports as soon as they are processed.

Events from user interactions (keyboard, mouse) are handled by InterViews *Interactors* (Fig. 3) and passed to the application variable referenced by the presentation variable list (Fig. 4).

3 ACTION LANGUAGE INTERPRETER

This general purpose mapper client consists of the software interface and an interpreter section. Due to the eventdriven dialog layer it has no apparent sequence flow. The program is best described by the data structures and functions of its C++ classes addressed by handle routines (Fig. 5).

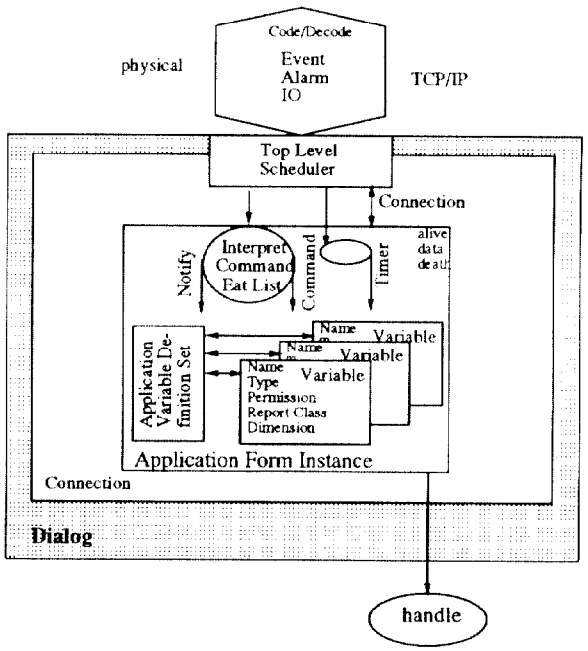

Figure 5: Dialog Layer: Applications Side

Form Handler

This class is central and unique to the running program. Here the global variables, actions, window interaction stati (freeze, unmap), alarm event instructions (timer) of all known GUI entities are referenced, time instances are reported and stored.

Subforms are interpreted and handled, trees of forms generated and managed.

Name Handler

Since mapper and application are separate programs that might run on different machines, pointers or addresses are unusable references. In this base class objects are identified by their names. Derived name handler classes are the sets 'subform' and 'action'.

Action Manager

In this set actions and their variables are administered by an action list. Actions are started by user requests or timer events. For all local variables the specification of the context (PARENT, HIGHLIGHT, DISABLE, CURRDIM, MAXDIM) is evaluated.

Expression Tree Handler

This is the implementation of the proper interpreter section. Language elements are the declaration of variables by the definition of simple types (*char, short, int, long, double*) and structures (*mesg, emesg* from the message database, *syscls_stat* etc. from the equipment access calls) or the definition of expressions like *goto, if(...), for(...;...;...)*. Statements and Syntax are very similar to 'C'.

All functions of the library of equipment access calls, some of libc.a like *strcmp(), strncpy(), exp(), log()* and some specific internal functions (*fupdate, exec_progs*) are built in.

Special Variables

In the forms the usable interpreter functionality is addressed by special variables (e.g. FI_ACTION, FI_AUTOTIME). In this class the corresponding semantic is implemented.

Two very specific variables are handled here: They allow for the change to another communication partner, namely the mapper of another display the next GUI entity should contact and the field level network deamon on another host that forwards the associated equipment access calls.

Timer Event Handler

The alarm event handler schedules the subsequent action on the periodic action list, i.e. *autoaction()*.

Dynamic Table IO

Servers for the maintenance of dynamic lists are started and configured. Equipment names incoming on an UDP socket of the dialog layer are added to the current equipment list, the actual list can be saved, existing configurations restored from disc.

Process Control

Programs are started by *fork()* and *exec()*, the return code is passed to the caller. This class keeps track of the associated process ID's and provides a method to terminate all child processes to the parent form.

Debug Module

Debug flags specific to the interpreter modules previously described allow for a convenient tracing of command flow.

4 GRAPHICAL USER INTERFACE

Based on the two application programs described above, a complex GUI has been composed with about 5300 lines of statements in our UIDL, that provides all required synoptic views and interaction tools to the operators.

Control panels provide push buttons for the commands, scrolled text fields for messages, chooser and edit fields for the input of strings. Color fields display status information. Analog values are shown as horizontal or vertical bars. A comparison of set points and actual values can be done by arbitrarily configurable lists. As layout elements simple graphical elements are provided.

5 SUMMARY

A run time system of graphics server and action language interpreter connected with a GUI definition in a description database has several advantages in comparison to the usage of most interactive GUI builders:

- No understanding of the underlaying window system is required. The GUI is easy to maintain and to extend.
- Graphical representations are already implemented, tested and stable.
- Adressing of graphical entities is simplified. Only a selection of needed and used graphical objects is implemented and set up properly.
- Flexible load management is possible. Mappers and the form interpreters can run on any host in the network.
- Maintenance requires only changes in ascii files and no recompile.

The bandwidth of the mapper–application connection which could be a matter of concern is perfectly satisfying.

6 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] R. Müller et. al., Conference Record of the 1991 IEEE Particle Accelerator Conference, San Francisco, pp. 1311–1313 (1991)

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

[5] R. W. Scheifler, J. Gettys, *The X window system*, ACM Transactions on Graphics, 5, 79 (1986)

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

[7] Robert Seacord, *User Interface Management Systems and Application Portability*, EUUG Newsletter, Vol. 10, Nr. 4 (1990)